Name: Tania Rajabally

Roll No: 43

UID: 2017130047

Batch: C

Course Code: CEL71 (AI and Soft Computing Lab)

Date:29/09/2020

**Experiment No 5**

**Designing An Expert System**

**Aim:** To design and implement an expert system, incorporating the match algorithm and the rule language.

**Theory:**

The expert systems are the computer applications developed to solve complex problems in a particular domain, at the level of extra-ordinary human intelligence and expertise.

Characteristics of Expert Systems

* High performance
* Understandable
* Reliable
* Highly responsive

The expert systems are capable of −

* Advising
* Instructing and assisting human in decision making
* Demonstrating
* Deriving a solution
* Diagnosing
* Explaining
* Interpreting input
* Predicting results
* Justifying the conclusion
* Suggesting alternative options to a problem

A naive implementation of an expert system might check each rule against known facts in a knowledge base, firing that rule if necessary, then moving on to the next rule (and looping back to the first rule when finished). For even moderate sized rules and facts knowledge-bases, this naive approach performs far too slowly. The Rete algorithm provides the basis for a more efficient implementation. A Rete-based expert system builds a network of nodes, where each node (except the root) corresponds to a pattern occurring in the left-hand-side (the condition part) of a rule. The path from the root node to a leaf node defines a complete rule left-hand-side. Each node has a memory of facts which satisfy that pattern. This structure is essentially a generalized trie. As new facts are asserted or modified, they propagate along the network, causing nodes to be annotated when that fact matches that pattern. When a fact or combination of facts causes all of the patterns for a given rule to be satisfied, a leaf node is reached and the corresponding rule is triggered.

SWI-Prolog is a versatile implementation of the Prolog language. Although SWI-Prolog gained its popularity primarily in education, its development is mostly driven by the needs for application development. This is facilitated by a rich interface to other IT components by supporting many document types and (network) protocols as well as a comprehensive low-level interface to C that is the basis for high-level interfaces to C++, Java (bundled), C#, Python, etc (externally available). Data type extensions such as dicts and strings as well as full support for Unicode and unbounded integers simplify smooth exchange of data with other components.

SWI-Prolog aims at scalability. Its robust support for multi-threading exploits multi-core hardware efficiently and simplifies embedding in concurrent applications. Its Just In Time Indexing (JITI) provides transparent and efficient support for predicates with millions of clauses.

SWI-Prolog unifies many extensions of the core language that have been developed in the Prolog community such as tabling, constraints, global variables, destructive assignment, delimited continuations and interactors.

SWI-Prolog offers a variety of development tools, most of which may be combined at will. The native system provides an editor written in Prolog that is a close clone of Emacs. It provides semantic highlighting based on real time analysis of the code by the Prolog system itself. Complementary tools include a graphical debugger, profiler and cross-referencer. Alternatively, there is a mode for GNU-Emacs and, Eclipse plugin called PDT and a VSC plugin, each of which may be combined with the native graphical tools. Finally, a computational notebook and web based IDE is provided by SWISH. SWISH is a versatile tool that can be configured and extended to suit many different scenarios.

SWI-Prolog provides an add-on distribution and installation mechanism called packs. A pack is a directory with minimal organizational conventions and a control file that describes the origin, version, dependencies and automatic upgrade support. Packs can be installed from an archive, GIT repository or URL using pack\_install/1. Packs are used to share code in the community. The pack system has grown a couple of eco systems for dealing with types, coroutining, etc.

**Problem Statement:**

There is a group of 5 friends, Johnny, Jack, Tom, Alice and Cathy. Each of them has a pet: Dog, Cat, Fish,

Turtle and Rabbit. They all have different professions: Athlete, Lawyer, Banker, Doctor and Historian.

They all stay in different cities: Mumbai, London, Tokyo, Paris and New York.

The following facts are known:

Cathy has neither a dog or cat and stays in Mumbai.

The historian stays in Tokyo.

Tom and Alice stay in the same continent.

Johnny has a rabbit and stays in New York.

The Athlete has a fish and stays in London.

The Doctor and Lawyer are both girls and the lawyer practices in the Mumbai High Court.

The dog and its owner see the Eiffel Tower on their daily walk.

**Questions:**

Which city does Alice stay in?

What pet does Jack own?

Who is the athlete?

What is Johnny’s profession?

**Procedure:**

We write the program in swish prolog.

All the combinations given are written. For some, we aren't sure so we put an or and put those. This is done by putting semicolons between the conditions.

We then run it and print the table which gives us the final answer.

