Laboratory Report

COURSE CODE: CSE2022L
ARTIFICIAL INTELLIGENCE TECHNIQUES AND METHODS LAB.

SCHOOL OF ENGINEERING AND SCIENCES
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Submitted By	
Student Name	Ayan Chowdhury
Enrollment No.	190020203008
Section/Group	A/T1
Department	Computer Science and Engineering
Session/Semester	2021-22/ Even Semester
Submitted To	
Faculty Name	Ms. Manka Sharma

GD GOENKA UNIVERSITY
Sohna road, Gurugram, Haryana

TABLE OF CONTENTS

6.	5/4/22	Write a program in prolog to implement Best First Search Algorithm.	
7.	21/3/22	a) Write a program in prolog to create the state space for the following and check the connections that exist between any two nodes. b) Write a program in prolog to find the path between any two states of the following state space. c) Write a program in prolog to find the path between any two states of the following state space. d) Write a program in prolog to accept the input from the user and perform string matching.	
8.	5/4/22	Write a program in prolog to implement Depth First Search Algorithm.	

EXPERIMENT NO-1 **DATE: 28/1/22**

AIM OF THE EXPERIMENT: STUDY OF PROLOG

THEORY :

Artificial Intelligence (AI), the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings. The term is frequently applied to the project of developing systems endowed with the intellectual processes characteristic of humans, such as the ability to reason, discover meaning, generalize, or learn from past experience.

Uses:

1. **Online shopping and advertising -** Artificial intelligence is widely used to provide personalised recommendations to people, based for example on their previous searches and purchases or other online behaviour. AI is hugely important in commerce: optimising products, planning inventory, logistics etc.

2. **Web search -** Search engines learn from the vast input of data, provided by their users to provide relevant search results.

3. **Digital personal assistants -** Smartphones use AI to provide services that are as relevant and personalised as possible. Virtual assistants answering questions, providing recommendations and helping organise daily routines have become ubiquitous.

4. **Machine translations -** Language translation software, either based on written or spoken text, relies on artificial intelligence to provide and improve translations. This also applies to functions such as automated subtitling.

5. **Smart homes, cities and infrastructure -** Smart thermostats learn from our behaviour to save energy, while developers of smart cities hope to regulate traffic to improve connectivity and reduce traffic jams.

6. **Cars -** While self-driving vehicles are not yet standard, cars already use AI-powered safety functions. The EU has for example helped to fund VI-DAS, automated sensors that detect possible dangerous situations and accidents. Navigation is largely AI-powered.

7. **Cybersecurity -** AI systems can help recognise and fight cyberattacks and other cyber threats based on the continuous input of data, recognising patterns and backtracking the attacks.

Goals of AI

- **To Create Expert Systems** – The systems which exhibit intelligent behaviour, learn, demonstrate, explain, and advice its users.
- **To Implement Human Intelligence in Machines** – Creating systems that understand, think, learn, and behave like humans.

Advantages of Artificial Intelligence:

1. **Reduction in Human Error:** The phrase "**human error**" was born because humans make mistakes from time to time. Computers, however, do not make these mistakes if they are programmed properly. With Artificial intelligence, the decisions are taken from the previously gathered information applying a certain set of algorithms. So errors are reduced and the chance of reaching accuracy with a greater degree of precision is a possibility.

2. **Takes risks instead of Humans:** This is one of the biggest advantages of Artificial intelligence. We can overcome many risky limitations of humans by developing an AI Robot which in turn can do the risky things for us. Let it be going to mars, defuse a bomb, explore the deepest parts of oceans, mining for coal and oil, it can be used effectively in any kind of natural or man-made disasters.

3. **Available 24x7:** An Average human will work for 4–6 hours a day excluding the breaks. Humans are built in such a way to get some time out for refreshing themselves and get ready for a new day of work and they even have weekly offed to stay intact with their work-life and personal life. But using AI we can make machines work 24x7 without any breaks and they don't even get bored, unlike humans.

4. **Helping in Repetitive Jobs:** In our day-to-day work, we will be performing many repetitive works like sending a thanking mail, verifying certain documents for errors and many more things. Using artificial intelligence, we can productively automate these mundane tasks and can even remove "**boring**" tasks for humans and free them up to be increasingly creative.

5. **Digital Assistance:** Some of the highly advanced organizations use digital assistants to interact with users which saves the need for human resources. The digital assistants also used in many websites to provide things that users want. We can chat with them about what we are looking for. Some chatbots are designed in such a way that it's become hard to determine that we're chatting with a chatbot or a human being.

6. **Faster Decisions:** Using AI alongside other technologies we can make machines take decisions faster than a human and carry out actions quicker. While taking a decision human will analyze many factors both emotionally and practically but AI-powered machine works on what it is programmed and delivers the results in a faster way.

<u>Disadvantages of Artificial Intelligence</u>:

1. **High Costs of Creation:** As AI is updating every day the hardware and software need to get updated with time to meet the latest requirements. Machines need repairing and maintenance which need plenty of costs. It' s creation requires huge costs as they are very complex machines.

2. **Making Humans Lazy:** AI is making humans lazy with its applications automating the majority of the work. Humans tend to get **addicted** to these inventions which can cause a problem to future generations.

3. **Unemployment:** As AI is replacing the majority of the repetitive tasks and other works with robots, human interference is becoming less which will cause a major problem in the employment standards. Every organization is looking to replace the minimum qualified individuals with AI robots which can do similar work with more efficiency.

4. **No Emotions:** There is no doubt that machines are much better when it comes to working efficiently but they cannot replace the human connection that makes the team. Machines cannot develop a bond with humans which is an essential attribute when comes to Team Management.

5. **Lacking Out of Box Thinking:** Machines can perform only those tasks which they are designed or programmed to do, anything out of that they tend to crash or give irrelevant outputs which could be a major backdrop.

Languages for AI development:

1. **JAVA**: Java by Oracle is one of the best programming languages available out there. Over the years, this language has adapted to the latest innovations and technological advancements. The same is true for AI. Using Java for AI development can help you get some scalable applications. For AI development, Java offers ease of usage and debugging and simplifies large-scale projects. You can represent the data in graphics and offer better user interaction.

2. **PYTHON**: Another one on the list is Python, the programming language that offers the least code among all others. There are many reasons why Python Development Companies are preferred the best for AI development. These reasons include:

    - Prebuilt Libraries for advanced computing like Numpy, Scipy, and Pybrain.

    - Open-source language with support by developers from across the world. There are many Forums and Tutorials for Python that you can seek help from.

    - The advantage of Python Machine Learning.

    - Python is an independent and flexible language compatible with multiple platforms with minimum tweaks.

    - There are options like Scripting, OOPs approach, and IDE that allows fast development with diverse algorithms.

    With all these features and many others, Python has become one of the best languages for AI development.

3. **JAVASCRIPT:** Just like Java, JavaScript is also an ideal match for AI development. However, it is used to develop more secure and dynamic websites. While Python is suitable for developers who don’t like coding, JavaScript is for those who don’t mind it.

    The AI capabilities of JavaScript help it interact and work smoothly with other source codes like HTML and CSS. Like Java, JavaScript also has a large community of developers that support the development process. With libraries like jQuery, React.js, and Underscore.js, AI development becomes more effective. From multimedia, buttons, to data storage, you can manage both frontend and backend functions using JavaScript. With JavaScript, you can ensure security, high performance, and less development time.

4. **JULIA:** While Julia does not come with a large community or support, it offers many high-end features for top-notch AI development. When it comes to handling data analysis and numbers, Julia is the best development tool.

   If you need to make a dynamic interface, catchy graphics, and data visuals, Julia provides you with the right tools for perfect execution. With features like debugging, memory management, and metaprogramming, this language makes AI development a breeze. For machine learning AI projects, Julia is the best bet. It comes with many packages like Metahead, MLJ.JL, Turing.JL, and Flux.JL.

5. **LISP:** Lisp is one of the oldest languages used for AI development. It was developed in the 1960s and has always been an adaptable and smart language. If your project requires modification of code, problem-solving, rapid prototyping, or dynamic development, Lisp is for you. Some successful projects made with Lisp are Routinic, Grammarly, and DART. Though it has its drawbacks, Lisp is still a promising programming language for AI development.

6. **R**: A statistical programming language, R is one of the most suitable choices for projects where you need statistical computations. It supports learning libraries like MXNet, TensorFlow, Keras, etc. The language is adopted by many industries like education, finance, telecommunication, pharmaceuticals, life sciences, etc. It is the language that fuels tech giants like Microsoft, Google, Facebook, and businesses like Uber, Airbnb, etc. R includes user-created packages like graphical devices, tools, import/export capabilities, statistical techniques, etc. With built-in graphic and data modeling support, the language allows developers to work on deep learning moderns without much hassle.

7. **PROLOG:** Prolog is short for Programming in Logic. The language was developed in 1972 in a rule-like form. It is majorly used for projects that involve computational linguistics and artificial intelligence. For the projects that require a database, natural language processing, and symbolic reasoning, Prolog is the best bet! It is the perfect language support for research when it comes to artificial intelligence. Used for automated planning, theorem proving, expert and type systems, Prolog still has limited usage. However, it is used to build some high-end NLP applications and by giants like IBM Watson.

8. **SCALA:** Scala makes the coding process fast, easy, and much more productive. The index Scaladex that has the Scala libraries and resources helps developers create some quality applications. It runs on the Java Virtual Machine (JVM) environment and helps developers program smart software. Scala is compatible with Java and JS and offers many features like pattern matching, high-performing functions, browser tools, and flexible interfaces. For AI development, Scala is one of the best options and it has impressed the developers in that area.

9. **RUST:** Everyone is looking for high-performance, fast, and safe software development, and Rust helps you achieve that! It is a general-purpose programming language that developers love to use for AI development. The syntax of Rust is similar to C++ but the former also offers memory safety and prevents garbage collection.

   Rust works at the backend of many well-known systems like Dropbox, Yelp, Firefox, Azure, Polkadot, Cloudflare, npm, Discord, etc. The memory safety, speed, and ease of expression make Rust the perfect choice for AI development and scientific computing.

10. **HASKELL:** While Haskell comes with limited support, it is another good programming language you can try for AI development. It offers pure functionality and abstraction capabilities that make the language very flexible. However, the lack of support might delay the AI development process. With Haskell, code reusability comes in handy for the developers along with other features like type system and memory management.

PROLOG LANGUAGE:

Prolog is a logic programming language. It has important role in artificial intelligence. Unlike many other programming languages, Prolog is intended primarily as a declarative programming language. In prolog, logic is expressed as relations (called as Facts and Rules). Core heart of prolog lies at the logic being applied. Formulation or Computation is carried out by running a query over these relations.

Key Features:

- Unification : The basic idea is, can the given terms be made to represent the same structure.
- Backtracking : When a task fails, prolog traces backwards and tries to satisfy previous task.
- Recursion : Recursion is the basis for any search in program.

Advantages:

- Easy to build database. Doesn't need a lot of programming effort.
- Pattern matching is easy. Search is recursion based.
- It has built in list handling. Makes it easier to play with any algorithm involving lists.

Disadvantages:

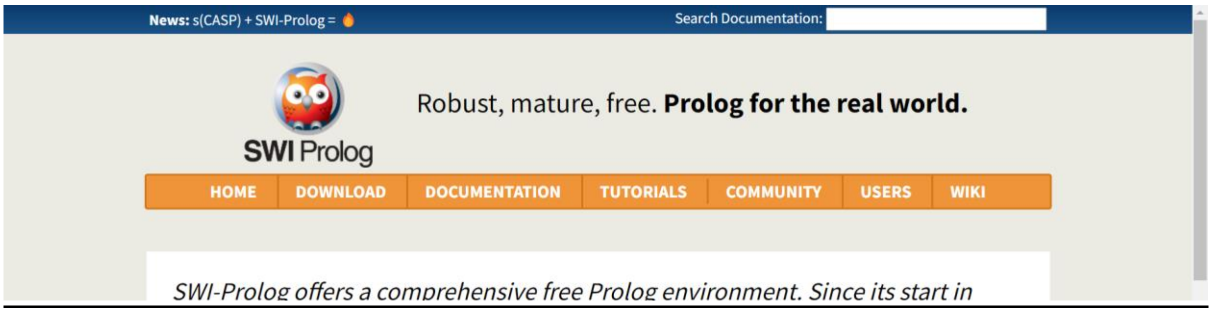
- LISP (another logic programming language) dominates over prolog with respect to I/O features.
- Sometimes input and output is not easy.

Applications:

Prolog is highly used in artificial intelligence(AI). Prolog is also used for pattern matching over natural language parse trees.

