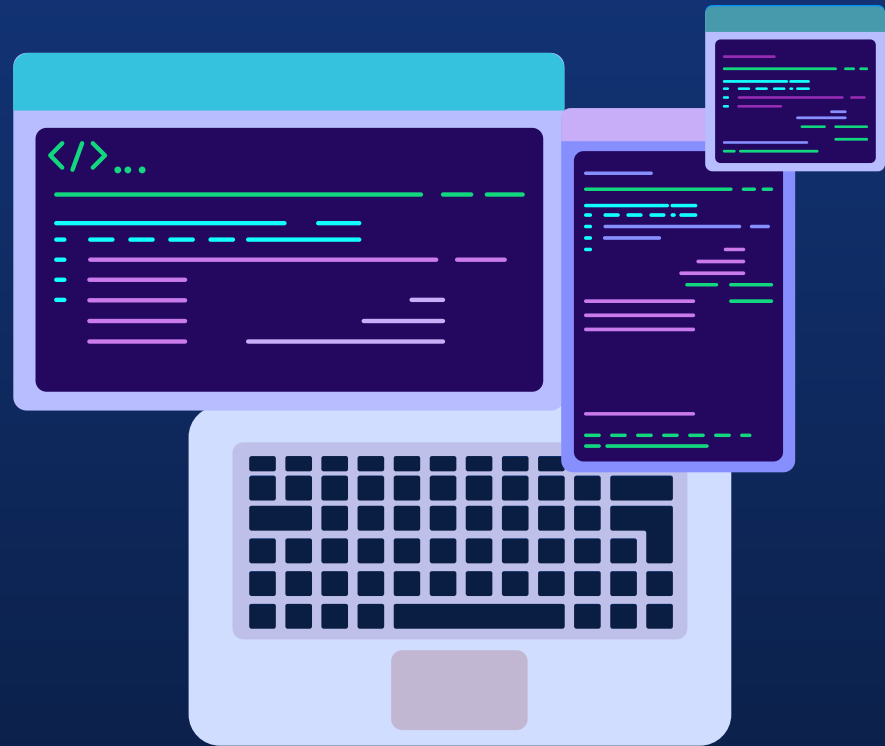


TUTORIAL PROLOG

María Fernanda Carbonell Santos
Juan David González Muñoz
Ivan Alejandro Cruz Tole
Yuli Beltran Rozo



CONTENIDO

01

Introducción

- *¿Qué es PROLOG?
- *Propósito
- *Aplicaciones
- *Ventajas y Desventajas

04

Conceptos Básicos

- *Términos
- *Hechos
- *Consultas
- *Reglas
- *Backtracking
- *Expresiones
- *Predicados Predefinidos
- *Listas
- *Arboles

02

Historia

- *Orígenes y Creadores de PROLOG

05

Ejemplo

- *Explicación de un ejemplo

03

Instalación

- *Instalación escritorio
- *PROLOG virtual

06

Referencias

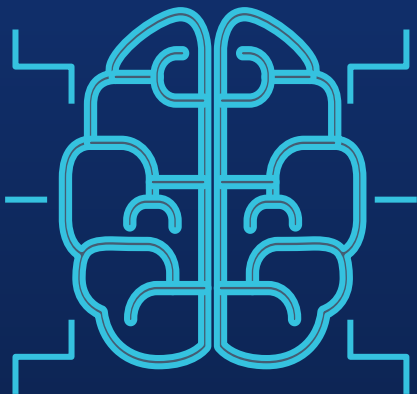
- *Bibliografía
- *Imágenes



01

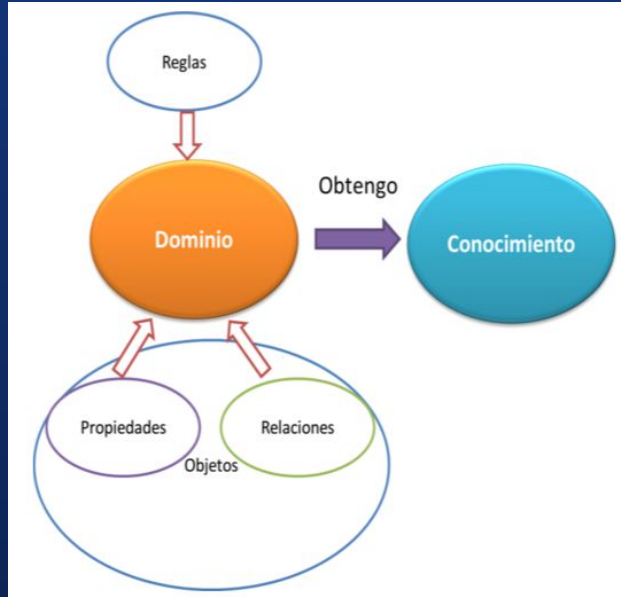
INTRODUCCIÓN





¿Qué es PROLOG?

PROLOG proveniente del francés **PRO**grammation en **LOG**ique, es un lenguaje de programación **declarativo** e **interpretado**, usado para representar conocimiento sobre un determinado dominio y las relaciones entre objetos en ese dominio.



[1]

Busca obtener conocimiento declarando **hechos reales** sobre los objetos y sus relaciones, creando **reglas** sobre dichos objetos y relaciones. Está basado en los siguientes mecanismos básicos: **unificación, estructuras de datos basadas en árboles y backtracking.**



Propósito

Es un lenguaje de programación muy **potente** y **flexible**; este es ampliamente utilizado en los campos de finanzas, defensa, telecomunicaciones, medicina, agricultura, ingeniería, manufactura y educación.



Aplicaciones

Sus aplicaciones van desde **la gestión de juegos**, a la **Inteligencia Artificial**, hasta los **sistemas expertos**; como lenguaje especialmente pensado para construir bases de conocimientos basados en la lógica.



