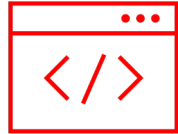




UNIVERSIDAD
NACIONAL
DE COLOMBIA

PROYECTO **CULTURAL, CIENTÍFICO Y COLECTIVO** DE NACIÓN



Prolog

Prof. Oscar Mauricio Salazar Ospina

Correo: omsalazaro@unal.edu.co

3007743 - Programación Lógica y Funcional
3010426 - Teoría de Lenguajes de Programación

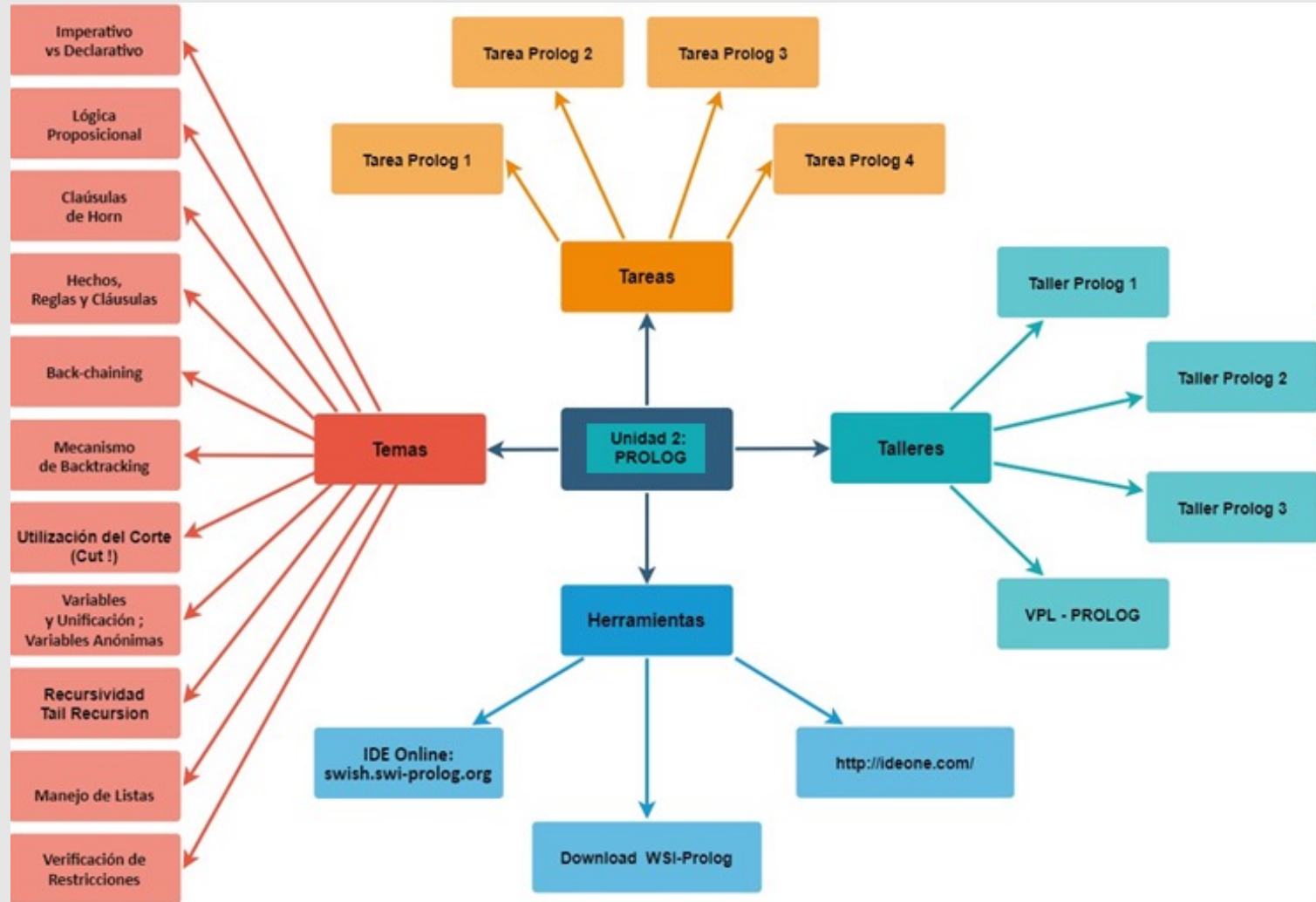
Facultad de Minas
Departamento de ciencias de la computación y la decisión