<u>INSTALLING PROLOG:</u>

1. Using an Internet browser navigate to SWI-Prolog home page: https://www.swi-prolog.org/ 277
2. Click on download tab.

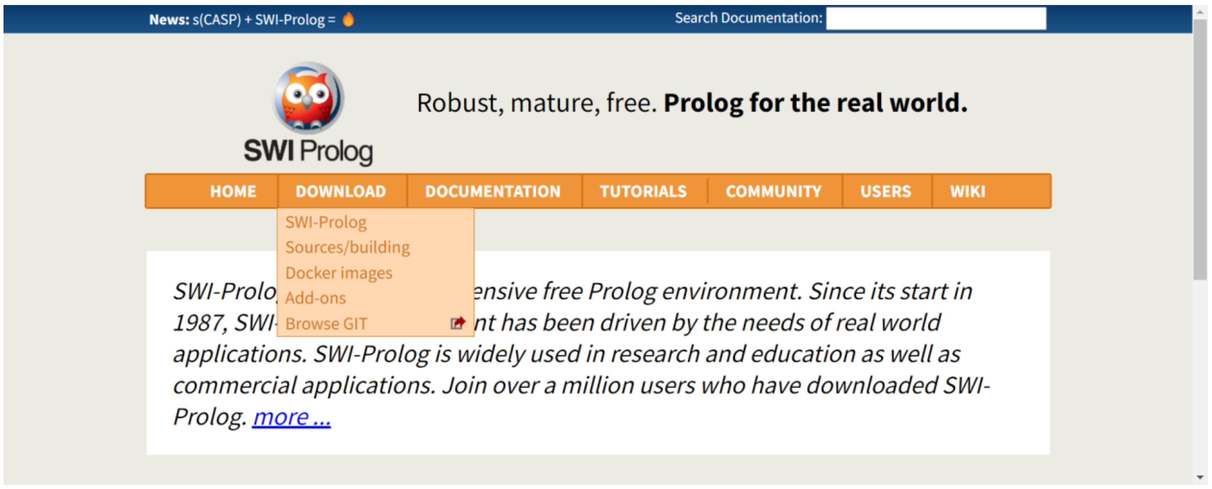


Robust, mature, free. **Prolog for the real world.**

SWI-Prolog offers a comprehensive free Prolog environment. Since its start in

Fig. 1.1 Home Page of SWI Prolog

3. Click SWI-Prolog.

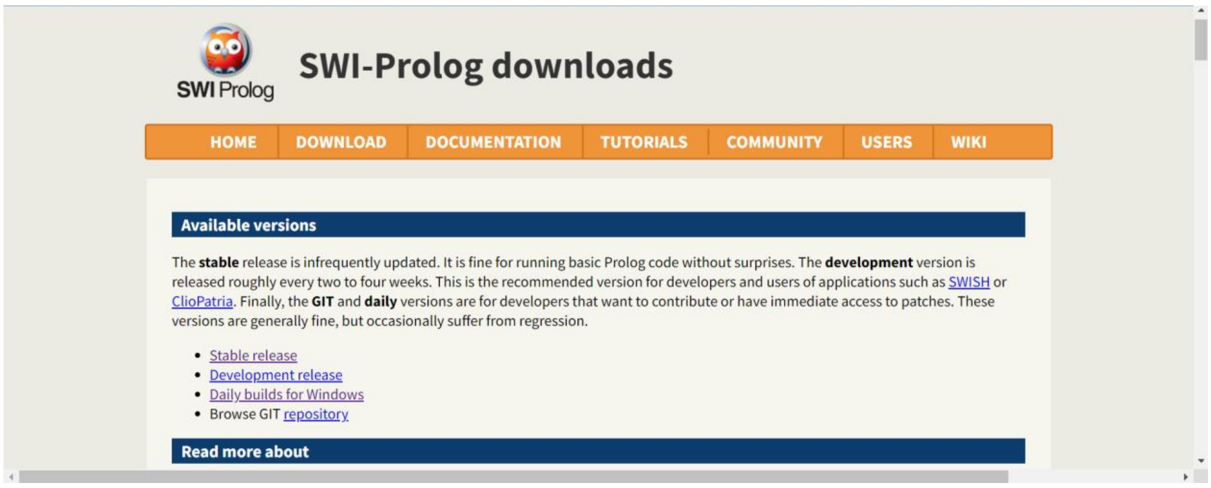


Robust, mature, free. **Prolog for the real world.**

SWI-Prolo ensive free Prolog environment. Since its start in 1987, SWI- nt has been driven by the needs of real world applications. SWI-Prolog is widely used in research and education as well as commercial applications. Join over a million users who have downloaded SWI-Prolog. more ...

Fig. 1.2 Download option in home page of SWI prolog

4. Then click on stable release



SWI-Prolog downloads

Available versions

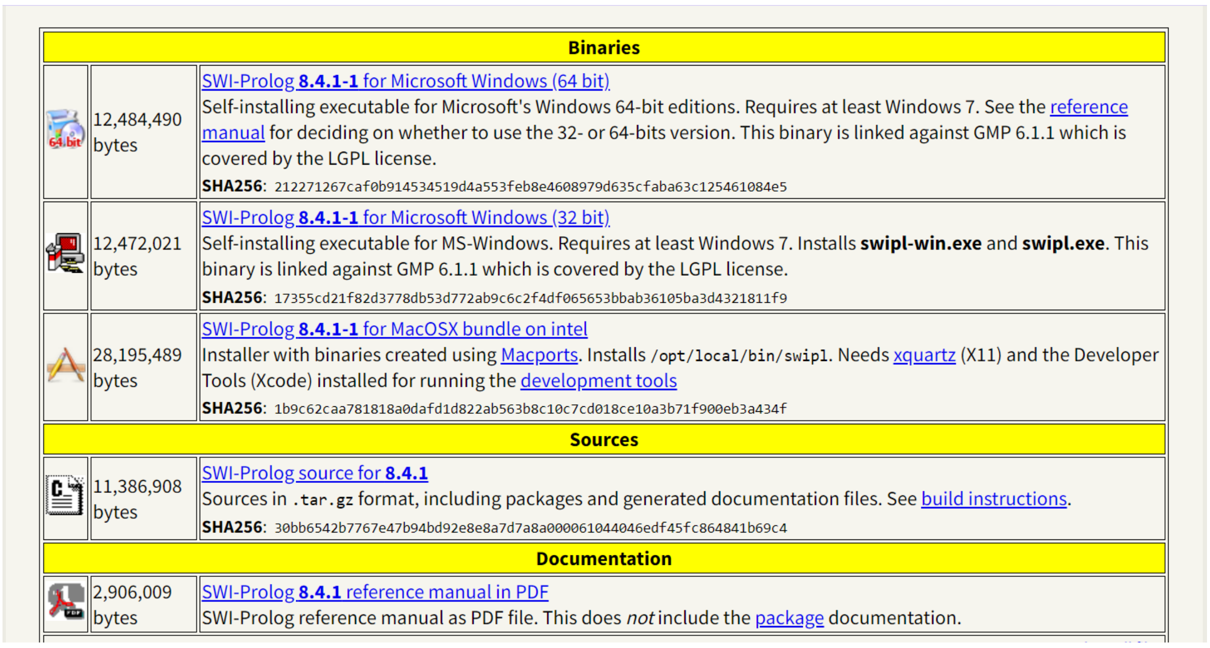
The **stable** release is infrequently updated. It is fine for running basic Prolog code without surprises. The **development** version is released roughly every two to four weeks. This is the recommended version for developers and users of applications such as SWISH or ClioPatria. Finally, the **GIT** and **daily** versions are for developers that want to contribute or have immediate access to patches. These versions are generally fine, but occasionally suffer from regression.

- Stable release
- Development release
- Daily builds for Windows
- Browse GIT repository

Read more about

Fig. 1.3 Versions of SWI Prolog available for download

5. Then download the setup which is suitable for your system.

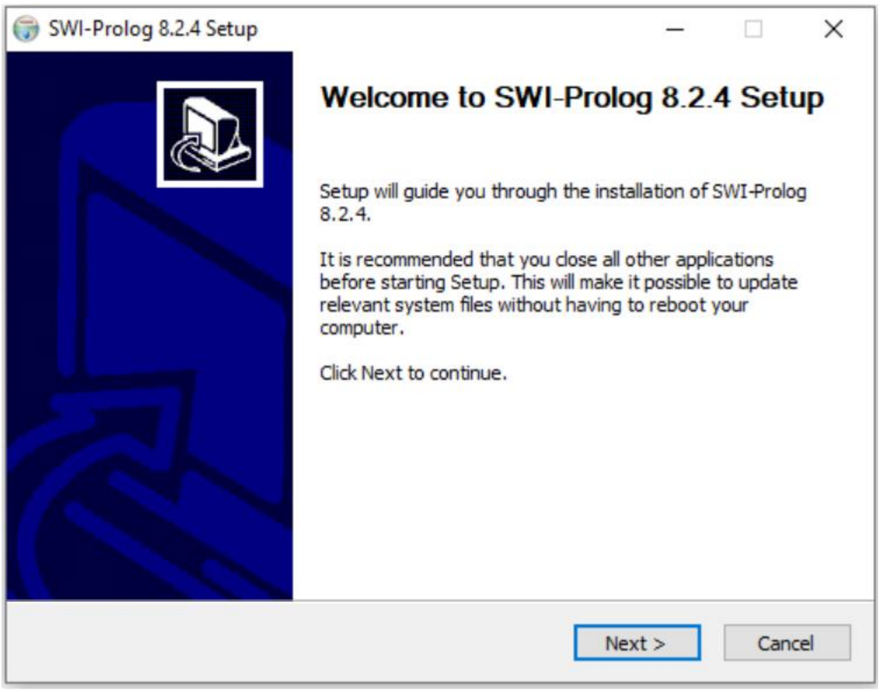


		Binaries
	12,484,490 bytes	SWI-Prolog **8.4.1-1** for Microsoft Windows (64 bit) Self-installing executable for Microsoft's Windows 64-bit editions. Requires at least Windows 7. See the reference manual for deciding on whether to use the 32- or 64-bits version. This binary is linked against GMP 6.1.1 which is covered by the LGPL license. **SHA256**: `212271267caf0b914534519d4a553feb8e4608979d635cfaba63c125461084e5`
	12,472,021 bytes	SWI-Prolog **8.4.1-1** for Microsoft Windows (32 bit) Self-installing executable for MS-Windows. Requires at least Windows 7. Installs **swipl-win.exe** and **swipl.exe**. This binary is linked against GMP 6.1.1 which is covered by the LGPL license. **SHA256**: `17355cd21f82d3778db53d772ab9c6c2f4df065653bbab36105ba3d4321811f9`
	28,195,489 bytes	SWI-Prolog **8.4.1-1** for MacOSX bundle on intel Installer with binaries created using Macports. Installs `/opt/local/bin/swipl`. Needs xquartz (X11) and the Developer Tools (Xcode) installed for running the development tools **SHA256**: `1b9c62caa781818a0dafd1d822ab563b8c10c7cd018ce10a3b71f900eb3a434f`
		Sources
	11,386,908 bytes	SWI-Prolog source for **8.4.1** Sources in `.tar.gz` format, including packages and generated documentation files. See build instructions. **SHA256**: `30bb6542b7767e47b94bd92e8e8a7d7a8a000061044046edf45fc864841b69c4`
		Documentation
	2,906,009 bytes	SWI-Prolog **8.4.1** reference manual in PDF SWI-Prolog reference manual as PDF file. This does *not* include the package documentation.

Fig. 1.4 Different versions of SWI Prolog

6. Then go to download and click on the setup.

7. Click on next



SWI-Prolog 8.2.4 Setup

Welcome to SWI-Prolog 8.2.4 Setup

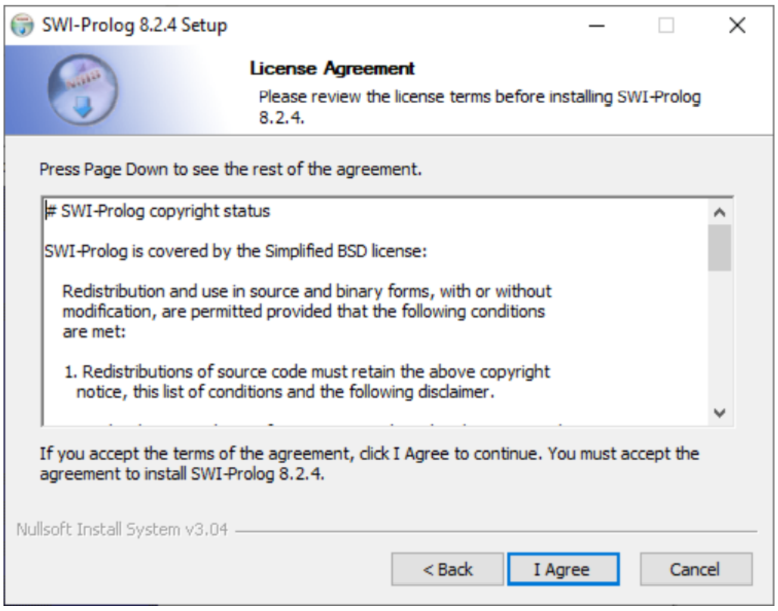
Setup will guide you through the installation of SWI-Prolog 8.2.4.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Fig. 1.5 Setup of SWI Prolog

8. Click on I Agree



SWI-Prolog 8.2.4 Setup

License Agreement

Please review the license terms before installing SWI-Prolog 8.2.4.

Press Page Down to see the rest of the agreement.

