Lab 1: Backward Chaining and Tracing in Prolog

CMPS 3560 - Artificial Intelligence

Alberto C. Cruz, Ph.D.

Department of Computer and Electrical Engineering and Computer Science

# Prelab

Prelabs are short homework assignments that must be completed **before the start of lab.** Refer to the syllabus for procedures about how the prelabs are checked off. Consider the following graph representing an inference chain:

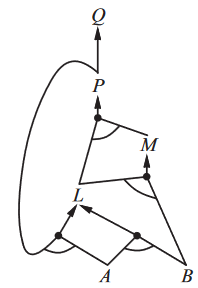


Figure : Prelab inference chain. Courtesy of Russel and Norvig, AI a Modern Approach.

1. Give the rules for the chain.
2. If A and B are fact, carry out backward chaining for the query: P.

# Introduction

## Goals

* Know how to use the Prolog interpreter gprolog or swiprolog
* Encode facts and rules in Prolog source code
* Invoke queries to a Prolog inference engine
* Trace a Prolog program to understand backward chaining

## Required Files

1. example.pl from Moodle

## Recommended Reading

1. Prolog slides from Moodle
2. Description of tracing in Prolog: <http://www.cse.unsw.edu.au/~billw/dictionaries/prolog/tracing.html>

## Background

Prolog is a declarative language first created in 1972 by Alain Colmerauer. A Prolog program consists of a set of statements called procedures. Statements are either a fact or a rule, similar to the description of rule-based expert systems in the text. Prolog it literally first order logic with a few exceptions. For this reason, Prolog lends itself well to implementing rule-based expert systems and will be our focus for the next three weeks. A user interacts with the Prolog interface, and the inference engine operates under the hood determining if the statement posited by the user is true or false. The only statements that are consulted, or executed, are statements that are goals or sub goals of the user’s statement. Thus, ordering of different rules in your source code does not effect when they are executed. *However, multiple declarations of a rule with the same name must be grouped together, and rules with the same name are visited from top to bottom (more on that later).* Each statement contains one or more clauses within the procedure which are ordered. The Prolog IDE we will use in class is gprolog*[[1]](#footnote-1)*, which is already on sleipnir. You can bring up gprolog by executing the following command while on sleipnir:

gprolog

You should see the following prompt:

GNU Prolog 1.3.0  
By Daniel Diaz  
Copyright (C) 1999-2007 Daniel Diaz  
| ?-

indicating that gprolog is waiting for you input. When you want to quit prolog execute the following command:

| ?- halt.

halt quits gprolog. Note the period. **The period terminates a gprolog procedure**. Your code must be entered in a text file separately and then loaded while running gprolog using the following command:

| ?- consult( ‘example.pl’ ).

which will load factorial.pl into gprolog.

### Facts

**Open up vi or emacs and take a look at example.pl.** As stated previously, you must write your code in a source file and then execute it later while running gprolog. Prolog code contains facts and rules. The following syntax is used to enter facts, also known as axioms and assertions:

car(gray).

This statement enters the fact ‘car’ is ‘gray.’ The following fact has multiple clauses:

color(a, orange).

This statement would be read as some object ‘a’ has the ‘color’ ‘orange.’ By convention, we will place the operator after the first clause. There are a few more facts. Try to understand them before moving on.

### Rules

Rules are encoded with the consequent first, followed by the operator :-, then the clauses of the antecedent separated by commas. For example:

hamburger(X) :- color(X,tan), shape(X,round).  
 full(X) :- ate(X, hamburger).

The first rule is interpreted as: IF ‘color’ is ‘red’ THEN ‘apple’ is ‘fruit.’ The second rule is interpreted as: IF ‘X’ ate ‘hamburger’ THEN ‘X’ is ‘full.’ Unlike the first rule, the second rule uses a variable. Variables in Prolog begin with an upper case letter. Invoking full(X) during runtime will check for all objects in the database that ate a hamburger. If you invoke full(jill) it will check if jill ate a hamburger. This is how you will implement queries in Prolog.

### Queries

**When you are done looking at example.pl, quit your text editor and then run gprolog and consult your source code (see above for the command to do this).** To check if a rule has been proven (e.g. it has fired/it is true/it is in the database) invoke the following command:

| ?- fruit(apple).  
true.

which attempts to prove that ‘fruit’ is ‘apple.’ Since ‘color’ is ‘red’ is fact, the rule fruit(apple) :- color(red). will always be true. This is an example of a ground query. To check if an object is full:

| ?- full(joe).  
false.

We know for a fact that ‘joe’ ate ‘salad’, so he can’t be full according to our rule. A non-ground query is a query invocation with a variable in it. Recall that variables start with an uppercase letter.

| ?- full(Y).

gprolog should respond with the following result:

Y = albert

It may seem like gprolog has hung at this point, but it is actually stepping through all objects which cause full(Y). to evaluate to true. To step through to the next object you must enter a semicolon, your screen output will look like this:

Y = albert ;  
Y = jill

This brief introduction should be all that is needed to complete the lab. For a glimpse into a more advanced Prolog program, here is an implementation of recursive factorial:

factorial(0,1).

factorial(N,F) :-

N>0,  
N1 is N-1,  
factorial(N1,F1),  
F is N \* F1.

Note that since clauses are ordered the antecedent can be written as if we were programming in an imperative language.

## Tracing a Prolog Source

Read recommended reading item #2 on how to trace a Prolog program.

# Technical Approach

## Inference Example from Textbook

To write a program in Prolog you will write your code in vi or emacs and then use gprolog to execute your code. Prolog code usually has the extension .pro or .pl. The following rules were discussed in Section 2.6.1 of the text:

Rule 1:

Rule 2:

Rule 3:

Rule 4:

Rule 5:

Enter these rules into your Prolog source code. Following the book’s example enter A, B, C, D and E as fact. Verify that all objects are fact except for N **using trace**. Be sure that you understand how backward chaining is being executed.

For check off, demonstrate to the instructor that you have implemented the example from the textbook, and execute a trace to determine if N is fact.

1. swiprolog will be used in CMPS 4560, but has a slightly different syntax for advanced commands that we will not use in this class. Feel free to use either. [↑](#footnote-ref-1)