Inteligencia Artificial

Se usa como interfaz de idiomas, busca en la base de datos para responder una pregunta



Sistemas Ambientales

MM4 Weather Modeling System, desarrollado en la Universidad Penn State y el Centro Nacional de Investigación Atmosférica



Automatas

Al fijar los hechos se pueden hacer consultas para verificar si una expresión es o no aceptada por el autómata

VENTAJAS VS DESVENTAJAS



VENTAJAS

- Fácil para programar.
- No hay que pensar demasiado en la solución del problema.
- El código tiende a ser mucho más corto.



DESVENTAJAS

- La resolución automática no siempre es eficiente.
- La representación del conocimiento.
- Incapaz de reconocer que un problema es inaplicable o insuficiente.
- Los motores de inferencia poseen algunos límites.



02

HISTORIA





Origen

Lenguaje de programación ideado a principios de los **años 70** en la Universidad de Aix-Marseille I (Marsella, Francia) por **Alain Colmerauer** y **Philippe Roussel**. Nació de un proyecto que no tenía como objetivo la traducción de un lenguaje de programación, sino el tratamiento algorítmico de lenguajes naturales.



[2]

Alain Colmerauer



[3]

Philippe Roussel

Alain Colmerauer y **Robert Pasero** trabajaban en la parte del procesamiento del lenguaje natural, y **Jean Trudel** y **Philippe Roussel** en la parte de deducción e inferencia del sistema. Philippe Roussel persuadió a **Robert Kowalski** para que colaborará con el proyecto, dando lugar a una versión preliminar del lenguaje Prolog a finales de **1971** y apareciendo la versión definitiva en **1972**.



03

INSTALACIÓN





[4]

SWI-Prolog ofrece un completo entorno Prolog gratuito. Desde su inicio en **1987**, el desarrollo de SWI-Prolog ha sido impulsado por las necesidades de las aplicaciones del mundo real. SWI-Prolog es ampliamente utilizado en la **investigación** y la **educación**, así como en **aplicaciones comerciales**.

Instalación Versión Escritorio






<https://www.swi-prolog.org/download/stable>

Linux versions are often available as a package for your distribution. We collect information about available packages and issues for building on specific distros [here](#). We provide a [PPA for Ubuntu](#) and [snap images](#).

Android binaries are available for [Termux](#) as the package `swi-prolog`. See also [Building SWI-Prolog on Android using LinuxOnAndroid](#).

Please check the [windows release notes](#) (also in the SWI-Prolog startup menu of your installed version) for details.

Examine the [Changelog](#).

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: 212271267caf9b914534519d4a553fab8c4608979d635cfaba63c125461084e5
	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 <code>swipl-win.exe</code> and <code>swipl.exe</code> . 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 <code>/opt/local/bin/swipl</code> . Needs xquartz (X11) and the Developer Tools (Xcode) installed for running the development tools . SHA256: 1b9c62cae781818a0dafd1d822ab563b8c10c7cd018ce10a3b71f900eb3a434f
Sources		
	11,386,908 bytes	SWI-Prolog source for 8.4.1 Sources in <code>.tar.gz</code> format, including packages and generated documentation files. See build instructions . SHA256: 30bb6542b7767e47b94bd92e8e8a7d7a8a000861044046edf45fc864841b69c4
Documentation		
	2,906,009 bytes	SWI-Prolog 8.4.1 reference manual in PDF SWI-Prolog reference manual as PDF file. This does <i>not</i> include the package documentation.

[Show all files](#)

Install scripts may download the SHA256 checksum by appending `.sha256` to the file name. Scripts can download the latest version by replacing the version of the file with `latest`. This causes the server to reply with the location of the latest version using an HTTP 303 See other message.

⚠ Windows antivirus software works using *signatures* and *heuristics*. Using the huge amount of viruses and malware known today, arbitrary executables are often falsily classified as malicious. [Google Safe Browsing](#), used by most modern browsers, therefore often classifies our Windows binaries as malware. You can use e.g., [virustotal](#) to verify files with a large number of antivirus programs.

Our Windows binaries are cross-compiled on an isolated Linux container. The integrity of the binaries on the server is regularly verified by validating its SHA256 fingerprint.

Please select the checkbox below to enable the actual download link.

☒ I understand

[Download swipl-8.4.1-1.x64.exe](#) (SHA256: 212271267caf0b914534519d4a553feb8e4608979d635cfaba63c125461084e5)

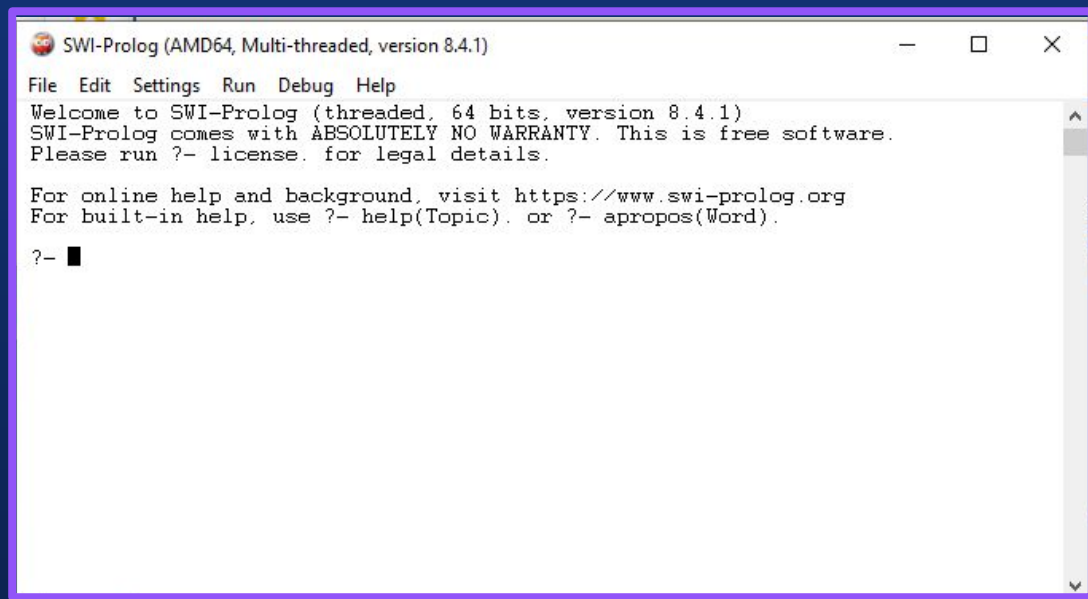
[VIRUSTOTAL Scan Result](#)

