



PROGRAMACIÓN LÓGICA

José Fernando Morantes Florez
Harold Nicolas Saavedra Alvarado
Juan Camilo Monterrosa Sanchez

**CLASIFICACIÓN DE
PARADIGMAS DE
PROGRAMACIÓN**

01

**PARADIGMA DE
PROGRAMACION
LOGICA**

02

**CONCEPTOS
CLAVES**

03

TABLA DE CONTENIDO

04

**VENTAJAS Y
DESVENTAJAS**

05

**LENGUAJES DE
PROGRAMACIÓN
LÓGICA**

06

**APLICACIONES DE LA
PROGRAMACIÓN
LÓGICA**



01

**CLASIFICACIÓN DE
PARADIGMAS DE
PROGRAMACIÓN**

CLASIFICACIÓN

LENGUAJES DE PROGRAMACIÓN

IMPERATIVOS

¿Cómo se debe llegar?
Usualmente usado en la creación de algoritmos

DECLARATIVOS

¿A qué cosa debe llegar?
más cercanos estos al razonamiento humano
Utilizan inferencia

CLASIFICACIÓN

LENGUAJES DE PROGRAMACIÓN

IMPERATIVOS

- Orientado a Objetos
- Por Procedimientos
- Procesamiento en Paralelo

DECLARATIVOS

- Funcionales
- Logico
- Relacionales



02

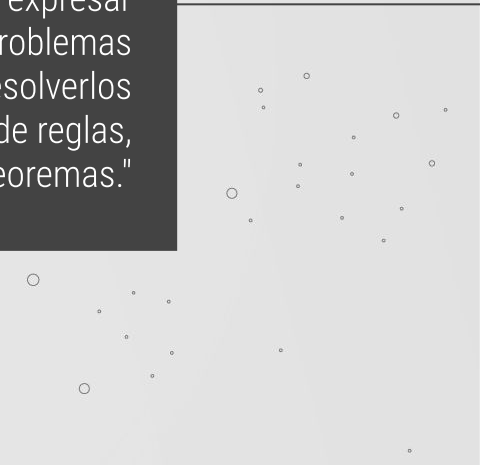
**PARADIGMA DE
PROGRAMACION LOGICA**



FILOSOFIA DEL PARADIGMA

"La programación lógica permite modelar problemas por medio de la abstracción, utilizando un sistema de lógica formal con la cual llegar a una conclusión por medio de hechos y reglas."

"la lógica es la manera más sencilla, para el intelecto humano, de expresar formalmente problemas complejos y de resolverlos mediante la aplicación de reglas, hipótesis y teoremas."





¿QUÉ TRATA DE RESOLVER?

Dado un problema S , saber si la proposición lógica A es solución o no de dicho problema o en qué casos lo es.



ALGORITMO



LÓGICA

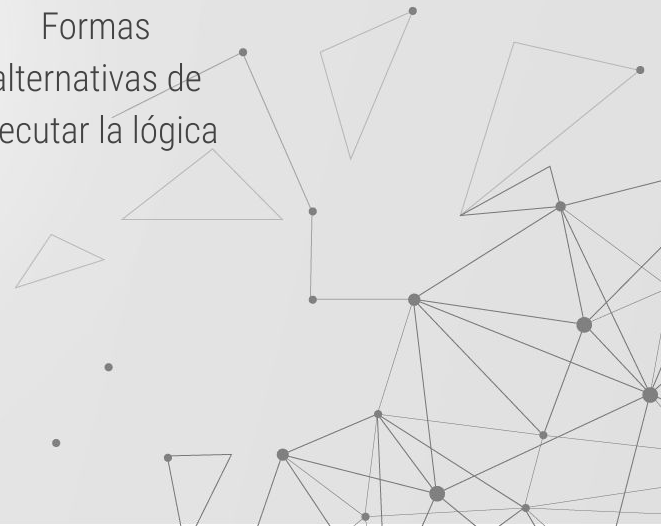


CONTROL

Programa bajo el
paradigma de
programación
lógica

Determina las
soluciones
producidas

Formas
alternativas de
ejecutar la lógica



CARACTERÍSTICAS DEL PARADIGMA

Unificación de términos.

Mecanismos de inferencia automática.

Recursión como estructura de control básica.

