CS3050: Practical 1, Programming with Prolog

Assignment: P1 - Practical 1

Deadline: 16th October 2020

Weighting: 45% of coursework weight

Please note that MMS is the definitive source for deadline and credit details. You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. You must contact the lecturer regarding any queries well in advance of the deadline.

---

1 Objective

This practical is aimed at designing and implementing automated reasoning solutions.

1.1 Competencies

- Develop experience in programming in Prolog
- Develop experience in modelling a logic problem
- Develop experience in using an automated reasoning engine

2 Practical Requirements

You are required to implement the following three exercises using GNU Prolog[1]. Each solution should be implemented in a different file called respectively **'ex1.pl','ex2.pl','ex3.pl',** etc. Please see Section 3.1 for information on how to structure the submission.

2.1 Exercise 1 - Structure of an organisation

Consider the following scenario in the context of the structure of a hypothetical university[2]. Provide sufficient facts to illustrate this scenario.

A hypothetical university is formed by a number of Schools, Faculties, and Departments, and runs a number of Modules. Each Department is part of a School. Each School is part of a Faculty. Each Module is offered by one or more Departments, and for convenience we say in this case a module is part of the department. A Student can take a number of Modules.

Write appropriate clauses and provide sufficient respective queries to fulfil each of following requirements:

[1] GNU Prolog, http://www.gprolog.org, also available on the lab machines.

[2] This example takes partial inspiration from the University Faculties https://www.st-andrews.ac.uk/about/governance/faculties-deans/

1. Identify whether a department collaborates with another department when offering a module.
2. Identify whether a student is enrolled in at least one module.
3. Identify whether two different students are taking a module in common.
4. Print all modules offered within a given school.
5. Define the relationship “X belongs to Y” where X and Y are two entities among modules, schools, departments, and faculties. ‘belongs to’ indicates that one is part or subpart of another entity.
6. Define the relationship “X member of Y” where X is a student, and Y is an entity among modules, schools, departments, and faculties. ’member of’ indicates that if the student takes a module, they are a member of that module as well as a member of the respective school, department, or faculty.

Please note that with the exception of point (3) where students must be distinct, it is ok to have duplicates.

2.2 Exercise 2 - Proofs by Truth Tables

We intend to develop a system that helps compute the validity of propositional inferences with at most two premises, and three propositions P,Q,R. We use a proof system based on the semantics of the propositional formulae, using the following method:

- Form a truth table for the propositions, add a column for Premis1, add a column for Premis2, and add a column for the conclusion.
- Identify the truth value of the premises for each row in the table
- Identify the truth value of the conclusion for each row in the table
- If every row that satisfies the premises also satisfies the conclusion, then the inference is valid. This means that for every row in the table, if Premis1=True, Premis2=True, then then conclusion must also be equal to True, otherwise the inference as a whole is invalid. For convenience, assume that to show this last step, we use an additional column in the table, writing ‘ok’ if the condition in that row is satisfied, ‘invalid’ otherwise.

Examples of valid and invalid inferences are shown in Figure 1 and Figure 2.

Prop	Prop	Premis1	Premis2	Conc	Check?
P	Q	P	$P \to Q$	Q	
T	T	T	T	T	ok
T	F	T	F	F	ok
F	T	F	T	T	ok
F	F	F	T	F	ok

Figure 1: Inference $P, P \to Q \vdash Q$, valid

The steps to be implemented are as follows:

1. Assume you have been given already the following clauses defining true, false, negation, and the operator $\wedge$:

Prop	Prop	Premis1	Premis2	Conc	Check?
P	Q	P	$P \rightarrow Q$	$\neg Q$	
T	T	T	T	F	invalid
T	F	T	F	T	ok
F	T	F	T	F	ok
F	F	F	T	T	ok

Figure 2: Inference $P, P \rightarrow Q \vdash \neg Q$, invalid

```
and(P,Q):- P, Q.

% negation
my_not(P) :- call(P), !, fail.
my_not(P).

%define true and false
t.
f :- fail.
```