Powered by SWI-Prolog 8.5.5-1-g9ed721542

[login](#)

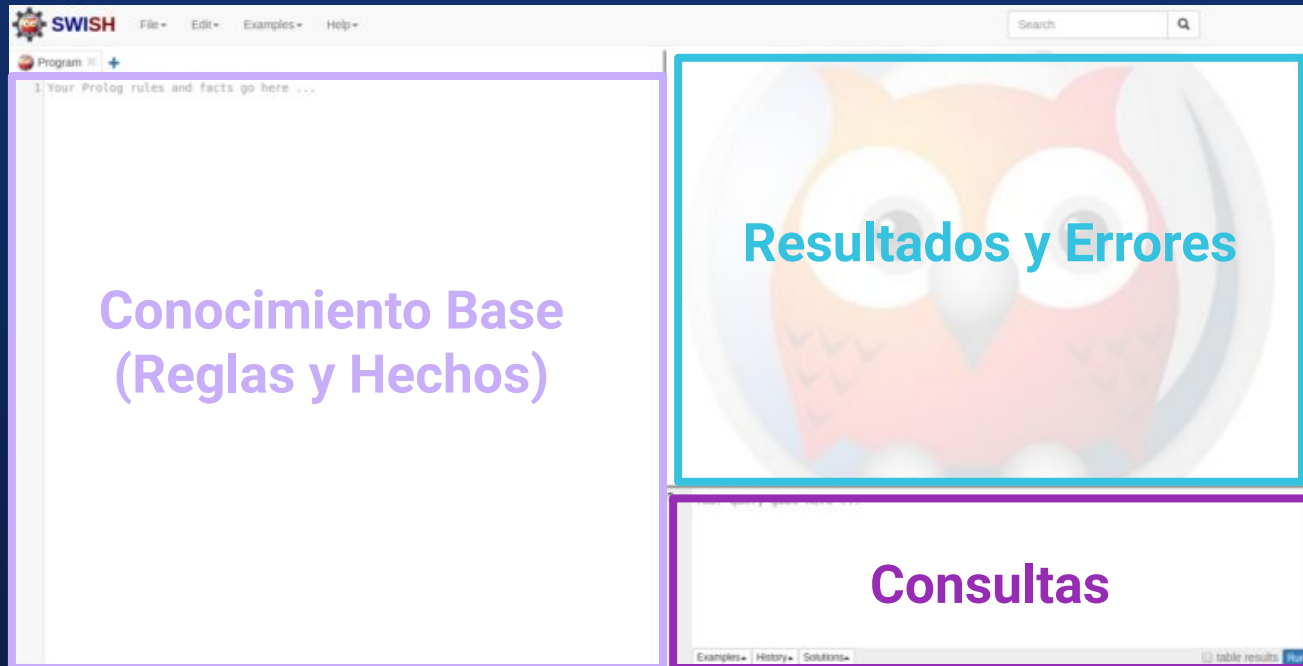
swipl-8.4.1-1.x64.exe

Mostrar todo



Versión en Línea

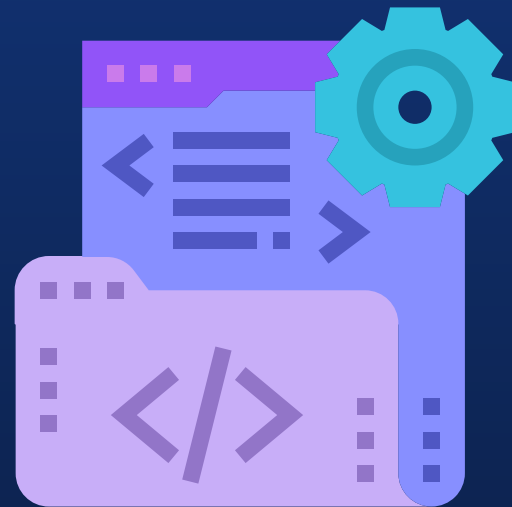
<http://swish.swi-prolog.org/>

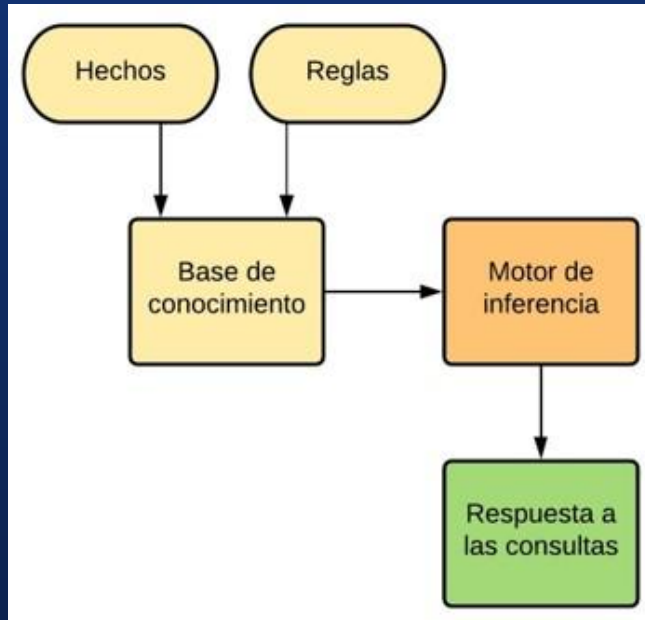




04

CONCEPTOS BÁSICOS

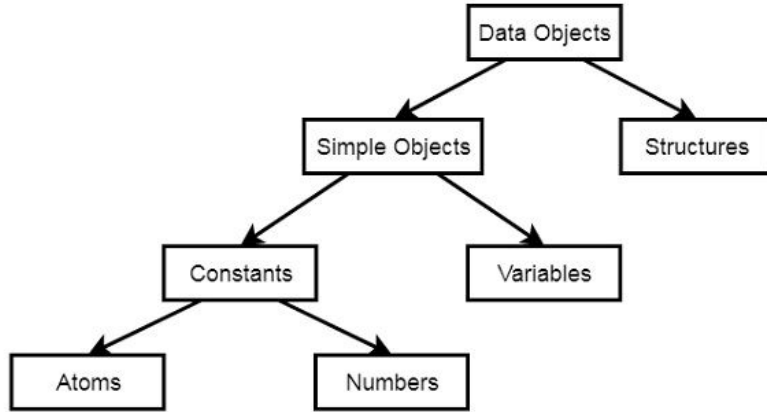




[1]

PROLOG

Consiste de una **base de datos** de **relaciones lógicas** y detalles que se cumplen para la aplicación, dicha base de datos no tiene una estructura impuesta, ni un procedimiento o clase principal. Escribir un programa en Prolog consiste en declarar el conocimiento disponible acerca de los **objetos**, además de sus **relaciones** y sus **reglas**.



Data objects in Prolog

[5]

Términos

Los **términos** son el único elemento del lenguaje, todos los datos están representados por términos; un término se compone de un **functor** seguido de **cero a N argumentos** entre paréntesis y separados por comas. Corresponden a un valor en PROLOG.

En lógica de primer orden, los términos se clasifican en tres categorías: **constantes**, **variables** y **términos compuestos**.

Constantes: En Prolog se distinguen dos tipos de constantes.

- **Números:** Este tipo de constantes se utiliza para representar **números enteros** y **números reales**, para poder realizar operaciones aritméticas entre ellos.

```
1 isNumber(2).  
2 isNumber(232423).  
3 isNumber(-3454).  
4 isNumber(-8.87).  
5 isNumber(2.1416).  
6 isNumber(0.234).  
7 isNumber(2e10).  
8 isNumber(3e-24).  
9 isNumber(2e2).  
10 isNumber(.87767).  
11  
Syntax error: Operator expected
```



Constantes: En Prolog se distinguen dos tipos de constantes.