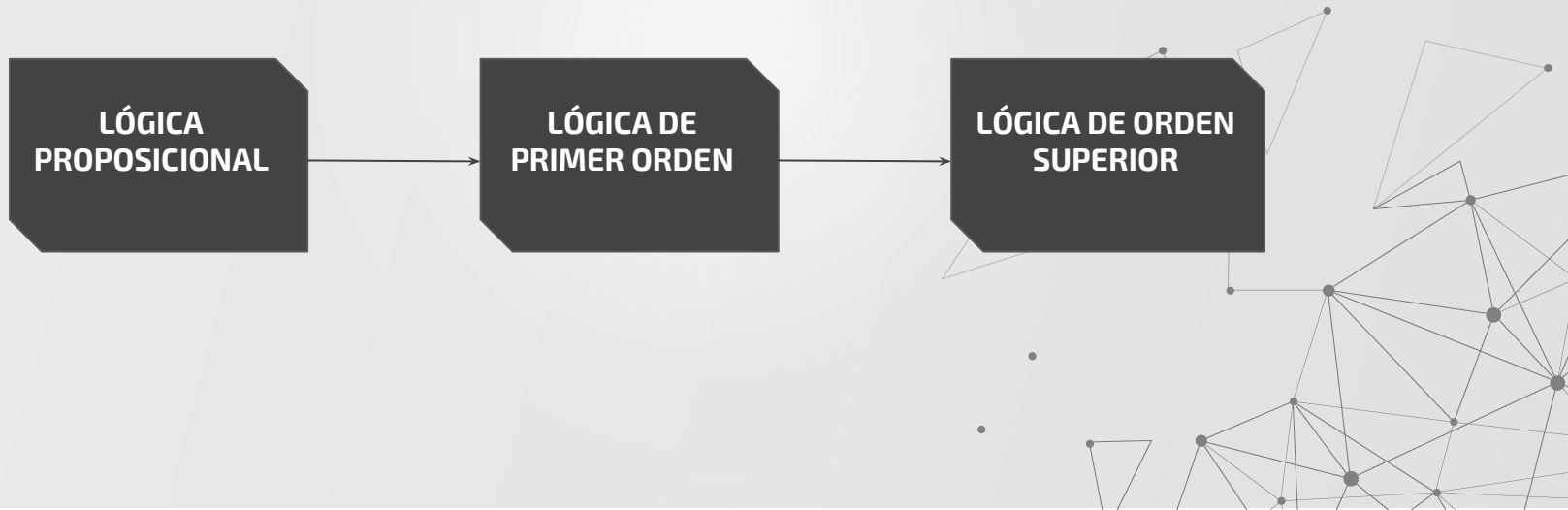
Visión lógica (matemática) de la computación.



03

CONCEPTOS CLAVES

La mayoría de los lenguajes de programación lógica se basan en la teoría lógica de primer orden, aunque también incorporan algunos comportamientos de orden superior.



CONCEPTOS DE LÓGICA



PROPOSICIÓN

Sentencia lógica que puede ser verdadera o falsa; atómica o compuesta.

Proposiciones, relaciones entre estas, inferencia.

LÓGICA SIMBÓLICA



CÁLCULO DE PREDICADOS

Tipo de lógica simbólica usado en programación lógica.

LÓGICA PROPOSICIONAL

<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>)
		(<i>Sentencia</i> \vee <i>Sentencia</i>)
		(<i>Sentencia</i> \Rightarrow <i>Sentencia</i>)
		(<i>Sentencia</i> \Leftrightarrow <i>Sentencia</i>)

EJEMPLOS

$\text{True} \wedge \text{False} = \text{False}$

$2 + 3 \Rightarrow 5$

S

$(P \wedge Q) \Rightarrow \neg R$

EJEMPLOS



$\text{True} \wedge \text{False} = \text{False}$



$2 + 3 \Rightarrow 5$



S



$(P \wedge Q) \Rightarrow \neg R$



REGLAS

Conjunto de operaciones que se pueden aplicar a los hechos para sacar un resultado lógico.

Conjunto de datos que conoce el sistema a priori(o que va adquiriendo a lo largo de su ejecución)

HECHO



BASE DE CONOCIMIENTO

Combinación entre hechos y reglas; es todo lo que ya estamos seguros de saber.

LOGICA DE PRIMER ORDEN



Características

- Más expresiva de la Lógica proposicional.
- Incluye proposiciones lógicas, predicados y cuantificadores.