Universidad Nacional de Colombia

PROYECTO **CULTURAL, CIENTÍFICO Y COLECTIVO** DE NACIÓN

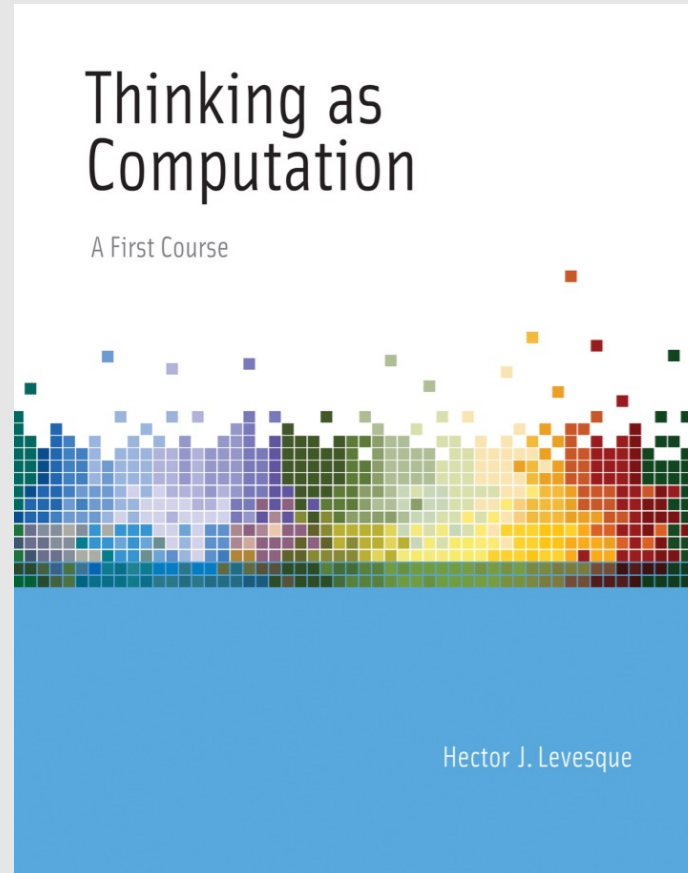
Prolog

Contenido Módulo Prolog



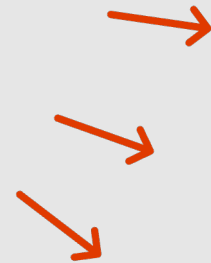
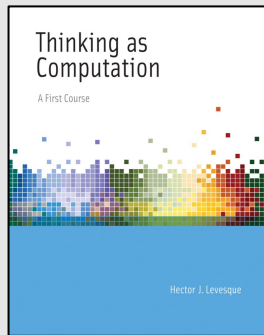
Prolog

Material complementario



Prolog

Material complementario



1 Thinking and Computation

Consider the following scenario:

A professor enters a classroom where a group of undergraduates is sitting, and announces "There is free pizza in the hall!" Suddenly, the students stand up and stampede toward the classroom door.

The events described here seem so ordinary that it is easy to miss how truly remarkable they are. But step back from them for a moment and imagine that you are studying these students as a curious scientist from another world. You observe that a certain sound emanates from the professor and that this causes a flurry of activity in the students. Now, as a scientist, ask yourself this: What sort of *physics* would explain how acoustic energy can be transformed into kinetic energy in this way? In particular, note that a very small change in the acoustic energy (like the professor's saying "There is free pizza in Nepal.") can result in *no* kinetic energy being produced at all, except maybe for some puzzled head shaking.

This is the wonder of *intelligent behavior*, perhaps the single most complex natural phenomenon that we are aware of. As a sheer mystery, it easily overshadows topics like dark matter, the source of gravity, and the mechanics of cancer.

One striking aspect of intelligent behavior of this sort is that it is clearly conditioned by *knowledge*: for a very wide range of activities, people make decisions about what to do based on what they know (or believe) about the world, effortlessly and often unconsciously. It's certainly not the *sounds* themselves that cause the students to stand up like animals that have been trained to respond to a bell. This is easy enough to confirm. The professor could have brought in a *sign* with a pizza message on it written in big letters, and the effect would have been just the same. In fact, one can imagine a situation where the following is written on the whiteboard at the front of the classroom:

As part of a psychology experiment, the professor will soon enter and tell you that free food is available. This is just a test. Please remain seated.

In this case, neither the sounds nor the sign would have any effect at all. One can try other small variations, and it will become clear that what makes the difference is whether the students come to *believe* there is free pizza to be had nearby.

3 The Prolog Language

...procedure called back-chaining could draw logical sentences. It was the first demonstration to be seen as computation. But so far this

...to write computer programs in a language up being knowledge bases like those in Queries will direct a computer to perform it" in the previous chapter. If nothing else, ted in this book, really is a procedure that

...in *logic*, is a language for writing programs and colleagues. There are several dialects ces. This book uses a popular one called n this and other dialects of Prolog.)

...Prolog is a bit like learning a foreign lan- i to memorize not just the new vocabulary w language.

...ning English. There will be rules of spelling r ("c"), and many more involving grammar. ; about the parent of a child. In French, one son and sa depending on whether the parent other), son père (father). In English, on the obsessive adjective depends on whether the oy, it's his mother, his father; for a girl, it's

...ut learning Prolog. The good news is that ed to get all the spelling and grammar of r language fits on one page (see figure 3.10).

2 A Procedure for Thinking

Chapter 1 showed that thinking, or at least some simple forms of thinking, could drawing conclusions from a large collection of sentences Leibniz's idea was that the rules of logic would tell how structures representing propositions the same way that r to manipulate symbolic structures representing num- is symbolic manipulation as a computational *procedure*, arithmetic.

is chapter is called *back-chaining*. It works on sentences are very restricted in their form. They have none of r any other natural language. On the other hand, these nterpreted symbolic structures: one can operate on them vance what they mean. This is in accordance with the at one can produce interesting answers to interesting now what the symbols stand for.

ms. The first section looks at the types of sentences ie and introduces a small example. Section 2 examines it in a bit more detail. The back-chaining procedure is 1.4 looks at some complex behavior of back-chaining summarizes very briefly what is good and less good ing.

