Universidad Autónoma de Baja California

Facultad de Ingeniería, Arquitectura y Diseño

Paradigmas de la Programación

Practica 4
Paradigma lógico

DIEGO OSUNA ACEVEDO

372273

17/mayo/2024.

Ir a la página del tutorial de Prolog en tutorialspoint (https://www.tutorialspoint.com/prolog/index.htm), y seguir el tutorial desde la sección Home hasta la sección Tree Data Structure (Case Study); son 17 secciones en total.

Agregar el reporte en PDF de la práctica sobre el paradigma lógico.

1. Home

Prolog (PROgramming in LOGics) es un lenguaje de programación lógico y declarativo. Es adecuado para programas que implican cálculo simbólico o no numérico.

2. Introduction

La programación lógica es uno de los paradigmas de programación informática, en el que las declaraciones del programa expresan los hechos y reglas sobre diferentes problemas dentro de un sistema de lógica formal.

Ejemplos de lenguajes de programación lógica

- ALF (algebraic logic functional programming language).
- ASP (Answer Set Programming)
- CycL
- Datalog
- FuzzyCLIPS
- Janus
- Parlog
- Prolog
- Prolog++
- ROOP

Prolog (PROgramming in LOGics).

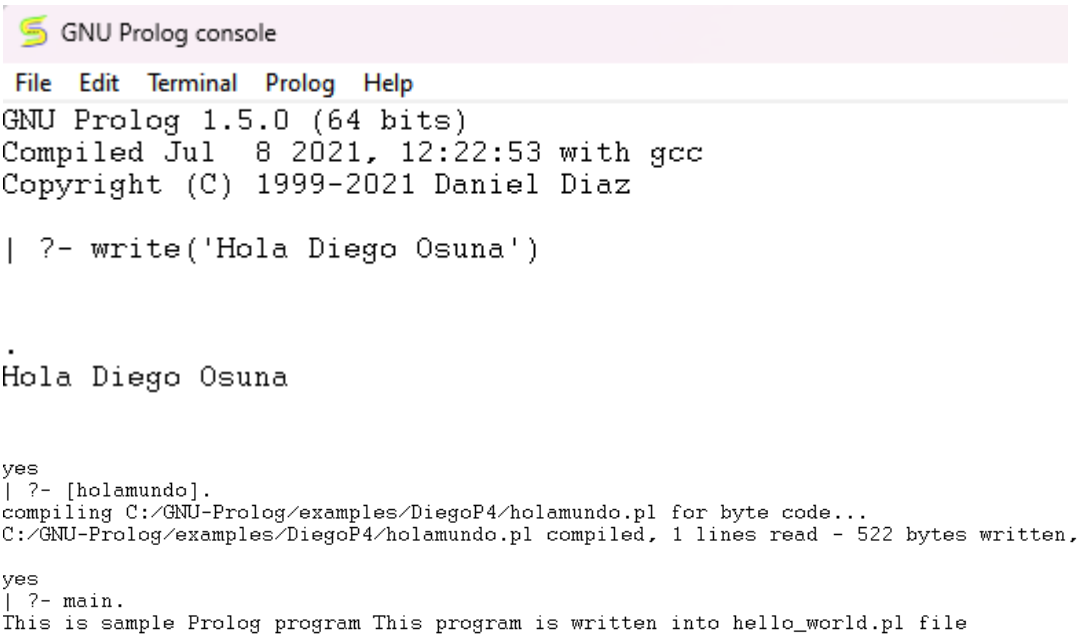
El lenguaje Prolog tiene tres principales elementos diferentes:
Hechos : Los hechos son un predicado que es verdadero, por ejemplo, si decimos "Tom es el hijo de Jack", entonces es un hecho.
Reglas : Las reglas son extinciones de hechos que contienen cláusulas condicionales. Para satisfacer una regla se deben cumplir estas condiciones.
Preguntas : Para ejecutar un programa de prólogo, necesitamos algunas preguntas, y esas preguntas pueden responderse mediante los hechos y reglas dados.

Aplicaciones de prolog:

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

3. Prolog - Environment Setup

4. Prolog - Hello World



5. Prolog - Basics

Hechos

Podemos definir un hecho como una relación explícita entre objetos y las propiedades que estos objetos puedan tener. Por lo tanto, los hechos son incondicionalmente verdaderos por naturaleza.