Es un sistema formal diseñado para estudiar la inferencia en lenguajes de primer orden

¿Qué son?



Utilidad

La lógica de primer orden tiene el poder expresivo suficiente para definir a prácticamente todas las matemáticas.

LOGICA DE PRIMER ORDEN

<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> ...

CUANTIFICADORES

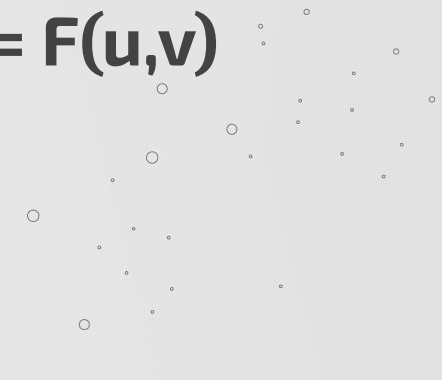
SIMBOLO	NOMBRE	SIGNIFICADO
\forall	Cuantificador Universal	Para todo
\exists	Cuantificador Existencial	Existe un
$\exists!$	Cuantificador Existencial Unico	Existe exactamente un



PREDICADOS

Funciones sobre objetos que se usan para expresar propiedades o relaciones entre éstos.

$$\mathbf{F: X \rightarrow Y}$$
$$\mathbf{x \rightarrow y = F(x)}$$

$$\mathbf{F: U \times V \rightarrow Y}$$
$$\mathbf{u, v \rightarrow y = F(u, v)}$$


LÓGICA DE ORDEN SUPERIOR

Es una extensión de la lógica de primer orden, en la que se permite reducir conjuntos (como podrían ser proposiciones, relaciones o funciones) a una variable sobre la cual se pueden expresar nuevas proposiciones o hacer uso de los cuantificadores. Así se expande el poder expresivo del lenguaje sin tener que agregar nuevos símbolos lógicos.

$$\forall P \forall x (Px \vee \neg Px)$$

Todo individuo o tiene una propiedad o no la tiene



CLÁUSULAS DE HORN

Secuencia de literales (disyunción de literales) que contiene a lo sumo uno de sus literales positivos .

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

$$(p \wedge q \wedge r \dots) \Rightarrow u$$

Reescritura equivalente como implicación





CLÁUSULA 'DEFINITE'

Cláusula de Horn con exactamente un literal positivo.

Cláusula 'definite' sin literales negativos.

HECHO



CLÁUSULA OBJETIVO

Sin ningún literal positivo (consulta).

EJEMPLO CLÁUSULAS DE HORN

“A es hija de B si A es mujer y B es padre de A”

$$\neg \text{mujer}(A) \vee \neg \text{padre}(B, A) \vee \text{hija}(A, B)$$
$$(\text{mujer}(A) \wedge \text{padre}(B, A)) \Rightarrow \text{hija}(A, B)$$

Reescritura equivalente como implicación

En PROLOG:


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

UNIFICACIÓN Y BACKTRACKING

Los programas en Prolog se componen de **cláusulas de Horn** que constituyen reglas del tipo "modus ponendo ponens", es decir, "**Si es verdad el antecedente, entonces es verdad el consecuente**".

$$\begin{array}{l} p \\ p \rightarrow q \\ \hline \therefore q \end{array}$$





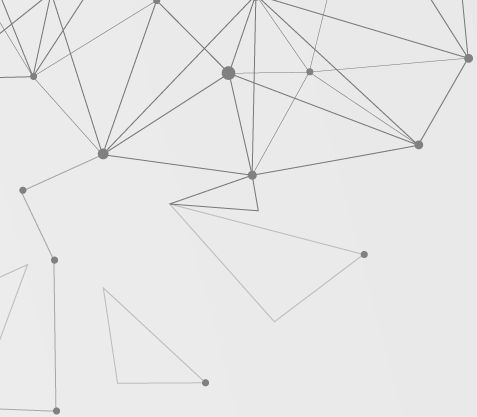
A partir de un **punto de elección** se determina un **subconjunto de cláusulas susceptibles de ser ejecutadas, estas a su vez son puntos de elección.**

UNIFICACIÓN

Si el objetivo de la **unificación** es falso se **deshace todo lo ejecutado** hasta antes de llegar al punto de elección para empezar un nuevo proceso”

