Faculty of Computer Science, Dalhousie University *22-Nov-2024*

CSCI 4152/6509 — Natural Language Processing

Lab 9: Prolog Tutorial 1

Lab Instructor: Sigma Jahan and Tymon Wranik-Lohrenz
Location: Mona Campbell 1108 (10am)/Goldberg CS 134 (4pm)
Time: Friday, 10:04–11:25 and 16:05–17:25
Notes author: Vlado Keselj and Magdalena Jankowska

Prolog Tutorial 1

Lab Overview

- In this lab we will learn about the Prolog programming language
- Introduction to Prolog

Files to be submitted:

1. `gcd.prolog`
2. `prog1.prolog`
3. `factorial.prolog`
4. `task.prolog`
5. `task-queries.txt`

Prolog in NLP

- Creation of Prolog was linked to NLP
- Over time it stayed related to NLP, e.g., in Definite Clause Grammars
- Prolog backtracking makes it easy to implement backtracking CFG parsers
- Prolog unification is directly related to unification-based grammar formalisms
- Prolog and First-Order Predicate Calculus are related to semantic processing

Prolog: Programming in Logic

- PROLOG has unusual control flow
  - built-in backtracking
- Program is a problem description rather than a recipe
- Program paradigm known as **declarative programming**
- Running a program is equivalent to proving a theorem
- Based on the First Order Predicate Logic

Slide notes:

Prolog Origins

- Based on Mathematical Logic
  - First-Order Predicate Logic
  - use of Horn clauses
- Robinson 1965
  - method for resolution for machine theorem proving
  - two important concepts: Resolution, Unification
- Alain Colmerauer, 1970s
  - Prolog—Programming in Logic
- Robert Kowalski *et al.*, U. of Edinburgh
- An additional important Prolog concept:
  - built-in backtracking

Robinson published an influential paper in 1965, introducing a logic inference system with a very small number of inference rules, and as such it was more convenient than other systems to be implemented on a computer. The main inference rule used in his system was *inference* and an important operation used was the *unification.*

Prolog as a Programming Language

- A few more characteristics:
- Running a program is equivalent to proving a theorem
- Output: values of variables found during a constructive proof
- Program is a set of axioms
- No internal state, no side effects
- Automatic garbage collection
- Extensive use of lists
- Use of recursion

Pros and Cons of Logic Programming

- Pros:
  - Absence of side-effects
  - No uninitialized or dangling pointers
    (this should ensure more reliability, and easiness of writing, debugging, and maintaining code)
  - Built-in backtracking and unification
- Cons:
  - Lack of libraries, development support tools
  - Less portable, no interfaces to other languages
  - An alternative programming pradigm (not mainstream)
- Pros and Cons similar to functional languages

Comparison of Different Programming Paradigms

- Let us consider the following problem and how it would be solved in three different programming paradigms:
  - Example problem:
    Calculate GCD (Greatest Common Divisor) of two numbers.
- Paradigms:
  - Imperative programming
  - Functional programming
  - Logic programming

Imperative programming: Recipe: to compute GCD of a and b, check to see if $a = b$. If so, output one of them and stop. Otherwise, replace the larger one with their difference and repeat.

Functional programming: $gcd(a, b)$ is: If $a = b$ then a; otherwise, it is $gcd(\min(a, b), |a - b|)$.

Logic programming: To prove that g is GCD of a and b, either show that $a = b = g$, or find c and d such that c is the smaller number of a or b, d is the absolute difference of a and b, and g is GCD of c and d.

Sample Programs

```
public static int GCD(int a, int b) {  // Java
    while (a != b) {
        if (a > b) a = a - b;
        else       b = b - a;
    }
    return a;
}

(define GCD (a b)              % Scheme
    (if (= a b) a
        (GCD (min a b) (abs (- a b)))))
```

Sample Prolog Program

```
gcd(A,A,A).                    ; Prolog
gcd(A,B,G) :- A =\= B, C is min(A,B),
       X is A - B, D is abs(X),
       gcd(C,D,G).
```

Step 1. Logging in to server timberlea

We will now start with the hands-on part of the lab. We are going to use an open-source version of the Prolog interpreter called SWI-Prolog (command name 'swipl'). As before, our first step is to login to the server timberlea and prepare the appropriate directory.

- Login to the sever timberlea

As in the previous labs, login to your account on the server timberlea.

- Change directory to `csci4152` or `csci6509`

Change your directory to `csci4152` or `csci6509`, whichever is your registered course. This directory should have been already created in one of your previous labs.

- `mkdir lab9`
- `cd lab9`

Now, using the command '`mkdir lab9`' create the directory `lab9`. After this, you should make this directory your current directory using the command: '`cd lab9`'.

