AI TOOLS & TECHNIQUES LAB

Using Prolog

Study of PROLOG

Introduction

Artificial Intelligence (or simply AI) is the study and development of machines that are capable of having an intelligence equal to or better than a human being. As a result, AI has many different applications today. AI is used in everything from gaming, to creating smarter computerized opponents, to robots that can assist humans in nearly every facet of life.

Additionally, AI can be used to create software such as facial recognition software and language processing software. Today, there are a couple of different programming languages that are used to create artificial intelligence, one of these is Prolog.

Prolog evolved out of research at the University of Aix-Marseille back in the late 60's and early 70's.

Alain Colmerauer and Phillipe Roussel, both of University of Aix-Marseille, colaborated with Robert Kowalski of the University of Edinburgh to create the underlying design of Prolog as we know it today.

Kowalski contributed the theoretical framework on which Prolog is founded while Colmerauer's research at that time provided means to formalize the Prolog language.

1972 is referred to by most sources as the birth date of Prolog.

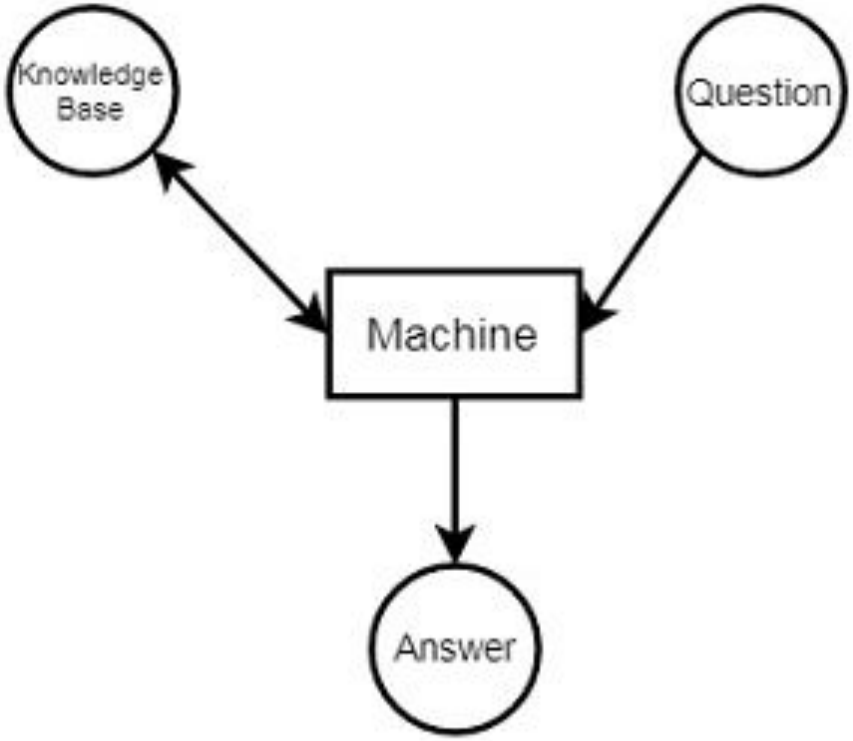
To this day Prolog has grown in use throughout North America and Europe. Prolog was used heavily in the European Esprit programme and in Japan where it was used in building the ICOT Fifth Generation Computer Systems Initiative. The Japanese Government developed this project in an attempt to create intelligent computers.

Prolog as the name itself suggests, is the short form of Logical Programming. It is a logical and declarative programming language.

Logic Programming is one of the Computer Programming Paradigm, in which the program statements express the facts and rules about different problems within a system of formal logic.

Prolog is a declarative language, which means that a program consists of data based on the facts and rules (Logical relationship) rather than computing how to find a solution. A logical relationship describes the relationships which hold for the given application.

Representation of Logical Programming:



Logical Programming

Prolog language basically has three different elements –

Facts – The fact is predicate that is true, for example, if we say, “Tom is the son of Jack”, then this is a fact.

