MIDTERM 2 in Artificial Intelligence

Ladok code:	**TAIK19**
Deadline:	**2019-09-27, 23:59**

The exam is open book, open internet etc., except where explicitly stated. However, you are not allowed to use any form of human assistance. Specifically, it is, of course, strictly prohibited to collaborate with other students in any way. When submitting your answers, you are required to give a statement that you have produced all your answers yourself (see below).

Grading:
Midterm 1 and Midterm 2 consist of **20 points** each. Midterm 3 consists of **30 points**. The maximum total score is consequently **70 points**.

Requirements for grading:

3:	28 points
4:	42 points
5:	56 points

Submission:
You must submit your final exam before the deadline by uploading your answers to the assignment named MIDTERM 2 in Canvas. Please note that you should use the word document answers.docx and give your answers to the different questions in the designated areas. You may submit either the word document or convert it into a pdf.

Good Luck!

1) Consider a wumpus world, where the following holds:
   - The Wumpus is at [4,1]
   - There are pits at [1,3], [4,3] and [3,4]
   - The gold is at [1,4]

   The agent is currently at square [3,3] and wants to infer that it is safe to move to [2,3], i.e. one square down. Safe in this context means that there is neither a pit nor a wumpus at the square. Previously, the agent has visited squares [1,1], [2,1], [3,1], [2,2], [3,2], [4,2].

   a) Using Russell & Norvig's naming conventions for propositional symbols, describe, in propositional logic, the facts about the world that the agents has been able to construct *directly from its percepts* by visiting the previous and current squares.

   (1p)

   b) How can the agent infer that square [2,3] is safe? Describe the reasoning in a mixture of words and formulas. You may need to take into account some general facts about the wumpus world, as described in Russell & Norvig.

   (1p)

   c) If the inference in b) were to be implemented using truth tables, how many rows would that truth table have? Hint: Think carefully about which facts and general rules of the world are needed! Give a thorough motivation/explanation for your answer.

   (1p)

   Finally, you will carry out a proof.

   d) Prove, *using truth tables*, that the inference rule:

   $$\frac{\alpha \wedge (\beta \vee \gamma)}{(\alpha \wedge \beta) \vee (\alpha \wedge \gamma)}$$

   is sound.

   (1p)

2) Consider the following knowledge base, given in natural language:

   All geniuses are mad, but it's not the case that everyone who is mad is a genius. Einstein was a genius, as was Turing. Everyone who is ahead of their time are misunderstood. Both Einstein and Turing were ahead of their time. Turing did not receive a Nobel Prize, but Einstein did. Both of them have awards named after them. Anyone who is mad and misunderstood are happy if and only if they both receive a Nobel Prize and have an award named after them.

   a) Translate the knowledge base into first-order logic, using one FOL sentence for each sentence above. Carefully consider which predicates, functions and constants to use.
   (1p)

   b) We now want to prove that the assertion ”Einstein was happy, but Turing was not” follows from the knowledge base. Conduct this proof by reasoning about the semantics of the logical sentences you wrote for task a) above.
   (1p)

   c) Convert the formalized knowledge base to CNF and divide into clauses. You must follow the algorithm for conversion, which is given in section 9.5.1 in Russell & Norvig. Document every step of your conversion into clause form.
   (2p)

   d) Use first-order resolution to prove the assertion ”Einstein was happy, but Turing was not” from the knowledge base.
   (1p)

3) For this task you should implement a program that performs proofs in propositional logic, using forward chaining. You may use any imperative or object-oriented language, e.g., C, C++, Java(script), C# or Python. You may not use functional languages (Haskell, ML, LISP etc.) or Prolog. You must produce the code yourself, i.e. it is not permissible to find and modify an existing program for the task.

   The program must take exactly two inputs:

   1. a set of sentences in Horn clause form, i.e. a knowledge base KB
   2. a single sentence in Horn clause form, i.e. the query A to be proved or disproved from the knowledge base

   Your solution should meet the following demands:

   - The program only handles sentences in the Horn clause form, represented as strings of the format: premise=>conclusion, where each propositional symbol is a single letter A-Z. Examples:
     - ABC=>D represents the clause $A \wedge B \wedge C \Rightarrow D$
     - A represents the clause A (i.e. the fact A)
   - Structurally, the forward chaining algorithm should be separated from the problem-specific part. In the final step you will test this by solving a different problem, which must be possible without changing anything in the forward chaining algorithm.
   - Your code should focus on clarity and readability, and must closely match the pseudocode for propositional logic forward chaining in chapter 8.