es

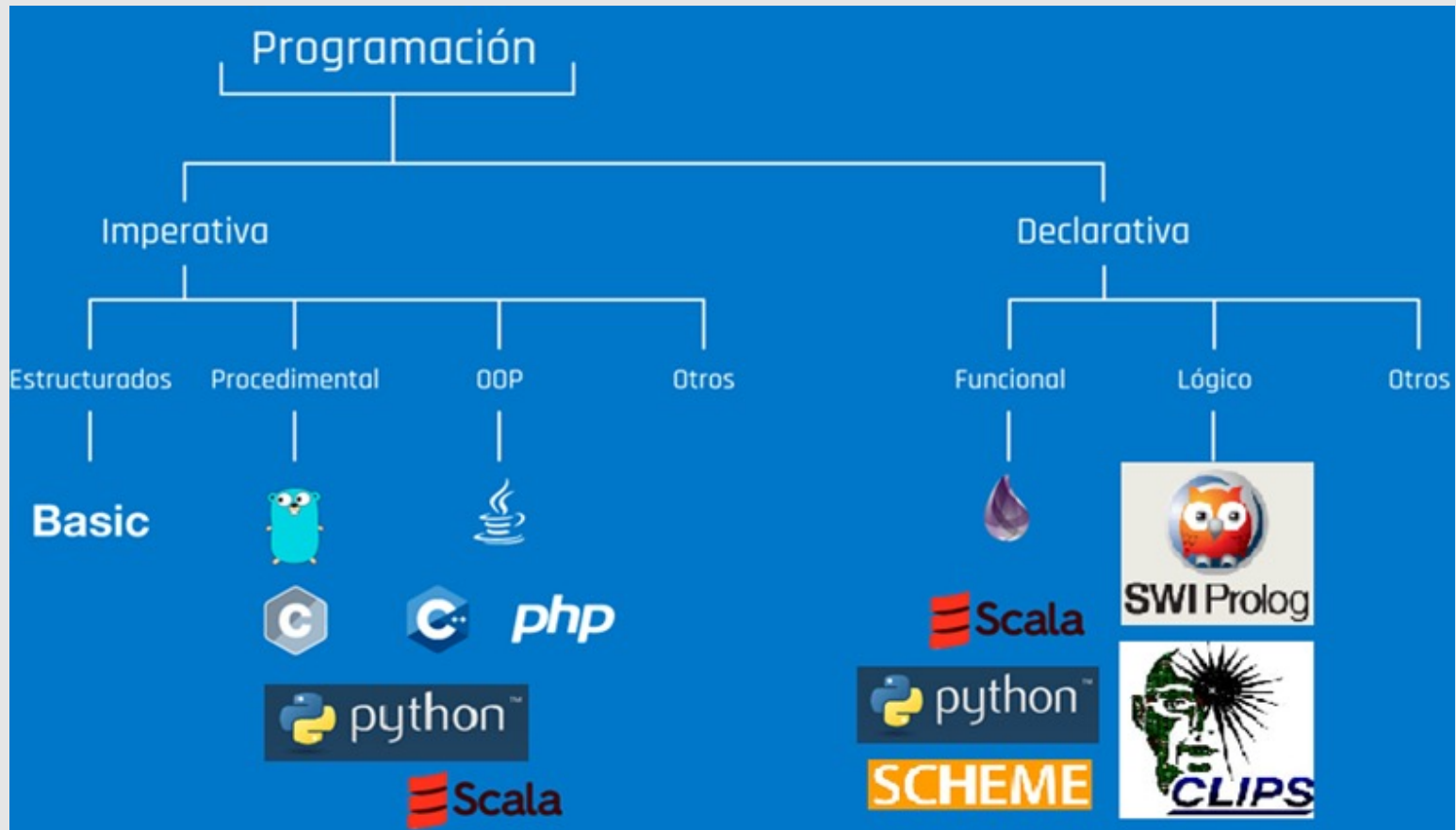
of a knowledge base is one consisting of just two sorts

s, simple basic sentences whose exact form is left

it is, sentences of the form *If P_1 and ... and P_n then Q* , re atomic sentences

Prolog

Paradigmas de programación



Prolog

Paradigma lógico



La **Programación Lógica** estudia el uso de la lógica para la modelación de problemas y el control sobre las reglas de inferencia para alcanzar una solución automática.

Prolog

Paradigma lógico



Objetivo: Aplicar reglas de la lógica para inferir conclusiones a partir de datos...

- Juan y María son pareja
- Rosa y Carlos son hijos de Juan y María
- Juan Tiene un Hermano
- María tiene otro hijo llamado Lucas
- La mamá de María es Matilde

¿Qué podemos inferir de estas frases?

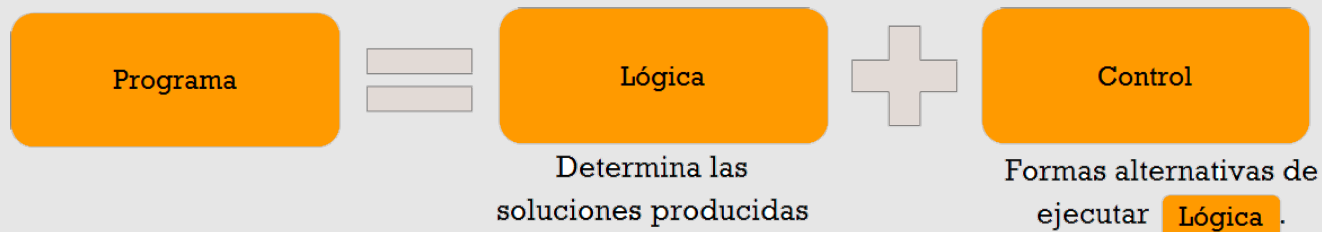
Prolog

Paradigma lógico



La Programación lógica y funcional, se conoce como Programación Declarativa

- Se trabaja en una forma descriptiva, estableciendo relaciones entre entidades, indicando qué se debe hacer y no el cómo.



- **Lógica (programador):** hechos y reglas para representar conocimiento
- **Control (interprete):** deducción lógica para dar respuestas (soluciones)

Prolog

Imperativo vs declarativo

Factorial de n

```
n=int(input("Entre número:"))
fact=1
while(n>0):
    fact=fact*n
    n=n-1
print("El factorial es: ")
print(fact)
```

Entre número:6
El factorial es: 720

%% PROLOG: Factorial de n

```
factorial(1,1).
factorial(N,M) :- N>1, N1 is N-1,
factorial(N1,K), M is N*K.
```

? factorial(6,Y)
-> Y=720

Lógica:
hechos y
reglas

Algoritmo: datos + procedimientos (secuencia de instrucciones)

Prolog

En este contexto se requiere solucionar problemas del tipo...



Dado un **problema X**, consultar o inferir si la **afirmación A** es solución o no de X o **en qué casos lo es**.



Además queremos implementar las **soluciones** de tal forma que la **resolución del problema** se haga de forma **automática o semi-automática**.



X: Comprar **bici** al **mejor precio**. ¿Cuáles son las reglas?



X: **Diagnosticar** una **enfermedad**. ¿Cuáles son los síntomas?



X: Realizar **detección** de **fallas de un dispositivo** cuando presenta falencias. ¿Cuáles son los errores?

Prolog

Otro objetivo, construir una base de conocimientos mediante reglas y hechos...



X: Comprar bici al mejor precio.

- ¿Cuáles son las reglas?
- ¿Cuáles son los hechos?



X: Determinar el diagnóstico de una enfermedad conociendo los síntomas.