- **Átomos o funtores:** Un átomo es **un nombre de propósito general**, se compone de una **secuencia de caracteres** que son analizados como una sola unidad. Se utilizan para nombrar **objetos, propiedades o relaciones**; deben empezar en minúscula.

```
1 isAtom(pedro).  
2 isAtom('Camilo Perez').  
3 isAtom('$&&/*').  
4 isAtom(atOmMm12345).  
5 isAtom(i_am_AToMm_23).  
6 |
```


Variables: Las variables se indican mediante una **cadena de caracteres** que puede constar de letras, números y guión bajo, adicionalmente debe comenzar con una **letra mayúscula** o con **guión bajo**, esto las diferencia de los átomos. Existe una variable especial que solo consta de un guión bajo (**_**), a esta variable se le llama variable **anónima**.

```
1 isVariable(Var) .  
2 isVariable(_var) .  
3 isVariable(Va2323sdf) .  
4 isVariable(V_a_w_3_) .  
5 isVariable(_) .  
6
```

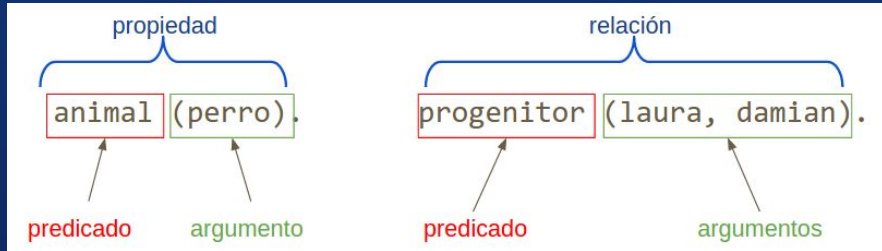
[1]



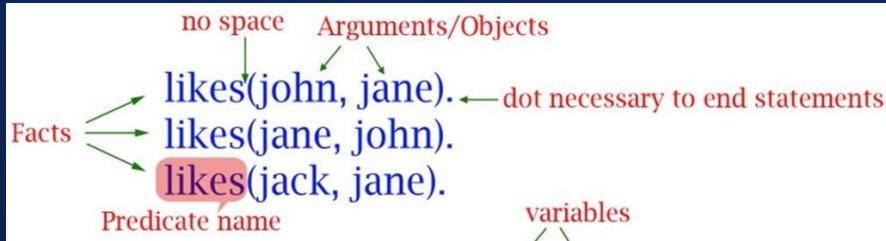
Estructuras: **Término compuesto** por otros términos. Una estructura se construye mediante un **átomo de función** llamado "**functor**", seguido entre paréntesis, por una conjunto de términos separados por comas, denominados **argumentos**.



```
1 name(arg1).  
  Functor Argumentos  
3 name(arg1, arg2, arg3).  
  Functor Argumentos  
5 name(Var).  
  Functor Argumentos  
7 name(2, 3, asd, _var).  
  Functor Argumentos
```



[1]



[6]

Hechos

Cláusula sin cuerpo que muestra una **relación** explícita entre **objetos** y **propiedades** que estos objetos puedan tener. No tienen que reflejar el mundo real necesariamente, pero será única y exclusivamente lo que Prolog tomará como **verdadero**. Están conformados por un **predicado** y un **argumento** u **objeto**.