Step 2: Running Prolog

- Run SWI Prolog using command: `swipl`
- To exit Prolog type: `halt.`
- Run Prolog again
- Access to helpful documentation: `help.`
- First chapter of the manual: `help(1).`

– Help on a specific command: `help(halt).`
– Command to load a program is `['file'].` but we first need to write a program

Step 3: GCD Program

– Exit Prolog
  In this step, we will write a short program for calculating GCD (Greatest Common Divisor) in Prolog. We first exit the Prolog interpreter (command 'halt.'), and we now use an editor to write the program. We will assume that you use Emacs, but if you are more comfortable with another plain-text editor in Linux, such as vi, you ca use it.
– Prepare the file named `gcd.prolog` with the following contents:

```
gcd(A,A,A).
gcd(A,B,G) :- A =\= B, C is min(A,B),
        X is A - B, D is abs(X),
        gcd(C,D,G).
```

Running GCD Program

– Save the file and suspend (or exit) the editor
  You can now save the file. You could exit the editor and run the Prolog again. A more convenient setup is to suspend the editor (with Ctrl+Z key combination on keyboard), so that after running the Prolog interpreter and exiting it, you can bring back the editor with Linux command 'fg'. An even more convenient setup is to have two terminal windows open, and being logged in into timberlea on both of them. In one you can use editor, and in another one you can use the interpreter, without exiting any of them until the end.
  In any case, we assume that you are now in the Linux shell and ready to run the Prolog interpreter.
– Run the Prolog interpreter (command '`swipl`').
– Load the program using the command:
  `['gcd.prolog'].`
– There should be no errors reported, otherwise you need to exit the interpreter and fix the program.
– In Prolog interpreter type: `gcd(24,36,X).`
  and then: `;`

Submission

– Submit the file `gcd.prolog` using `nlp-submit` command
– It will be marked as a part of the next Assignment

Step 4: Prolog syntax

Constants

Constants in Prolog start with a lowercase letter, e.g.,

```
bill
car
```

Numbers

Numbers (integer or float) are used in Prolog as constants. e.g.,

```
5
7.1
```

Variables

Variable names start with an uppercase letter or an underscore ('_').

e.g.,

```
X
T1
_a
```

Anonymous variable

_ is a special, anonymous variable; two occurrences of this variable can represent different values, with no connection between them.

Predicate

A predicate can be considered a function. It is written as a string starting with a lowercase character, followed by `(`, followed by a list of arguments separated by commas, and followed by `)`, e.g.,

```
happy(george)
father(george,X)
```

Facts

A fact is a statement that a given predicate for given arguments is true:

```
happy(bill).
parent(bill,george).
```

These facts should be understood as: "happy(bill) is true", "parent(bill,george) is true".

If a fact contains a variable, it means that the predicate is true for any value of the variable, e.g.,

```
isFactor(X,X).
```

should be understood "for any value of X, isFactor(X,X) is true"

Rules

A rule corresponds to the following form of a logical formula:

$$p_1 \land p_2 \land \ldots \land p_n \Rightarrow q$$

where $n \geq 1$, and $p_1, ..., p_n, q$ are predicates for some arguments, e.g.,

```
happy(bill) :- jogging(bill),rested(bill).
```

should be understood: “if jogging(bill) is true, and rested(bill) is true, then happy(bill) is true”. Notice that `,` corresponds to $\land$, and `:-` corresponds to $\Leftarrow$

Rules usually contain variables. It means that the logical formula is true for any values of the variables, e.g.,

```
older(Y,X) :- isChild(X), isAdult(Y).
```

should be understood: “for any X and Y, if isChild(X) is true, and isAdult(Y) is true, then older(Y,X) is true”.

Prolog program

- a Prolog program is a collection of facts and rules
- it is called a knowledge base.
  For example:

  ```
  older(Y,X) :- isChild(X), isAdult(Y).
  isChild(bill).
  isChild(jane).
  isAdult(rob).
  ```

Querying Prolog knowledgebase

A query is typed after the Prolog prompt `?-`

- A query without a variable:

  ```
  ?- isChild(bill).
  ```

  means “Is isChild(bill) true?”
- A query with a variable:

  ```
  ?- older(X,jane).
  ```

  means “List all values of X such that older(X,jane) is true”

  ```
  ?- older(A,B).
  ```

  means “List all pairs of values of A and B, such that older(A,B) is true”

Step 5: Roland and Franklin Example

- Type a ‘roland and franklin’ example in a file named `prog1.prolog` with the following contents:

```
hare(roland).
turtle(franklin).
faster(X,Y) :- hare(X), turtle(Y).
```

- After loading the file, on Prolog prompt, type:

```
faster(roland,franklin).
```

The Prolog interpreter will simply respond with the answer ‘true.’ and print the prompt again.

Try `faster(X,franklin).` and `faster(X,Y).` and you will see that the interpreter will print the correct assignments for the variables `X` and `Y`.

The above example shows some basic elements of a Prolog program. The lines:

```
hare(roland).
turtle(franklin).
```

are called facts. In the above lines we want to express that the constant ‘roland’ is a hare and ‘franklin’ is a turtle. A convention in Prolog is that constants start with lowercase letters. The words ‘hare’ and ‘turtle’ are called predicates. Predicates are similar to functions or subroutines in other programming languages, and we recognized them since they are followed by a open parenthesis (‘`(`’), a list of arguments, and a closed parenthesis.

