# CSCI 561: Foundations of Artificial Intelligence
# Instructor: Prof. Laurent Itti
# Homework #3: Inference in First Order Logic

## Introduction:

As mobiles, wearables and other tiny gadgets surround us more and more in our day to day life, we are sharing more and more information with enterprises that provide us with their services. But what do we lose in return? How much control do we have on over our private information? These question seem simple to answer at the first glance. You share what you want to share. But, it's not like that with the use of AI methods that can mine your shared data to infer your private non-shared personal information. In this assignment, we want to help users to find out what an enterprise can do with their data before giving permission to their applications to access that.

We are going to implement a backward chaining system that gets the rules of data-mining and the abilities that an enterprise has at its disposal. Then it will help customers to find out if a certain type of personal information can be extracted by that enterprise if it gains access to another set of information about the user.

## Problem:

You are given a knowledge base and a number of queries. Your job is to determine if the queries can be inferred from the knowledge base or not. You have to use backward chaining algorithm for solving this problem.

## Input format

You will be given an input file. Read the input file name from the command line.

The first line of the input will be the number of queries (n). Following n lines will be the queries, one per line. For each of them, you have to determine whether it can be proved form the knowledge base or not.

Next line of the input will contain the number of clauses in the knowledge base (m).

Following, there will be m lines each containing a statement in the knowledge base. Each clause is in one of these two formats:

1- $p1 \wedge p2 \wedge \ldots \wedge pn => q$

2- facts: which are atomic sentences. Such as p or ~p

All the p s and also q are either a literal such as HasPermission(Google,Contacts) or negative of a literal such as ~HasPermission(Google,Contacts).

- Queries will not contain any variables.
- Variables are all single lowercase letters
- All predicates (such as HasPermission) and constants (such as Google) are case-sensitive alphabetical strings that begin with uppercase letters.
- Each predicate has at least one argument. (There is no upper bound for the number of arguments). Same predicate will not appear with different number of arguments.
- See the sample inputs for spacing patterns.
- All of the arguments of the facts are constants. i.e. you can assume that there will be no fact such as HasPermission(x,Contacts) ( which says that everyone has permission to see the contacts!) in the knowledge base.
- You can assume that the input format is exactly as it is described. There are no errors in the given input.

## Output format

You need to create a file “inference.xxx” where ‘xxx’ is the extension for the programming language you choose. (“py” for python, “cpp” for C++, and “java” for Java). If you are using C++11, then the name of your file should be “inference11.cpp” and if you are using python3.4 then the name of your file should be “inference3.py”. The command to run your program would be the same as was for previous assignments. Make sure that your program runs on Vocareum.

You have to create a file named ‘output.txt’. For each query, determine if that query can be inferred from the knowledge base or not, one query per line. If true, print “TRUE” and if not, print “FALSE”. (without the double quotes)

## Some clarifications

- If you decide that the given statement can be inferred from the knowledge base, every variable in each rule of the proving process should be unified with a Constant. So, if you have something like A(x) -> B(John) , and you cannot find any x to fulfill the A(x) premise, you cannot say that B(John) is true.
- The knowledge base that you get is consistent. So there are no contradicting rules or facts in the knowledge base.
- If you run to a loop and there is no alternative paths you can try, report False. An example for this would be having just two rules 1) A(x) -> B(x) 2) B(x) -> A(x) and wanting to prove A(John) . In this case your program should report false. In other words, if all the alternatives for proving A(x) lead back to A(x), this is considered a loop thus you have to report false.
- There will be at most 100 queries and 1000 clauses in the knowledge base.
- Considering the size of knowledge base, we recommend using an indexing method for storing your knowledge base. Such as table-based indexing or tree-based index that you have learned in class.
- You are free to use any parsing tool such as LEX and YACC if you want. ( As long as it runs on Vocareum)