Rules – Rules are extinctions of facts that contain conditional clauses. To satisfy a rule these conditions should be met. For example, if we define a rule as –

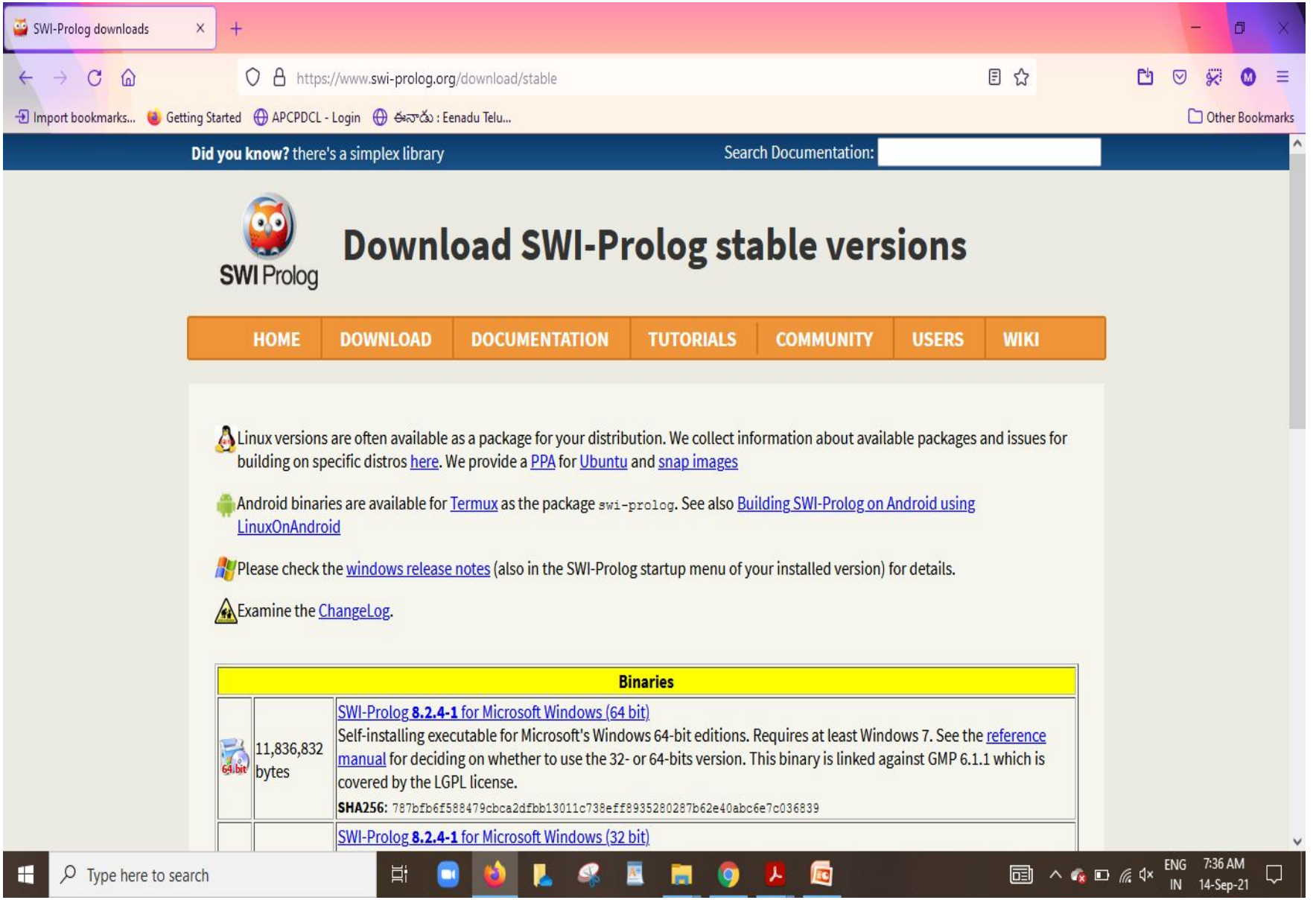
grandfather(X, Y) :- father(X, Z), parent(Z, Y) This implies that for X to be the grandfather of Y, Z should be a parent of Y and X should be father of Z.

Questions – And to run a prolog program, we need some questions, and those questions can be answered by the given facts and rules.