- ¿Cuáles son las reglas?
- ¿Cuáles son los hechos?



X: Realizar detección de fallas de un dispositivo cuando presenta errores de ejecución.

- ¿Cuáles son las reglas?
- ¿Cuáles son los hechos?

Generalidades

Otras características de este paradigma...



Unificación de términos.

- Evitar el uso de sinónimos



Mecanismos de inferencia **automáticos**.



Recursión como estructura de **control** básica.



Visión lógica de la computación.

- Un computador no es hinja de nadie
- Un computador no gusta de un color u otro
- ...

Prolog

Lógica proposicional...

La **lógica proposicional**, conocida como **lógica de enunciados**, toma como **elemento básico** las **frases declarativas simples o proposiciones**.

<i>Sentencia</i>	→	<i>Sentencia Atómica</i> <i>Sentencia Compleja</i>
<i>Sentencia Atómica</i>	→	Verdadero Falso <i>Símbolo Proposicional</i>
<i>Símbolo Proposicional</i>	→	P Q R ...
<i>Sentencia Compleja</i>	→	\neg <i>Sentencia</i> (<i>Sentencia</i> \wedge <i>Sentencia</i>) y (<i>Sentencia</i> \vee <i>Sentencia</i>) o (<i>Sentencia</i> \Rightarrow <i>Sentencia</i>) Entonces (<i>Sentencia</i> \Leftrightarrow <i>Sentencia</i>) sii, ssi y syss: si y solo si

¿La expresión **$((P \wedge Q) \Rightarrow R)$** hace parte del lenguaje de esta gramática ?

Prolog

Lógica proposicional – conceptos básicos

Proposiciones: Elementos de una frase que constituyen por sí solos una **unidad de comunicación de conocimientos** y pueden ser **considerados verdaderos o falsos**.

Proposición Simple: “Sebastián va a comprar unos tenis”.

Proposición Compuesta: “Sebastián comprará unos tenis, si y solo sí, Ana le paga el dinero que le debe”

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
falso	falso	verdadero	falso	falso	verdadero	verdadero
falso	verdadero	verdadero	falso	verdadero	verdadero	falso
verdadero	falso	falso	falso	verdadero	falso	falso
verdadero	verdadero	falso	verdadero	verdadero	verdadero	verdadero

Prolog

Lógica de primer orden

La lógica de primer orden, conocida como **lógica de predicados**, es un **sistema deductivo basado** en un **Lenguaje Lógico Matemático Formal**.

<i>Sentencia</i>	→	<i>Sentencia Atómica</i> (<i>Sentencia Conectiva Sentencia</i>) <i>Cuantificador Variable ... Sentencia</i> \neg <i>Sentencia</i>
<i>Sentencia Atómica</i>	→	<i>Predicado (Término...)</i> <i>Término = Término</i>
<i>Término</i>	→	<i>Función(Término)</i> <i>Constante</i> <i>Variable</i>
<i>Conectiva</i>	→	\wedge \vee \Rightarrow \Leftrightarrow
<i>Cuantificador</i>	→	\neg <i>Sentencia</i>
<i>Variable</i>	→	<i>a</i> <i>x</i> <i>s</i> ...
<i>Predicado</i>	→	<i>TieneColor</i> <i>EstáLloviendo</i> ...
<i>Función</i>	→	<i>Hombre</i> <i>Humano</i> <i>Mujer</i> ...

Permite Formalizar:

¿Qué se afirma?
(predicado o relación)
¿De quién se afirma?
(objeto)

**!!!Si se puede Formalizar
lo puedo programar!!!**

Prolog

Equivalencia en cláusulas de Horn

¿ Son equivalentes ?



$$(\neg p \vee \neg q \vee u) \leftrightarrow ((p \wedge q) \rightarrow u)$$

Probar con tablas de verdad ...

Tiempo estimado: **5 minutos**

Prolog

Equivalencia en cláusulas de Horn

									Equivalencia Clausula de Horn
p	q	u	$\neg p$	$\neg q$	$\neg p \vee \neg q$	$\neg p \vee \neg q \vee u$	$p \wedge q$	$(p \wedge q) \rightarrow u$	$(\neg p \vee \neg q \vee u) \leftrightarrow ((p \wedge q) \rightarrow u)$
v	v	v	f	f	f	v	v	v	v
v	v	f	f	f	f	f	v	f	v
v	f	v	f	v	v	v	f	v	v
v	f	f	f	v	v	v	f	v	v
f	v	v	v	f	v	v	f	v	v
f	v	f	v	f	v	v	f	v	v
f	f	v	v	v	v	v	f	v	v
f	f	f	v	v	v	v	f	v	v

↑
tautología

Prolog

Claúslas de Horn

Definición Formal: En lógica proposicional, una **fórmula lógica** es una **cláusula de Horn** si es una **cláusula** (disyunción de literales) con, como **máximo**, un literal **positivo**.

$$\neg p \vee \neg q \vee \dots \vee \neg t \vee u$$

Se puede reescribir como:

$$(p \wedge q \wedge \dots \wedge t) \rightarrow u$$

Inferencias:

antecedente \rightarrow consecuente

"Si es verdadero el antecedente, entonces es verdadero el consecuente"

Ejemplo:

Cláusula

$$(mujer(A) \wedge padre(B, A)) \rightarrow hija(A, B)$$

Verbal

"A es hija de B si A es mujer y B es padre de A"

Prolog

hija(A,B) :- mujer(A), padre(B,A)