La sintaxis para los hechos es la siguiente:
relacion(objeto1, objeto2...).

Reglas

Podemos definir una regla como una relación implícita entre objetos. Por lo tanto, los hechos son condicionalmente verdaderos. Entonces, cuando una condición asociada es verdadera, el predicado también es verdadero.

Sintaxis:
nombre_regla(objeto1, objeto2, ...) :- hecho/regla(objeto1, objeto2, ...)
Supongamos que una cláusula es como:
P :- Q;R.
Esto también se puede escribir como:

P :- Q.
P :- R.

Consultas
Las consultas son preguntas sobre las relaciones entre objetos y las propiedades de los objetos.

```
yes
| ?- consult('5.pl').
compiling C:/GNU-Prolog/examples/DiegoP4/5.pl for byte code...
C:/GNU-Prolog/examples/DiegoP4/5.pl compiled, 5 lines read - 495 bytes wri

yes
| ?- girl(priya)
.

yes
| ?- girl(jamini)
.

no
| ?- can_cook(priya).

yes
| ?- can_cook(jaya).

no
```

6. Prolog - Relations

Relaciones en Prolog

En los programas Prolog, se especifica la relación entre objetos y las propiedades de los objetos.

Family.pl

```
| ?-  consult('6family.pl').
compiling C:/GNU-Prolog/exampl
C:/GNU-Prolog/examples/DiegoP4.

yes
| ?- parent(X,jim).

X = pat ? ;

X = peter

yes
| ?- pam

.
uncaught exception: error(exis
| ?- mother(X,Y).

X = pam
Y = bob ? ;

X = pat
Y = jim ? ;

(16 ms) no
```

```
| ?- haschild(X).

X = pam ? ;

X = tom ? ;

X = tom ? ;

X = bob ? ;

X = bob ? ;

X = pat ? ;

X = bob ? ;

X = peter

(31 ms) yes
| ?- sister(X,Y).

X = liz
Y = bob ? ;

X = ann
Y = pat ? ;

X = ann
Y = peter ? ;

X = pat
Y = ann ? ;

X = pat
Y = peter ? ;

no
| ?-
```

Family_Ext.pl

```
| ?- consult('6family_ext
compiling C:/GNU-Prolog/e
C:/GNU-Prolog/examples/Di

yes
| ?- uncle(X,Y).

X = peter
Y = jim ? ;

no
| ?- grandparent(X,Y).

X = pam
Y = ann ? .
Action (; for next soluti

X = pam
Y = pat ? ;

X = pam
Y = peter ? ;

X = tom
Y = ann ? ;

X = tom
Y = pat ? ;

X = tom
Y = peter ? ;

X = bob
Y = jim ? ;

X = bob
Y = jim ? ;

(16 ms) no
```

```
(16 ms) no
| ?- wife(X,Y).

X = pam
Y = tom ? ;

X = pat
Y = peter ? ;

no
| ?-
```

Trazando la salida

En Prolog podemos rastrear la ejecución. Para rastrear la salida, debes entrar en el modo de rastreo escribiendo "trace.". Luego, en la salida podemos ver que solo estamos rastreando "¿pam es madre de quién?".

```
| ?- mother(X,Y).

X = pam
Y = bob ? ;

X = pat
Y = jim ? ;

no
| ?- trace.
The debugger will first creep -- showing everything (trace)

yes
{trace}
| ?- mother(pam,Y).
      1    1  Call: mother(pam,_23) ?
      2    2  Call: parent(pam,_23) ?
      2    2  Exit: parent(pam,bob) ?
      3    2  Call: female(pam) ?
      3    2  Exit: female(pam) ?
      1    1  Exit: mother(pam,bob) ?

Y = bob

yes
{trace}
| ?- notrace.
The debugger is switched off

yes
```

7. Prolog - Data Objects

Átomos y Variables

Átomos
Los átomos son una variación de las constantes. Pueden ser cualquier nombre u objeto.

Cadenas de caracteres especiales

Debemos tener en cuenta que al usar átomos de esta forma, se requiere cierta precaución ya que algunas cadenas de caracteres especiales ya tienen un significado predefinido; por ejemplo, ':-'.

