COL226
Assignment 6

Jai Arora
2018CS50219

Implementation details→

1. The database from which the queries will be answered is stored in a file, and the filename is passed as a parameter to the executable.
2. A lexing buffer is made from the input file channel, and they are passed to the first parser and lexer.
3. The object returned after parsing is of type myProg, which is a clause list.
4. Clause is of type term * (term list).
5. This database generated is passed to another function, which loops and asks for a query until the user enters the exit prompt.
6. The query for the database is taken via the read_line () function, so t**he query has to be entered as a single line. It is of the form g1,g2,g3,....**
7. The query is passed into a second lexer and parser, and an object of type query option is returned. A query is a term list.
    a. If the object was None → then the query was invalid, and the function asks for a query again.
    b. If the object was of the form Some(_) → The query is resolved with the database, and the function loops back to ask for another query.
8. The internal representation of the database and query uses data types defined in assignment 5.
9. The variables in the database have to be kept different from the ones in the query for proper functioning, so I **convert the variables of the database to such internal representation which are not allowed for Prolog variables, and keep the query variables as it is.**
10. Also, in the database, the variables should not be clashing between the clauses itself, so for the internal representation, I append the clause number in front of the variable name.This representation is not valid wrt Prolog, so the variables won't clash.
11. One change from assignment 5 → Earlier the "compose" function, which took 2 substitutions and composed them, was of the form: (compose s1 s2) return s1.s2, now it return s2.s1, which means s2 was applied first, and then s1.
12. Appropriate function were made to print a substitution, which is a (variable*term) list.

13. For query resolution, the database and the query are sent to the ‘resolveQuery’ function, which calls another helper function.
14. The helper function has the current query, original database, current un-searched database, substitution list and a list of variables in the original query as parameters.
15. The helper function returns a pair of two boolean values: (b1,b2), where b1 tells if at least one solution was found or not, and b2 denotes if I need to backtrack and find the next solution or not.
    a. If the query is empty, which is an indication that all the sub goals are resolved, then **the substitution list would carry exactly one substitution, as it is the leaf-call of the execution tree.** In this case I print the head of the substitution list, and ask for the prompt from the user.
        i. If the user enters “;”, then I return (true,true)
        ii. Else If the user enters “.”, then I return (true,false)
        iii. Else I ask for the prompt again.
    b. For printing the substitution, I use varList to remove all those variables which are not present in varList, as they are not supposed to be printed.
    c. If the query is non empty, then I look for a clause in the un-searched database, whose head unifies with the first goal in the current query.
        i. If the database is empty, then I return (false,true), as you should still backtrack and search for the solution in this case.
        ii. If the database is not empty, and the head of the clause at the start of this database does not unify, then I continue the search in the remaining database with the same query.
        iii. Else, the mgu is found, and the goal in the query is replaced with the rule of that particular clause.This mgu is composed with every substitution in the list, and **the un-searched database is replaced by the original database.**
        iv. These new parameters are passed into the recursive call. Based on the boolean pair (b1,b2) returned, the following actions are taken:
            1. If b2 is false, which means that user does not require more solution, I just return (b1,b2), so that other recursive calls do the same.

2. Else, I backtrack, and search further in the database as if those terms did not unify. Let (b3,b4) be the result of this call. In this case, I return (b1||b3,b4).

Execution Details:

1. A makefile is present in the folder. Upon calling “make”, all the files are compiled and a final executable is generated.
2. Upon calling “make execute”, the executable is called with the input file. The default input file name in the make file is “input.txt”.
3. Calling “make clean” removes the extraneous compiled files.