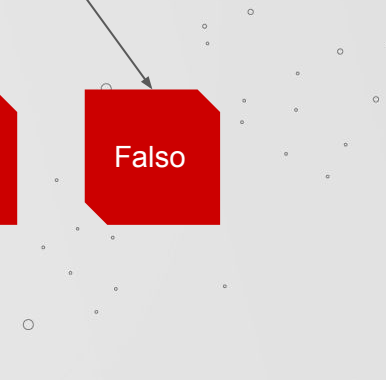
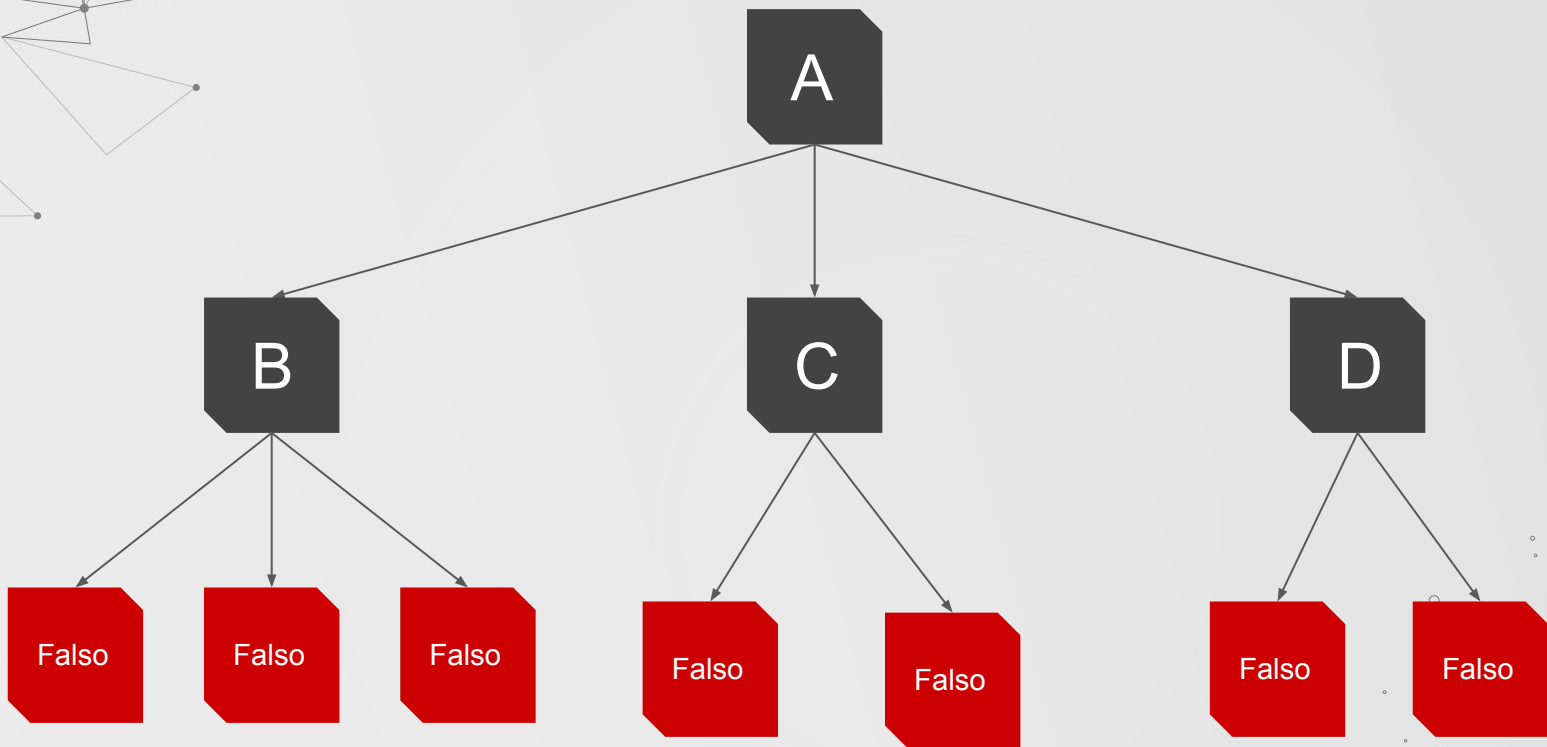
BACKTRACKING

