COMP3411/9814 Artificial Intelligence
Term 1, 2022

DRAFT

Assignment 2 – Prolog and Machine Learning

Due: Week 10 - 10pm Friday 22 April

Marks: 20% of final assessment for COMP3411/9814 Artificial Intelligence

This assignment has two parts: Prolog programming and using a machine learning toolkit. The first part will be submitted as a Prolog source file and the second part as a PDF file.

Part 1 - Prolog

At the top of your file, place a comment containing **your full name, student number and assignment name**. You may add additional information like the date the program was completed, etc. if you wish.

At the start of each Prolog predicate, write a comment describing the operation of the predicate.

A significant part of completing this assignment will be testing the code you write to make sure that it works correctly. To do this, you will need to design test cases that exercise every part of the code.

It is important to use *exactly* the names given below for your predicates, otherwise the automated testing procedure will not be able to find your predicates and you will lose marks. Even the capitalisation of your predicate names must be as given below.

This assignment will be marked on functionality in the first instance. However, you should always adhere to good programming practices in regard to structure, style and comments, as described in the Prolog Dictionary. Submissions which score very low in the automarking will be examined by a human marker, and may be awarded a few marks, provided the code is readable.

Your code must work under the version of SWI Prolog used on the Linux machines in the UNSW School of Computer Science and Engineering. If you develop your code on any other platform, it is your responsibility to re-test and, if necessary, correct your code when you transfer it to a CSE Linux machine prior to submission.

Your code will be run on a few simple tests when you submit. So, it is a good idea to submit early and often so that potential problems with your code can be detected early. You will be notified at submission time if your code produces any compiler warnings. Please ensure that your final submission does not produce any such warnings (otherwise, marks will be deducted).

Question 1.1: List Processing (4 marks)

Write a predicate `sumsq_even(Numbers, Sum)` that sums the squares of only the even numbers in a list of integers.

Example:

```
?- sumsq_even([1,3,5,2,-4,6,8,-7], Sum).
Sum = 120
```

The example computes 2*2 + (-4)*(-4) + 6*6 + 8*8). The sum for an empty list should be 0.

To decide if a number is even or odd, you can use the built-in Prolog operator **N mod M**, which computes the remainder after dividing the whole number **N** by the whole number **M**. Thus a number **N** is even if the goal 0 is **N mod 2** succeeds. Remember that arithmetic expressions like **X + 1** and **N** mod **M** are only evaluated, in Prolog, if they appear after the is operator. So 0 is **N mod 2** works, but **N mod 2** is 0 doesn't work.

Question 1.2: Planning (5 marks)

Modify the simple planner from the week 5 lecture so that it can solve the planning problem in the textbook, Poole and Mackworth Section 6.1. Here, there is a robot can can move around an office building, pickup mail and deliver coffee.

Like the rubbish pickup problem, actions can be representation by a triple:

```
action(Action, State, NewState)
```

where the state of the world is represented by a quintuple:

```
state(RLoc, RHC, SWC, MW, RHM)
```

- RLoc - Robot Location,
- RHC - Robot Has Coffee,
- SWC - Sam Wants Coffee,
- MW - Mail Waiting,
- RHM - Robot Has Mail

You are required to replace the action specifications by a new set that specify the state transitions for each of the actions:

- mc - move clockwise
- mcc - move counterclockwise
- puc - pickup coffee
- dc - deliver coffee
- pum - pickup mail
- dm - deliver mail.

Do not alter the planner code, only the *action* clauses should be replaced.

You should only need to write one clause for each action, however, you might need some addition clauses to help with the *mc* and *mcc* actions. Think about how to represent the room layout.

Your program should be able to satisfy queries such as:

```
?- id_plan(
     state(lab, false, true, false, false),
     state(_, _, false, _, _),
     Plan).
Plan = [mc, mc, puc, mc, dc]
```

which asks the robot to get Sam coffee. Note that the values of the state variables, RHC, SWC, MW, RHM are boolean and we've used the constants **true** and **false**, to represent the boolean values. The initial state in this example has the robot in the lab, the robot does not have any coffee, Sam wants coffee, there is no mail waiting and the robot does not have any mail. The goal state is where the the Sam no longer wants coffee. Note that the anonymous variable, '_', indicates that we don't care what the other values are.

To test your action specifications, think about how you would ask the robot to pickup mail. What about if Sam want coffee and there was mail waiting?

Do not leave any testing code on your submitted file.

Question 1.3: Inductive Logic Programming

Duce is a program devised by Muggleton [3] to perform learning on a set of propositional clauses. The system includes 6 operators that transform pairs of clauses into a new set of clauses. Your job is to write Prolog programs to implement 2 of the 6 operators. One is given as an example to get you started.

Inter-construction. This transformation takes two rules, with *different* heads, such as

```
x <- [b, c, d, e]
y <- [a, b, d, f]
```

and replaces them with rules

```
x <- [c, e, z]
y <- [a, f, z]
z <- [b, d]
```

i.e. we find the intersection of the bodies of the first two clauses, invent a new predicate symbol, z, and create the clause $z \leftarrow [\boldsymbol{b}, \boldsymbol{d}]$. Create two new clauses which are the original clauses rewritten to replace $[\boldsymbol{b}, \boldsymbol{d}]$ by z.

To make processing the rules easier, we represent the body of the clause by a list and we define the operator `<-` to mean "implied by" or "if".

