


The page features abstract geometric designs in the corners. The top-left corner is filled with overlapping triangles in shades of blue, green, and red. The bottom-right corner contains a similar pattern of overlapping triangles in various shades of gray.

Tutorial Prolog

A vertical bar with three parallel lines in blue, green, and red colors.

Wilson Nicolás Arévalo Rodríguez
Víctor Alfredo Barragán Páez
Jonathan López Castellanos

Contenido

1

Introducción

Origen y propósito de Prolog

2

Primeros pasos

Descripción del entorno de programación y la sintaxis del lenguaje

3

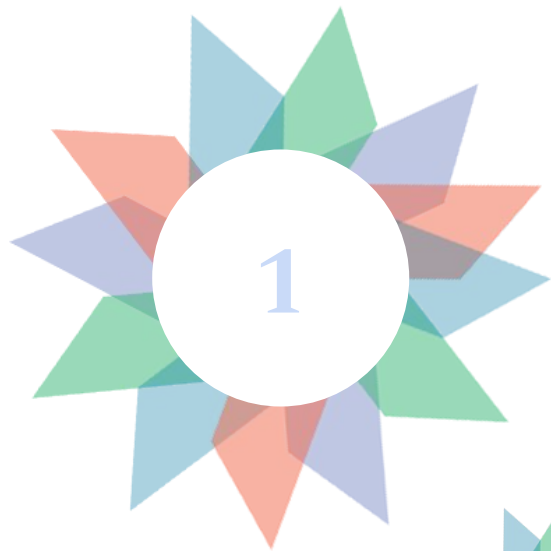
Conceptos básicos

Términos, operadores, expresiones, hechos, reglas y consultas

4

Conceptos intermedios

Reglas recursivas, backtracking, listas y árboles



Introducción

Contexto del lenguaje

Origen



Alain Colmerauer



Philippe Roussel

Su nombre proviene del francés
“**PRO**grammation en **LOG**ique”.

El lenguaje de programación Prolog se originó del trabajo hecho por **Robert A. Kowalski** en la Universidad de Edinburgh y **Alain Colmerauer** en la Universidad de Aix-Marseille (Francia) en los años 70. La investigación de **Kowalski** en el área de deducción automatizada, llevó al desarrollo con **Colmerauer** al uso formal de lógica como un lenguaje de programación.

Colmerauer y **Philippe Roussel** desarrollaron el primer intérprete (1972), y **David Warren** de la Universidad de Edinburgh desarrolló el primer compilador Prolog

Definición

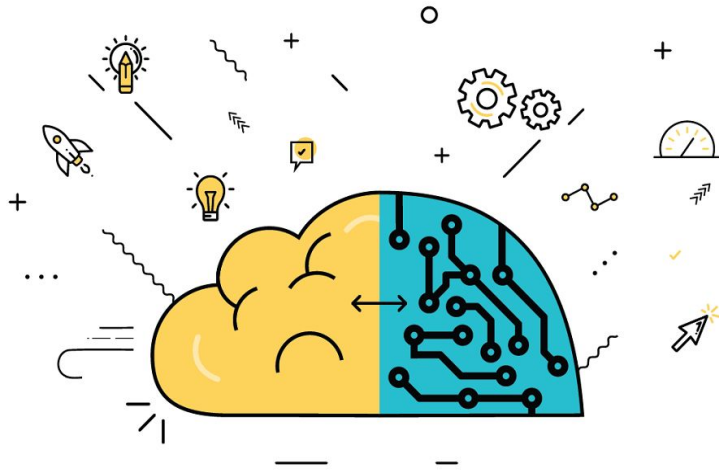
Es un lenguaje **declarativo** e **interpretado**, esto quiere decir que el lenguaje se usa para representar **conocimientos** sobre un determinado dominio y las relaciones entre objetos de ese dominio.

Se basa en nociones matemáticas de relaciones de inferencia.



Prolog es un lenguaje de programación especialmente indicado para **modelar problemas** que impliquen objetos y las relaciones entre ellos. Está basado en los siguientes **mecanismos básicos**: unificación, estructuras de datos basadas en árboles y backtracking automático.