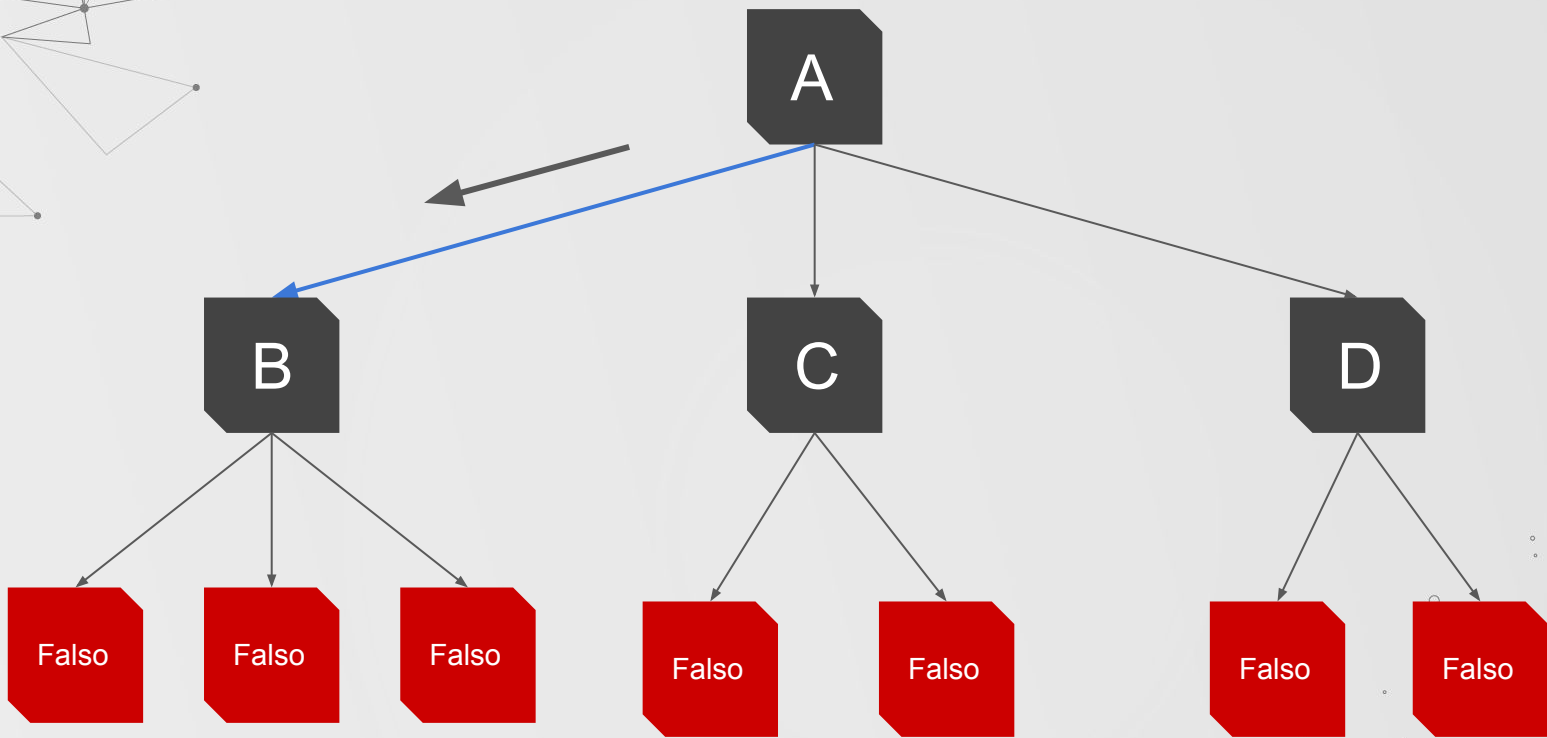


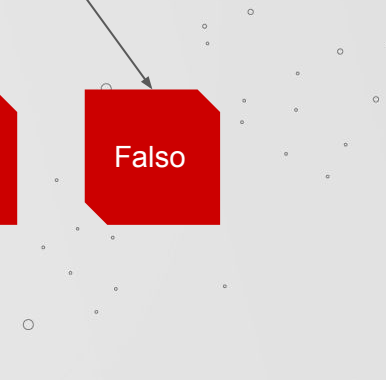