2. Define the propositional operators $\vee, \neg, \rightarrow$.
3. Define a clause with functor $table(\_)$ which takes as arguments propositions, premises and conclusions. This clause prints the required truth table one row at a time for each column "Proposition, Premis1, Premis2, Conc". We assume that there are always three propositions P,Q,R.
4. Extend the clause in point (3) with the additional column "Check" which prints 'ok' or 'invalid' depending on the current row of the table as described above.
5. Test your approach with queries similar to Figure 1 and Figure 2. You can also check for additional queries such as [$q, \neg p \vee q \vdash p$ not valid], [$r, \neg p \wedge \neg q \vdash r \vee p$ valid].
6. Provide two additional example queries to show that your system works correctly.

Please note that column headers do not need to be printed.

2.3 Exercise 3 - Logical Equivalence Tree

We intend to develop a proof assistant system that helps compute proofs using the simplification laws of propositional logic as seen in the lectures. This system assumes that a formula has already been parsed, and a binary tree representing it has been stored in the knowledge base. In particular, facts represent nodes of the tree where each node includes the following information: an Id representing the node, e.g. n_x; an element representing either a i) proposition if the node is a leaf or ii) an operator $\wedge, \vee, \neg, \rightarrow$ if this is an internal node; pointers to its two children, e.g., n_l,n_r. Note that 'void' can be used if the child of the node is not present, and for operator $\neg$ we assume that the right-hand side child is void. An example tree is provided in Figure 3. Assume that our proof assistant allows us to apply a law to a specific part of the logical formula. The system will transform the formula and print the new formula such that we will be able to check the result of the application of this law.

The system should achieve the following tasks:

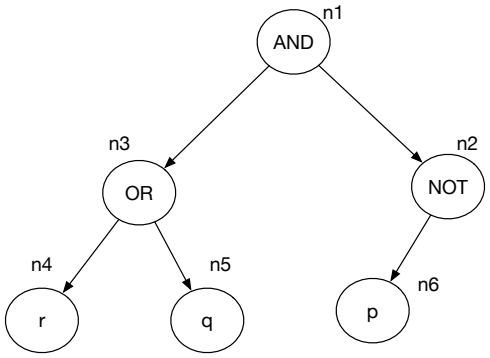

Figure 3: Equation Tree

1. Print the formula associated to the tree or subtree given a node Id. For example querying `printTree(n1).` should return $and(or(r, q), not(p))$. If the node points to non existing nodes in its subtrees, no formula should be printed and the query should fail.

2. We intend to apply a simplification law to a given **single** node Id in the tree, as a starting point. Provide a clause that modifies the tree according to the *And commutativity*, the first *De Morgan* law ($\neg(A \vee B)$), and *Or associativity* (One clause per rule). The new tree resulting from calling for example `and-commutativity(n1)` is shown in Figure 4.

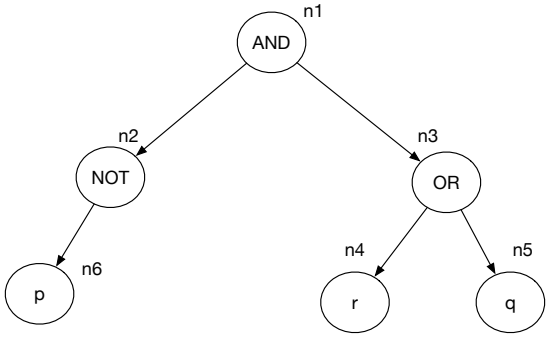

Figure 4: Equation Tree after calling and-commutativity

3. By calling print from step (1) we should now be able to see the new transformed formula. For example, after we have called the and-commutativity on $n1$, calling again `printTree(n1).` should return $and(not(p), or(r, q))$.

4. Provide at least three example queries (one per rule) of your system working.

For step (2) you might find useful to consider the built-in GNU prolog predicates `asserta`, and `retract`.