The next line in the above program is called a rule:

```
faster(X,Y) :- hare(X), turtle(Y).
```

It should be understood as the logical formula:

$$\text{hare}(X) \wedge \text{turtle}(Y) \Rightarrow \text{faster}(X, Y)$$

with the meaning that for any two expressions X and Y, such that hare(X) is true (or satisfied) and turtle(Y) is true, we can conclude, or infer, that faster(X, Y) is true. it is a rule, which we assume is always true. As we can notice, X and Y are variables, which can take values of arbitrary expressions. A convention in Prolog is that variables start with uppercase letters or an underscore character (‘_’).

Try examples:

```
?- faster(roland,franklin).
Yes ;
...
?- faster(roland,X).
X = franklin ;
...
?- hare(X).
X = roland ;
```

Step 6: Taking Courses

- Let us program the following rule:
  - If a student X is taking a course Y, and the course Y has lecture on a day D, then X is busy on D.
- In our database of facts, we will add the following facts:
  - a student named ‘joe’ is taking a course named ‘nlp’
  - ‘nlp’ has a lecture on ‘friday’
- We will see how Prolog infers that ‘joe’ is busy on ‘friday’
- You can notice how we use lowercase letters for constants

Taking Courses Code

- Instead of starting a new file, you can simply add the following code to the file ‘`prog1.prolog`’

```
busy(X,D) :- taking_course(X,Y), haslecture(Y,D).
taking_course(joe,nlp).
haslecture(nlp,friday).
```

- Try in Prolog interpreter (do not type ‘`?-`’ part):

```
?- busy(joe,friday).
?- busy(X,friday).
?- busy(joe,Y).
?- busy(X,Y).
```

- Remember to type '`;`' after each answer

Step 7: Lists (Arrays), Structures.

Lists are implemented as linked lists. Structures (records) are expressed as terms or predicates.

As an example, add the following line to `prog1.prolog`:

`person(john,public,'123-456').`

In the Prolog interpreter, try: `?- person(john,X,Y).`

An empty list is: `[]` and is used as a constant.

A list is created as a nested term using special predicate `.` (dot). Example: `.(a, .(b, .(c, [])))`

List Notation

We can use predicate '`is_list`' to check that this is a list:

```
?- is_list(.(a, .(b, .(c, [])))).
```

A better way to write lists:

`.(a, .(b, .(c, [])))` is the same as `[a,b,c]`

This is also equivalent to:

`[ a | [ b | [ c | [] ]]]`

or

`[ a, b | [ c ] ]`

Programming with Lists

A frequent Prolog expression is: `[H|T]`
where H is head of the list, and T is the tail, which is another list.

Example with testing membership of a list:

Add the following code to `prog1.prolog`:

```
member(X, [X|_]).
member(X, [_|L]) :- member(X,L).
```

Try the following query in the interpreter:

```
?- member(a, [b,a,c]).
Yes
```

More Queries with Predicate `member`

```
?- member(2, [1,3,4,5]).
```

```
No
?- member(X, [1,2,3,4,5]).
X = 1;
X = 2;             ...and so on
```

Submit the file `prog1.prolog` using the command `nlp-submit`.

Step 8: Arithmetic in Prolog

- Terms can be constructed using arithmetic function symbols (declared to be infix), e.g.:
  `X+3, X-Y, 5*4`
- Special predicate `is` forces evaluation of the right-hand side part, e.g.:

```
?- X = 5*3.
X = 5*3 ;
yes
?- X is 5*3.
X = 15 ;
yes
```

Example: Calculating Factorial

```
factorial(0,1).
factorial(N,F) :- N>0, M is N-1,
  factorial(M,FM), F is FM*N.
```

After saving in `factorial.prolog` and loading to Prolog:

```
?- ['factorial.prolog'].
% factorial.prolog compiled 0.00 sec, 1,000 bytes

Yes
?- factorial(6,X).

X = 720 ;
```

Submit the file `factorial.prolog` using the command `submit-nlp`.

Step 9: Task

a) `task.prolog`

Write and submit using `submit-nlp` a Prolog program named `task.prolog` that is a knowledgebase containing three facts and one rule:

- There should be three facts, each of them encoding one of the following statements:
  - "John is a child of Mary"
  - "Ann is a child of John"
  - "Tom is a child of John"
- There should be one rule, encoding the following statement:
  - "A child of one's child is one's grandchild" (in other words: "If a person is a child of some other person, and this other person is a child of some third person, then the first person is a grandchild of the third person").

Your program needs to allow for queries stated in part (b) below.

b) `task-queries.txt` You will write Prolog queries for your program task.prolog from part (a) and submit them in a text file task-queries.txt, one query per line. For each line of the submitted task-queries.txt file, one should be able to copy the entire line and paste it into Prolog prompt, after loading your program task.prolog from part (a), to obtain an answer from Prolog.

Consider the following questions:

- "Is Ann a child of Mary?"
- "Is Ann a grandchild of Mary?"
- "Who is a grandchild of Mary?"
- "Whose grandchild is Tom?"

For each of these questions, write a Prolog query to obtain an answer to the question. You need to keep the above order when you put your queries in a text file, i.e., a query in the n-th line of your text file is to be the query for n-th one of the above questions.

The end of Lab 9.