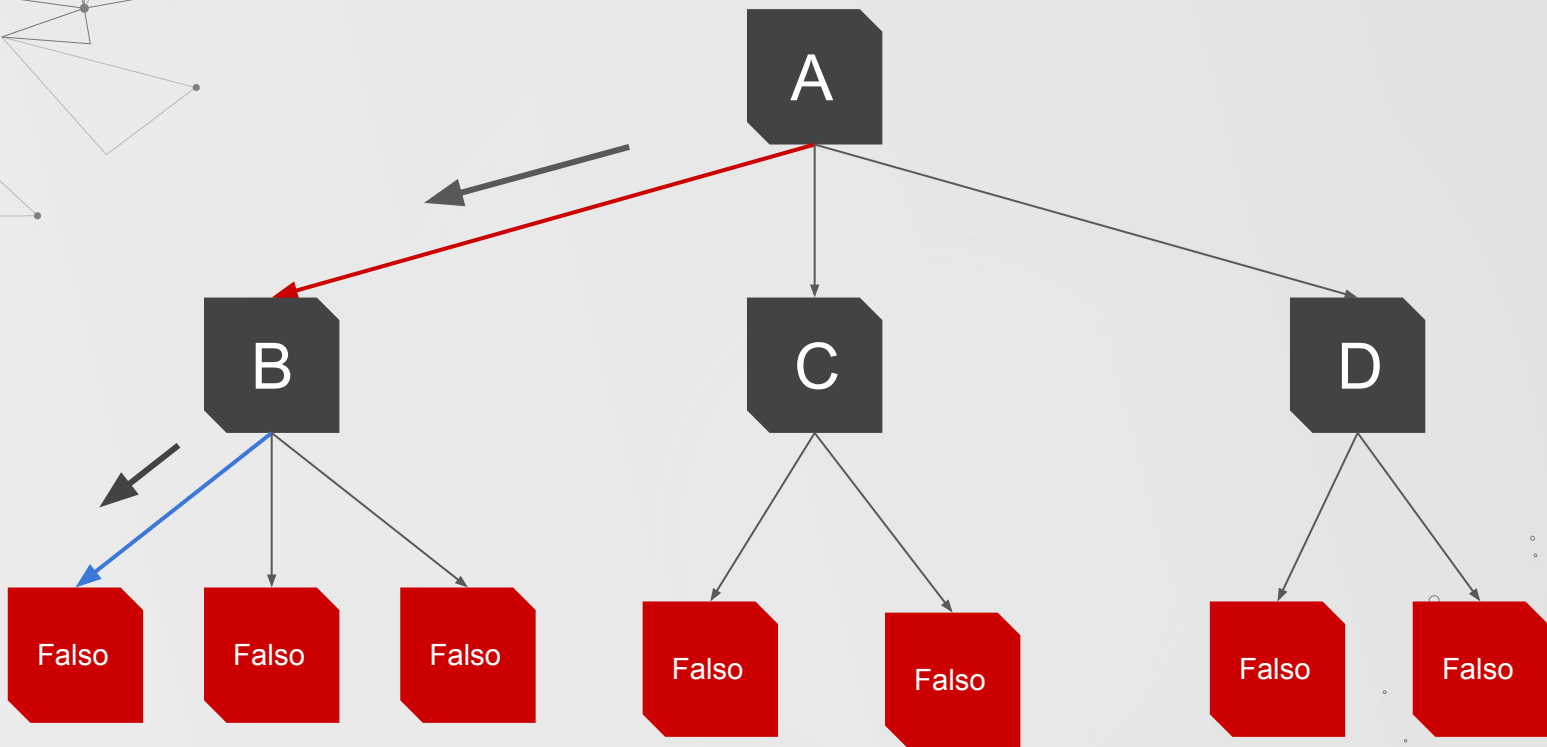