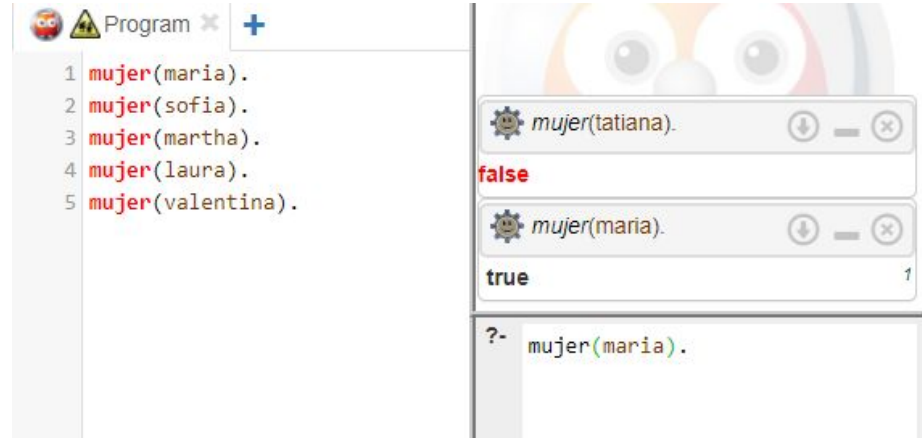
Propósito



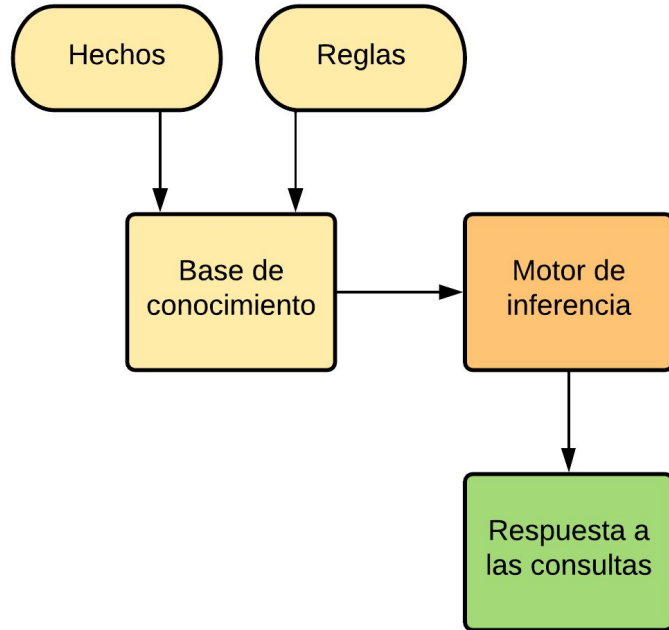
La definición de un pequeño conjunto de conceptos consigue un lenguaje de programación muy **potente** y **flexible**, este es ampliamente utilizado (junto con el lenguaje de programación **Lisp**) en aplicaciones que utilizan técnicas de **Inteligencia Artificial**. La popularidad de este lenguaje se debe a su **capacidad de deducción** y además es un lenguaje fácil de usar por su semántica y sintaxis. Sólo busca relaciones entre los objetos creados, las variables y las listas, que son su estructura básica. También es ampliamente utilizado en el procesamiento de **lenguaje natural** y **razonamiento lógico**.

Ventajas

- Modularidad
- Polimorfismo
- Expresividad (en la creación de un programa o base de conocimiento).
- Ejecución y búsqueda incorporada en el lenguaje



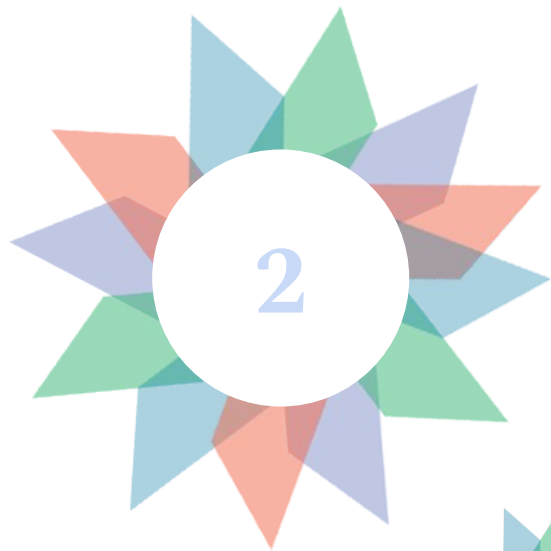
Método



Un programa en Prolog consiste en declarar el conocimiento disponible acerca de los objetivos, además de sus relaciones y sus reglas.

En lugar de tener que abrir un programa para ejecutar cierta aplicación y obtener una solución, se hace una pregunta, el programa revisa la base de hechos para encontrar la solución a la pregunta.

En caso de tener más de una solución, Prolog realiza backtracking para encontrar soluciones distintas.



Primeros pasos

Configuración del entorno de programación y
descripción de la sintaxis del lenguaje


IDE





SWI Prolog


Descarga de la versión de 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](#)

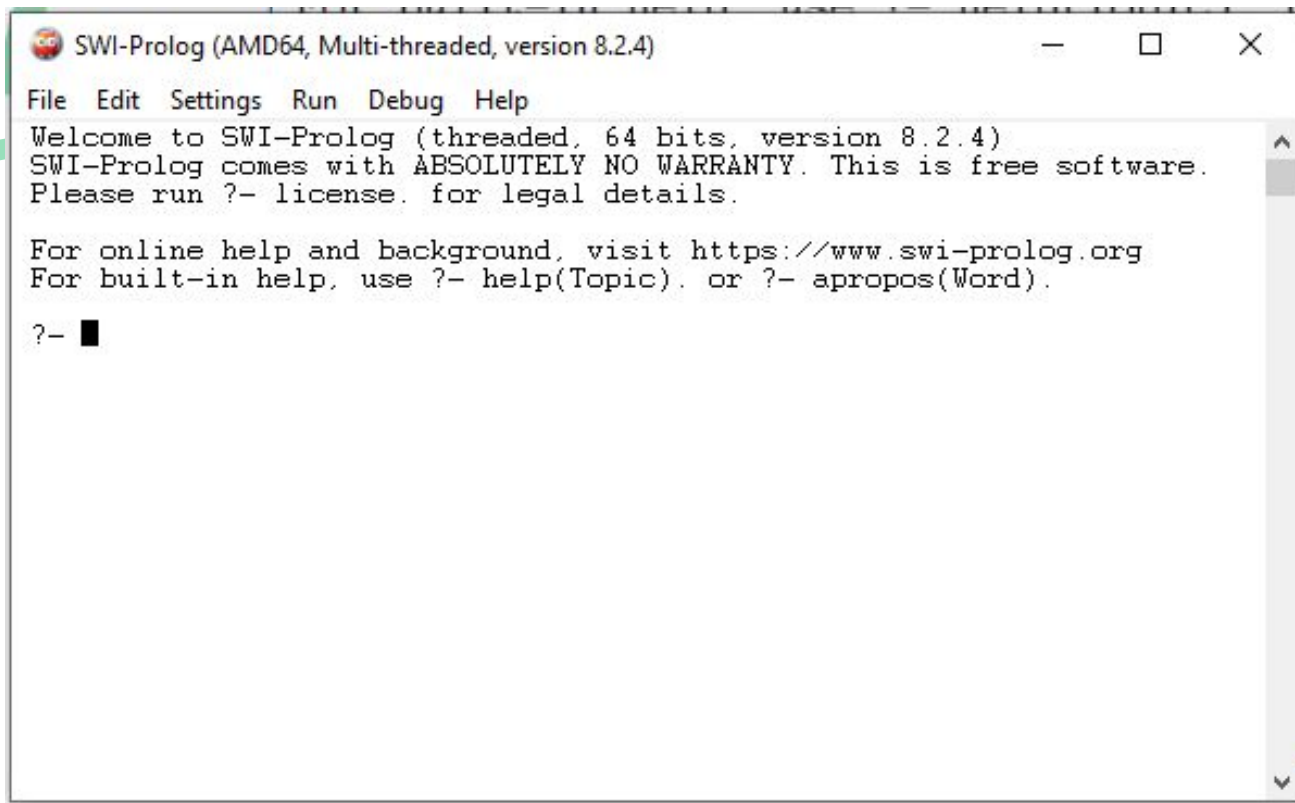
 Android binaries are available for [Termux](#) as the package `swi-prolog`. See also [Building SWI-Prolog on Android using LinuxOnAndroid](#)