SWI-Prolog copyright status

SWI-Prolog is covered by the Simplified BSD license:

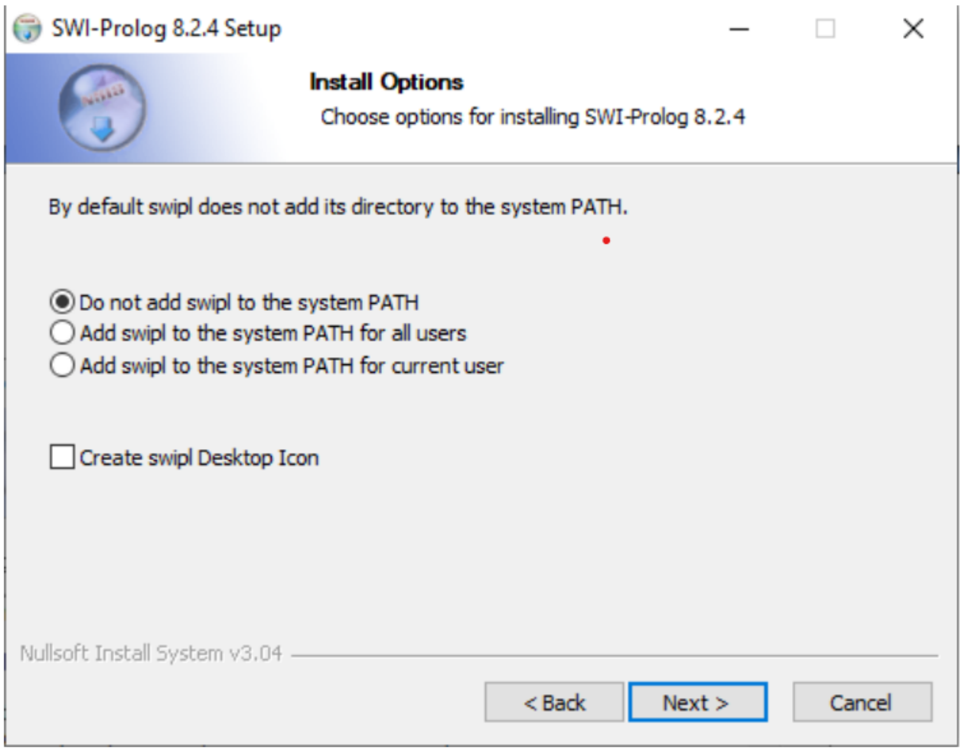
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install SWI-Prolog 8.2.4.

Fig. 1.6 Terms and Conditions page on SWI Prolog setup

9. Click on next


SWI-Prolog 8.2.4 Setup

Install Options

Choose options for installing SWI-Prolog 8.2.4

By default swipl does not add its directory to the system PATH.

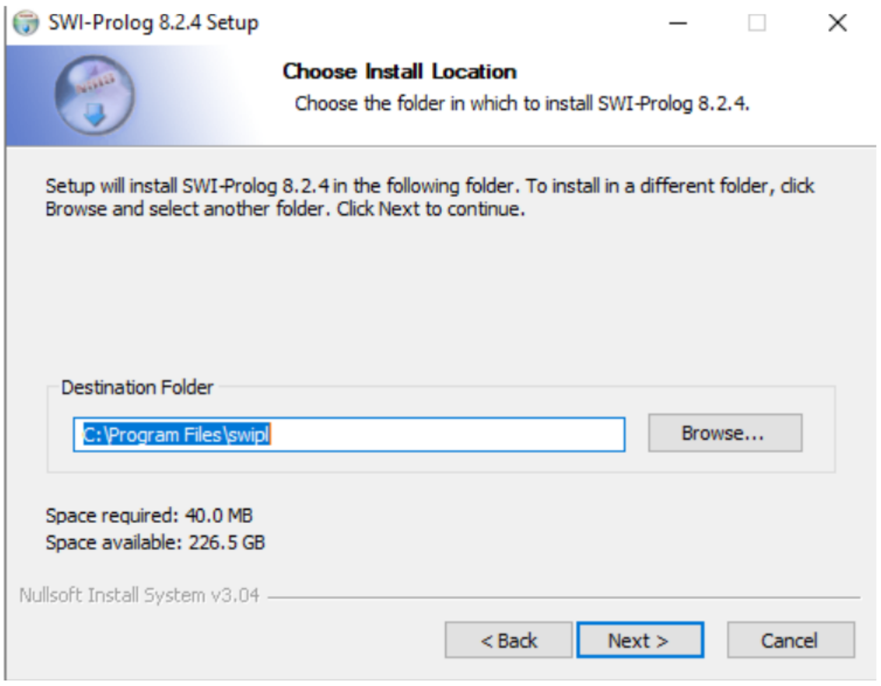
- [✓ Do not add swipl to the system PATH]
- Add swipl to the system PATH for all users
- Add swipl to the system PATH for current user

- [ ] Create swipl Desktop Icon

Nullsoft Install System v3.04

Fig. 1.7 Options for adding swipl in system path

10. Select the folder where you want to install and then click on next.


SWI-Prolog 8.2.4 Setup

Choose Install Location

Choose the folder in which to install SWI-Prolog 8.2.4.

Setup will install SWI-Prolog 8.2.4 in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

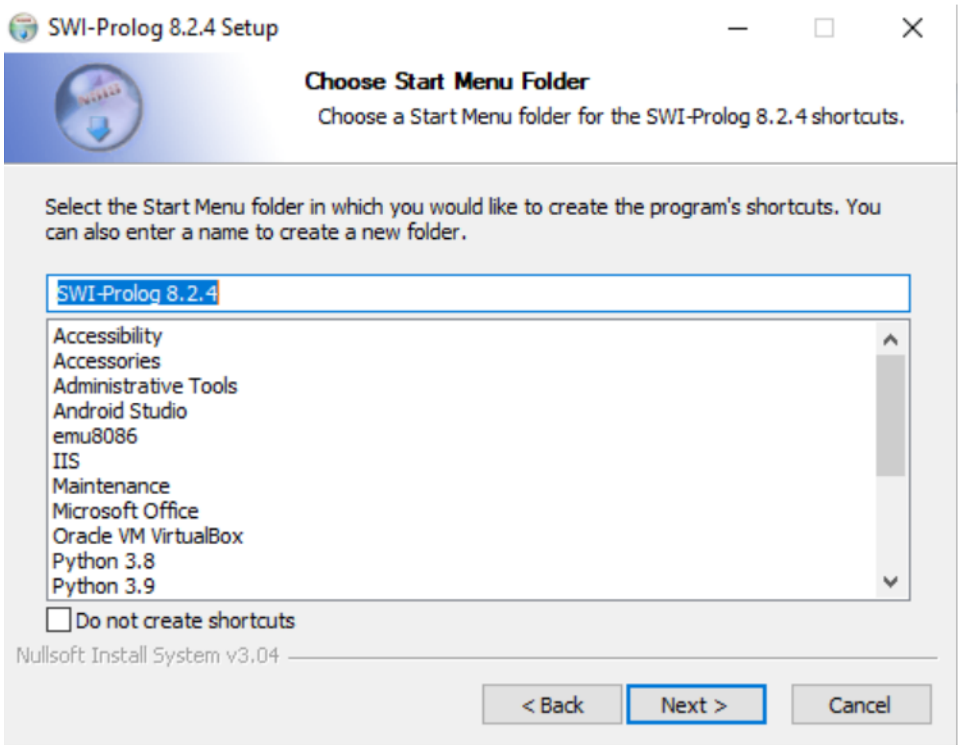
Destination Folder: C:\Program Files\swipl

Space required: 40.0 MB
Space available: 226.5 GB

Nullsoft Install System v3.04

Fig. 1.8 Setting swipl installation path

11. Click on next


SWI-Prolog 8.2.4 Setup

Choose Start Menu Folder

Choose a Start Menu folder for the SWI-Prolog 8.2.4 shortcuts.

Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

SWI-Prolog 8.2.4

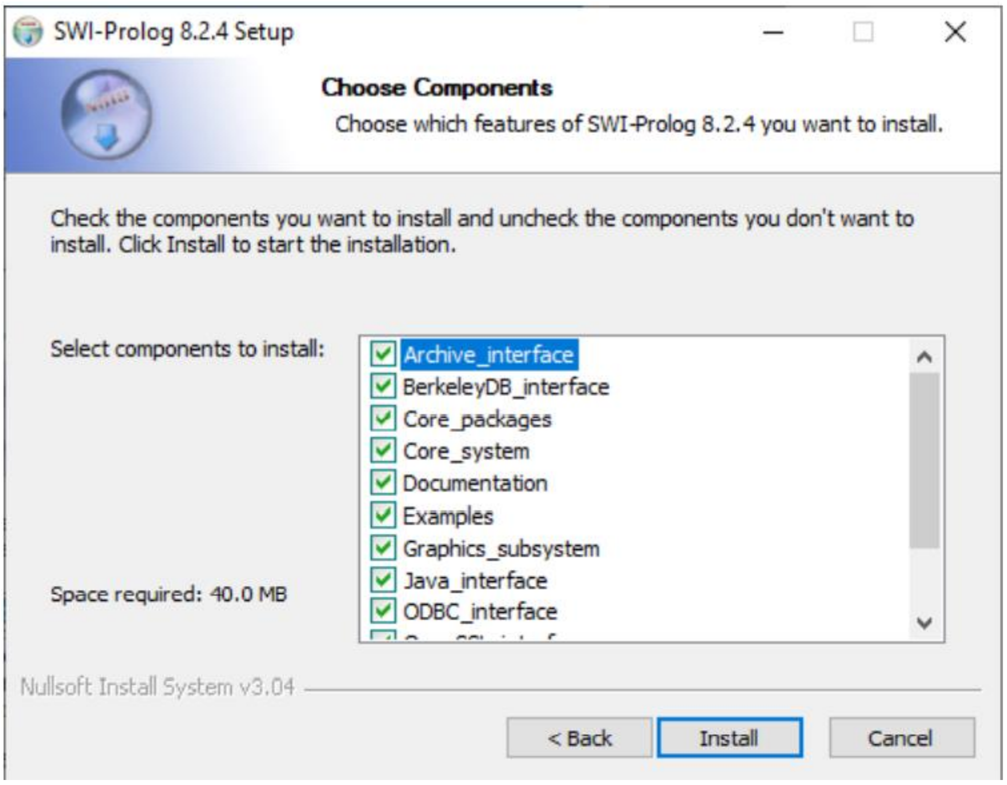
Accessibility
Accessories
Administrative Tools
Android Studio
emu8086
IIS
Maintenance
Microsoft Office
Oracle VM VirtualBox
Python 3.8
Python 3.9

☐ Do not create shortcuts

Nullsoft Install System v3.04

Fig. 1.9 Selecting the start menu folder we want

12. Click on Install


SWI-Prolog 8.2.4 Setup

Choose Components

Choose which features of SWI-Prolog 8.2.4 you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Install to start the installation.

Select components to install:

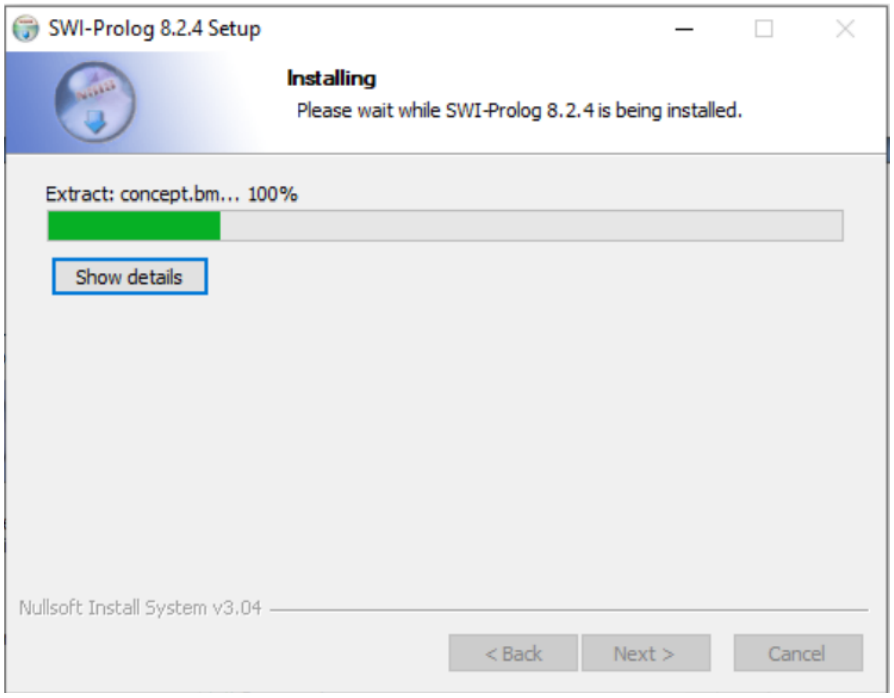
☑ Archive_interface
☑ BerkeleyDB_interface
☑ Core_packages
☑ Core_system
☑ Documentation
☑ Examples
☑ Graphics_subsystem
☑ Java_interface
☑ ODBC_interface

Space required: 40.0 MB

Nullsoft Install System v3.04

Fig. 1.10 Checking the components to install

13. Wait until it gets installed.


SWI-Prolog 8.2.4 Setup

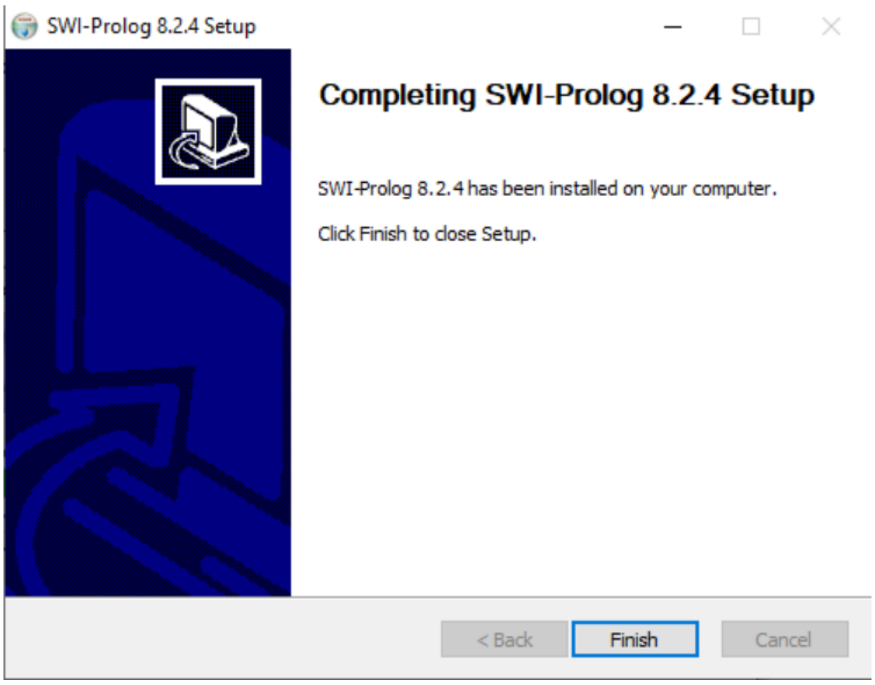
Installing

Please wait while SWI-Prolog 8.2.4 is being installed.

