Prolog

**Key Concepts for Logic Programming:** Unification, recursion, and first order logic

**Research 1**

Install a prolog IDE <http://www.swi-prolog.org/>

Quick start IDE guide <http://www.swi-prolog.org/pldoc/man?section=quickstart>

What is first order logic?

Provide two examples of statements using first order logic

What is a logic paradigm?

Provide two examples of languages that can use this programing paradigm

**Reading Material 1**

Read and make notes of the tutorial in <http://www.doc.gold.ac.uk/~mas02gw/prolog_tutorial/prologpages/> from simple facts to recursion.

* Pay specific attention to the logic behind predicates and unification, this will be the heart of everything we do.
* Use google or imdb to create a knowledge base in Prolog, (you will be consulting it to see what actors are in which movies). That knowledge base should have at least the following:
  + - 1 movies with "Chuck Norris"
    - 2 movies with "Jackie Chan"
    - 2 movies with "Bruce Lee"
    - 1 movie with "Chuck Norris" and "Bruce Lee"
    - Create a predicate that binds each actor to each movie.
    - Create another predicate that contains any misc information you think is interesting and relates to your previous KB.

Query your knowledge base to know if you created it correctly.

**During class**

* What is first order logic.
* Context of the language.
* Logic programming paradigm.
* Family example.
* What is the definition of logic programming?
* What are the characteristics of a language that uses logic programming paradigm?
* Name two advantages logic programming has over other paradigms
* Name two disadvantages that logic programming has compared with imperative, object oriented programing.
* Could this type of programming be done in parallel?
* What are the basic elements of logic programming languages?
* Family tree example or movie sample. **(family.pl)**
* Trace on Backtracking
* Representation of relations and graphs in the KB, directionality.
* Representation <http://www.cs.oswego.edu/~odendahl/coursework/notes/prolog/synopsis/con.html>

**Lab 1**

After doing this lab you should be able to understand how to represent facts and graphs in a logic programming language. You should also grasp how Prolog’s basic operations work.

* Given the following facts

hobby(juan,kaggle).   
hobby(luis,hack).   
hobby(elena,tennis).   
hobby(midori,videogame).   
hobby(simon,sail).  
hobby(simon,kaggle).  
hobby(laura,hack).  
hobby(hans,videogame).   
  
Define a predicate “compatible(X,Y)”. We say that X and Y are compatible if they share at least 1 hobby.

* Use the following map to generate a KB of the paths that lead to Rome (ROMA!), assume that every road is a **one way road** and the all lead to the capital, because “*all roads lead to Rome*”. The directions in your predicates should be from left (start) to right (end). For example:

road(placentia, ariminum).

road(ariminum, ancona).

road(ancona, roma).

road(messana, capua).

road(rhegium ,messana).

road(catina, rhegium).

Define the rule can\_get\_to(Origin, Destination) which is true if there is a path that starts in Origin and following the directionality of the roads can get to Destination.

Example:

can\_get\_to(lilibeum, capua) is true,

can\_get\_to(roma, pisae) is false.

Define the predicate size(X, Y, Z) which returns in Z the number of cities crossed in the path from X to Y, the optimal path is not required, yet.

* Define the predicate min(A, B, C, Z), which returns Z as the minimal value between A, B, and C.
* Define the predicate gcd(A, B, Z), which returns Z as the greatest common divisor (or highest common factor) of A and B.

**Reading Material 2**

Read and make notes from the following links [Lists](http://www.doc.gold.ac.uk/~mas02gw/prolog_tutorial/prologpages/lists.html) and [List Processing](http://kti.ms.mff.cuni.cz/%7Ebartak/prolog/lists.html).

* Are there any advantages to using recursion instead of iteration?

**During class**

* Compare a cycle between a procedural, functional and a logic paradigm. **(recusion\_example.pl, fibbo.pl)**
* Do example with head recursion and tail recursion **(implement list\_example.pl)**
* Use of a stack in sorts example True use of Unification **(implement merge sort.pl Bubble sort.pl and insert sort.pl)**

**Lab 2**

After this lab you should have a deeper understanding of recursion, and you should know how lists work in Prolog.

* Define the following predicates

Any positive.

?- any\_positive ([-1,-2,3,-4]).  
true.

Substitute.

?- substitute (2, 3, [1, 2, 2, 2, 3, 2], X).  
X = [1, 3, 3, 3, 3, 3]

Eiminate duplicates.

?- eliminate\_duplicates ([a, a, a, a, b, c, c, a, a, d, e, e, e, e],X).  
X = [a, b, c, d, e]

Intersect.

?- intersect ([a, b, c, d], [b, d, e, f], X).  
X = [b, d]

Invert.

?- invert([a, b, e, c, e],X).  
X = [e, c, e, b, a]

Less than.

?- less\_then(5, [1, 6, 5, 2, 7], X).  
X = [1, 2]

More than (or equal).

?- more\_than((5, [1, 6, 5, 2, 7], X).  
X = [6, 5, 7]

rotate.

?- rotate([1, 6, 5, 2, 7], 3, X).  
X = [2, 7, 1, 6, 5]  
?- rotate([1, 6, 5, 2, 7], -3, X).  
X = [5, 2, 7, 1, 6]

Given the Italy Knowledge base you used in the last lab, modify it so that the relations between part are bidirectional. Define the predicate path(Origin, Destiny, Path) which returns in Path the path between Origin and Destiny.

**+ 10 points in lab if the path is the shortest possible one.**

**Reading Material 3**

Read and make notes from <http://cs.union.edu/~striegnk/learn-prolog-now/html/node88.html#sec.l10.cut> and the prolog\_cutfail.pdf, it is a compact presentation.

* In what cases should we use cut?
* What is the purpose of fail?

**During class**

* Some structure implementations in prolog **Stack queue Set.pl**
* First depth search algorithm. **Dfs.pl**

**Lab 3**

After this lab you should understand in what situations it is necessary to stop prologs backtracking and how to do so.

This is an implemented depth fist search algorithm in prolog:

<https://www.cs.unm.edu/~luger/ai-final/code/PROLOG.depth.html>

1. Read it carefully and use it as an example to implement your own **Limited Depth First Search Algorithm**
2. Implement the quicksort algorithm using prolog such that

?- quick\_sort([1, 6, 2, 7], X).  
X = [1, 2, 6, 7]  
?- quick\_sort ([1, 6, 2, 7], X).  
X = [1, 2, 6, 7]  
?- quick\_sort ([13, 46, 25, 12, 27, 1], X).  
X = [1, 12, 13, 25, 27, 46]

**Cool examples:**

<https://www.cs.auckland.ac.nz/~j-hamer/07.363/explore.html>

Kevin Bacon center of the Universe.

Understanding prolog compiler <http://www.swi-prolog.org/pldoc/man?section=compilation>

Generate a compiled version of your program in prolog <http://www.swi-prolog.org/FAQ/MakeExecutable.html>