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

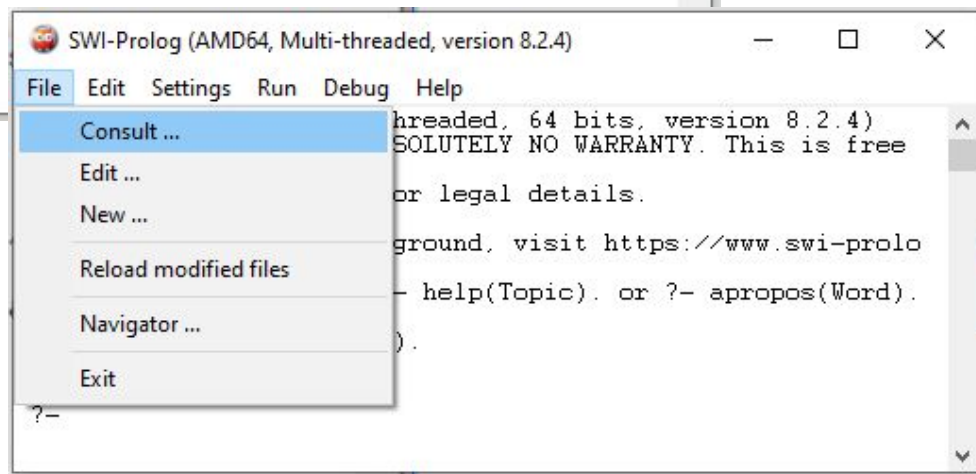
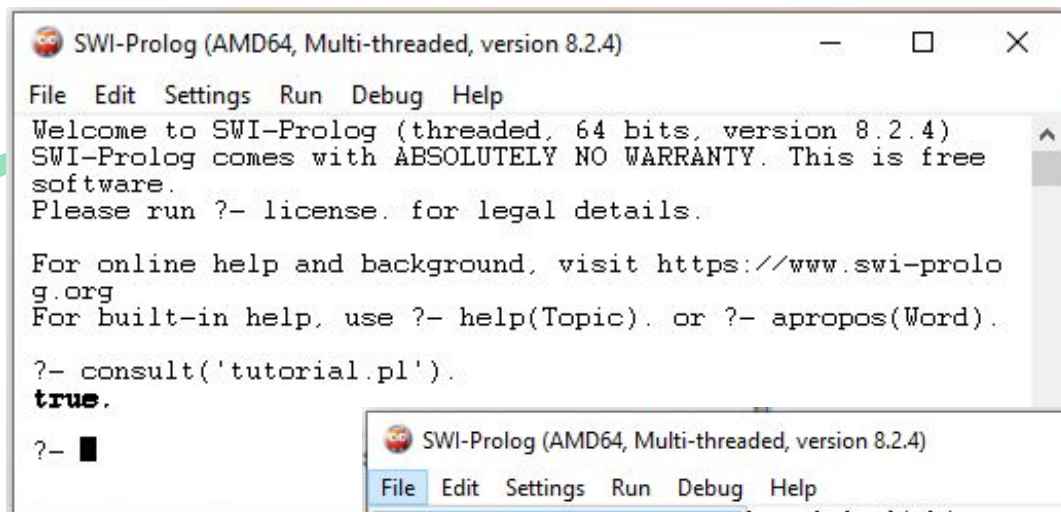
 Examine the [ChangeLog](#).

Binaries		
	11,836,832 bytes	SWI-Prolog 8.2.4-1 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: 787bf6f588479cbca2dfbb13011c738eff8935280287b62e40abc6e7c036839
	11,478,340 bytes	SWI-Prolog 8.2.4-1 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: 29df223fbace7a07d0b56d32cd86b482c272ac9b1c987a4503c9af0f5ea1de13
	27,794,581 bytes	SWI-Prolog 8.2.4-1 for MacOSX 10.12 (Sierra) and later 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: c1b34dea44950af9c1c2e413e8f86cf16bbdb15b43b3098617a48a7f85841dd4
Sources		
	10,998,859 bytes	SWI-Prolog source for 8.2.4 Sources in <code>.tar.gz</code> format, including packages and generated documentation files. See build instructions . SHA256: f4bcc78437f9080ab089762e9e6afa7071df7f584c14999b92b9a90a4efbd7d8
Documentation		
	2,726,944 bytes	SWI-Prolog 8.2.4 reference manual in PDF SWI-Prolog reference manual as PDF file. This does <i>not</i> include the package documentation.