   First, test your program by running the example from Lecture 6, i.e.:
   KB = $\{A, B, A \wedge B \Rightarrow L, A \wedge P \Rightarrow L, B \wedge L \Rightarrow M, L \wedge M \Rightarrow P, P \Rightarrow Q\}$
   Query is Q.

   a) Hand in (copy-paste making sure that coloring and indentation are correct) the following:
      - Your main program, including the forward chaining algorithm.
      - Your problem-specific part.
      - A screen printout or similar showing the program reporting the solution to the forward chaining proof.

   (2 p)

   After you have run the sample problem above, you should replace the problem-specific part with the following problem instance:
   KB = $\{A \Rightarrow G, A \wedge G \Rightarrow F, C, C \Rightarrow A, F \wedge C \Rightarrow H, H \Rightarrow F, H \wedge F \Rightarrow M, M \wedge A \Rightarrow B\}$
   Query is B

   When doing so, you should not have to do any modifications of the main program, i.e., the forward chaining module.

   b) Hand in (copy-paste making sure that coloring and indentation are correct) the following:
      - Your new problem-specific part.
      - A screen printout or similar showing the program reporting the solutions from a forward chaining proof of the inference above.

   (1p)

PROLOG PART

4.1) Cut (1p)

Write a predicate split/3 that splits a list of integers into two lists: one containing the positive ones (and zero), the other containing the negative. For example:

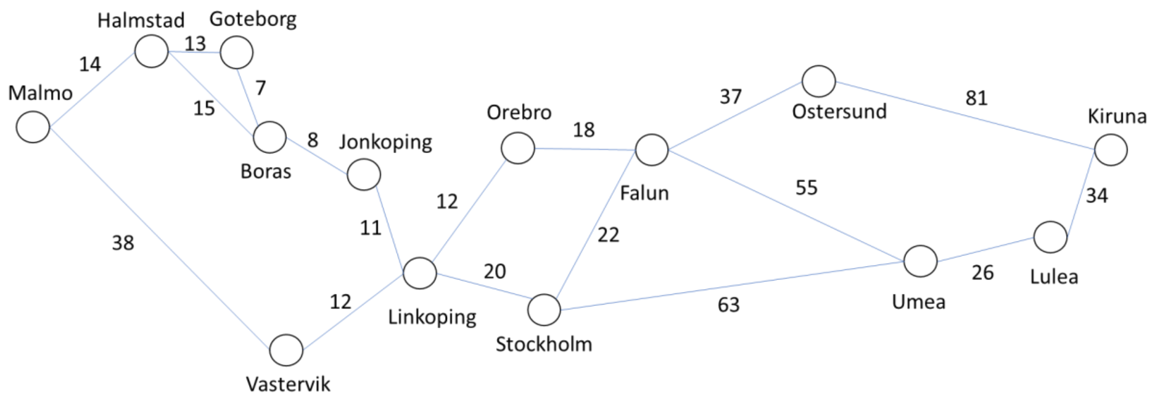
```
?- split([2,7,-1,-5,8,0,-6],P,N).
P  =  [2,7,8,0]
N  =  [-1,-5,-6].
```

You should use **cut** to make your program efficient!

4.2) Data Structure (1p for task a, 3p for task b)

a) Write a predicate/2 that collects the leaves in a binary tree. Use the atom **`nil`** to represent an empty tree, and the term **`t(L,X,R)`** to represent a non-empty binary tree where X denotes the root node and L and R denote the left and right subtree, respectively. Use cut to make your program more efficient if possible.

b) Have a look at the graph showing some Swedish cities above. The costs on the edges represent road distances. Write a program in Prolog that can return the path between two of the cities with a specific cost. If there is no path with the specific cost, fail.



4.3) Problem Solving (3p)

A farmer with a sheep, a goat and a bag of hay wants to cross a river. He has a boat that provides place only for him and one other object. He should not leave the sheep or the goat unsupervised with the hay. How can he manage the crossing? Write a program in Prolog to help the farmer to resolve the problem.

Note: You should comment your program in such a manner as to easily describe the purpose of the code.