Jordan Rowley
Computer Science
University of St. Andrews

CS3050 Practical 1 Report

Programming with Prolog

In exercise 1 I have implemented a knowledge base of facts and then a variety of clauses to satisfy the practical requirements. As displayed in the ex1_test file, all 6 of the requirements have been fulfilled. In exercise 2 I have defined the additional propositional operators ∨, ¬, →, and then built out some propositions to facilitate printing the table, generating the table, and evaluating clauses. As shown in ex2_test all 6 requirements have been fulfilled. In exercise 3 I have implemented the printTree query and the three simplification laws from part 2 (and-commutativity, first-de-morgan, and or-associativity). I have fulfilled requirements 1, 2, and 3 for this exercise unfortunately leaving out requirement 4.

For the 'Structure of organisation' part of the practical, I equated the university structure hierarchy to a family tree hierarchy in which elements have 'parents'. A student is related to their modules through a studies(Student, Module) functor. A department collaborates with another on a module if they are both 'parents' of the module. A student is considered 'enrolled' when they are taking at least one module which is checked using an anonymous variable; studies(Student, _). To determine if two students share classes, the students must be unique from one another and they must both study a module in common. To print all modules given by a school I printed all 'grandchildren' of the school. I defined the 'belongs to' relationship using the parent functor and a recursive call to the belongs functor. This looks through all descendants of X looking for Y. I took a different approach for the 'memberOf' relationship. Since I knew that the tree has a fixed depth, I hard-coded in the relationship between a student and each level of university hierarchy.

For the 'Proofs by truth tables' exercise, I started by defining the propositional operators. I then wrote an evaluate clause which evaluates another clause in a variable to true or false. The table functor has an arity of 6 with parameters being: P, Q, R, Premise1, Premise2, and

Conclusion; one for each row ow the truth table. The table function prints column labels out and then calls generateTable with the same parameters to evaluate each element and write out the correct boolean value in each cell. Once this was all working I wrote the check clause which is just a simple if-else statement where if the premises are both true and the conclusion is false then invalid is written. Some examples of functionality are in ex2_test.

For the 'Logical Equivalence Tree' part of the practical, I changed the code design a few times during development and ended up showing a tree as a set of nodes and a set of branches. Initially, each node would store their ID, element, and pointers to two children but I divorced the pointers and the node data to help me understand how the recursion needed to work. My printTree(NodeId) is essentially a depth first search starting from a given node with some extra formatting so that the formula is printed out nicely. For the second requirement of this exercise I used retract to delete involved branches/nodes and asserta to replace the deleted part of the tree with new nodes and branches in accordance with the simplification laws. I didn't have time to provide test queries for these but logically they should work as expected.

In ex3 I could've implemented more checks for logical validity e.g if node is an and node then it must have two children.