Números

Otra variación de las constantes son los números. Así, los números enteros pueden ser representados como 100, 4, -81, 1202. En Prolog, el rango normal de los enteros es de -16383 a 16383.

Variables Anónimas en Prolog

Las variables anónimas no tienen nombres. Las variables anónimas en Prolog se escriben con un solo carácter de subrayado '_'. Y una cosa importante es que cada variable anónima individual se trata como diferente.

```
compiling C:/GNU-Prolog/examples/DiegoP4/7var_anonymous.pl for byte code.
C:/GNU-Prolog/examples/DiegoP4/7var_anonymous.pl compiled, 3 lines read -

yes
| ?- hates(X,tom).

X = jim ? ;

X = peter

yes
| ?- hates(_,tom).

true ? ;

yes
| ?- hates(_,pat).

no
| ?- hates(_,fox).

true ? ;

no
| ?-
```

8. Prolog - Operators

Operadores de Comparación
Los operadores de comparación se utilizan para comparar dos ecuaciones o estados.

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 X and Y values are equal
X =\= Y	the X and Y values are not equal

```
| ?- 1+2=:=2+1.

yes
| ?- 1+A=B+2.

A = 2
B = 1

yes
| ?- 5>10.

no
| ?- 10=\=100.

yes
```

Operadores Aritméticos en Prolog

Los operadores aritméticos se utilizan para realizar operaciones aritméticas. Hay varios tipos de operadores aritméticos, que son los siguientes:

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

```
Copyright (C) 1999-2021 Daniel Diaz

| ?- X is 100 + 200,write('100 + 200 is '),write(X),nl,
        Y is 400 - 150,write('400 - 150 is '),write(Y),nl,
        Z is 10 * 300,write('10 * 300 is '),write(Z),nl,
        A is 100 / 30,write('100 / 30 is '),write(A),nl,
        B is 100 // 30,write('100 // 30 is '),write(B),nl,
        C is 100 ** 2,write('100 ** 2 is '),write(C),nl,
        D is 100 mod 30,write('100 mod 30 is '),write(D),nl.
100 + 200 is 300
400 - 150 is 250
10 * 300 is 3000
100 / 30 is 3.3333333333333335
100 // 30 is 3
100 ** 2 is 10000.0
100 mod 30 is 10

A = 3.3333333333333335
B = 3
C = 10000.0
D = 10
X = 300
Y = 250
Z = 3000

(16 ms) yes
| ?-
```

9. Prolog - Loop & Decision Making

Bucles

Las declaraciones de bucle se utilizan para ejecutar un bloque de código varias veces. En general, for, while, do-while son estructuras de bucle en los lenguajes de programación (como Java, C, C++).

```
yes
| ?- consult('9loop.pl').
compiling C:/GNU-Prolog/examples/DiegoP4/9loop.pl for byte code...
C:/GNU-Prolog/examples/DiegoP4/9loop.pl compiled, 4 lines read - 763

yes
| ?- count_to_10(3).
3
4
5
6
7
8
9
10

true ?

(31 ms) yes
| ?-
```

Toma de Decisiones

Las declaraciones de decisión son declaraciones If-Then-Else. Entonces, cuando intentamos coincidir con alguna condición y realizar alguna tarea, utilizamos las declaraciones de toma de decisiones.

```
| ?- consult('9decision.pl').
compiling C:/GNU-Prolog/examples/DiegoP4/9decision.pl for byte code...
C:/GNU-Prolog/examples/DiegoP4/9decision.pl compiled, 9 lines read - 111

yes
| ?- gt(10,100).
X is smaller

yes
| ?- gt(150,100).
X is greater or equal

true ?

(32 ms) yes
| ?- gte(10,20).
X is smaller

yes
| ?- gte(100,20).
X is greater

true ?

yes
| ?- gte(100,100).
X and Y are same

true ?
Action (; for next solution, a for all solutions, RET to stop) ?

yes
| ?-
```

10. Prolog - Conjunctions & Disjunctions

La conjunción (lógica AND) se puede implementar usando el operador coma (,). Así, dos predicados separados por una coma están unidos por una declaración AND.