2.4 Additional Functionalities

It is strongly recommended that you ensure you have completed the Requirements in Exercise 1, 2, and 3 before attempting any of these requirements. You may consider **one or two** additional requirements for this assignment. Some suggestions are presented here, but you may also present your own requirements. Additional requirements should aim to demonstrate additional interesting uses of Prolog to extend the functionalities of the systems provided in the Exercises above. Please describe well the purpose of your additional requirements in the report and place your additional requirements in separate **ex?.pl** files.

- Extend Exercise 1 with additional elements in the scenario, for example Lecturers, Deans, Societies, Cohorts etc.. showing interesting results you can obtain through queries.
- In Exercise 2, we only focus on 2 premises and 3 propositions at the most. Extend the system to make this more flexible.
- In Exercise 3, we only focus on 3 laws. Provide additional simplification laws to make a more comprehensive proof assistant.
- In Exercise 3, we focus on the application of a law to a single node. Extend step(2), to apply the law on all nodes where the law is applicable.
- In Exercise 3, the input nodes have already been given as facts. Modify the system such that it is able to parse a logical equation.
- Adapt the approach developed in Exercise 3 for reducing the logical equation to a formula in CNF format.

3 Code and Report Requirements

3.1 Code and Libraries

Please develop your programs using GNU Prolog. The solution for each exercise should be saved on an appropriate file **'ex?.pl'**. These files should be placed on the top directory of your submission. For each exercise list your queries in a separate and appropriately named file **'ex?_test'**. The submission should only make use of standard Prolog instructions and GNU Prolog built-in predicates.

3.2 Report

You are required to submit a report describing your submission. The report should include:

- A brief checklist of the functionalities implemented
- If any of the functionalities is only partially working, ensure that this is discussed in your report
- Design: A description and justification for the functionalities implemented as part of your solutions
- Examples and Testing: Selected examples and queries of the functionalities implemented.
- Evaluation: A critical evaluation of your solutions and what can be improved
- Please cite any source consulted.

The suggested length for the report is between 1000–2000 words.

4 Deliverables

You should submit a ZIP file via MMS by the deadline containing:

- A PDF report as discussed in Section 3.2.
- Your implementation of the exercises (see Section 3.1).

5 Assessment Criteria

Marking will follow the guidelines given in the school student handbook (see link in the next section). The following issues will be considered:

- Achieved requirements
- Quality of the code and the solution provided
- Example queries
- Insights demonstrated in the report

Some guideline descriptors for this assignment are given below:

8-10	The submission implements fully the functionalities of exercise 1, adequately documented or reported.
11-13	The submission implements fully the functionalities of exercise 1 & some attempt to exercise 2. The code submitted is of an acceptable standard, and the report describes clearly what was done, with good style.
14-16	The submission implements fully the functionalities of exercise 1, and exercise 2. The code submitted is clear and well-structured, and the report is clear showing a good level of understanding. For the upper part of this band some attempt to exercise 3 should be provided.
up to 17	The submission implements fully the functionalities of exercise 1, exercise 2 and exercise 3. It contains clear, well-designed code, together with a clear, insightful and well-written report, showing in-depth understanding of the solutions presented.
above 17	The submission implements fully the functionalities of exercise 1, exercise 2 and exercise 3, with one or two additional functionalities completed. The submission demonstrates unusual clarity of design and implementation, together with an outstandingly well-written report showing evidence of extensive background reading, a full knowledge of the subject and insight into the problem.

6 Policies and Guidelines

Marking: See the standard mark descriptors in the School Student Handbook

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_-Descriptors`

Lateness Penalty: The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#latenesspenalties`

Good Academic Practice: The University policy on Good Academic Practice applies:

`https://www.st-andrews.ac.uk/students/rules/academicpractice/`

Alice Toniolo & Susmit Sarkar

cs3050.lec@cs.st-andrews.ac.uk

October 1, 2020

New version October 3, 2020