

CSCI 4152/6509
Natural Language Processing

Lab 9:
Prolog Tutorial 1

Lab Instructor: Sigma Jahan and Mayank Anand

Faculty of Computer Science

Dalhousie University

Lab Overview

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

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

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

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 paradigm (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

- Starting the hands-on part of the lab
- Login to the sever timberlea
- Change directory to `csci4152` or `csci6509`
- `mkdir lab8`
- `cd lab8`

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
- 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
- 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:

$$p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q$$

where $n \geq 1$, and p_1, \dots, 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 \wedge , 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 .

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

Prepare and submit two files, as described in notes.