La disyunción (lógica OR) se puede implementar usando el operador punto y coma (;). Así, dos predicados separados por un punto y coma están unidos por una declaración OR.

```
| ?- consult('10discon.pl').
compiling C:/GNU-Prolog/examples/DiegoP4/10discon.pl for byte cod
C:/GNU-Prolog/examples/DiegoP4/10discon.pl compiled, 11 lines rea

yes
| ?- father(jhon,bob).

yes
| ?- child_of(jhon,bob).

true ?

yes
| ?- child_of(lili,bob).

yes
| ?-
```

11. Prolog - Lists

Representación de Listas
La lista es una estructura de datos simple que se utiliza ampliamente en la programación no numérica. Una lista consiste en cualquier número de elementos, por ejemplo, rojo, verde, azul, blanco, oscuro. Se representará como [rojo, verde, azul, blanco, oscuro].

Operaciones Básicas en Listas

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.

```
| ?- consult('11list.pl').
compiling C:/GNU-Prolog/examples/DiegoP4/11list.pl for byte code...
C:/GNU-Prolog/examples/DiegoP4/11list.pl compiled, 1 lines read - 467 byte

yes
| ?- list_member(b,[a,b,c]).

true ?

yes
| ?- list_member(b,[a,[b,c]]).

no
| ?- list_member([b,c],[a,[b,c]]).

true ?

yes
| ?- list_member(d,[a,b,c]).

no
| ?- list_member(d,[a,b,c]).

no
| ?-
```

12. Prolog - Recursion and Structures

Recursión
La recursión es una técnica en la que un predicado se utiliza a sí mismo (puede ser con otros predicados) para encontrar el valor de verdad.

Estructuras
Las estructuras son objetos de datos que contienen múltiples componentes.

Coincidencia en Prolog
La coincidencia se utiliza para comprobar si dos términos dados son iguales (idénticos) o si las variables en ambos términos pueden tener los mismos objetos después de ser instanciadas.

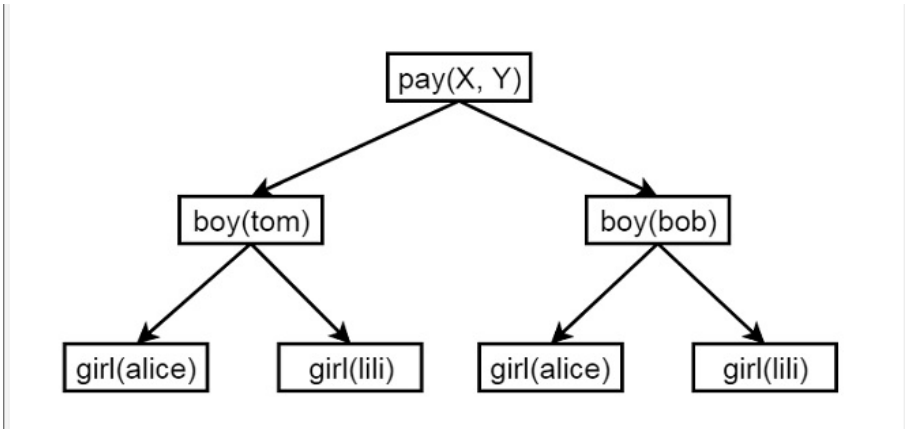
Árboles Binarios

A continuación se muestra la estructura de un árbol binario utilizando estructuras recursivas.

```
node(2, node(1,nil,nil), node(6, node(4,node(3,nil,nil), node(5,nil,nil)),
node(7,nil,nil))
```