Prolog

Claúulas

- `hijo(sebas,estefa). hijo(sebas,mario).`
- `hijo(juanita,estefa). hijo(juanita,mario).`
- `hijo(estefa,carlos). hijo(estefa,claudia).`
- `hombre(sebas). hombre(mario). hombre(carlos).`
- `mujer(estefa). mujer(juanita). mujer(cata).`
- $(\text{hijo}(X,Y) \wedge \text{hombre}(Y)) \rightarrow \text{padre}(Y,X)$
- $(\text{hijo}(X,Z) \wedge \text{mujer}(Z)) \rightarrow \text{madre}(Z,X)$
- $(\text{hombre}(X) \wedge \text{mujer}(Y)) \rightarrow \text{sexo_opuesto}(X,Y)$
- $(\text{hombre}(X) \wedge \text{mujer}(Y)) \rightarrow \text{sexo_opuesto}(Y,X)$
- $(\text{padre}(X,Y), \text{hijo}(Z,Y)) \rightarrow \text{abuelo}(X,Z)$
- $(\text{madre}(Y,R), \text{hijo}(Z,R)) \rightarrow \text{abuela}(Y,Z)$



Prolog

Claúsulas en Prolog

- `hijo(sebas,estefa). hijo(sebas,mario). ?- abuelo(X,juanita).`
- `hijo(juanita,estefa). hijo(juanita,mario).`
- `hijo(estefa,carlos). hijo(estefa,claudia). ?- abuelo(carlos,sebas).`
- `hombre(sebas). hombre(mario). hombre(carlos).`
- `mujer(estefa). mujer(juanita). mujer(cata).`
- `padre(P,X) :- hijo(X,P), hombre(P).`
- `madre(M,X) :- hijo(X,M), mujer(M).`
- `sexo_opuesto(X,Y) :- hombre(X), mujer(Y).`
- `sexo_opuesto(Y,X) :- hombre(X), mujer(Y).`
- `abuelo(X,Z) :- padre(X,Y), hijo(Z,Y).`
- `abuela(Y,Z) :- madre(Y,R), hijo(Z,R).`
- `hermanos(X, Y) :- hijo(X, Z), hijo(Y, Z), X\=Y.`



Prolog

Claúsulas en Prolog



Prolog es un lenguaje lógico **adecuado** para **programas** que implican **cálculos simbólicos o no numéricos**.

- Su nombre proviene del francés **“PROgrammation en LOGique”**, creado por **Alain COLMERAUER** de la Universidad de **Aix-Marseille (Marsella, Francia)**.
- Tras una **estancia científica en Montreal**, trabajó sobre los sistemas de traducción automática (**NLP – Natural Language Processing**), en especial del **inglés al francés**. Inventó el llamado **sistema Q**, primer paso hacia el nacimiento del lenguaje Prolog.
- Asesor principal del proyecto japonés para equipos computacionales de 5ª Generación.



Alain COLMERAUER

Prolog

Claúsulas en Prolog



Prolog es un lenguaje lógico **adecuado** para **programas** que implican **cálculos simbólicos o no numéricos**.

- Es un lenguaje de uso frecuente en **Inteligencia Artificial**, donde la manipulación de símbolos e inferencia sobre ellos es una tarea común.
- Un **programa** en Prolog **consiste** en una **serie de hechos, reglas** (Claúsulas de Horn) y **consultas** (objetivos).
- Un **programa** se **ejecuta verificando** algunas **consultas** y **validando** si estas pueden ser **probadas** al utilizar los hechos conocidos e **infiriendo** a partir de las **reglas existentes**.

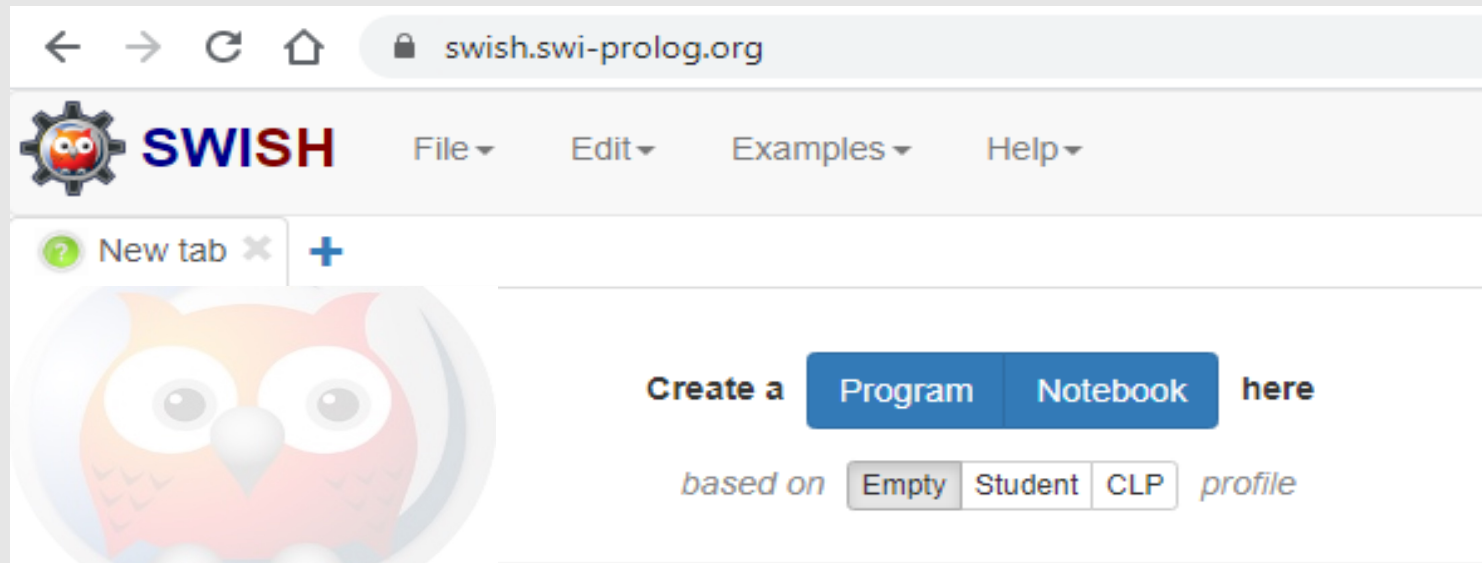


Alain COLMERAUER

Prolog

Herramientas

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



2. Download swi-prolog

3. <https://ideone.com>

Prolog

Hechos simples



- Podemos hacer algunas **declaraciones** usando **hechos**.
- Los **hechos** consisten en un **aspecto particular de algo** (evidencia) o una **relación** entre los aspectos.
- Para **describir un hecho** es suficiente con **escribir una palabra**, por ejemplo: llueve.
- Ahora podemos hacer una **consulta** de Prolog preguntando
?- llueve.