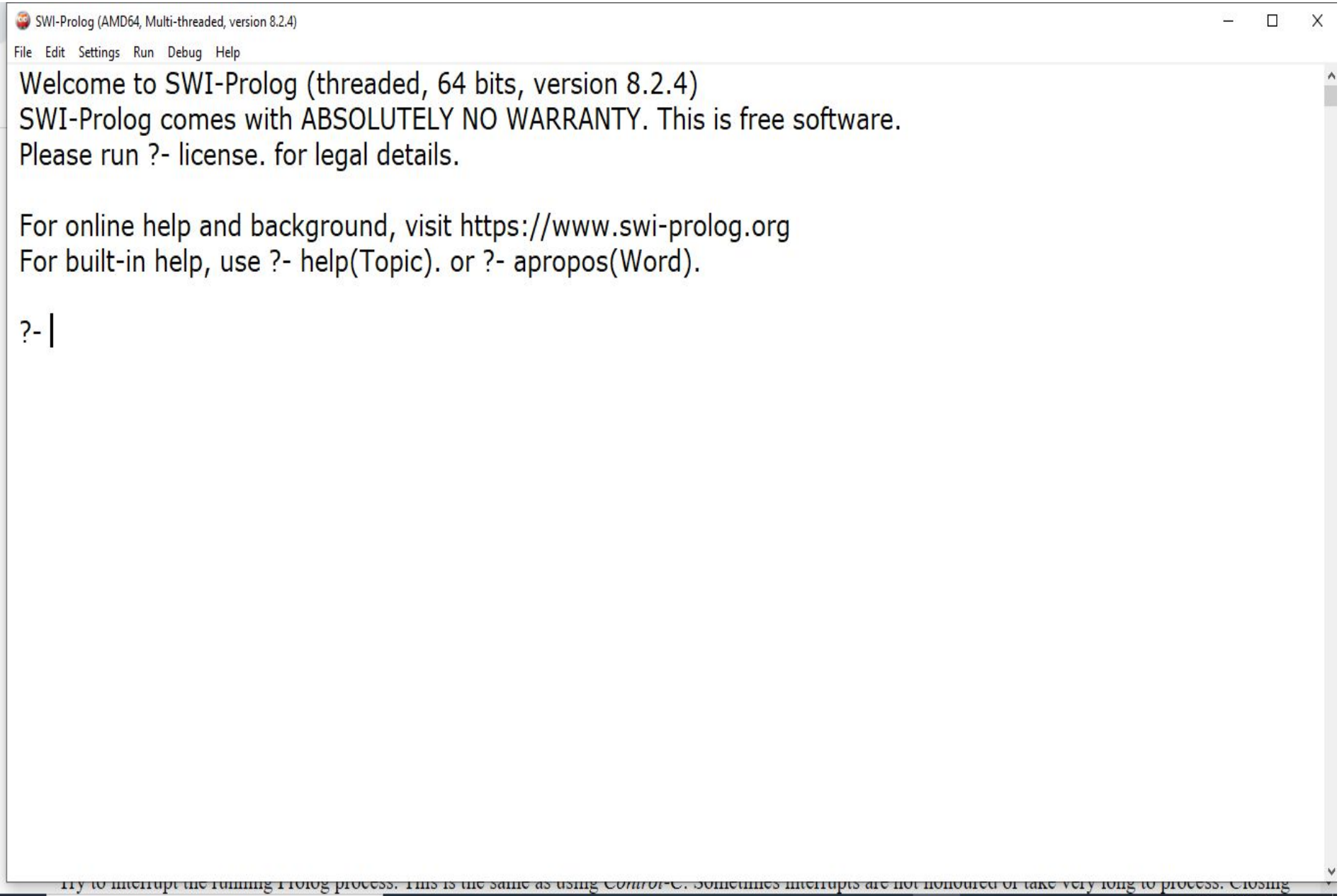
Applications of Prolog

- Intelligent Database Retrieval
- Natural Language Understanding
- Specification Language
- Machine Learning
- Robot Planning
- Automation System

To start Prolog, download the software from website
https://www.swi-prolog.org/



SWI-Prolog has the following menu commands



Try to interrupt the running Prolog process. This is the same as using *Control*-C. Sometimes interrupts are not honoured or take very long to process. Closing

File/Reload modified files:
This menu reloads all loaded source files that have been modified.

File/Navigator ... Opens an explorer-like view on Prolog files and the predicates they contain.

Settings/Font ... Allows for changing the font of the console. On some installations the default font gives redraw and cursor dislocation problems. In this case you may wish to select an alternative. Some built-in commands assume non-proportional fonts.

Run/Interrupt: Try to interrupt the running Prolog process.

Run/New thread: Creates a new interactor window running in a separate thread of execution. This may be used to inspect the database or program while the main task continues.

Debug/Edit spy points .. Edit break points on predicates

Prolog Program Structure:

domains

It is an optional section. This keyword is used to mark a section declaring the domains that would be used in the code. E.g.:

```
person = symbol
disease, indication = symbol
```

predicates

This section contains the declarations of the predicates that would be later defined in the clauses section of the code. e.g.:

```
father(person,person)
symptom(disease, indication)
```

clauses

It contains the actual *definitions* of the previously declared predicates.

goal

This section is used to query prolog database

Example:

1. Open notepad type the following clauses

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

This program consists of six clauses .Each of these clauses declares one fact about the parent relation.