13. Prolog - Backtracking

Retroceso es un procedimiento en el cual Prolog busca el valor de verdad de diferentes predicados comprobando si son correctos o no. El término "retroceso" es bastante común en el diseño de algoritmos y en diferentes entornos de programación. En Prolog, hasta que alcanza el destino adecuado, intenta retroceder. Cuando se encuentra el destino, se detiene.




```
yes
{trace}
| ?- pay(X,Y).
      1    1  Call: pay(_23,_24) ?
      2    2  Call: boy(_23) ?
      2    2  Exit: boy(tom) ?
      3    2  Call: girl(_24) ?
      3    2  Exit: girl(alice) ?
      1    1  Exit: pay(tom,alice) ?

X = tom
Y = alice ? ;
      1    1  Redo: pay(tom,alice) ?
      3    2  Redo: girl(alice) ?
      3    2  Exit: girl(lili) ?
      1    1  Exit: pay(tom,lili) ?

X = tom
Y = lili ? ;
      1    1  Redo: pay(tom,lili) ?
      2    2  Redo: boy(tom) ?
      2    2  Exit: boy(bob) ?
      3    2  Call: girl(_24) ?
      3    2  Exit: girl(alice) ?
      1    1  Exit: pay(bob,alice) ?

X = bob
Y = alice ? ;
      1    1  Redo: pay(bob,alice) ?
      3    2  Redo: girl(alice) ?
      3    2  Exit: girl(lili) ?
      1    1  Exit: pay(bob,lili) ?

X = bob
Y = lili

(15 ms) yes
{trace}
| ?-
```

```
{trace}
| ?- f(1,Y), 2<Y.
      1    1  Call: f(1,_23) ?
      2    2  Call: 1<3 ?
      2    2  Exit: 1<3 ?
      1    1  Exit: f(1,0) ?
      3    1  Call: 2<0 ?
      3    1  Fail: 2<0 ?
      1    1  Redo: f(1,0) ?
      2    2  Call: 3=<1 ?
      2    2  Fail: 3=<1 ?
      2    2  Call: 6=<1 ?
      2    2  Fail: 6=<1 ?
      1    1  Fail: f(1,_23) ?

(47 ms) no
{trace}
| ?-
```

Negación como Falla

Aquí realizaremos un fallo cuando la condición no se cumpla. Supongamos que tenemos una afirmación, "Mary le gusta todos los animales excepto las serpientes", lo expresaremos en Prolog.

```
{trace}
| ?- likes(mary,dog).
      1    1  Call: likes(mary,dog) ?
      2    2  Call: snake(dog) ?
      2    2  Fail: snake(dog) ?
      2    2  Call: animal(dog) ?
      2    2  Exit: animal(dog) ?
      1    1  Exit: likes(mary,dog) ?

(31 ms) yes
{trace}
| ?- likes(mary,python).
      1    1  Call: likes(mary,python) ?
      2    2  Call: snake(python) ?
      2    2  Exit: snake(python) ?
      3    2  Call: fail ?
      3    2  Fail: fail ?
      1    1  Fail: likes(mary,python) ?

no
{trace}
| ?-
```

14. Prolog - Different and Not

El predicado **diferenciador** verificará si dos argumentos dados son iguales o no. Si son iguales, devolverá falso; de lo contrario, devolverá verdadero.

```
| ?- consult('14not.pl').
compiling C:/GNU-Prolog/examples/DiegoP4/14not.pl for byte code...
C:/GNU-Prolog/examples/DiegoP4/14not.pl:2: warning: singleton variables [X
C:/GNU-Prolog/examples/DiegoP4/14not.pl compiled, 1 lines read - 331 bytes

yes
| ?- different(100,200).

yes
| ?- different(100,100).

no
| ?- different(abc,def).

yes
| ?- different(abc,abc).

no
| ?-
```

El predicado **"not"** se utiliza para negar alguna afirmación, lo que significa que cuando una afirmación es verdadera, entonces "not(afirmación)" será falsa; de lo contrario, si la afirmación es falsa, entonces "not(afirmación)" será verdadera.

```
| ?- consult('14not.pl').
compiling C:/GNU-Prolog/examples/DiegoP4/14not.pl for byte code...
C:/GNU-Prolog/examples/DiegoP4/14not.pl compiled, 0 lines read - 636 bytes

yes
| ?- not(true).

no
| ?- not(fail).

yes
| ?-
```

15. Prolog - Inputs and Outputs

El predicado write()

Para escribir la salida podemos usar el predicado write(). Este predicado toma el parámetro como entrada y escribe el contenido en la consola por defecto. write() también puede escribir en archivos.

```
.
| ?- write(44).
44

yes
| ?- write('hola').
hola

yes
| ?- write('hola'),nl,write('mundo').
hola
mundo

yes
| ?- write("ABCDE")
.
[65,66,67,68,69]

yes
```

El predicado read()