Prolog

Hechos simples – sintaxis Prolog

- Los hechos siempre **deben comenzar** con una **letra minúscula** y **terminar** con un **punto**.

maria_toca_piano.



- Los hechos en sí pueden consistir en cualquier combinación de letra o número, así como el **carácter de subrayado** `_`.
- Sin embargo, se deben **evitar los nombres que contienen los caracteres** `-`, `+`, `*`, `/` u otros operadores matemáticos.
- Los **comentarios** van entre `/* */`

Prolog

Declaración de hechos simples

The screenshot shows the SWISH Prolog environment. The code editor on the left contains the following Prolog facts:

```
1 cielo_es_azul.      /* El cielo es de color azul */
2 azul_es_color.      /* El azul es un color */
3
4 luisa_tiene_frio.    /* Luisa Tiene Frio */
5 llueve.              /* Está lloviendo */
6
7 luis_compro_cerveza. /* Luis Compró Cerveza */
8 mari_es_cantante.    /* Mari es cantante */
9
10 luis_gano_1000_apostando. /* Luis gana 1000 apostando */
11
12
13
```

The query window on the right shows the execution of the following queries:

- `llueve.` returns `true`.
- `soleado.` returns the error `procedure `soleado` does not exist`.

Red arrows and circles highlight the relationship between the `llueve.` fact in the code and its successful execution in the query window. A blue arrow and circle highlight the error for the `soleado.` query, which is not defined in the provided code.

Prolog

Declaración de hechos simples - constantes

The screenshot displays the SWISH Prolog environment. The main editor window shows a Prolog program with the following facts:

```
1 'Cartagena'.  
2 capital('Bolívar', 'Cartagena').  
3 numero(menos_uno, -1).  
4 numero(uno, 1).  
5 numero(dos, 2).  
6
```

Below the editor, three separate execution windows are shown, each with a query and its result:

- Query: `numero('uno', X).`
Result: `X = 1`
- Query: `numero(menos_uno, X).`
Result: `X = -1`
- Query: `capital('Bolívar', 'Cartagena').`
Result: `true`

Below these, a fourth execution window shows a query with a variable:

- Query: `?- \+ capital('Bolívar', 'Bogotá').`
Result: `true`

Prolog

Declaración de hechos simples - constantes

- La **solución de un problema** se puede **realizar** con un nivel de **abstracción** considerablemente **alto**, **sin entrar en detalles** de implementación irrelevantes.
- Las **soluciones son más fáciles** de **entender** por las **personas** y así hay una generación rápida de prototipos e ideas complejas.
- La **resolución** de problemas complejos es **resuelta por el intérprete** a partir de la **declaración** de las condiciones **dadas**.
- Potencia de **inferencia**.
- **Simplicidad**.
- **Sencillez** en la **implementación** de **estructuras complejas**.
- Puede **mejorarse la eficiencia** modificando el componente de control sin tener que **modificar** la **lógica** del algoritmo.

Prolog

Cláusulas (lógica proposicional)

antecedente \rightarrow consecuente

"Si es verdadero el antecedente, entonces es verdadero el consecuente"

Ejemplo:

Cláusula

$(mujer(A) \wedge padre(B, A)) \rightarrow hija(A, B)$

Prolog

hija(A, B) :- mujer(A), padre(B, A)

Verbal

"A es hija de B si A es mujer y B es padre de A"

Claúsulas

$(hijo(X, Y) \wedge hombre(Y)) \rightarrow padre(Y, X)$
 $(hijo(X, Z) \wedge mujer(Z)) \rightarrow madre(Z, X)$
 $(padre(X, Y) \wedge hijo(Z, Y)) \rightarrow abuelo(X, Z)$
 $(madre(Y, R) \wedge hijo(Z, R)) \rightarrow abuela(Y, Z)$

↑
antecedentes

↑
consecuente

Reglas en Prolog

$padre(P, X) :- hijo(X, P), hombre(P).$
 $madre(M, X) :- hijo(X, M), mujer(M).$
 $abuelo(X, Z) :- padre(X, Y), hijo(Z, Y).$
 $abuela(Y, Z) :- madre(Y, R), hijo(Z, R).$

↑
consecuente

↑
antecedentes

Prolog

Predicados

Los **predicados** se utilizan para **expresar propiedades** de los **objetos**, predicados **monádicos**, y relaciones entre ellos, predicados **poliádicos**.

En **Prolog** los llamaremos **hechos**. Debemos tener en cuenta que:

- Los nombres de todos los **objetos y relaciones** deben comenzar con una letra **minúscula**.
- **Primero se escribe la relación o propiedad:** predicado
- Los **objetos** se escriben **separándolos** mediante **comas** y encerrados entre **paréntesis**: argumentos.
- Al final del hecho debe ir un punto (".").

Ejemplos:

Monádicos

- mujer(clara). mujer(chelo).
- capital('Bolívar','Cartagena'). capital('Antioquia','Medellín').
- le_gusta_a(jorge,informatica). regala(jorge, flores, clara).

Poliádicos

Prolog

Términos (constantes o variables)

Los **términos** pueden ser **constantes o variables**, y toman **valores** en un **dominio no vacío** (Universo del Discurso). Para saber **cuántos individuos** del universo **cumplen** una determinada **propiedad o relación**, **cuantificamos** los términos.

Las **constantes** se utilizan para **dar nombre** a **objetos** concretos del dominio, o sea, **representan individuos conocidos** de nuestro **Universo**.

Hay dos tipos de **constantes**: **átomos** y **números**.

- **Átomos**: existen tres clases de constantes atómicas:
 - **Cadenas de letras, dígitos y subrayado** () empezando por letra minúscula.
 - Cualquier **cadena de caracteres encerrada entre comillas** simples (').
 - **Combinaciones** especiales de **signos**: "?-", ":-", ...