A Prolog program to implement inter-construction is given as an example to give you hints about how to write the two operators.

```
:- op(300, xfx, <-).

inter_construction(C1 <- B1, C2 <- B2, C1 <- Z1B, C2 <- Z2B, C <- B) :-
       C1 \= C2,
       intersection(B1, B2, B),
       gensym(z, C),
       subtract(B1, B, B11),
       subtract(B2, B, B12),
       append(B11, [C], Z1B),
       append(B12, [C], Z2B).
```

First, we define `<-` as an operator that will allow us to use the notation `x <- y`. The program uses Prolog built-in predicates to perform set operations on lists and to generate new symbols.

1. The first line the program assumes that the first two arguments are given as propositional clauses. The remaining three arguments are the output clauses, as in the example above.
2. inter_construction operates on two clauses that gave different heads, i.e. C1 \= C2.
3. We then find the interaction of the bodies of the clauses and call it, B.
4. **gensym** is a builtin predicate that creates a new name from a base name. So **gensym(z, C)** binds **C** to **z1**. Every subsequent call to gensym, with base z, will create names, z2, z3, …. Calling **reset_gensym** will restart the numbering sequence.
5. At this point the last argument `C <- B = z1<-[b, d]` because **C** is bound to **z1** and **B** is the intersection `[b, d]`.
6. The bodies **Z1B** and **Z2B**, are obtained by subtracting the intersection, **B**, from **B1** and **B2** and appending the single element **[C].** So we can run the program:

```
?- inter_construction(x <- [b, c, d, e], y <- [a, b, d, f], X, Y, Z).
X = x<-[c, e, z1],
Y = y<-[a, f, z1],
Z = z1<-[b, d].
```

obtaining the desired result.

You will write programs for the other five operators, defined below. All of these programs can be created from the builtin predicates used in the **inter-construction** code. So all you have to do is work out how to combine them. The only other builtin predicate you may need is to test if one list is a subset of another with **subset(X, Y)**, where **X** is a subset of **Y**.

You can assume that the inputs will always be valid clauses and the lists will always be non-empty, i.e. with at least one element.

Question 1.3 (a) (2 marks):

Intra-construction. This transformation takes two rules, with the *same* head, such as

```
x <- [b, c, d, e]
x <- [a, b, d, f]
```

and replaces the with rules

```
x <- [b, d, z]
z <- [c, e]
z <- [a, f]
```

That is, we merge the two, **x**, clauses, keeping the intersection and adding a new predicate, **z**, that distributes the differences to two new clauses.

```
?- intra_construction(x <- [b, c, d, e], x <- [a, b, d, f], X, Y, Z).
X = x<-[b, d, z1],
Y = z1<-[c, e],
Z = z1<-[a, f].
```

Question 1.3 (b) (2 marks):

Absorption. This transformation takes two rules, with the *different* heads, such as

```
x <- [a, b, c, d, e]
y <- [a, b, c]
```

and replaces the with rules

```
x <- [y, d, e]
y <- [a, b, c]
```

Note that the second clause is unchanged. This operator checks to see if the body of one clause is a subset of the other. If it is, the common elements can be removed from the larger clause and replaced by the head of the smaller one.

```
?- absorption(x <- [a, b, c, d, e], y <- [a, b, c], X, Y).
X = x<-[y, d, e],
Y = y<-[a, b, c].
```

Question 1.3 (c) (2 marks):

Truncation. This is the simplest transformation. It takes two rules that have the same head and simply drops the differences to leave just one rule. For example

```
x <- [a, b, c, d]
x <- [a, c, j, k]
```

are replaced by

```
x <- [a, c]
```

That is, the body of the new clause is just the intersection of the bodies of the input clauses.

```
?- truncation(x <- [a, b, c, d], x <- [a, c, j, k], X).
X = x<-[a, c].
```

The complete Duce algorithm performs a search, looking for combination of these operators that will lead to the greater compression over the database of examples. Here compression is measured by the reduction in the number of symbols in the database. You don't have to worry about the search algorithm, just implement the operators, as above

Part 2 - Machine Learning

Question 2.1 (5 marks): Machine Learning

For this question, you will the Weka machine learning toolkit to apply a variety of algorithms to a given dataset so that you can compare the results.

Go to the Weka website and download the program and see the accompanying documentation for how to use it. There is much more on the Weka website and it also has online help.

Use the data set **Soybean Data Set** that is provided as an example with Weka. The example data are in a directory called **data**, that is in Weka's installation directory. On the Mac, this is in /Applications/weka-3.8.6.app/Contents/app/data. On Windows it is in C:\Program Files\Weka-3-8-6\data. On Linux, you can install Weka in either your home directory or somewhere central like /opt or /usr/local.

You will try four learning algorithms:

- J48 - decision tree learner
- NaiveBayes - a naïve Bayes classifier
- JRip - a rule learner
- MultilayerPerceptron - a multilayer neural net algorithm

Create a table based on the output:

Algorithm	Accuracy (correctly classified instances)	Time to build model	Readablity
NaiveBayes			
Multilayer Perceptron			
JRip			
J48			

The last column is your subjective assessment of the readability of the output model.

References

1. Poole &Mackworth, Artificial Intelligence: Foundations of Computational Agents, Chapter 7, Supervised Machine Learning)
2. http://archive.ics.uci.edu/ml/datasets/Adult).
3. Muggleton, S. (1987). Duce, An oracle based approach to constructive induction. Proceedings of the International Joint Conference on Artificial Intelligence, 287–

292.

Submitting your assignment

This assignment must be submitted electronically.

Put your zID and your name at the top of every page of your submission!

give cs3411 assign2 part1.pl part2.pdf

- part1.pl should contain all the your Prolog code.
- part2.pdf should contain the table from part 2.

Late submissions will incur a penalty of 5% per day, applied to the maximum mark.

Group submissions are not allowed. By all means, discuss the assignment with your fellow students. But you must write (or type) your answers individually. **Do NOT copy anyone else's assignment, or send your assignment to any other student.**