Extract: concept.bm... 100%

Fig. 1.11 Installing swipl

14. Click on Finish.


SWI-Prolog 8.2.4 Setup

Completing SWI-Prolog 8.2.4 Setup

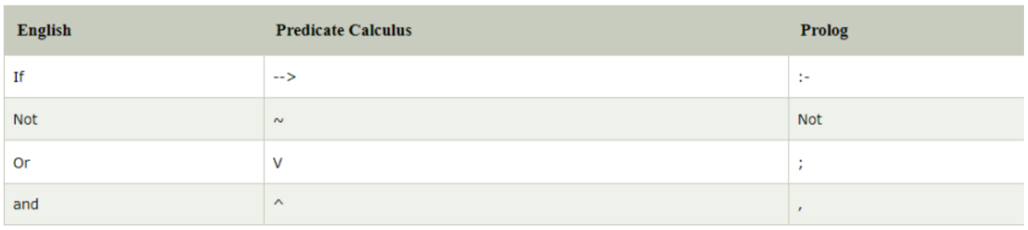
SWI-Prolog 8.2.4 has been installed on your computer.

Click Finish to close Setup.

Fig. 1.12 Finished installation

Prolog Syntax:

1. **Symbol –**



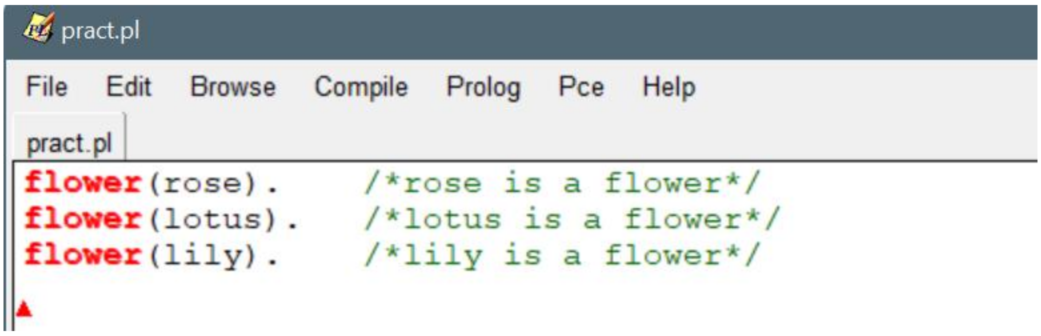
English	Predicate Calculus	Prolog
If	-->	:-
Not	~	Not
Or	V	;
and	^	,

Fig. 1.13 Symbols used in Prolog

2. **Variable -** Variable is a string. The string can be a combination of lower case or upper case letters. The string can also contain underscore characters that begin with an underscore or an upper-case letter. Rules for forming names and predicate calculus are the same.

3. **Facts -** A fact is a predicate expression that makes a declarative statement about the problem domain. Whenever a variable occurs in a Prolog expression, it is assumed to be universally quantified. Note that all Prolog sentences must end with a period.

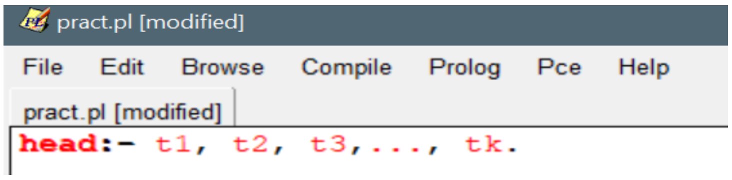
   Example:



```
flower(rose).     /*rose is a flower*/
flower(lotus).    /*lotus is a flower*/
flower(lily).     /*lily is a flower*/
```

Fig. 1.14 Facts in Prolog

4. **Rules -** A rule is a predicate expression that uses logical implication (:-) to describe a relationship among facts. Thus, a Prolog rule takes the form:

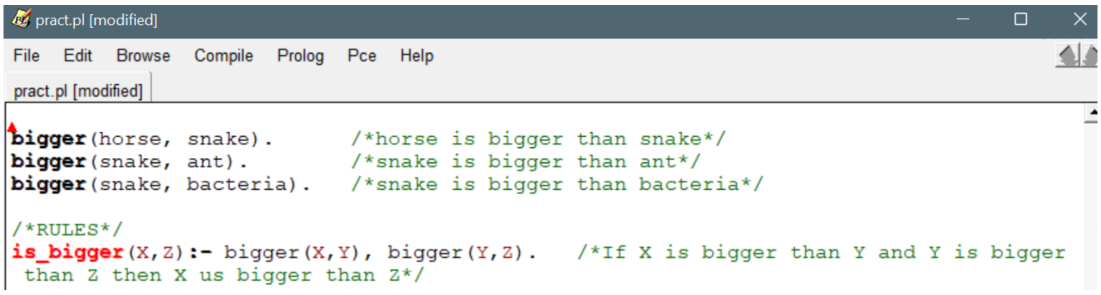


```
head:- t1, t2, t3,..., tk.
```

Fig. 1.15 Syntax for writing rules in Prolog

where k>=1.
The head is known as the clause of the head.
:- is known as the clause neck. It is read as 'if'. The body of the clause is specified by t1, t2, t3, …, tk. It contains one or more components, and it can be separated using the commas. A rule will read as 'head is true if t1, t2, t3, …., tk are all true'.

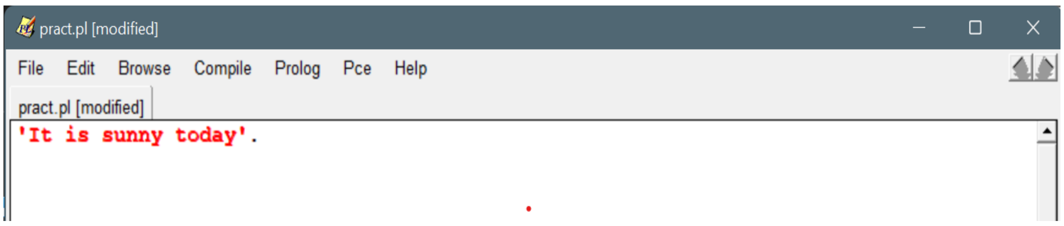
Example:



```
bigger(horse, snake).      /*horse is bigger than snake*/
bigger(snake, ant).        /*snake is bigger than ant*/
bigger(snake, bacteria).   /*snake is bigger than bacteria*/

/*RULES*/
is_bigger(X,Z):- bigger(X,Y), bigger(Y,Z).   /*If X is bigger than Y and Y is bigger
 than Z then X us bigger than Z*/
```

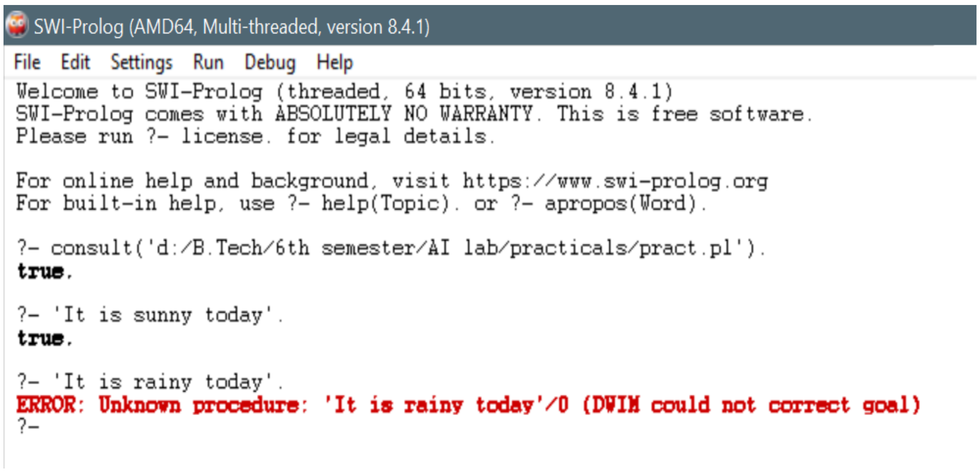
Fig. 1.16 Rules in Prolog

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:



```
'It is sunny today'.
```

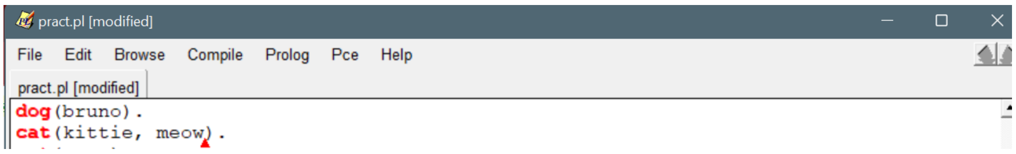
Fig. 1.17 Queries in Prolog


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.1)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- consult('d:/B.Tech/6th semester/AI lab/practicals/pract.pl').
true.

?- 'It is sunny today'.
true.

?- 'It is rainy today'.
ERROR: Unknown procedure: 'It is rainy today'/0 (DWIM could not correct goal)
?-
```

Fig. 1.18 Output of queries typed in file

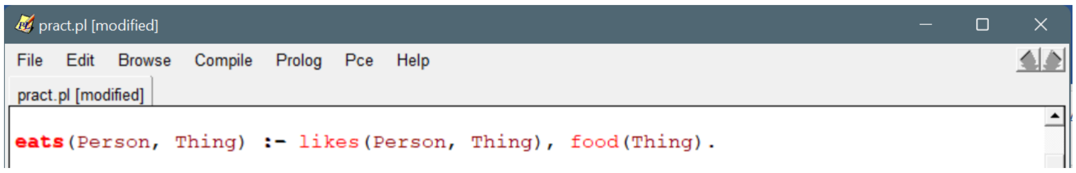
6. **Clause** - A clause in Prolog is a unit of information in a Prolog program ending with a full stop ("."). A clause may be a fact, like:



```
dog(bruno).
cat(kittie, meow).
```

Fig. 1.19 Clause in Prolog in the form of fact

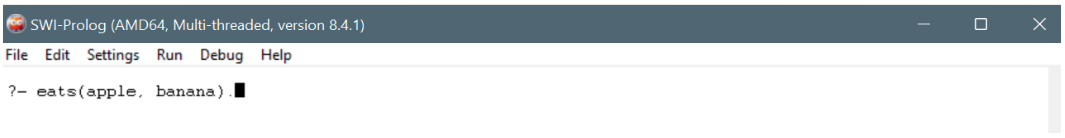
or a rule, like:



```
eats(Person, Thing) :- likes(Person, Thing), food(Thing).
```

Fig. 1.20 Clause in Prolog in the form of rule

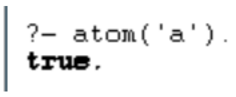
A clause may also be a query to the Prolog interpreter, as in:


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
?- eats(apple, banana).
```

Fig. 1.21 Clause in Prolog terminal

7. **Atoms** – Atoms are identifiers. They are typically used in cases where identity comparison is the main operation and that are typically not composed nor taken apart.



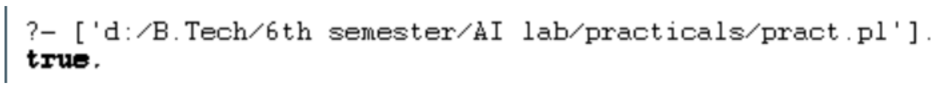
```
?- atom('a').
true.
```

Fig. 1.22 Atoms in Prolog

Important points and commands in PROLOG:

1. **Creating a file-** Name of the file should be in the following extension:
   Name_of_file.pl

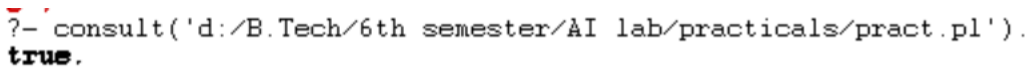
2. **To compile file-**
   a. Method 1: Writing path of file inside [‘’] with forward slash.



```
?- ['d:/B.Tech/6th semester/AI lab/practicals/pract.pl'].
true.
```

Fig. 1.23 Saving file in Prolog

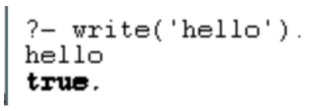
   b. Method 2: Using consult



```
?- consult('d:/B.Tech/6th semester/AI lab/practicals/pract.pl').
true.
```

Fig. 1.24 Saving file in Prolog using consult

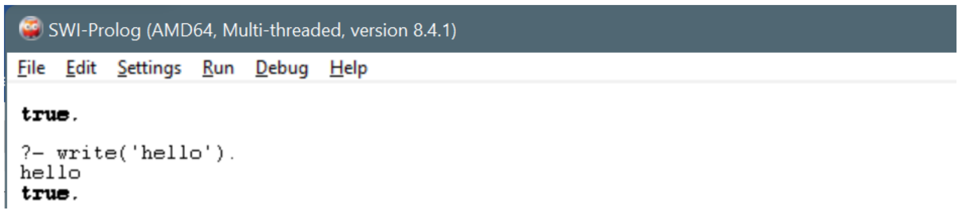
3. **write()- to print text.**



```
?- write('hello').
hello
true.
```

Fig. 1.25

4. **write('\33\[2J')-** To clear screen.