Ejm. llueve. a10. constante_x10. 'Cartagena'. 'Antioquia'.

No válidos: 2carros. Universidad. _x10. julian-zapata.

Prolog

Términos (constantes o variables)

Constantes (átomos y números)


- **Números:** se utilizan para **representar números** de forma que se puedan realizar **operaciones aritméticas**.

Enteros: en la implementación de Prolog puede utilizarse cualquier entero en intervalos muy grandes.

Ejm. `integer(-999999999999999999).` 

- Reales: decimales con coma flotante, consistentes en al menos un dígito, opcionalmente un punto decimal y más dígitos, u opcionalmente E, un «+» o «-» y más dígitos.

Ejm. `float(+20E20).` `float(-999999999999999999.9).`
`float(-1.2E6).`



No válidos:

`float(+1.2+3)`

```
?- float(.9)
      ^
Syntax error: Operator expected
```

Prolog

Términos (constantes o variables)

Las **variables** se utilizan para representar objetos del Universo u **objetos desconocidos** en ese momento, es decir, son las **incógnitas del problema**.

Se **diferencian de los átomos** en que empiezan siempre con una **letra mayúscula o con el signo de subrayado (_)**.

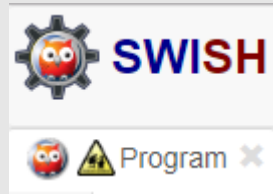
Cualquier identificador que empiece por mayúscula, será tomado por Prolog como una variable.

Ejm. X. Y10. _Cartagena. _variable. _1. _2.

Prolog

Predicados predefinidos

true
fail
var(X)
nonvar(X)
atom(X), atomic(X)
integer(X), float(X)



<code>integer(cuatro)</code>
false
<code>integer(28)</code>
true
<code>integer(2,5)</code>
procedure `integer(A,B)` does not exist
<code>integer(2.5)</code>
false

<code>true</code>
true
<code>fail</code>
false
<code>false</code>
false

<code>atom(z).</code>
true
<code>integer(1)</code>
true
<code>integer(casa)</code>
false
<code>atom(1)</code>
false

<code>var(Y)</code>
true
<code>var('Medellín')</code>
false
<code>var(casa)</code>
false
<code>nonvar(casa)</code>
true
<code>nonvar('Hola')</code>
true
<code>nonvar(Z)</code>
false

<code>float(-9999999999999999.9)</code>	<code>float(-1.2E6).</code>
true	true

<code>atomic(Z)</code>
false
<code>atomic('San Andrés')</code>
true
<code>atomic(IVA)</code>
false
<code>atomic(iva)</code>
true
<code>atomic(23)</code>
true
<code>atomic(maria)</code>
true

Prolog

Conceptos

Figure 3.10. A review of Prolog programs and queries

Here are the pieces that make up Prolog programs and queries:

- A *constant* is either a string of characters enclosed within single quotes or a lowercase letter optionally followed by letters, digits, and underscores.
- A *variable* is an uppercase letter or an underscore optionally followed by letters, digits, and underscores.
- A *number* is a sequence of one or more digits optionally preceded by a minus sign and optionally containing a decimal point.
- A *term* is a constant, variable, or number.
- A *predicate* is written as a constant.
- An *atom* is a predicate optionally followed by terms (called the *arguments* of the predicate) enclosed within parentheses and separated by commas.
- An *equality* is two terms separated by the = symbol.
- A *literal* is an atom or an equality optionally preceded by the \+ symbol.
- A *query* is a sequence of one or more literals separated by commas and terminated with a period.
- A *clause* is an atom (called the *head* of the clause) followed by a period or by the :- symbol and then a query (called the *body* of the clause).
- A *program* is a sequence of one or more clauses.

Prolog

Conceptos

1. Un término es una constante, una variable, un número
llueve, hace_sol, jorge_3, 'Medellín' ...
X, Y, Actor, Película,
2, 876, -1
1. Un predicado `hijo(manuel, mario)`
2. Un átomo `atom('Cartagena')`
3. Una igualdad de términos `R == R`
4. Un literal `llueve. \+ capital('Antioquia', 'Bogotá').`
5. Una consulta (query) `literal_1, literal_2, literal_3. hijo(manuel, P), hombre(P).`
6. Una clausula `padre(P, X) :- hijo(X, P), hombre(P). llueve :- true ~ llueve.`
7. Un programa `conjunto de cláusulas.`



Prolog

Unificación de variables en reglas

Supongamos que tenemos el siguiente hecho:

escucha(luisa, rock).

Podríamos pensar en consultar ¿Qué escucha luisa?, así:

escucha(luisa, what).

Desafortunadamente no se puede, porque “what” no coincidirá con “rock”

Para hacer coincidir o mapear los argumentos de esta manera, **debemos usar una Variable.**

→ El proceso de emparejar elementos con variables se conoce como **unificación.**

Prolog

Unificación de variables en reglas

escucha(luisa, romantica).

escucha(luisa, salsa).

escucha(luisa, merengue).

escucha(pepe, rock).

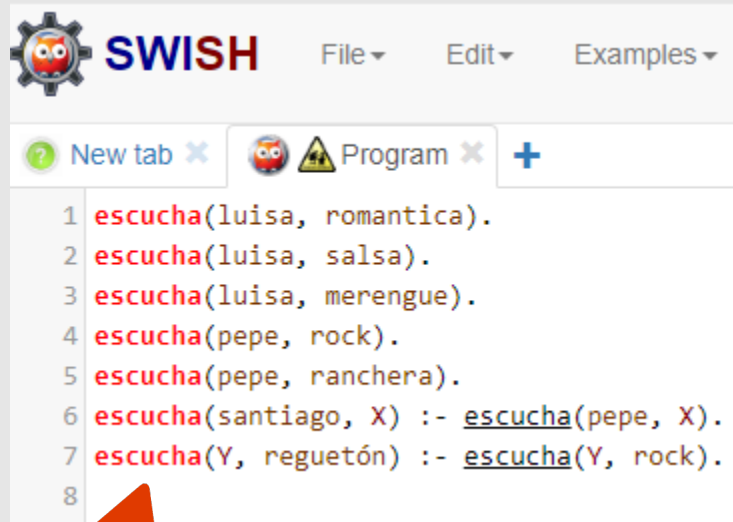
escucha(pepe, ranchera).