Consultas

Sobre el archivo de **base de datos** se pueden realizar una serie de **preguntas** para extraer **conocimiento** generado por los hechos. Comienzan con un signo de interrogación seguido de un guión **?-** y terminan en punto. Ante una consulta, Prolog intenta hacer un **matching** sobre la base de conocimiento.

```
?- progenitor(patricia,jaime).
```

Singleton variables: [X]

true

```
?- progenitor(X,jaime).
```

Singleton variables: [X]

X = patricia

```
?- progenitor(tomas,X), progenitor(X,Y), progenitor(Y,Z).
```

Singleton variables: [X]

X = jose,
Y = patricia,
Z = jaime

[1]

cabeza :- objetivo1, objetivo2, ..., objetivon.

Rule → friends(X, Y) :- likes(X, Y), likes(Y, X).

head body

[6]

```
1 %Primer ejemplo:
2 %hechos
3 mamifero(perro).
4 mamifero(gato).
5 %reglas
6 animal(X) :- mamifero(X).
7
8
```

animal(gato)

true

?- animal(gato)

[1]

Reglas

Cuando la **verdad** de un hecho **depende** de la verdad de **otro hecho** o de un grupo de hechos se usa una regla. Declaran las **condiciones** para que un predicado sea cierto. Funcionan como las fórmulas condicionales habituales en lógica. Una regla está compuesta por una **cabeza** y un **cuerpo**.

Tipos de Reglas

- Emplea el operador lógico AND
- Se utiliza la coma (,)

`tia(X,Y):-hermana(X,Z),padre(Z,Y).`

Regla

AND

Conjunciones

- Emplea el operador lógico OR
- Se utiliza el punto y coma (;)

`hijo(X,Y):-padre(Y,X);madre(Y,X).`

Regla

OR



Disyunciones

Reglas Recursivas

La gran potencia de Prolog está en la definición de **reglas recursivas**. Podría ser interesante definir la relación `predecesor(X,Y)`. Un predecesor de X podrá ser el progenitor de X. También será predecesor si es abuelo/a, si es tatarabuelo/a, etc., es decir, podríamos definir un conjunto de reglas como:

<code>predecesor(X,Y):-progenitor(X,Y).</code>	<code>%padre</code>
<code>predecesor(X,Y):-progenitor(X,Z), progenitor(Z,Y).</code>	<code>%abuelo</code>
<code>predecesor(X,Y):-progenitor(X,Z), progenitor(Z,V), progenitor(V,Y).</code>	<code>%bisabuelo</code>

Reglas Recursivas

Para hacer uso de reglas recursivas se debe considerar 2 casos:

- **Caso básico:** Define cuándo se detiene el cálculo.
- **Caso recursivo:** Suponiendo que ya se ha solucionado un caso más simple, define cómo descomponer el caso actual hasta llegar al caso básico

```
predecesor(X,Y):-progenitor(X,Y).           %caso base  
predecesor(X,Y):-progenitor(X,Z), predecesor(Z,Y).  %caso recursivo
```

Reglas Recursivas

```
1 /* Equivale a:  
2 Para todo X e Y, si X es mujer y X es el progenitor de Y,  
3 entonces X es la madre de Y */  
4 madre(X,Y):-  
5     mujer(X),  
6     progenitor(X,Y).  
7  
8 /* Equivale a:  
9 Para todo X e Y, si X es hombre y X es el progenitor de Y,  
10 entonces X es el padre de Y */  
11 padre(X,Y):-  
12     hombre(X),  
13     progenitor(X,Y).  
14  
15 %Hechos.  
16 mujer(clara).  
17 mujer(isabel).  
18 mujer(ana).  
19 mujer(patricia).  
20 hombre(tomas).  
21 hombre(jose).  
22 hombre(jaime).
```

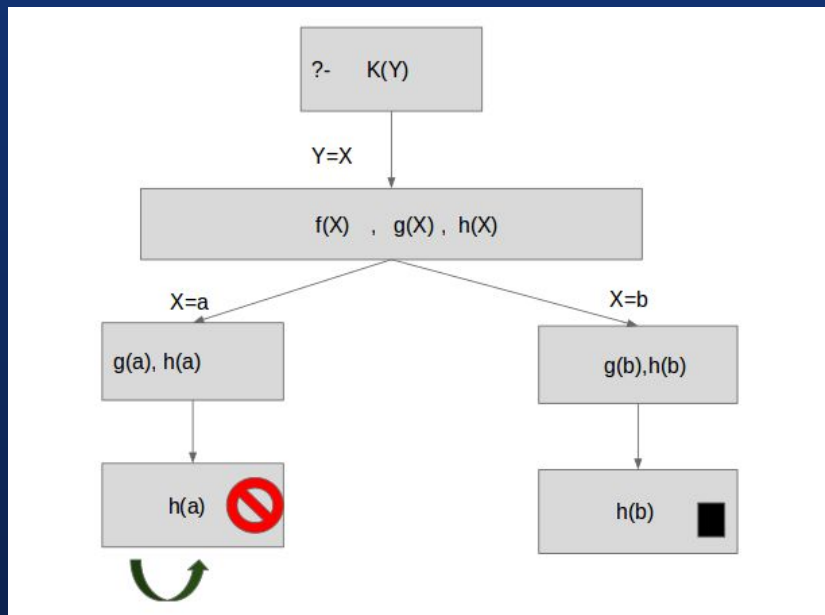
[1]

?- madre(X,Y).