Versión de escritorio



Versión de escritorio



Uso de la versión en línea (web)

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

The screenshot displays the SWISH web interface. At the top, there is a navigation bar with the SWISH logo, a menu (File, Edit, Examples, Help), a search bar, and a notification bell showing 25 alerts. Below the navigation bar, a sidebar on the left contains a 'New tab' button and a 'Create a' section with buttons for 'Program' and 'Notebook'. A search bar in the sidebar contains the text 'user:"me"' and shows 'No matching files'. A message below the search bar suggests using the Examples menu or the Program/Notebook button. The main content area features a large, stylized owl graphic. At the bottom, there is a query input field with the placeholder text 'Your query goes here ...' and buttons for 'Examples', 'History', and 'Solutions'. A checkbox for 'table results' and a 'Run!' button are also present.

SWISH File Edit Examples Help

239 users online

Search

New tab

Create a Program Notebook here

based on Empty Student CLP profile

user:"me" Filter Type

No matching files

If you are a new user you may

- Use the Examples menu from the navigation bar
- Use the Program or Notebook button above

help on search

?- Your query goes here ...

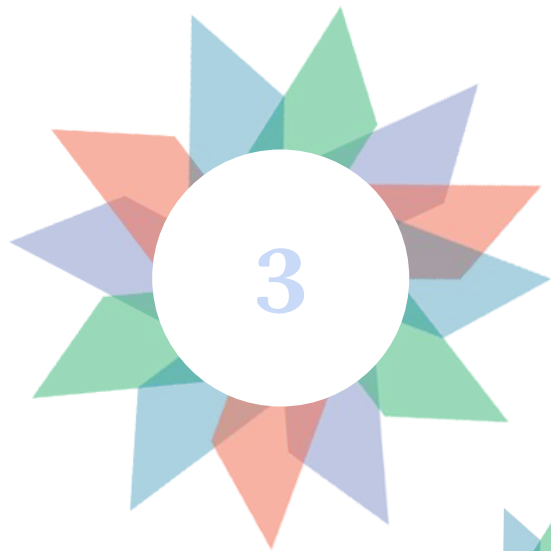
Examples History Solutions

☐ table results Run!

Uso de la versión en línea (web)

The image shows a screenshot of the SWISH web interface. The interface has a top navigation bar with the SWISH logo, a search bar, and a user count (262 users online). The main content area is divided into three sections:

- Conocimiento** (Knowledge): A large red-bordered box on the left side of the interface. It contains the text "1 Your Prolog rules and facts go here ...".
- Resultados de las consultas** (Query Results): A green-bordered box on the right side of the interface. It features a large, stylized owl graphic in the background and the text "Resultados de las consultas" in green.
- Consultas** (Queries): A blue-bordered box at the bottom right of the interface. It contains the text "Your query goes here ...".



Conceptos básicos



Términos

Todos los datos están representados por términos; corresponden a un valor en Prolog.

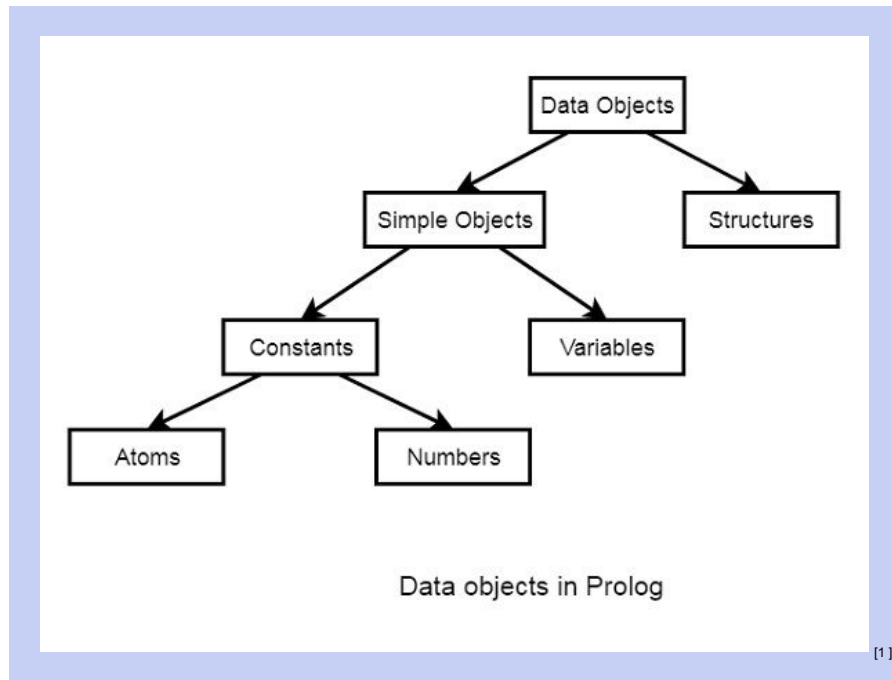
- Átomos: constantes textuales. Ej: tom, x25, 'Black'
- Variables: valores que no han sido 'vinculados'. Comienzan con una letra mayúscula o un guión bajo.
 - Variables **anónimas**: Usadas cuando no es necesario conocer el valor de la variable.
'_'
- **Estructuras**: Objetos compuestos de otros objetos.