2. After designing in notepad save it as “sample.pl”

3. Now open the swi prolog interactive window and select the file menu and open the designed pl fie.

File new sample.pl

4. Now select compile menu to compile the prolog file.

compile compile buffer

This command will display error messages if there is any error otherwise it display compiled.

5 When this program has been communicated to the Prolog system Prolog can be posed some questions about the parent relation. For example:

```
?- parent( bob, pat).
true.
```

2. simple fact for the statements using PROLOG.

A Prolog program consists of a number of clauses. Each clause is either a fact or a rule. After a Prolog program is loaded (or *consulted*) in a Prolog interpreter, users can submit goals or queries, and the Prolog interpreter will give results (answers) according to the facts and rules.

FACT:

A **fact** is a predicate expression that makes a declarative statement about the problem domain. A fact must start with a predicate (which is an atom) and end with a fullstop. The predicate may be followed by one or more arguments which are enclosed by parentheses.

The arguments can be atoms (atoms are treated as constants), numbers, variables or lists. Arguments are separated by commas.

If we consider the arguments in a fact to be objects, then the predicate of the fact describes a property of the objects.

In a Prolog program, a presence of a fact indicates a statement that is true. An absence of a fact indicates a statement that is not true.

RULE:

A rule can be viewed as an extension of a fact with added conditions that also have to be satisfied for it to be true.

It consists of two parts. The first part is similar to a fact (a predicate with arguments). The second part consists of other clauses (facts or rules which are separated by commas) which must all be true for the rule itself to be true.

These two parts are separated by ":-". You may interpret this operator as "if" in English.

```
father(jack, susan).                                    /* Fact  1 */
father(jack, ray).                                      /* Fact  2 */
father(david, liza).                                    /* Fact  3 */
father(david, john).                                    /* Fact  4 */
father(john, peter).                                    /* Fact  5 */
father(john, mary).                                     /* Fact  6 */
mother(karen, susan).                                   /* Fact  7 */
mother(karen, ray).                                     /* Fact  8 */
mother(amy, liza).                                      /* Fact  9 */
mother(amy, john).                                      /* Fact 10 */
mother(susan, peter).                                   /* Fact 11 */
mother(susan, mary).                                    /* Fact 12 */

parent(X, Y) :- father(X, Y).                           /* Rule  1 */
parent(X, Y) :- mother(X, Y).                           /* Rule  2 */
grandfather(X, Y) :- father(X, Z), parent(Z, Y). /* Rule  3 */
grandmother(X, Y) :- mother(X, Z), parent(Z, Y). /* Rule  4 */
grandparent(X, Y) :- parent(X, Z), parent(Z, Y). /* Rule  5 */
```

Queries

The Prolog interpreter responds to **queries** about the facts and rules represented in its database. The database is assumed to represent what is true about a particular problem domain.

In making a query you are asking Prolog whether it can prove that your query is true. If so, it answers "yes" and displays any **variable bindings** that it made in coming up with the answer. If it fails to prove the query true, it answers "No".

Example:

Facts

English meanings

```
food(burger).       // burger is a foodf
food(sandwich).     // sandwich is a food
food(pizza).        // pizza is a food
lunch(sandwich).    // sandwich is a lunch
dinner(pizza).      // pizza is a dinner
```

Rules

```
meal(X) :- food(X).     // Every food is a meal OR Anything is a meal if it is a food
```

Queries / Goals

```
?- food(pizza).         // Is pizza a food?
true.
?- meal(X), lunch(X).       // Which food is meal and lunch?
X = sandwich .
?- dinner(sandwich).        // Is sandwich a dinner?
false.
```

Facts	English meanings
studies(charlie, csc135).	// charlie studies csc135
studies(olivia, csc135).	// olivia studies csc135
studies(jack, csc131).	// jack studies csc131
studies(arthur, csc134).	// arthur studies csc134
teaches(kirke, csc135).	// kirke teaches csc135
teaches(collins, csc131).	// collins teaches csc131
teaches(collins, csc171).	// collins teaches csc171
teaches(juniper, csc134).	// juniper teaches csc134

Rules

professor(X, Y) :- teaches(X, C), studies(Y, C).	// X is a professor of Y if X teaches C and Y studies C.

Queries / Goals

?- studies(charlie, What).	// charlie studies what? OR What does charlie study?
?- professor(kirke, Students).	// Who are the students of professor kirke.

Write predicates One converts centigrade temperatures to Fahrenheit, the other checks if a temperature is below freezing.

```
c_to_f(C,F) :-F is C * 9 / 5 + 32.

freezing(F) :-F =< 32.
```

Output:

```
c_to_f(100,X).
X = 212.
```