Singleton variables: [X]

X		Y	
clara		jose	1
patricia		jaime	2

[1]



[1]

Backtracking


En **prolog** se devuelve hasta que encuentra que un hecho **base verdadero** y de ahí construye la respuesta. Si un predicado ofrece múltiples reglas **se intentan una por una hasta que una tiene éxito**. Si una alternativa no satisface la consulta, se rechaza la regla elegida previamente y se prueba la siguiente.



EXPRESIONES aritméticas

Expresión	Operación
$X+Y$	suma de X e Y
$X-Y$	X menos Y
$X*Y$	producto de X por Y
X/Y	cociente real de la división de X por Y
$X//Y$	cociente entero de la división de X por Y
X^Y	potencia entera de X a la Y
$X^{**}Y$	potencia real de X a la Y
$X \bmod Y$	resto de la división entera de X por Y
$abs(X)$	valor absoluto de X

$acos(X)$	arco coseno de X
$asen(X)$	arco seno de X
$atan(X)$	arcotangente de X
$cos(X)$	coseno de X
$exp(X)$	exponencial de X; $[e^X]$
$ln(X)$	logaritmo neperiano de X
$log(X)$	logaritmo en base 2 de X
$sin(X)$	seno de X
$sqrt(X)$	raíz cuadrada de X
$tan(X)$	tangente de X
$round(X,N)$	redondeo del real X con N decimales



EXPRESIONES Comparación Términos

Expresión	Operación
$X < Y$	cierto si el valor numérico de X es menor que el de Y
$X > Y$	cierto si el valor numérico de X es mayor que el de Y
$X \leq Y$	cierto si el valor numérico de X es menor o igual que el de Y
$X \geq Y$	cierto si el valor numérico de X es mayor o igual que el de Y


EXPRESIONES Comparación Expresiones

Expresión	Operación
$X == Y$	la expresión X es igual que la expresión Y
$X \neq Y$	la expresión X es distinta que la expresión Y
$X < Y$	la expresión X es menor que la expresión Y
$X > Y$	la expresión X es mayor que la expresión Y
$X \leq Y$	la expresión X es menor o igual que la expresión Y
$X \geq Y$	la expresión X es mayor o igual que la expresión Y
$X \text{ is } Y$	Si Y es una expresión aritmética, ésta se evalúa y el resultado se intenta unificar con X.



Predicados Predefinidos

Los predicados predefinidos son aquellos que **ya están definidos en PROLOG**, es decir, no necesitamos especificarlos mediante cláusulas. Existen **dos tipos** de predicados predefinidos:

- Aquellos **predicados de uso frecuente** que ya los proporciona PROLOG, aunque podríamos definirlos nosotros.
 - Predicados con **un efecto colateral distinto** a la instanciación de variables a valores.
- 

Predicados que permiten determinar el tipo de términos que estamos usando

Predicado	Función
var	El objetivo <code>var(X)</code> se cumple si X es una variable no instanciada.
nonvar	El objetivo <code>nonvar(X)</code> se cumple si X es una variable instanciada
atom	El objetivo <code>atom(X)</code> se cumple si X representa un átomo.
integer	El objetivo <code>integer(X)</code> se cumple si X representa un número entero.
atomic	El objetivo <code>atomic(X)</code> se cumple si X representa un entero o un átomo.

predicados predefinidos que permiten controlar otros predicados

Predicado	Función
!(cut)	Fuerza al sistema a mantener ciertas elecciones que ha realizado.
true	Este objetivo siempre se cumple.
fail	Este objetivo siempre fracasa.
not	El objetivo not(X) se cumple si fracasa el intento de satisfacer X. El objetivo not(X) fracasa si el intento de satisfacer X tiene éxito. Es similar a la negación en la lógica de predicados.
repeat	forma auxiliar para generar soluciones múltiples mediante el mecanismo de reevaluación.
call	Se cumple si tiene éxito el intento de satisfacer X.
;	Especifica una disyunción de objetivos
,	Especifica una conjunción de objetivos




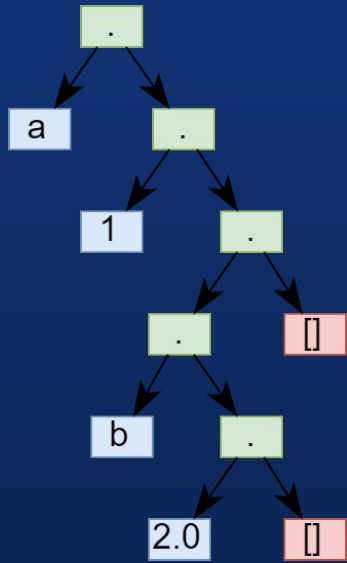
Predicados de lectura y escritura

Predicado	Función
write	escribe el término X en la consola de salida.
nl	genera una nueva línea en la consola de salida.
read	lee el siguiente término en la consola de entrada.
display	funciona exactamente igual que write, excepto que pasa por alto las declaraciones de operadores.

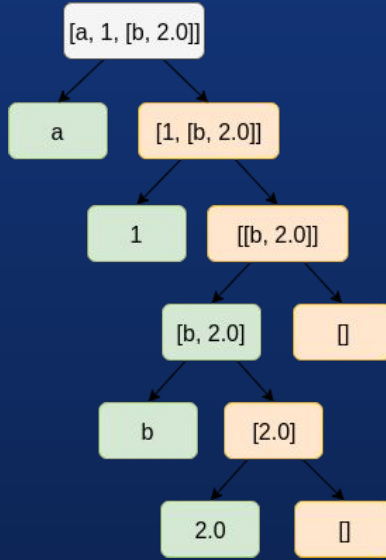
Predicados para modificación de la base de conocimiento.

Predicado	Función
assertz	Inserta un hecho al final de la lista de hechos de un predicado.
asserta	Inserta un hecho al inicio de la lista de hechos de un predicado.
retract	Remueve un hecho de la base de conocimiento.
retractall	Remueve todos los hechos que coincidan con el argumento recibido.