line(point(5, 10), point(6, 30))

↑
Nombre

↑
Argumentos: otros términos
Aridad: número de argumentos

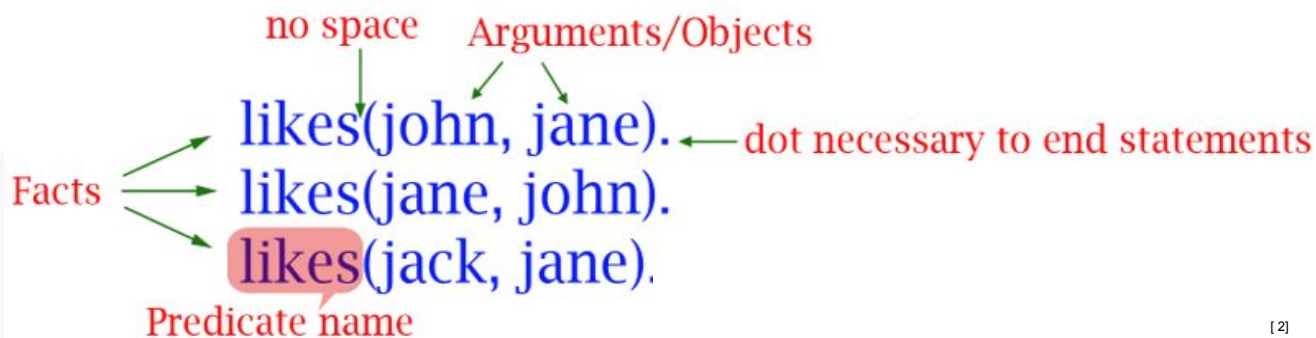
Functor: Combinación de nombre y aridad de un término compuesto.



Hechos

Cláusulas sin cuerpo que muestran una relación explícita entre objetos y propiedades que estos objetos puedan tener.

Declaraciones que se deben considerar como **verdaderas**, naturalmente.



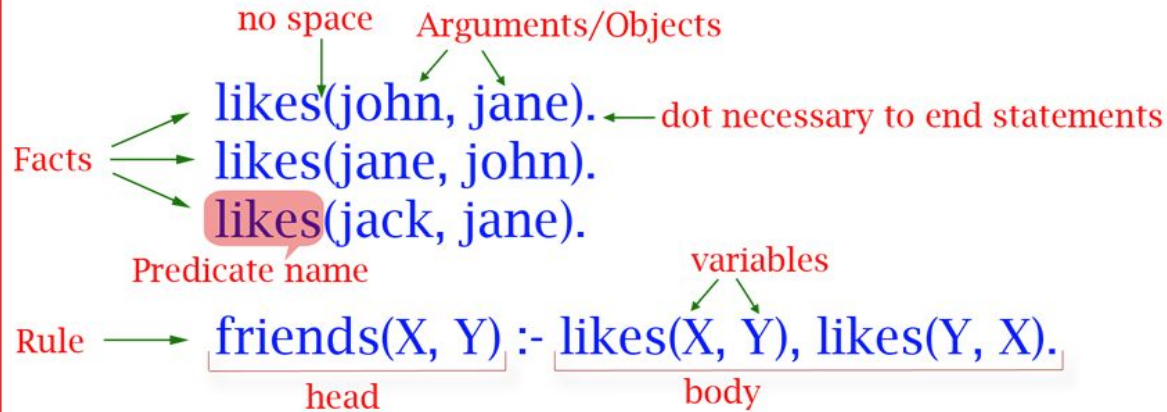
[2]

Son de la forma: relacion(objeto1,objeto2...).

Los nombres de propiedades / relaciones comienzan con letras minúsculas.

Una colección de hechos con el mismo functor se denomina **predicado**.

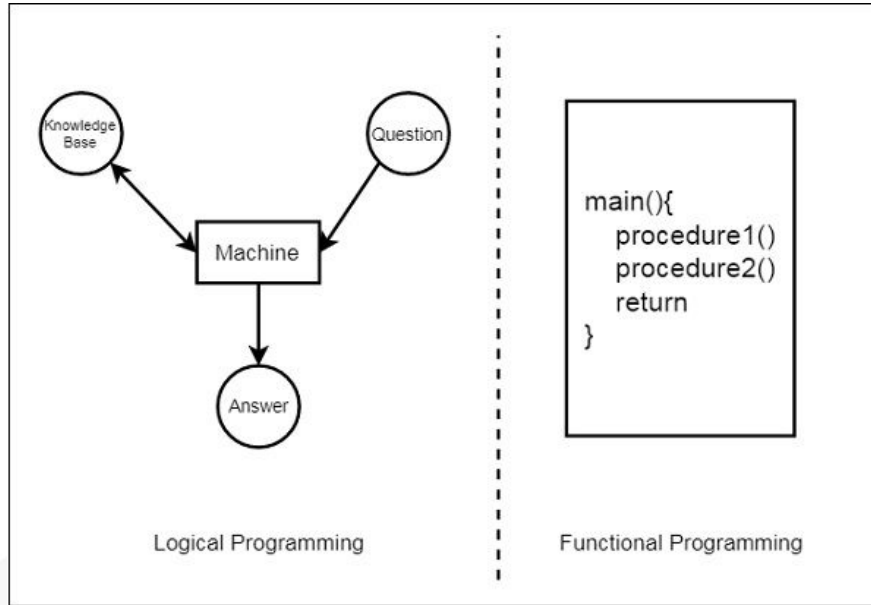
Reglas



- Relaciones implícitas entre objetos.
- De la forma "concluir que (algo) es cierto **si** (algo más) es cierto".
- Constan de una cabeza y un cuerpo separados por el operador ":-", que significa "Si" o "está implícito por".