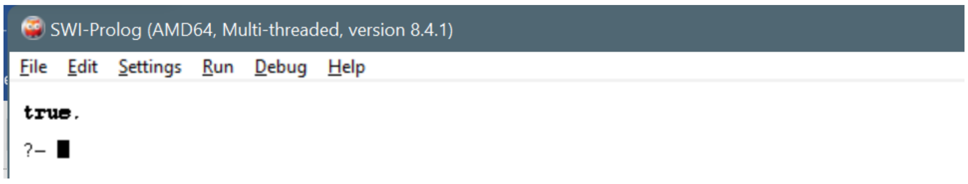


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)

File Edit Settings Run Debug Help

```
true.

?- write('hello').
hello
true.
```

Fig. 1.26 Command to clear screen in Prolog



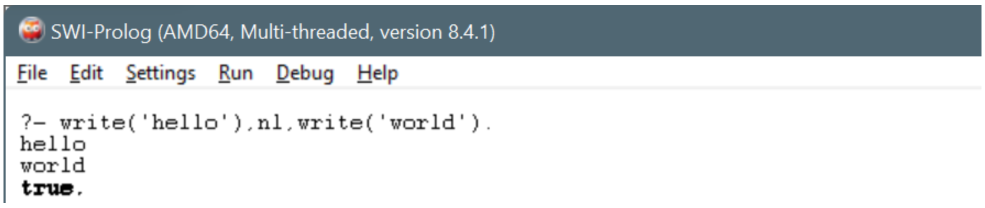
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)

File Edit Settings Run Debug Help

```
true.

?- █
```

Fig. 1.27 Output on using clear command in Prolog

5. **nl-** For print text on new line.


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)

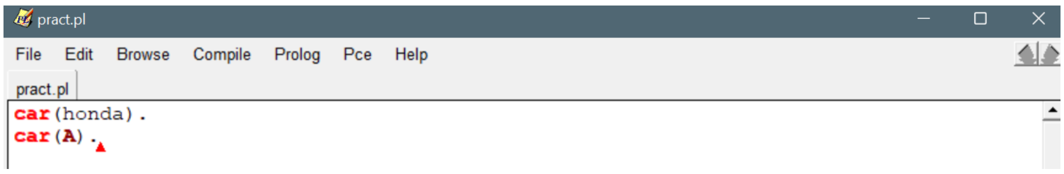
```
?- write('hello'),nl,write('world').
hello
world
true.
```

Fig. 1.28 Command to add new line in Prolog

6. **halt-** to exit SWI-Prolog.

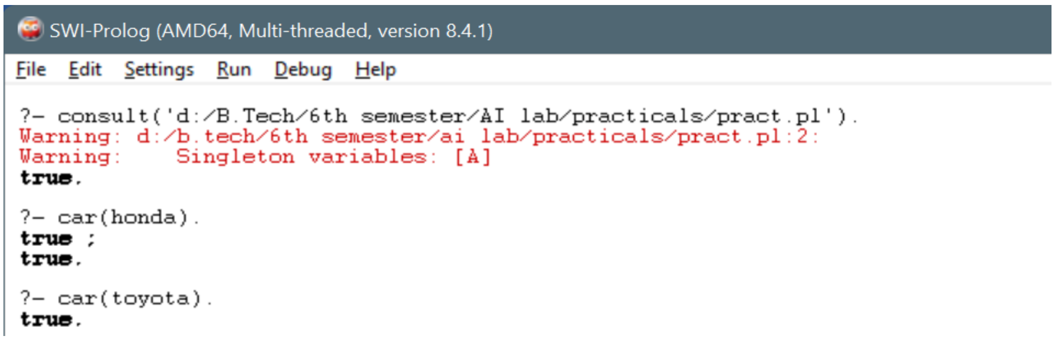
7. **Variable declaration in PROLOG-**

   The variable in the prolog should always be declared in the caps. For example, X, Y, Z, etc. Here X, Y, Z is a variable.


pract.pl

```
car(honda).
car(A).
```

Fig. 1.29 Example of variable in Prolog

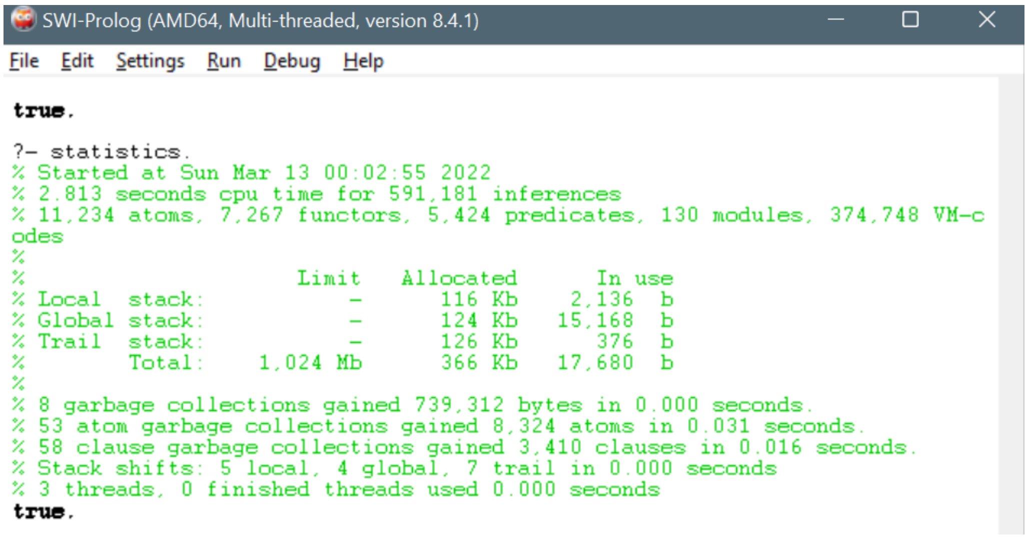

SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)

```
?- consult('d:/B.Tech/6th semester/AI lab/practicals/pract.pl').
Warning: d:/b.tech/6th semester/ai lab/practicals/pract.pl:2:
Warning:    Singleton variables: [A]
true.

?- car(honda).
true ;
true.

?- car(toyota).
true.
```

Fig. 1.30 Ways in which variable should not be declared in Prolog

8. **Anonymous variables-**
   The anonymous variables in prolog are written by a single underscore character '_'. And one important thing is that each individual anonymous variable is treated as different. They are not same. When we want to use the variable, but do not want to reveal the value of the variable, then we can use anonymous variables.

9. **Statistics-**
   This library predicts to obtain information about resources usage by program.

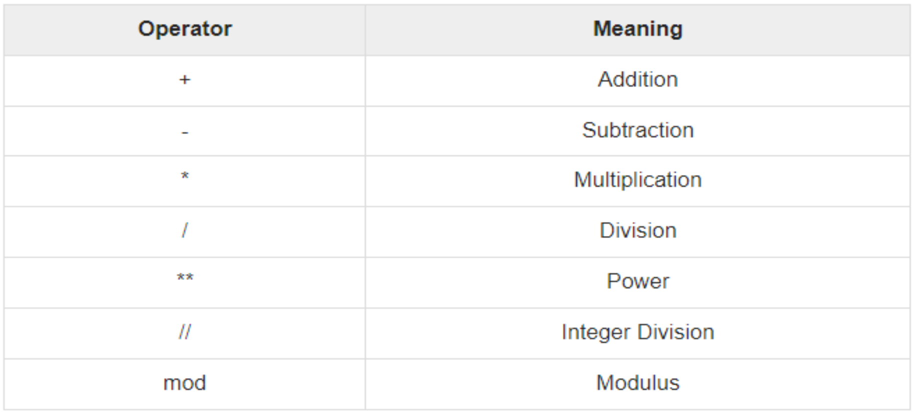

SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
true.

?- statistics.
% Started at Sun Mar 13 00:02:55 2022
% 2.813 seconds cpu time for 591,181 inferences
% 11,234 atoms, 7,267 functors, 5,424 predicates, 130 modules, 374,748 VM-c
odes
%
%                    Limit   Allocated      In use
% Local  stack:          -      116 Kb    2,136  b
% Global stack:          -      124 Kb   15,168  b
% Trail  stack:          -      126 Kb      376  b
%        Total:   1,024 Mb      366 Kb   17,680  b
%
% 8 garbage collections gained 739,312 bytes in 0.000 seconds.
% 53 atom garbage collections gained 8,324 atoms in 0.031 seconds.
% 58 clause garbage collections gained 3,410 clauses in 0.016 seconds.
% Stack shifts: 5 local, 4 global, 7 trail in 0.000 seconds
% 3 threads, 0 finished threads used 0.000 seconds
true.
```

Fig. 1.31 Statistics command in Prolog

Operators in Prolog.

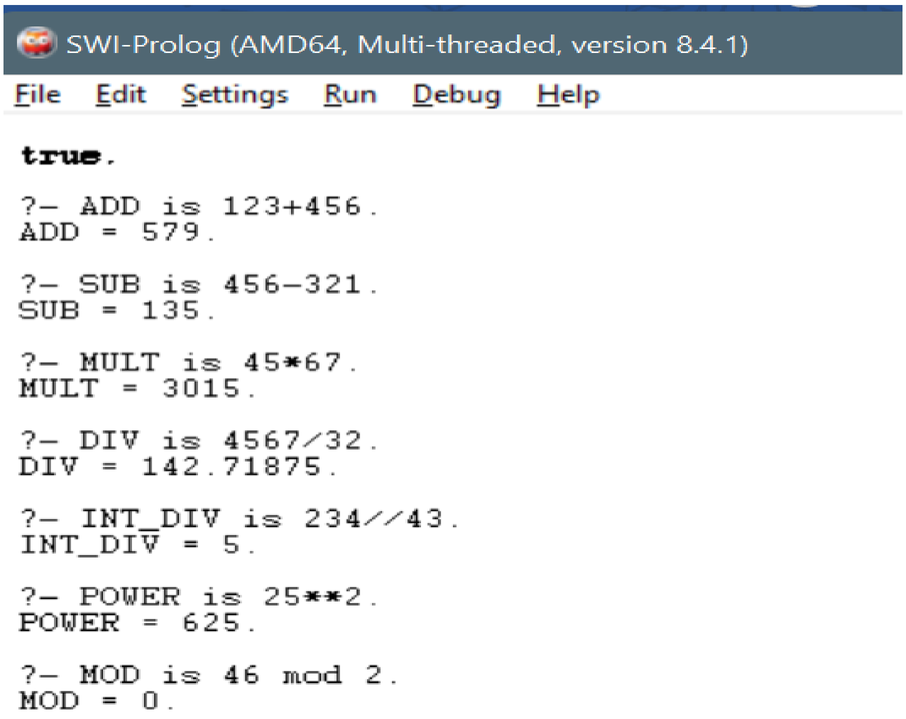
- Arithmetic Operators in Prolog: Arithmetic operators are used to perform arithmetic operations.



Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
//	Integer Division
mod	Modulus

Fig 1.32 Operators in Prolog

Source code for arithmetic operators.


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

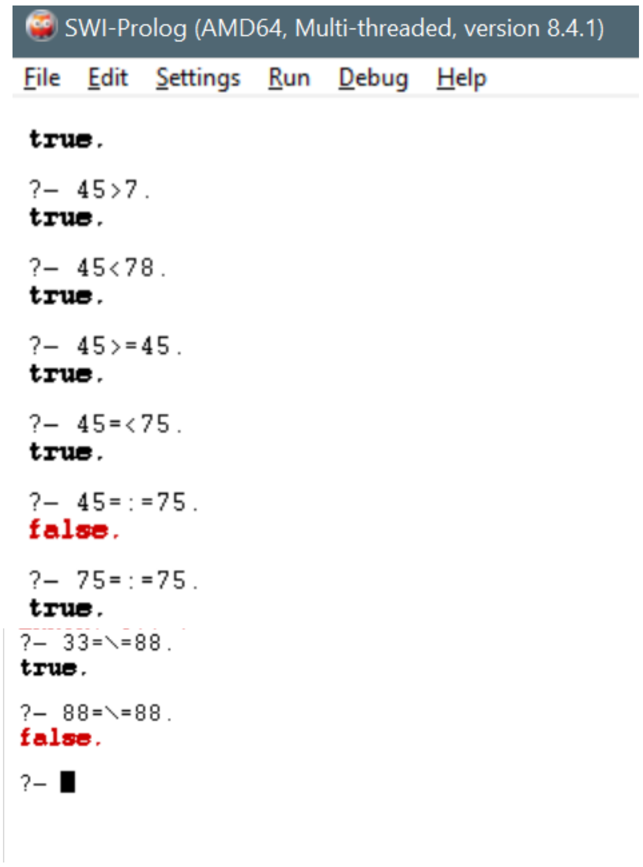
```
true.