[a, b, c, d]



[1]

Listas

Es un tipo concreto de **estructura de datos** simple que almacena secuencialmente elementos de **cualquier tipo**. Es una secuencia ordenada de elementos que puede tener cualquier longitud. Un elemento puede ser cualquier tipo de dato e incluso otra lista. En Prolog las listas están formadas por **cabeza** y **cola**.

Unificación de Listas

En Prolog se puede unificar una lista con otra, **una variable** que no está instanciada se puede **unificar con cualquier objeto**, por tanto se puede unificar una lista con una variable. Para unificar una variable con una lista pero separando su cabeza y cola se debe hacer de la forma **[A | B]** donde el símbolo **(|)** separa la cabeza de la **cola A** de la **cola B**.

[a,b,c] = [Cabeza|Cola]

Cabeza = a

Cola = [b,c]

[X,Y,Z] = [a,b,c]

X = a

Y = b

Z = c

Terminación de Listas

1. Cuando la lista es **vacía** :

`predicado([]):- procesar([]).`

`predicado([Cabeza | Cola]):- procesar(Cabeza), predicado(Cola).`

2. Cuando un **elemento es encontrado**:

`predicado(Cabeza, [Cabeza | Cola]):- procesar algo.`

`predicado(X, [Cabeza | Cola]):- procesar algo, predicado(X, Cola).`

3. Cuando una **posición es encontrada**:

`predicado(1,Cabeza, [Cabeza | Cola]):- procesar algo.`

`predicado(P, X, [| L]):- P1=P-1, predicado(P1,X, L).`

Arboles

Como un árbol es una **estructura recursiva**, necesitaremos un caso base y un caso recursivo.

- X es un árbol vacío
- X es un árbol con hijos

```
binary_tree(void).
```

```
%caso base
```

```
binary_tree(t(K,L,R)) :-
```

```
%caso recursivo
```

```
    binary_tree(L),
```

```
    binary_tree(R).
```

Arboles

Como se puede observar se necesitan 3 elementos para definir un árbol: **la raíz**, **el subárbol izquierdo** y el **subárbol derecho**.

En árbol anterior **t(K,L,R)** queda definido como:

```
tree1(t(6, t(4, t(2, nil, nil), t(5, nil, nil)), t(9, t(7, nil, nil), nil))).
```

Arboles

También podemos definir reglas para realizar los posibles **recorridos en el árbol**:

- **Inorder:**

```
inorder(nil, []).
```

```
inorder(t(K,L,R), List):-inorder(L,LL),  
                           inorder(R, LR),  
                           append(LL, [K|LR],List).
```

- **Preorder:**

```
preorder(nil, []).
```

```
preorder(t(K,L,R), List):-preorder(L,LL),  
                           preorder(R, LR),  
                           append([K|LL], LR, List).
```

Arboles

- **Postorder:**
postorder(nil, []).
postorder(t(K,L,R), List):-postorder(L,LL),
postorder(R, LR),
append(LL, LR,R1),
append(R1, [K], List).



05

EJEMPLO



Doomie de **analizador sintáctico del inglés**

El modelo del lenguaje inglés consta de siete categorías o **tipos de lexema** :

Sustantivo, verbo,

adjetivo, adverbio,

artículo, conjunción,

preposición.

La gramática es **sensible al contexto** en tanto que obliga que haya coherencia cuantitativa entre sustantivos y verbos, así como entre sustantivos y artículos.

e.g., *a man likes a dog* (es coherente) ; *a man likes a dogs* (no es coherente)

El problema

Doomie de **analizador sintáctico del inglés**

```
isa(noun, word).  
isa(verb, word).  
isa(article, word).
```

```
isa(singular_noun, noun).  
isa(plural_noun, noun).  
isa(singular_verb, verb).  
isa(plural_verb, verb).  
isa(plural_article, article).  
isa(singular_article, article).  
isa(neutral_article, article).
```

isa(object, parent)

* se utiliza una estructura

arbórea

```
isa(man, singular_noun).  
isa(men, plural_noun).  
isa(dog, singular_noun).  
isa(dogs, plural_noun).  
isa(woman, singular_noun).  
isa(women, plural_noun).  
isa(leg, singular_noun).  
isa(legs, plural_noun).  
isa(bite, plural_verb).  
isa(bites, singular_verb).  
isa(like, plural_verb).  
isa(likes, singular_verb).  
isa(a, singular_article).  
isa(the, neutral_article).
```

Doomie de **analizador sintáctico del inglés**

```
hasprop(noun, type, noun).  
hasprop(verb, type, verb).  
hasprop(article, type, article).
```

```
hasprop(singular_noun, quant, singular).  
hasprop(plural_noun, quant, plural).  
hasprop(singular_verb, quant, singular).  
hasprop(plural_verb, quant, plural).  
hasprop(singular_article, quant, singular).  
hasprop(plural_article, quant, plural).  
hasprop(neutral_article, quant, singular).  
hasprop(neutral_article, quant, plural).
```

hasprop(object, property, value)

```
hasprop(man, quant, singular).  
hasprop(men, quant, plural).  
hasprop(dog, quant, singular).  
hasprop(dogs, quant, plural).  
hasprop(woman, quant, singular).  
hasprop(women, quant, plural).  
hasprop(leg, quant, singular).  
hasprop(legs, quant, plural).  
hasprop(bites, quant, singular).  
hasprop(bite, quant, plural).  
hasprop(likes, quant, singular).  
hasprop(like, quant, plural).  
hasprop(a, quant, singular).  
hasprop(the, quant, plural).  
hasprop(the, quant, singular).
```

Doomie de **analizador sintáctico del inglés**

```
hasproperty(Object, Property, Value):-  
    hasprop(Object, Property, Value).  
  
hasproperty(Object, Property, Value):-  
    isa(Object, Parent),  
    hasproperty(Parent, Property, Value).
```