[2]

La regla concluye que dos personas (X,Y) son amigas si a X le agrada Y y viceversa.



[3]

Consultas

Preguntas sobre las relaciones entre los objetos y las propiedades de los objetos.
Realizadas en el entorno principal de Prolog.

Todos los hechos y reglas definidas se almacenarán en un archivo llamado base de datos o **base de conocimientos**.

```
db.pl
happy(alice).
happy(bob).
happy(bill).
with_albert(alice).

runs(albert) :-
    happy(albert).
dances(alice) :-
    happy(alice),
    with_albert(alice).
```



```
?- dances(alice).
true.
```

De acuerdo a las consultas, el lenguaje de programación lógica puede encontrar una respuesta en la base y devolverla.

Operadores

Aclaraciones:

Primera columna: precedencia.

Segunda columna: Tipo, f indica la posición del operador, x/y posición de los argumentos.

Operadores a destacar:

- `:=` igualdad aritmética entre expresiones.
- `=/=` desigualdad aritmética entre expresiones.
- `\+` **negación**.
- `“ , ”` y `“ ; ”`: operadores AND y OR respectivamente, comúnmente utilizados en reglas.
- `**` potencia.
- `//` división entera .
- `=` unificación.
- `is`: **evalúa** expresiones matemáticas.

1200	xfx	-->, :-
1200	fx	:-, ?-
1150	fx	dynamic, discontinuous, initialization, meta_predicate, module_transparent, multifile, public, thread_local, thread_initialization, volatile
1105	xfy	
1100	xfy	;
1050	xfy	->, *->
1000	xfy	,
990	xfx	:=
900	fy	\+
700	xfx	<, =, . ., =@=, \@=, =:=, =<, ==, =\=, >, >=, @<, @=, @>, @>=, \=, \==, as, is , >:<, :<
600	xfy	:
500	yfx	+, -, /\, \/, xor
500	fx	?
400	yfx	*, /, //, div, rdiv, <<, >>, mod, rem
200	xfx	**
200	xfy	^
200	fy	+, -, \
100	yfx	.
1	fx	\$

Algunas funciones matemáticas

random_between(A, B, Y).

Asigna a Y un valor aleatorio entre A y B

truncate(X).

Remueve la parte fraccionaria del flotante X.

**pi.
e.**

between(A, B, X).

Obtiene todos los valores entre A y B

**floor(X).
ceiling(X).**

Retorna el piso de X.
Retorna el techo de X.

Valores de pi y
épsilon resp.

succ(A, X).

Añade 1 a A y lo asigna a X.

sqrt(A).

Retorna la raíz cuadrada de A.

abs(X).

Valor absoluto de X.

**sin(A).
cos(A).
tan(A).**

Seno, coseno y
tangente de A resp.

**max(X,Y).
min(X,Y).**

Valor máximo entre X y Y
Valor mínimo entre X y Y

**asin(A).
acos(A).
atan(A).**

Arcoseno, arcocoseno y
arcotangente de A resp.

round(X).

Redondea un valor
cercano a X

**log(A).
log10(A).
exp(A).**

Logaritmo natural de A.
Logaritmo base 10 de A.
Exponencial de A.

Operaciones de entrada/salida

nl = new line

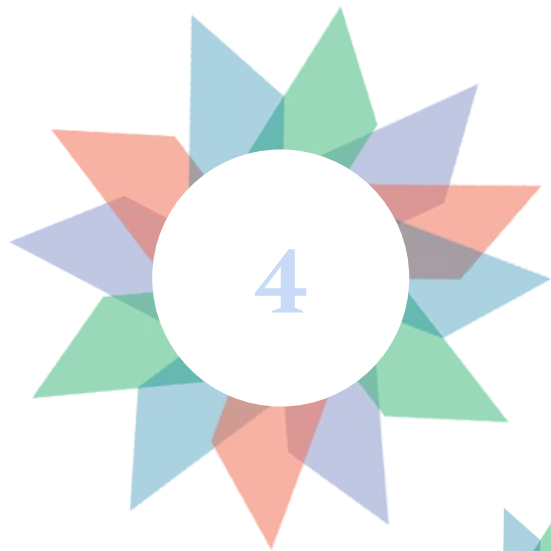
Predicado	Función
write(X)	Escribe el término en la salida
writeln(X)	Escribe el término en la salida mostrando comillas
read(X)	Lee el término en consola
get(X)	Recibe un carácter que se guarda ASCII
put(X)	Escribe un salida un solo carácter
format(formato, argumentos)	Formato de uno o más términos en salida

Cambios dinámicos a la base de conocimiento

Predicados que se deseen cambiar se deben marcar como dinámicos antes de usarse.