?- ADD is 123+456.
ADD = 579.

?- SUB is 456-321.
SUB = 135.

?- MULT is 45*67.
MULT = 3015.

?- DIV is 4567/32.
DIV = 142.71875.

?- INT_DIV is 234//43.
INT_DIV = 5.

?- POWER is 25**2.
POWER = 625.

?- MOD is 46 mod 2.
MOD = 0.
```

Fig 1.33 Arithmetic Operations in Prolog

Comparison Operators:

Operator	Meaning
X>Y	X is greater than Y
X<Y	X is less than Y.
X>=Y	X is greater than or equal to Y.
X=<Y	X is less than or equal to Y.
X=:=Y	The values of X and Y are equal.
X=\=Y	The values of X and Y are not equal.

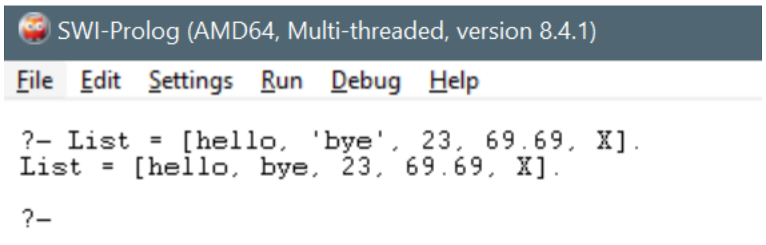
Source code:


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
true.

?- 45>7.
true.

?- 45<78.
true.

?- 45>=45.
true.

?- 45=<75.
true.

?- 45=:=75.
false.

?- 75=:=75.
true.
?- 33=\=88.
true.

?- 88=\=88.
false.

?-
```

Fig 1.34 Comparison Operator in Prolog

Concept of Lists:
The list is a simple data structure that is widely used in non-numeric programming. List consists of any number of items, for example, red, green, blue, white, dark. It will be represented as, [red, green, blue, white, dark]. The list of elements will be enclosed with square brackets.


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

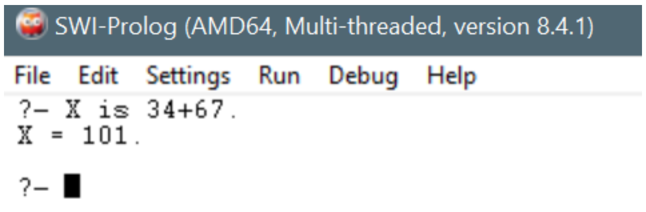
```
?- List = [hello, 'bye', 23, 69.69, X].
List = [hello, bye, 23, 69.69, X].

?-
```

Fig 1.35 Lists in Prolog

List Operations:

Operations	**Definition**
Membership Checking	During this operation, we can verify whether a given element is member of specified list or not?
Length Calculation	With this operation, we can find the length of a list.
Concatenation	Concatenation is an operation which is used to join/add two lists.
Delete Items	This operation removes the specified element from a list.
Append Items	Append operation adds one list into another (as an item).
Insert Items	This operation inserts a given item into a list.

Concept of 'Is'
The is built-in predicate is used in Prolog to force the evaluation of arithmetic expressions.


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
?- X is 34+67.
X = 101.

?- 
```

Fig 1.36 Is in ProLog

Built-In Predicates
In Prolog, we have seen the user-defined predicates in most of the cases, but there are some built-in predicates. There are three types of built-in predicates as given below –

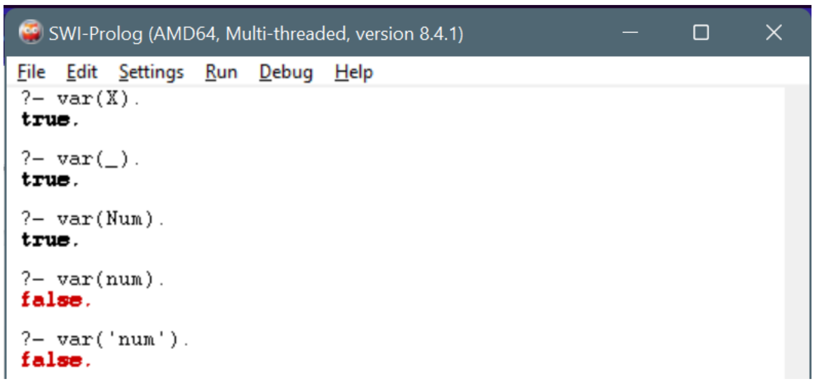
- Identifying terms
- Decomposing structures
- Collecting all solutions

So this is the list of some predicates that fall under the identifying terms group –

Predicate	Description
var(X)	succeeds if X is currently an un-instantiated variable.
novar(X)	succeeds if X is not a variable, or already instantiated
atom(X)	is true if X currently stands for an atom
number(X)	is true if X currently stands for a number
integer(X)	is true if X currently stands for an integer
float(X)	is true if X currently stands for a real number.
atomic(X)	is true if X currently stands for a number or an atom.
compound(X)	is true if X currently stands for a structure.
ground(X)	succeeds if X does not contain any un-instantiated variables.

Figure 1.37 Built-in Predicates in Prolog

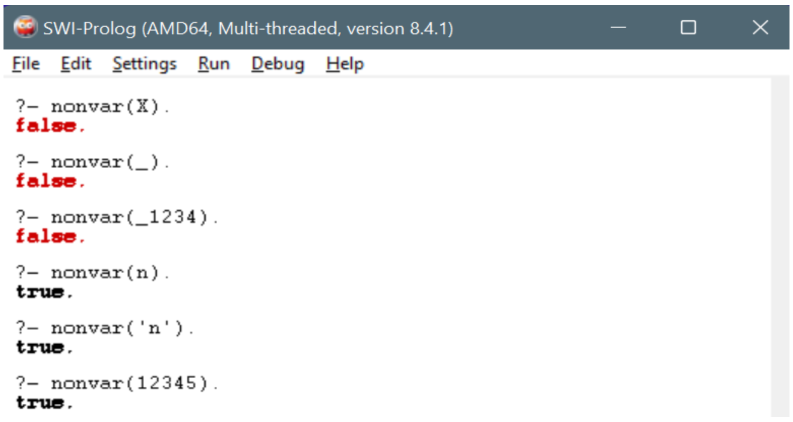
1. The var(X) Predicate: When X is not initialized, then, it will show true, otherwise false. So let us see an example.


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
?- var(X).
true.

?- var(_).
true.

?- var(Num).
true.

?- var(num).
false.

?- var('num').
false.
```

Figure 1.38 var in Prolog

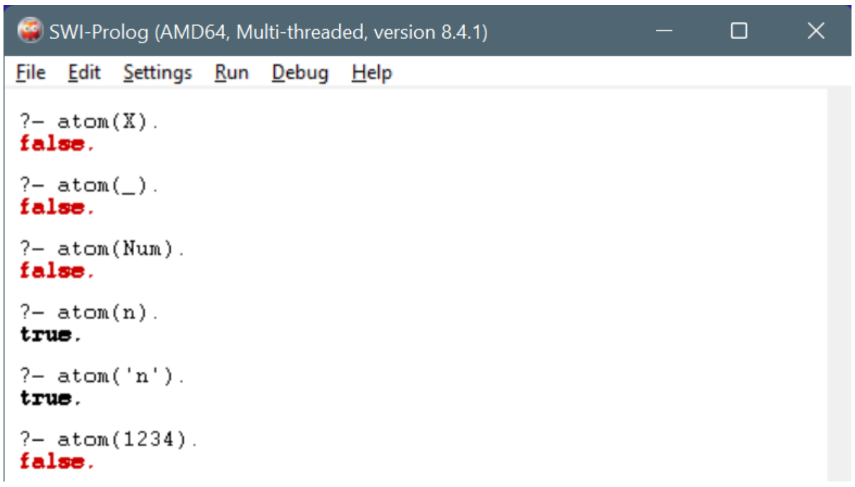
2. nonvar(X): When X is not initialized, the, it will show false, otherwise true. So let us see an example.


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
?- nonvar(X).
false.

?- nonvar(_).
false.

?- nonvar(_1234).
false.

?- nonvar(n).
true.

?- nonvar('n').
true.

?- nonvar(12345).
true.
```

Figure 1.39 nonvar in Prolog

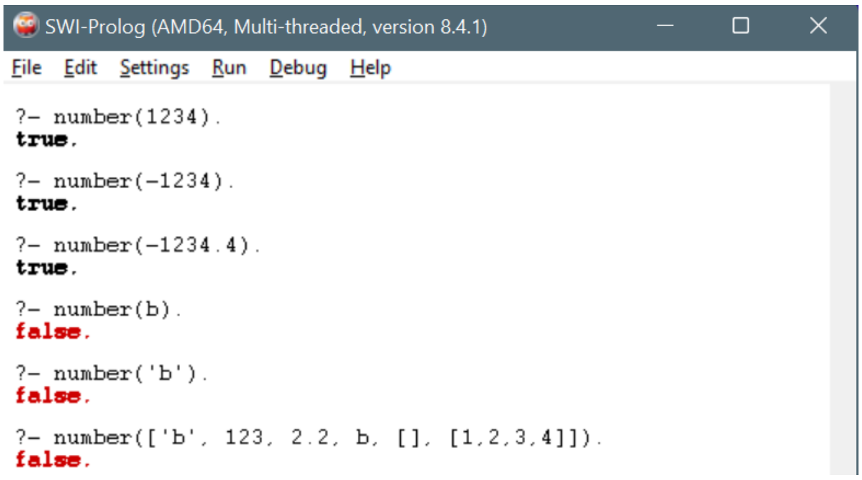
3. The atom(X) Predicate: This will return true when a non-variable term with 0 arguments and a not numeric term is passed as X, otherwise false.


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
?- atom(X).
false.

?- atom(_).
false.

?- atom(Num).
false.

?- atom(n).
true.

?- atom('n').
true.

?- atom(1234).
false.
```

Figure 1.40 atom in Prolog

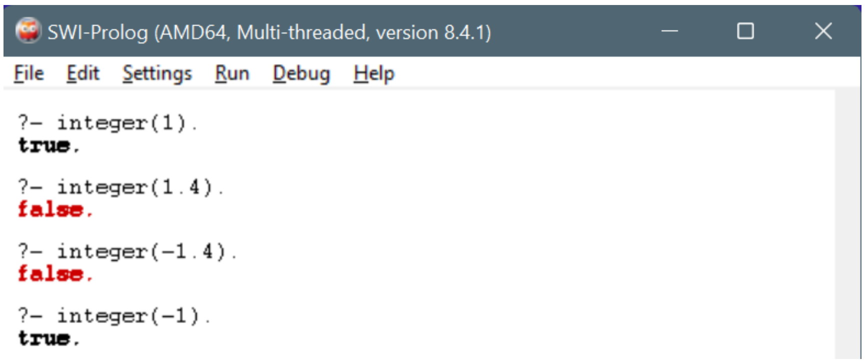
4. The number(X) Predicate: This will return true, X stands for any number, otherwise false.


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
?- number(1234).
true.

?- number(-1234).
true.

?- number(-1234.4).
true.

?- number(b).
false.

?- number('b').
false.

?- number(['b', 123, 2.2, b, [], [1,2,3,4]]).
false.
```

Figure 1.41 number in Prolog

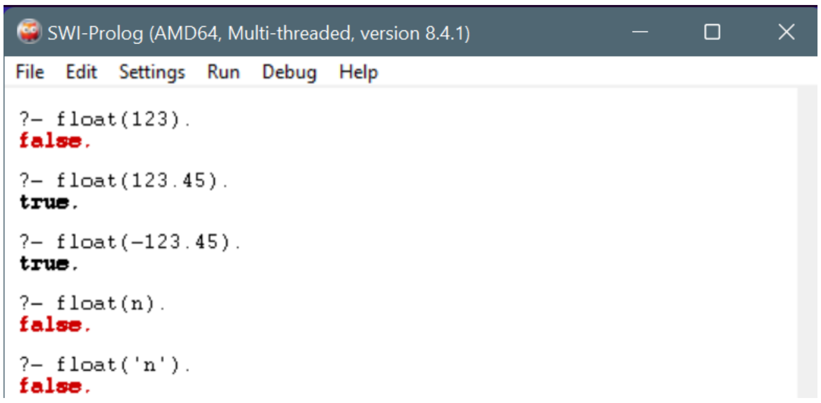
5. The integer(X) Predicate: This will return true, when X is a positive or negative integer value, otherwise false. The float(X) Predicate: This will return true, X is a floating number, otherwise false.


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
?- integer(1).
true.

?- integer(1.4).
false.

?- integer(-1.4).
false.

?- integer(-1).
true.
```

Figure 1.42 integer in Prolog

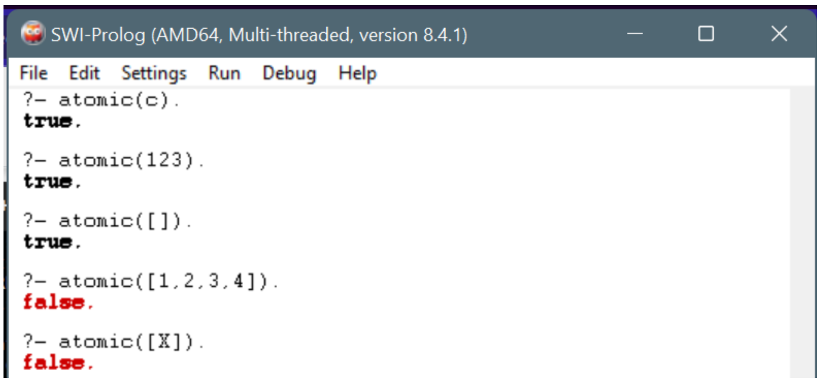
6. The float(X) Predicate: This will return true, X is a floating number, otherwise false.