**Program:**

**:- use\_rendering(table,**

**[header(h('Name','Pet','Profession','City','Gender','Continent'))]).**

**people(PI) :-**

**length(PI, 5),**

**member(h(\_,\_,\_,mumbai,\_,asia),PI),**

**member(h(\_,\_,\_,tokyo,\_,asia),PI),**

**member(h(\_,\_,\_,paris,\_,europe),PI),**

**member(h(\_,\_,\_,london,\_,europe),PI),**

**member(h(\_,\_,\_,newyork,\_,america),PI),**

**member(h(johnny,\_,\_,\_,male,\_),PI),**

**member(h(jack,\_,\_,\_,male,\_),PI),**

**member(h(tom,\_,\_,\_,male,\_),PI),**

**member(h(alice,\_,\_,\_,female,\_),PI),**

**member(h(cathy,\_,\_,\_,female,\_),PI),**

**member(h(\_,dog,\_,\_,\_,\_),PI),**

**member(h(\_,turtle,\_,\_,\_,\_),PI),**

**member(h(\_,fish,\_,\_,\_,\_),PI),**

**member(h(\_,rabbit,\_,\_,\_,\_),PI),**

**member(h(\_,cat,\_,\_,\_,\_),PI),**

**member(h(\_,\_,banker,\_,\_,\_),PI),**

**member(h(cathy,\_,\_,mumbai,female,asia), PI),**

**member(h(\_,\_,historian,tokyo,\_,asia), PI),**

**member(h(johnny,rabbit,\_,newyork,male,america), PI),**

**member(h(\_,fish,athlete,london,\_,europe), PI),**

**member(h(\_,\_,lawyer,mumbai,female,asia), PI),**

**member(h(\_,\_,doctor,\_,female,\_), PI),**

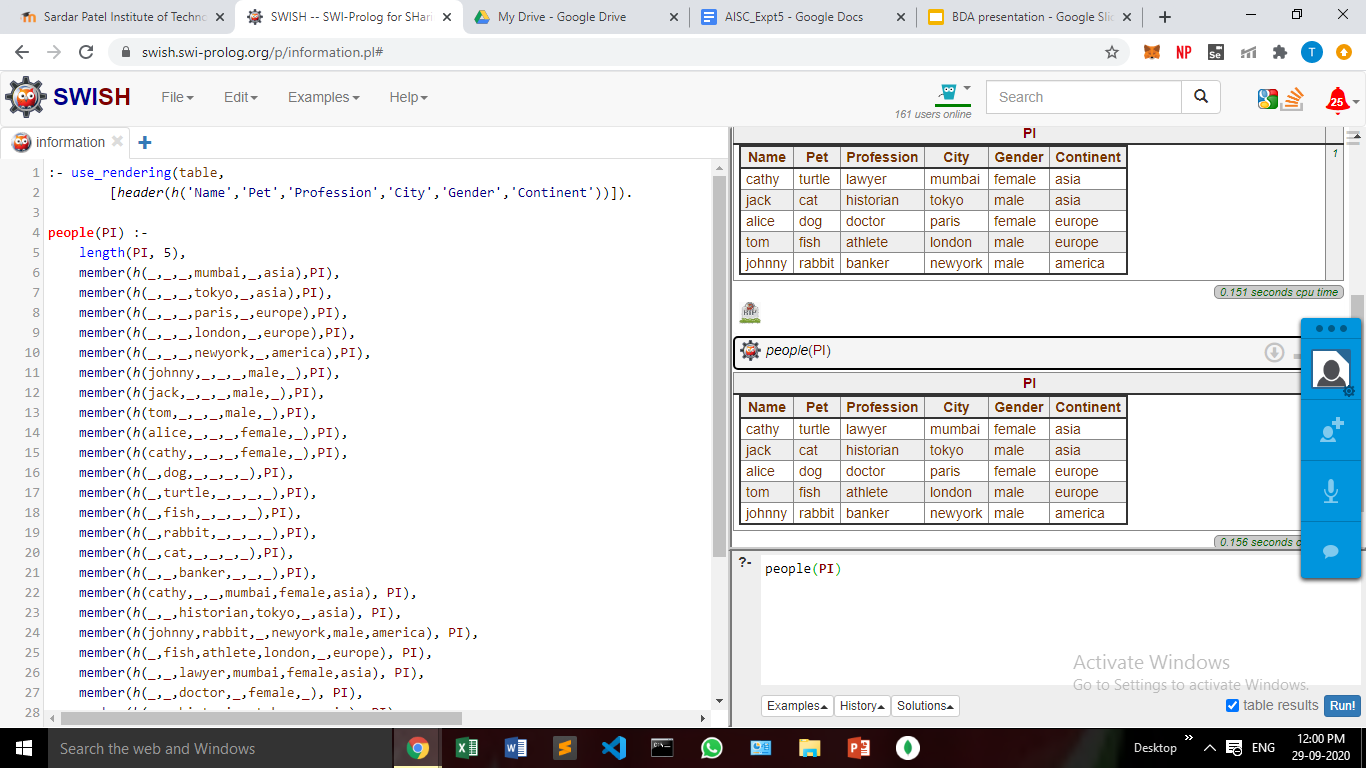
**member(h(\_,\_,historian,tokyo,\_,asia), PI),**

**member(h(\_,dog,\_,paris,\_,europe), PI),**

**member(h(tom,\_,\_,\_,\_,europe),PI);member(h(alice,\_,\_,\_,\_,europe),PI);member(h(tom,\_,\_,\_,\_,asia),PI);member(h(alice,\_,\_,\_,\_,asia),PI),**

**member(h(cathy,turtle,\_,\_,\_,\_),PI);member(h(cathy,fish,\_,\_,\_,\_),PI);member(h(cathy,rabbit,\_,\_,\_,\_),PI).**

**Output:**

****

**Answers:**

1. Paris
2. Cat
3. Tom
4. Banker

**Conclusion:**

In this experiment, we solved the given problem statement using Swish. We got the answer to the questions from the table generated by the Swish program.