`:- dynamic(predicado/aridad)`

Predicado	Función
assertz(H)	Inserta un hecho al final de la lista para ese predicado
asserta(H)	Inserta un hecho al inicio de la lista para ese predicado
retract(H)	Remueve un hecho
retractall(H)	Remueve todos los hechos que coincidan con H



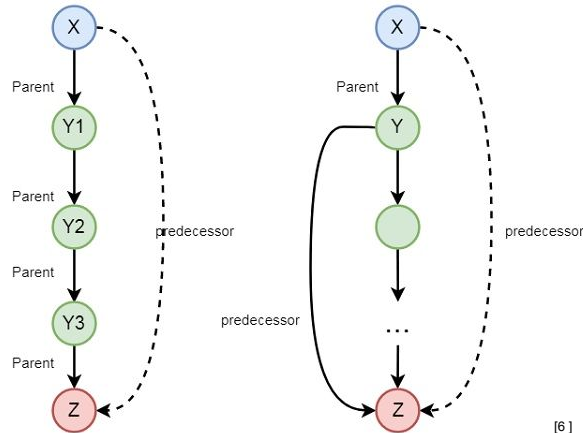
Conceptos intermedios



Reglas recursivas

Una regla es recursiva si uno de sus objetivos (consulta) se refiere al predicado que la misma regla define, es decir, un predicado se usa a sí mismo para encontrar un valor de verdad.

Ej:



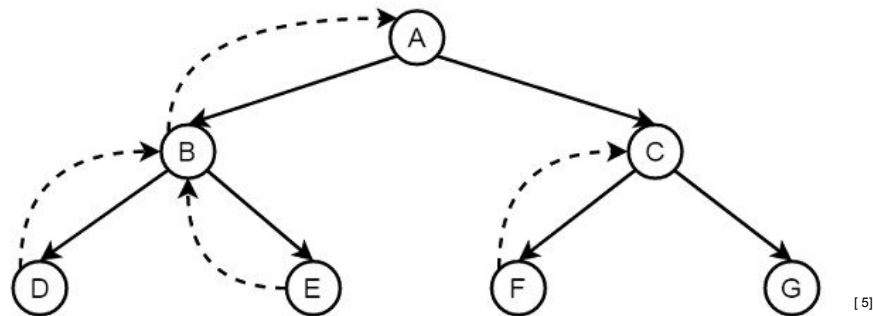
`predecessor(X,Z) :-
parent(X,Z).`

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

X es predecessor de Z si X es padre de Z o si X es padre de un Y, y Y es predecessor de Z.

Backtracking

Si un predicado ofrece múltiples reglas para resolver un objetivo, 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.



[5]

```
[debug] ?- trace.  
true.
```

```
[trace] ?- female(X), happy(X).  
Call: (11) female(_7386) ? creep  
Exit: (11) female(alice) ? creep  
Call: (11) happy(alice) ? creep  
Exit: (11) happy(alice) ? creep  
X = alice ;  
Redo: (11) female(_7386) ? creep  
Exit: (11) female(betsy) ? creep  
Call: (11) happy(betsy) ? creep  
Fail: (11) happy(betsy) ? creep  
Redo: (11) female(_7386) ? creep  
Exit: (11) female(diana) ? creep  
Call: (11) happy(diana) ? creep  
Fail: (11) happy(diana) ? creep  
false.
```

Knowledge Base

```
female(alice).  
female(betsy).  
female(diana).
```

```
happy(albert).  
happy(alice).  
happy(bob).  
happy(bill).  
with_albert(alice).
```

Listas

La lista es una estructura de datos simple que almacena secuencialmente elementos de cualquier tipo (incluso listas). Es flexible, por lo que no tiene un tamaño fijo. Cada lista tiene una cabeza (el primer elemento) y una cola (la lista conformada por todos los elementos de la lista excluido el primero).

Ejemplo:

[maria, camion, 32, ['dato', 3.56]]

Cabeza: maria

Cola: [camion, 32, ['dato', 3.56]]

[maria, camion, 32, ['dato', 3.56]]

maria

[camion, 32, ['dato', 3.56]]

camion

[32, ['dato', 3.56]]

32

[['dato', 3.56]]

['dato', 3.56] []

'dato' [3.56]

3.56

[]

Deestructuración de una
lista en su cabeza y cola

Unificación de listas



Algunos ejemplos de unificación de listas mediante variables usando la notación [Cabeza | Cola]:

Lista de ejemplo:

$[maria, camion, 32, ['dato', 3.56]] = [V \mid Z]$

$V \rightarrow Cabeza = maria$

$Z \rightarrow Cola = [camion, 32, ['dato', 3.56]]$

$[maria, camion, 32, ['dato', 3.56]] = [V, W \mid Z]$

$V = maria$

$W = camion$

$Z \rightarrow Cola = [32, ['dato', 3.56]]$

$[maria, camion, 32, ['dato', 3.56]] = [V, W, X \mid Z]$

$V = maria$

$W = camion$

$X = 32$

$Z \rightarrow Cola = [['dato', 3.56]]$

$[maria, camion, 32, ['dato', 3.56]] = [V, W, X, Y \mid Z]$

$V = maria$

$W = camion$

$X = 32$

$Y = ['dato', 3.56]$

$Z \rightarrow Cola = []$

Criterios de terminación para listas



Lista vacía

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

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

Un elemento determinado

Una posición determinada

`predicado(1, Cabeza, [Cabeza | Cola]):-objetivos....`
`predicado(P, E, [| Cola]):-P1=P-1, objetivos..., predicado(P1, E Cola).`

Árbol SLD

Selección-Resolución Lineal-Cláusulas definitivas, el método de resolución básico de Prolog para resolver casos de no-determinismo

```
longitud([], 0).  
longitud([_ | T], N):-  
    longitud(T, N1),  
    N is N1 + 1.
```

?-longitud([1, 2], L).
 ↓ { T / [2], N / L }
?-longitud([2], N1), L is N1 + 1.

Árbol SLD

Selección-Resolución Lineal-Cláusulas definitivas, el método de resolución básico de Prolog para resolver casos de no-determinismo

```
longitud([], 0).  
longitud([_ | T], N):-  
    longitud(T, N1),  
    N is N1 + 1.
```

?-longitud([1, 2], L).