- ¿Qué escucha santiago?
- ¿Quién escucha reguetón?
- Todo el que escucha rock escucha reggaetón
- santiago escucha todo lo que escuche pepe

Prolog


Unificación de variables en reglas



The image shows the SWISH Prolog editor interface. It has a menu bar with 'File', 'Edit', and 'Examples'. Below the menu bar, there are tabs for 'New tab' and 'Program'. The main area contains a list of Prolog rules:

```
1 escucha(luisa, romantica).  
2 escucha(luisa, salsa).  
3 escucha(luisa, merengue).  
4 escucha(pepe, rock).  
5 escucha(pepe, ranchera).  
6 escucha(santiago, X) :- escucha(pepe, X).  
7 escucha(Y, reguetón) :- escucha(Y, rock).  
8
```

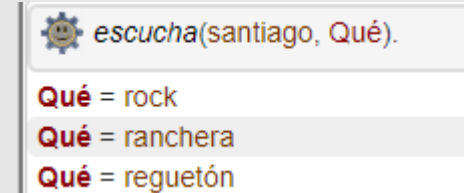
An orange arrow points from the bottom left towards the code area.



The image shows the SWISH Prolog editor interface. It has a menu bar with 'File', 'Edit', and 'Examples'. Below the menu bar, there are tabs for 'New tab' and 'Program'. The main area contains a list of Prolog rules:

```
1 escucha(luisa, romantica).  
2 escucha(luisa, salsa).  
3 escucha(luisa, merengue).  
4 escucha(pepe, rock).  
5 escucha(pepe, ranchera).  
6 escucha(santiago, X) :- escucha(pepe, X).  
7 escucha(Y, reguetón) :- escucha(Y, rock).  
8
```

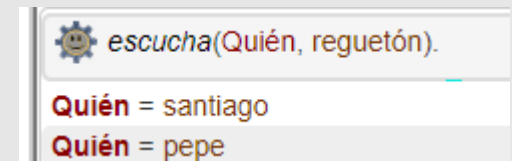
An orange arrow points from the bottom left towards the code area.



The image shows the SWISH Prolog editor interface. It has a menu bar with 'File', 'Edit', and 'Examples'. Below the menu bar, there are tabs for 'New tab' and 'Program'. The main area contains a list of Prolog rules:

```
1 escucha(luisa, romantica).  
2 escucha(luisa, salsa).  
3 escucha(luisa, merengue).  
4 escucha(pepe, rock).  
5 escucha(pepe, ranchera).  
6 escucha(santiago, X) :- escucha(pepe, X).  
7 escucha(Y, reguetón) :- escucha(Y, rock).  
8
```

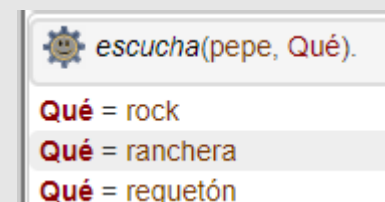
An orange arrow points from the bottom left towards the code area.



The image shows the SWISH Prolog editor interface. It has a menu bar with 'File', 'Edit', and 'Examples'. Below the menu bar, there are tabs for 'New tab' and 'Program'. The main area contains a list of Prolog rules:

```
1 escucha(luisa, romantica).  
2 escucha(luisa, salsa).  
3 escucha(luisa, merengue).  
4 escucha(pepe, rock).  
5 escucha(pepe, ranchera).  
6 escucha(santiago, X) :- escucha(pepe, X).  
7 escucha(Y, reguetón) :- escucha(Y, rock).  
8
```

An orange arrow points from the bottom left towards the code area.



The image shows the SWISH Prolog editor interface. It has a menu bar with 'File', 'Edit', and 'Examples'. Below the menu bar, there are tabs for 'New tab' and 'Program'. The main area contains a list of Prolog rules:

```
1 escucha(luisa, romantica).  
2 escucha(luisa, salsa).  
3 escucha(luisa, merengue).  
4 escucha(pepe, rock).  
5 escucha(pepe, ranchera).  
6 escucha(santiago, X) :- escucha(pepe, X).  
7 escucha(Y, reguetón) :- escucha(Y, rock).  
8
```

An orange arrow points from the bottom left towards the code area.

Unificación de variables en reglas.

Generalidades

Referencias

- Bratko, Ivan. Prolog Programming for Artificial Intelligence (4th Edition) (International Computer Science Series) Aug 31, 2011
- Programming in Prolog: Using the ISO Standard Oct 4, 2013 by William F. Clocksin and Christopher S. Mellish
- Thinking as Computation (The MIT Press), Jan 6, 2012. by Hector J. Levesque
- Clause and Effect: Prolog Programming for the Working Programmer Apr 29, 2003. by William F. Clocksin
- Programming in Haskell Sep 12, 2016 by Graham Hutton
- Learn You a Haskell for Great Good!: A Beginner's Guide Apr 15, 2011 by Miran Lipovaca
- Learning Haskell Data Analysis, May 28, 2015 by James Church
- Advanced Computer Programming in Python, Advanced Computer Programming in Python Mar 22, 2017, by Karim Pichara and Christian Pieringer
- Functional Python Programming - Create Succinct and Expressive Implementations with Python Jan 31, 2015 by Steven Lott
- Functional Python Programming: Discover the power of functional programming, generator functions, lazy evaluation, the built-in itertools library, and monads, 2nd Edition Apr 13, 2018 by Steven F. Lott
- Building Web Applications with Python and Neo4j, Jul 16, 2015
- Learning Neo4j 3.x - Second Edition: Effective data modeling, performance tuning and data visualization techniques in Neo4j, Oct 20, 2017 by Jerome Baton and Rik Van Bruggen

Gracias 

Universidad Nacional de Colombia

PROYECTO **CULTURAL, CIENTÍFICO Y COLECTIVO** DE NACIÓN