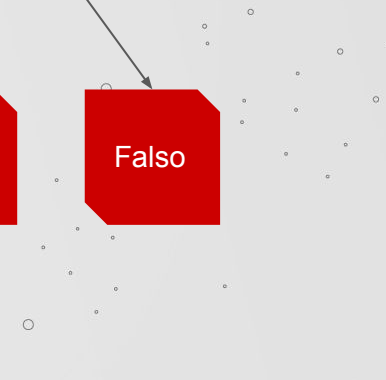
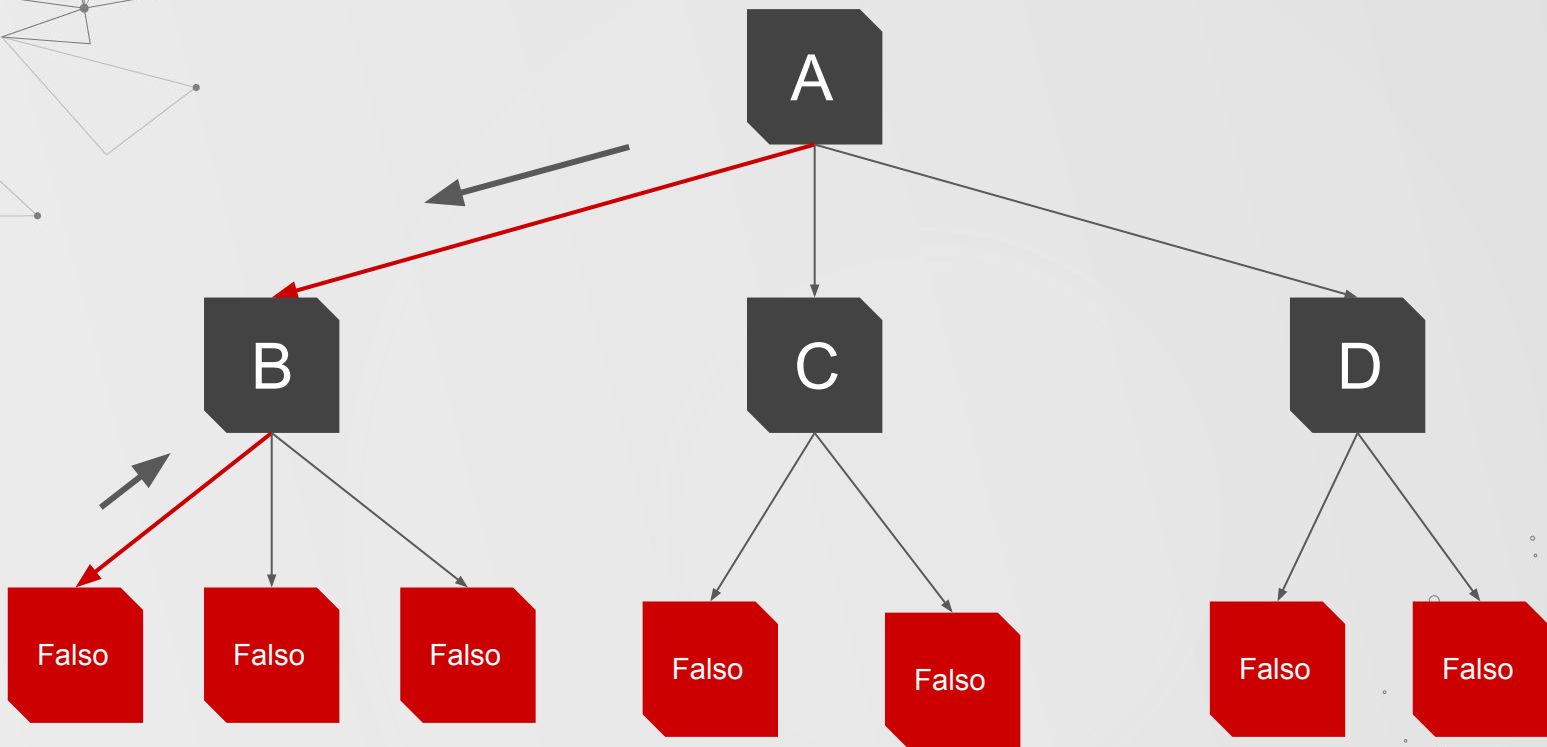
B V C V D →
A