El predicado read() se utiliza para leer desde la consola. El usuario puede escribir algo en la consola, lo que puede ser tomado como entrada y procesarlo.

```
.
| ?- consult('15read.pl').
compiling C:/GNU-Prolog/examples/DiegoP4/15read.pl for byte code...
C:/GNU-Prolog/examples/DiegoP4/15read.pl compiled, 7 lines read - 1228 byt

yes
| ?- cube.
Write a number: 4
.
Cube of 4: 64
Write a number: 10.
Cube of 10: 1000
Write a number: 14.
Cube of 14: 2744
Write a number: 8.
Cube of 8: 512
Write a number: stop.

(15 ms) yes
```

El predicado tab()

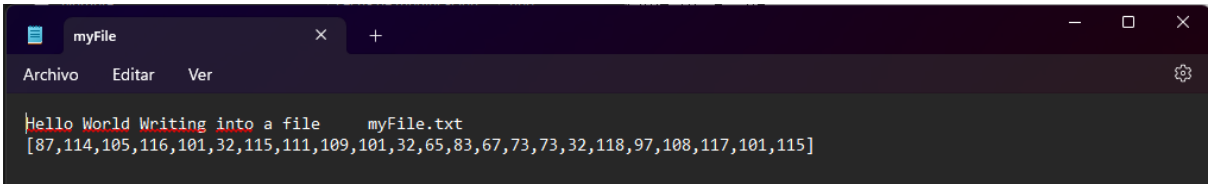
El predicado tab() es un predicado adicional que se puede utilizar para insertar algunos espacios en blanco mientras escribimos algo. Toma un número como argumento e imprime esa cantidad de espacios en blanco.

```
| ?- write('hola'),tab(24),write('mundo').
hola                        mundo

yes
| ?- write('We'),tab(5),write('will'),tab(5),write('use'),tab(5),write('ta
We     will     use     tabs

yes
| ?-
```

Los predicados tell y told

Si queremos escribir en un archivo, en lugar de en la consola, podemos usar el predicado tell(). Este predicado tell() toma el nombre del archivo como argumento. Si ese archivo no está presente, entonces crea un archivo nuevo y escribe en él. Ese archivo se mantendrá abierto hasta que escribamos el comando told.

```
(15 ms) yes
| ?- write('hola'),tab(24),write('mundo').
hola                        mundo

yes
| ?- write('We'),tab(5),write('will'),tab(5),write('use'),tab(5),write('ta
We     will     use     tabs

yes
| ?- tell('myFile.txt').

yes
| ?- write('Hello World').

yes
| ?- write(' Writing into a file'),tab(5),write('myFile.txt'),nl.

yes
| ?- write("Write some ASCII values").

yes
| ?- told.
```



16. Prolog - Built-In Predicates

En Prolog, hemos visto que en la mayoría de los casos se utilizan predicados definidos por el usuario, pero también existen algunos predicados integrados. Hay tres tipos de predicados integrados:

- Identifying terms
- Decomposing structures
- Collecting all solutions

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.

Mathematical Predicates

Predicates	Description
random(L,H,X).	Get random value between L and H
between(L,H,X).	Get all values between L and H
succ(X,Y).	Add 1 and assign it to X
abs(X).	Get absolute value of X
max(X,Y).	Get largest value between X and Y
min(X,Y).	Get smallest value between X and Y
round(X).	Round a value near to X
truncate(X).	Convert float to integer, delete the fractional part
loor(X).	Round down
ceiling(X).	Round up
sqrt(X).	Square root

17. Tree Data Structure (Case Study)

```
| ?- consult('17tree.pl').
compiling C:/GNU-Prolog/examples/DiegoP4/17tree.pl for byte code...
C:/GNU-Prolog/examples/DiegoP4/17tree.pl:19: warning: singleton variables
C:/GNU-Prolog/examples/DiegoP4/17tree.pl compiled, 28 lines read - 3248 by

yes
| ?- i is_parent p.

yes
| ?- i is_parent s.

no
| ?- is_parent(i,p).

yes
| ?- e is_sibling_of f.

true ?

yes
| ?- is_sibling_of(e,g).

no
| ?- leaf_node(v).

yes
| ?- leaf_node(a).

no
| ?- is_at_same_level(l,s).

true ? l
Action (; for next solution, a for all solutions, RET to stop) ? ;

(16 ms) no
```