↓ { T / [2], N / L }

?-longitud([2], N1), L is N1 + 1.

↓ { T / [], N2 / N1 }

?-longitud([], N2), N1 is N2 + 1, L is N1 + 1.

Árbol SLD

Selección-Resolución Lineal-Cláusulas definitivas, el método de resolución básico de Prolog para resolver casos de no-determinismo

```
longitud([], 0).  
longitud([_ | T], N):-  
    longitud(T, N1),  
    N is N1 + 1.
```

?-longitud([1, 2], L).

↓ { T / [2], N / L }

?-longitud([2], N1), L is N1 + 1.

↓ { T / [], N2 / N1 }

?-longitud([], N2), N1 is N2 + 1, L is N1 + 1.

↓ { N2 / 0 }

?-N1 is 0 + 1, L is N1 + 1.

Árbol SLD

Selección-Resolución Lineal-Cláusulas definitivas, el método de resolución básico de Prolog para resolver casos de no-determinismo

```
longitud([], 0).  
longitud([_ | T], N):-  
    longitud(T, N1),  
    N is N1 + 1.
```

?-longitud([1, 2], L).
 ↓ { T / [2], N / L }
?-longitud([2], N1), L is N1 + 1.
 ↓ { T / [], N2 / N1 }
?-longitud([], N2), N1 is N2 + 1, L is N1 + 1.
 ↓ { N2 / 0 }
?-N1 is 0 + 1, L is N1 + 1.
 ↓ { N1 / 1 }
?-L is 1 + 1.

Árbol SLD

Selección-Resolución Lineal-Cláusulas definitivas, el método de resolución básico de Prolog para resolver casos de no-determinismo

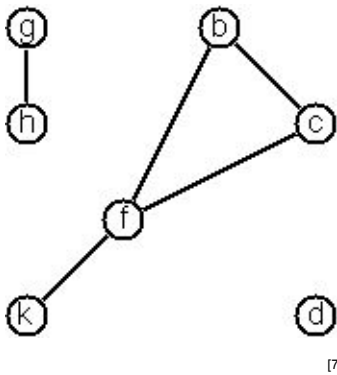
```
longitud([], 0).  
longitud([_ | T], N):-  
    longitud(T, N1),  
    N is N1 + 1.
```

?-longitud([1, 2], L).
 ↓ { T / [2], N / L }
?-longitud([2], N1), L is N1 + 1.
 ↓ { T / [], N2 / N1 }
?-longitud([], N2), N1 is N2 + 1, L is N1 + 1.
 ↓ { N2 / 0 }
?-N1 is 0 + 1, L is N1 + 1.
 ↓ { N1 / 1 }
?-L is 1 + 1.
 ↓ { L / 2 }

Grafos

Estructuras definidas como un conjunto de nodos y un conjunto de aristas, donde cada arista es un par de nodos.

Múltiples formas de representar en Prolog:



Grafos no dirigidos

Forma arista-hecho: Cada arista es un hecho diferente.

```
arista(b,c).      arista(b,f).  
arista(c,f).      ...
```

Forma grafo-término: Grafo como objeto con dos tipos de conjuntos. (nodos, aristas)

```
grafo([b,c,d,f,g,h,k],[e(b,c),e(b,f),e(c,f),e(f,k),e(g,h)])
```

Forma lista de adyacencia: asocia cada nodos a sus adyacentes.

```
[n(b,[c,f]), n(c,[b,f]), n(d,[ ]), ...]
```

“Forma amigable”:

```
[b-c, f-c, g-h, d, f-b, k-f, h-g]
```

Grafos dirigidos

Poseen arcos (aristas con sentido definido)

Forma arco-hecho:

```
arco(s,u).      arco(u, s).  
arco(u,r).      ...
```

Forma grafo-término:

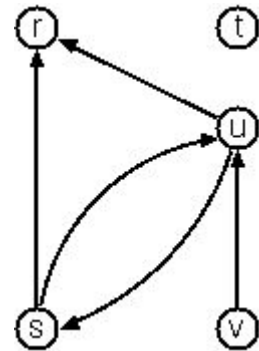
```
digrafo([r,s,t,u,v],[a(s,r),a(s,u),a(u,r),a(u,s),a(v,u)])
```

Forma lista de adyacencia:

```
[n(r,[ ]),n(s,[r,u]),n(t,[ ]),n(u,[r]),n(v,[u])]
```

“Forma amigable”:

```
[s > r, s > u, u > r, u > s, v > u, t]
```



Referencias

1. Prolog-Data Objects. Tomado de https://www.tutorialspoint.com/prolog/prolog_data_objects.htm
2. Logic Programming: a courseware (2015). Prolog Examples. Tomado de <http://athena.ecs.csus.edu/~mei/logicp/prolog/programming-examples.htm>
3. Prolog - Introduction. Logic and Functional Programming. Tomado de https://www.tutorialspoint.com/prolog/prolog_introduction.htm
4. Prolog Operators. Tomado de <https://www.swi-prolog.org/pldoc/man?section=operators>
5. Prolog-Backtracking. Tomado de https://www.tutorialspoint.com/prolog/prolog_backtracking.htm
6. Prolog-Recursion and Structures. Tomado de https://www.tutorialspoint.com/prolog/prolog_recursion_and_structures.htm
7. P-99: Ninety-Nine Prolog Problems. Graphs. Tomado de <https://www.ic.unicamp.br/~meidanis/courses/mc336/2009s2/prolog/problemas/>



**Gracias por
su atención**