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
?- float(123).
false.

?- float(123.45).
true.

?- float(-123.45).
true.

?- float(n).
false.

?- float('n').
false.
```

Figure 1.43 float in Prolog

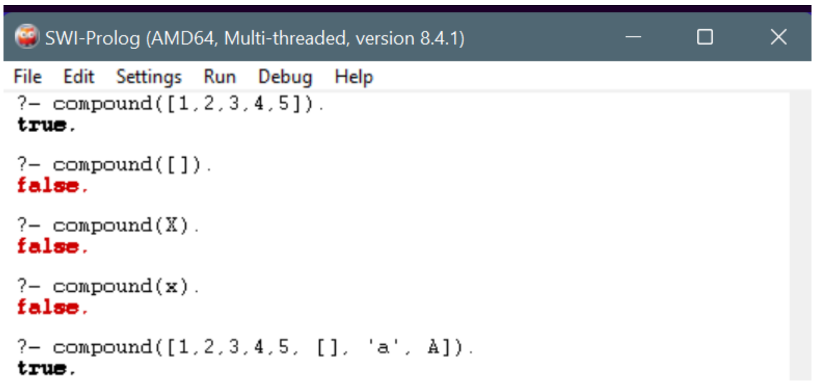
7. The atomic(X) Predicate: We have atom(X), that is too specific, it returns false for numeric data, the atomic(X) is like atom(X) but it accepts a number.


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
?- atomic(c).
true.

?- atomic(123).
true.

?- atomic([]).
true.

?- atomic([1,2,3,4]).
false.

?- atomic([X]).
false.
```

Figure 1.44 atomic in Prolog

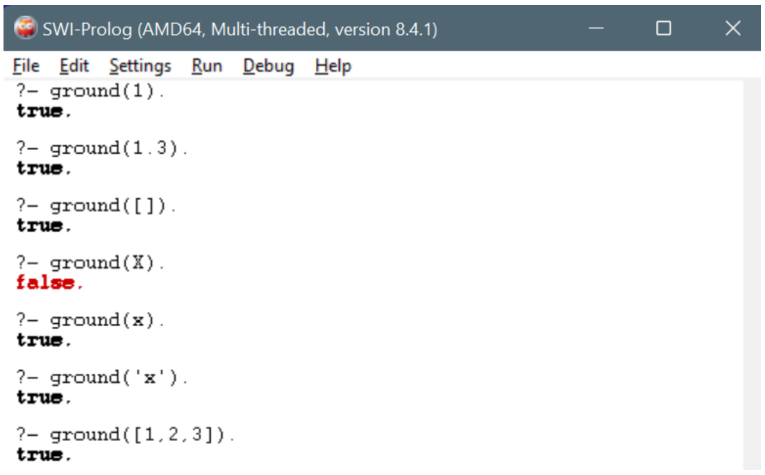
8. The compound(X) Predicate: If atomic(X) fails, then the terms are either one non-instantiated variable (that can be tested with var(X)) or a compound term. Compound will be true when we pass some compound structure.


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

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

?- compound([]).
false.

?- compound(X).
false.

?- compound(x).
false.

?- compound([1,2,3,4,5, [], 'a', A]).
true.
```

Figure 1.45 compound in Prolog

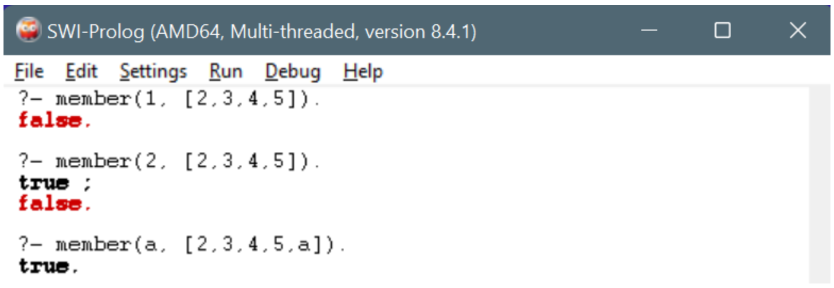
9. The ground(X) Predicate: This will return true if X does not contain any un-instantiated variables. This also checks inside the compound terms, otherwise returns false.


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
?- ground(1).
true.

?- ground(1.3).
true.

?- ground([]).
true.

?- ground(X).
false.

?- ground(x).
true.

?- ground('x').
true.

?- ground([1,2,3]).
true.
```

Figure 1.46 ground in Prolog

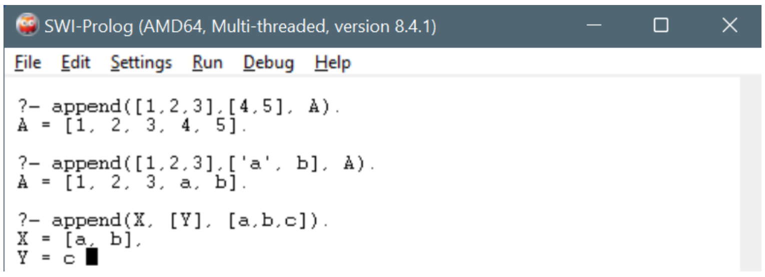
10. Predicate member/2: member(?Elem, ?List).
    - True if Elem is a member of List. The SWI-Prolog definition differs from the classical one. Our definition avoids unpacking each list element twice and provides determinism on the last element. E.g. this is deterministic: member(X, [One]).


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

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

?- member(2, [2,3,4,5]).
true ;
false.

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

Figure 1.47 member in Prolog

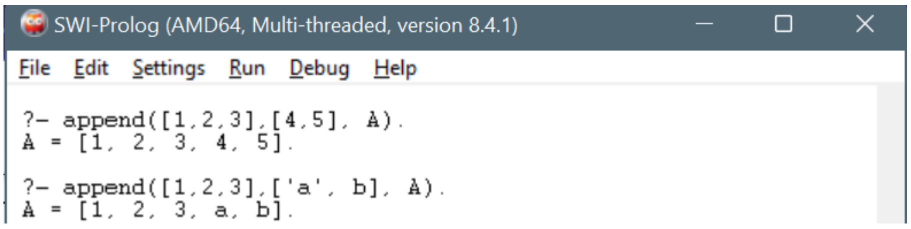
11. Predicate append/3: append(?List1, ?List2, ?List1AndList2)
    - List1AndList2 is the concatenation of List1 and List2


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
?- append([1,2,3],[4,5], A).
A = [1, 2, 3, 4, 5].

?- append([1,2,3],['a', b], A).
A = [1, 2, 3, a, b].

?- append(X, [Y], [a,b,c]).
X = [a, b],
Y = c
```

Figure 1.48 append in Prolog

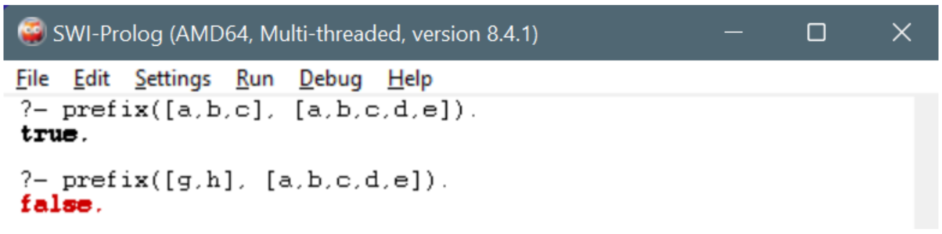
12. Predicate append/2: append(+ListOfLists, ?List)
    - Concatenate a list of lists. Is true if ListOfLists is a list of lists, and List is the concatenation of these lists.


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
?- append([1,2,3],[4,5], A).
A = [1, 2, 3, 4, 5].

?- append([1,2,3],['a', b], A).
A = [1, 2, 3, a, b].
```

Figure 1.49 append in Prolog

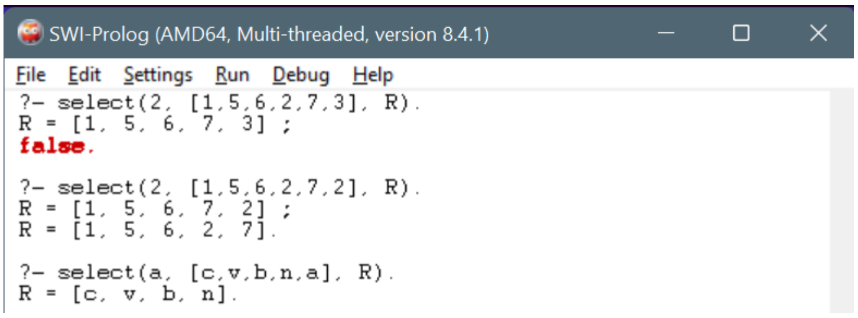
13. Predicate prefix/2: prefix(?Part, ?Whole).
    - True if Part is a leading substring of Whole. This is the same as append(Part, _, Whole).


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
?- prefix([a,b,c], [a,b,c,d,e]).
true.

?- prefix([g,h], [a,b,c,d,e]).
false.
```

Figure 1.50 prefix in Prolog

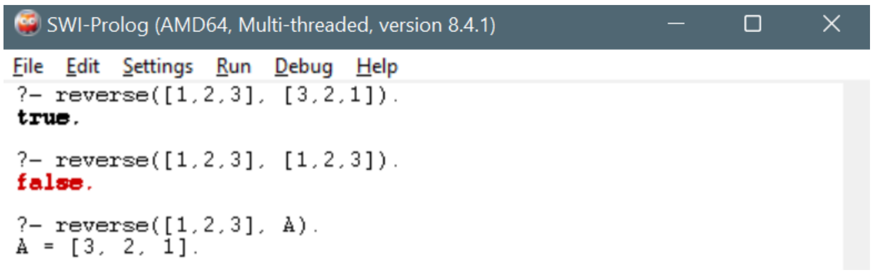
14. Predicate select/3: select(?Elem, ?List1, ?List2)
    - Is true when List1, with Elem removed, results in List2. This implementation is deterministic if the last element of List1 has been selected.


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
?- select(2, [1,5,6,2,7,3], R).
R = [1, 5, 6, 7, 3] ;
false.

?- select(2, [1,5,6,2,7,2], R).
R = [1, 5, 6, 7, 2] ;
R = [1, 5, 6, 2, 7].

?- select(a, [c,v,b,n,a], R).
R = [c, v, b, n].
```

Figure 1.51 select in Prolog

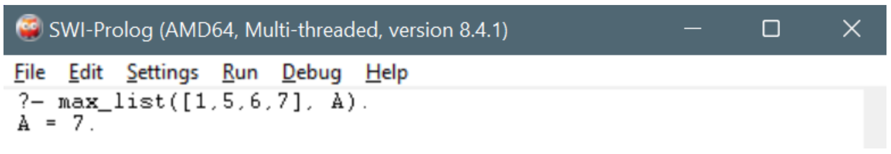
15. Predicate reverse/2: reverse(?List1, ?List2)
    - Is true when the elements of List2 are in reverse order compared to List1.


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
?- reverse([1,2,3], [3,2,1]).
true.

?- reverse([1,2,3], [1,2,3]).
false.

?- reverse([1,2,3], A).
A = [3, 2, 1].
```

Figure 1.52 reverse in Prolog

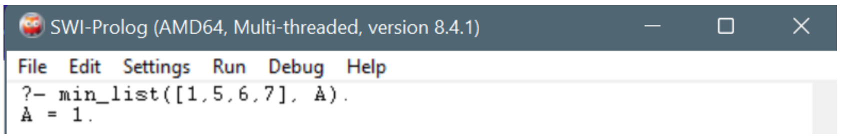
16. Predicate max_list/2: max_list(+List:list(number), -Max:number)
    - True if Max is the largest number in List. Fails if List is empty.


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
?- max_list([1,5,6,7], A).
A = 7.
```

Figure 1.53 max_list in Prolog

17. Predicate min_list/2: min_list(+List:list(number), -Min:number)
    - True if Min is the smallest number in List. Fails if List is empty.