* el predicado **hasproperty** contempla la herencia de propiedades de los lexemas padres a sus hijos.

```
adjective(big).  
adjective(small).  
adverb(strongly).  
adverb_adj(very).  
ad_adverb(quite).  
conjunction(and).  
conjunction(but).  
preposition(on).
```

Doomie de **analizador sintáctico del inglés**

```
utterance(X):-  
    sentence(X, []);  
    (sentence(X, Next_phrase), connector(Next_phrase, Rest), utterance(Rest));  
    (sentence(X, Next_phrase), connector(Next_phrase, Rest), prepphrase(Rest, [])).
```

Doomie de **analizador sintáctico del inglés**

```
sentence(Start, End):-  
    (nounphrase(Start, Rest, Number), verbphrase(Rest, End, Number)) ;  
    (nounphrase(Start, Rest, Number), verbphrase(Rest, LastBit, Number), prepphrase(LastBit, End)).
```

Doomie de **analizador sintáctico del inglés**

```
connector([X | Rest], Rest):-  
    conjunction(X).
```

```
prepphrase([Preposition | Rest], End):-  
    preposition(Preposition),  
    nounphrase(Rest, End, _).
```

Doomie de **analizador sintáctico del inglés**

```
nounphrase([Word | End], End, Number):-  
    hasproperty(Word, type, noun),  
    hasproperty(Word, quant, Number).  
nounphrase([Word1, Word2 | End], End, Number):-  
    hasproperty(Word1, type, article),  
    hasproperty(Word1, quant, Number),  
    hasproperty(Word2, type, noun),  
    hasproperty(Word2, quant, Number).  
nounphrase([Adjective, Word | End], End, Number):-  
    hasproperty(Word, type, noun),  
    hasproperty(Word, quant, Number),  
    adjective(Adjective).
```

nounphrase

Reglas de inferencia

Doomie de **analizador sintáctico del inglés**

```
nounphrase([Word1, Adjective, Word2 | End], End, Number):-  
    hasproperty(Word1, type, article),  
    hasproperty(Word1, quant, Number),  
    adjective(Adjective),  
    hasproperty(Word2, type, noun),  
    hasproperty(Word2, quant, Number).
```

```
nounphrase([Adverb_Adj, Adjective, Word | End], End, Number):-  
    hasproperty(Word, type, noun),  
    hasproperty(Word, quant, Number),  
    adjective(Adjective),  
    adverb_adj(Adverb_Adj).
```

nounphrase

Reglas de inferencia

Doomie de **analizador sintáctico del inglés**

```
nounphrase([Word1, Adverb_Adj, Adjective, Word2 | End], End, Number):-  
    hasproperty(Word1, type, article),  
    hasproperty(Word1, quant, Number),  
    hasproperty(Word2, type, noun),  
    hasproperty(Word2, quant, Number),  
    adjective(Adjective),  
    adverb_adj(Adverb_Adj).
```

```
nounphrase([Preposition | Rest], End, Number):-  
    preposition(Preposition),  
    nounphrase(Rest, End, Number).
```

nounphrase

Reglas de inferencia

Doomie de **analizador sintáctico del inglés**

```
verbphrase([Word | End], End, Number):-  
    hasproperty(Word, type, verb),  
    hasproperty(Word, quant, Number).  
verbphrase([Word | Rest], End, Number):-  
    hasproperty(Word, type, verb),  
    hasproperty(Word, quant, Number),  
    nounphrase(Rest, End, _).  
verbphrase([Adverb, Word | Rest], End, Number):-  
    adverb(Adverb),  
    hasproperty(Word, type, verb),  
    hasproperty(Word, quant, Number),  
    nounphrase(Rest, End, _).
```

verbphrase

Reglas de inferencia

Doomie de **analizador sintáctico del inglés**

```
verbphrase([Ad_adverb, Adverb, Word | Rest], End, Number):-  
    ad_adverb(Ad_adverb),  
    adverb(Adverb),  
    hasproperty(Word, type, verb),  
    hasproperty(Word, quant, Number),  
    nounphrase(Rest, End, _).
```

verbphrase

Reglas de inferencia



06

REFERENCIAS



REFERENCIAS

BIBLIOGRAFIA

- <http://ferestrepoca.github.io/paradigmas-de-programacion/proglogica/tutoriales/prolog-gh-pages/index.html#16-Des-Ven>
- <https://es.wikipedia.org/wiki/Prolog>
- <https://users.dcc.uchile.cl/~abassi/IA/Prolog.html>
- <http://www.di-mare.com/adolfo/cursos/2007-2/pp-Prolog.pdf>
- <https://es.scribd.com/document/104202187/Aplicaciones-desarrolladas-en-Prolog>

IMAGENES

- [1] <http://ferestrepoca.github.io/paradigmas-de-programacion/proglogica/tutoriales/prolog-gh-pages/index.html#16-Des-Ven>
- [2] https://es.wikipedia.org/wiki/Alain_Colmerauer
- [3] https://es.wikipedia.org/wiki/Philippe_Roussel
- [4] <https://snapcraft.io/swi-prolog>

REFERENCIAS

BIBLIOGRAFIA

- Logic Programming: a courseware (2015). Prolog Examples. Tomado de <http://athena.ecs.csus.edu/~mei/logicp/prolog/programming-examples.htm>

IMAGENES

- [5] https://www.tutorialspoint.com/prolog/prolog_data_objects.htm
- [6] Logic Programming: a courseware (2015). Prolog Examples. Tomado de <http://athena.ecs.csus.edu/~mei/logicp/prolog/programming-examples.htm>

THANKS



Para más información consultar en:
<http://ferestrepoca.github.io/paradigmas-de-programacion/proglogica/tutoriales/prolog-gh-pages/index.html#16-Des-Ven>

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, infographics & images by **Freepik**