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
?- min_list([1,5,6,7], A).
A = 1.
```

Figure 1.54 min_list in Prolog

Backtracking

The backtracking term is quite common in algorithm designing, and in different programming environments. In Prolog, until it reaches proper destination, it tries to backtrack. When the destination is found, it stops.

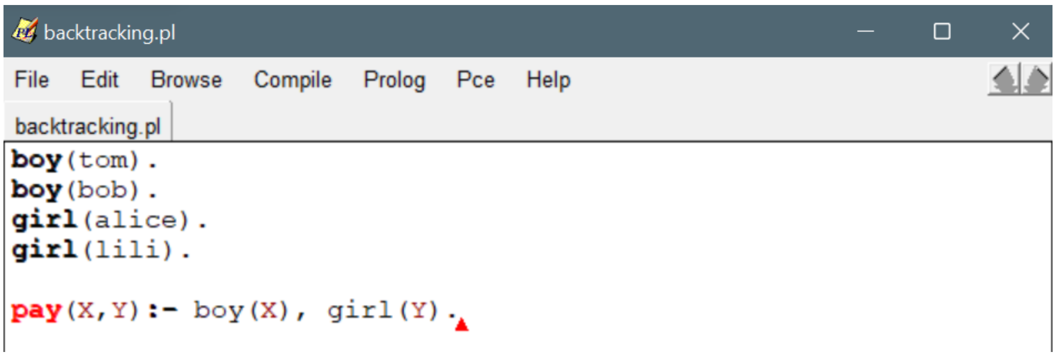
How Backtracking works-

Now we know, what is the backtracking in Prolog. Let us see one example,

Note – While we are running some prolog code, during backtracking there may be multiple answers, we can press **semicolon** (;) to get next answers one by one, that helps to backtrack. Otherwise when we get one result, it will stop.

Now, consider a situation, where two people X and Y can pay each other, but the condition is that a boy can pay to a girl, so X will be a boy, and Y will be a girl. So for these we have defined some facts and rules –

Knowledge Base



```
boy(tom).
boy(bob).
girl(alice).
girl(lili).

pay(X,Y):- boy(X), girl(Y).
```

Figure 1.55 Knowledge Base

Following is the illustration of the above scenario –

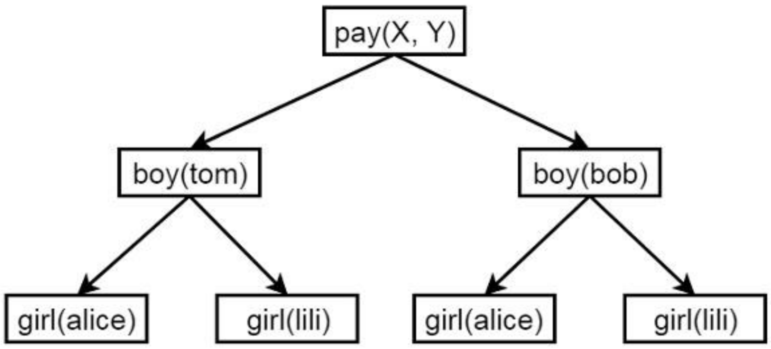
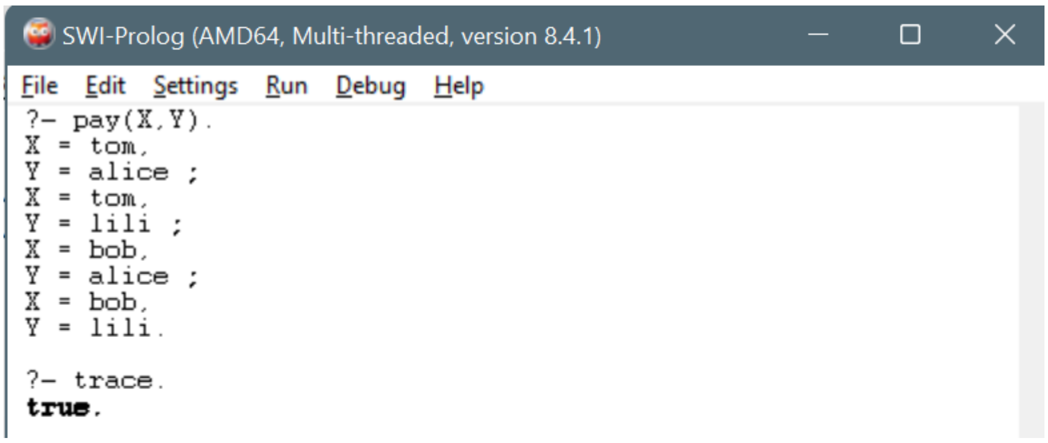


Figure 1.56 Representation of the scenario

As X will be a boy, so there are two choices, and for each boy there are two choices alice and lili. Now let us see the output, how backtracking is working.

Output


SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

```
?- pay(X,Y).
X = tom,
Y = alice ;
X = tom,
Y = lili ;
X = bob,
Y = alice ;
X = bob,
Y = lili.

?- trace.
true.
```

Figure 1.57 Output

Recursion

Recursion is a technique in which one predicate uses itself (may be with some other predicates) to find the truth value.

Let us understand this definition with the help of an example –

- is_digesting(X,Y) :- just_ate(X,Y).
- is_digesting(X,Y) :-just_ate(X,Z),is_digesting(Z,Y).

So this predicate is recursive in nature. Suppose we say that *just_ate(deer, grass)*, it means *is_digesting(deer, grass)* is true. Now if we say *is_digesting(tiger, grass)*, this will be true if *is_digesting(tiger, grass)* :- *just_ate(tiger, deer), is_digesting(deer, grass)*, then the statement *is_digesting(tiger, grass)* is also true.

There may be some other examples also, so let us see one family example. So if we want to express the predecessor logic, that can be expressed using the following diagram –

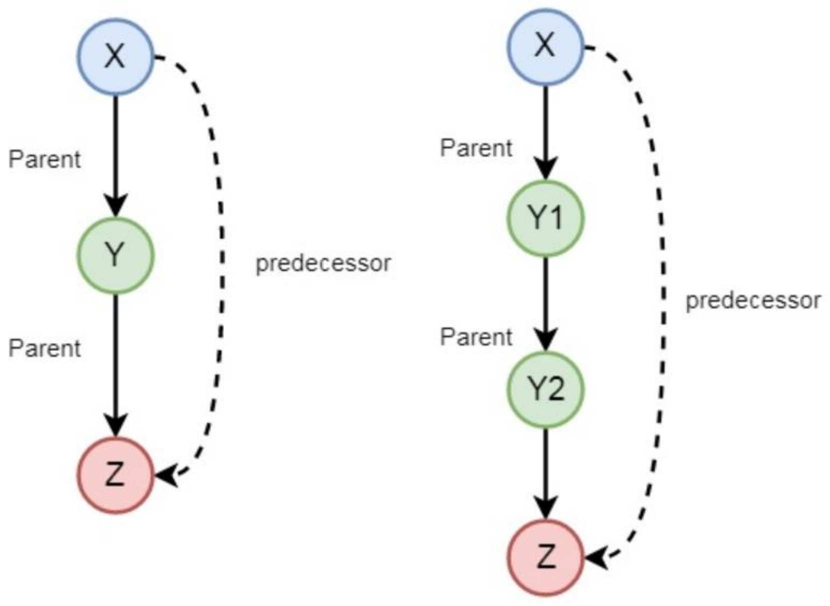


Fig 1.58 Recursion explanation using diagram

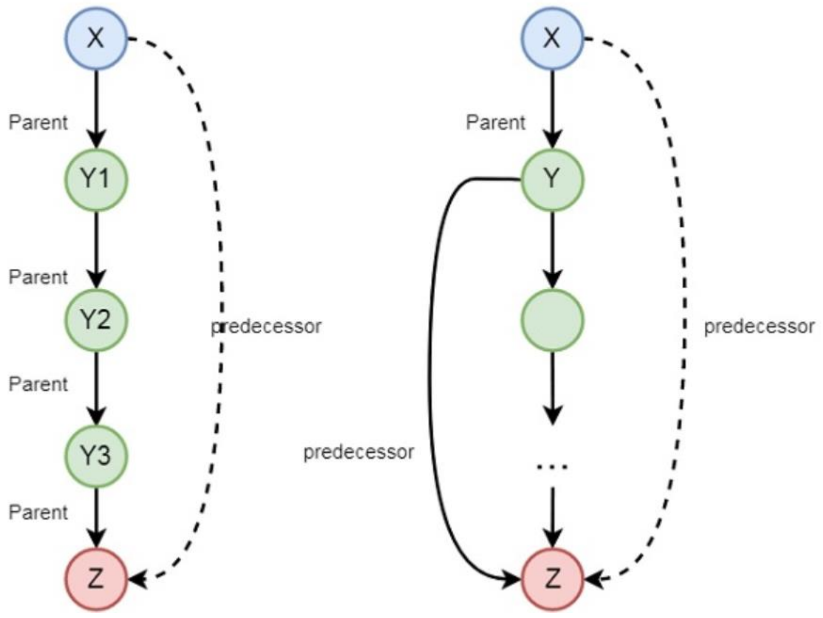


Fig 1.59 Recursion explanation using diagram

So we can understand the predecessor relationship is recursive. We can express this relationship using the following syntax –

- `predecessor(X, Z) :- parent(X, Z).`
- `predecessor(X, Z) :- parent(X, Y),predecessor(Y, Z).`

Breadth First Search:
Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'), and explores all of the neighbour nodes at the present depth prior to moving on to the nodes at the next depth level.

Example:

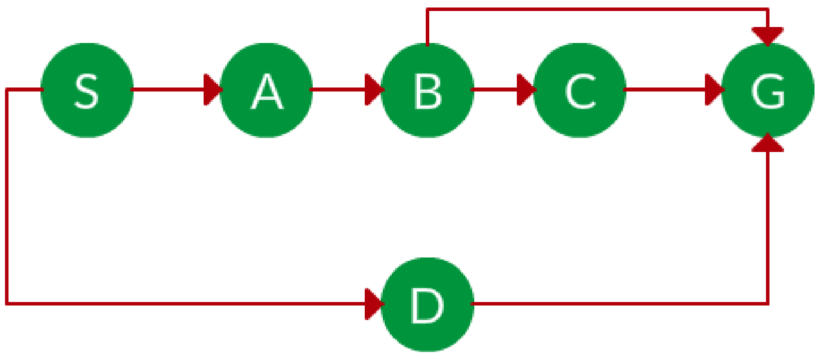


Fig 1.60 Graph

The equivalent search tree for the above graph is as follows:

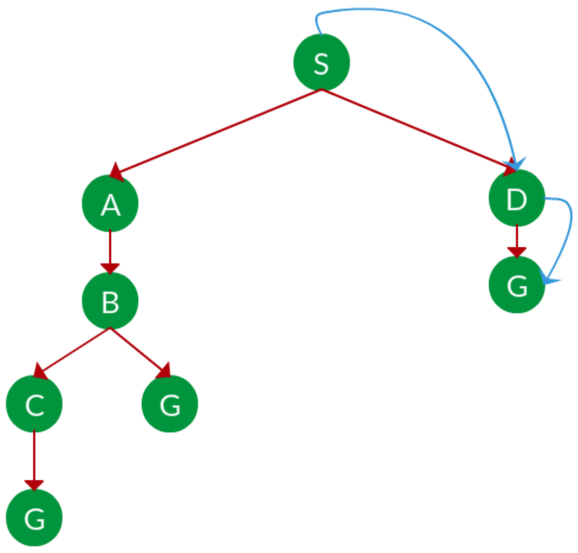


Fig 1.61 Tree

Path: S -> D -> G

Depth First Search:
Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at

the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

Example:

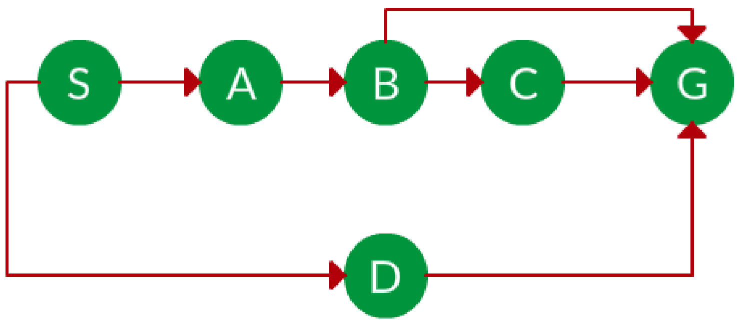


Fig 1.62 Graph

The equivalent search tree for the above graph is as follows:

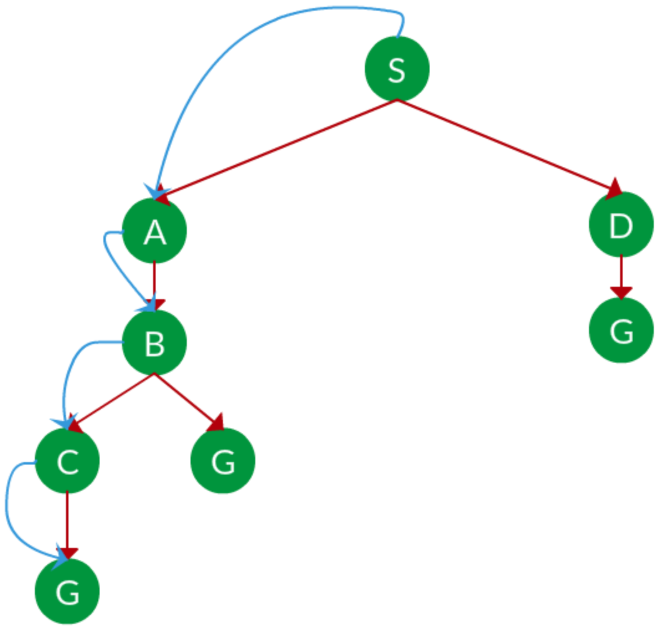


Fig 1.63 Tree

Path: S -> A -> B -> C -> G