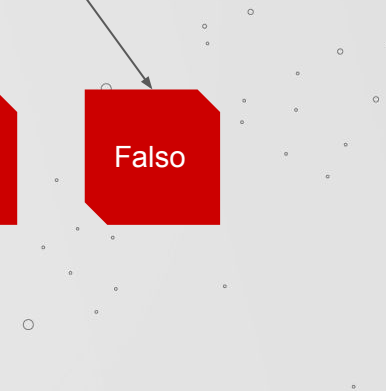
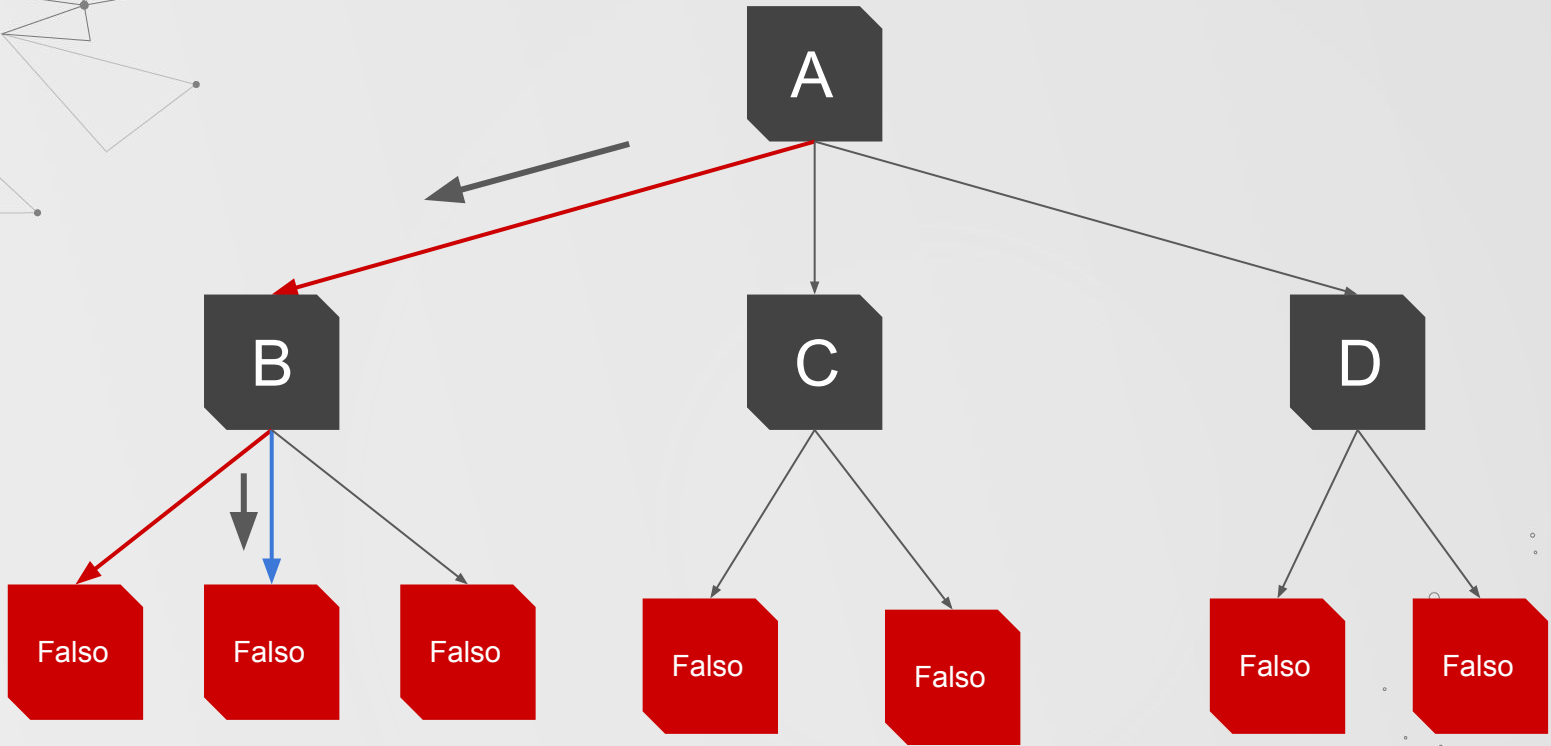
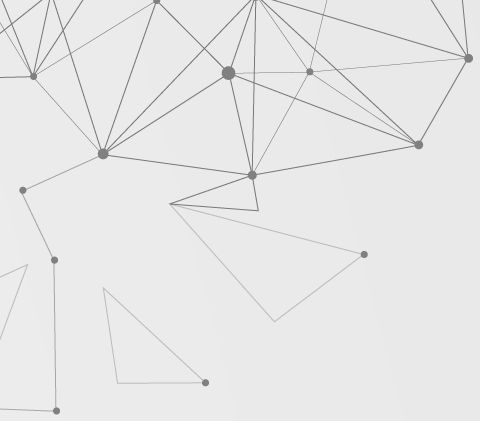


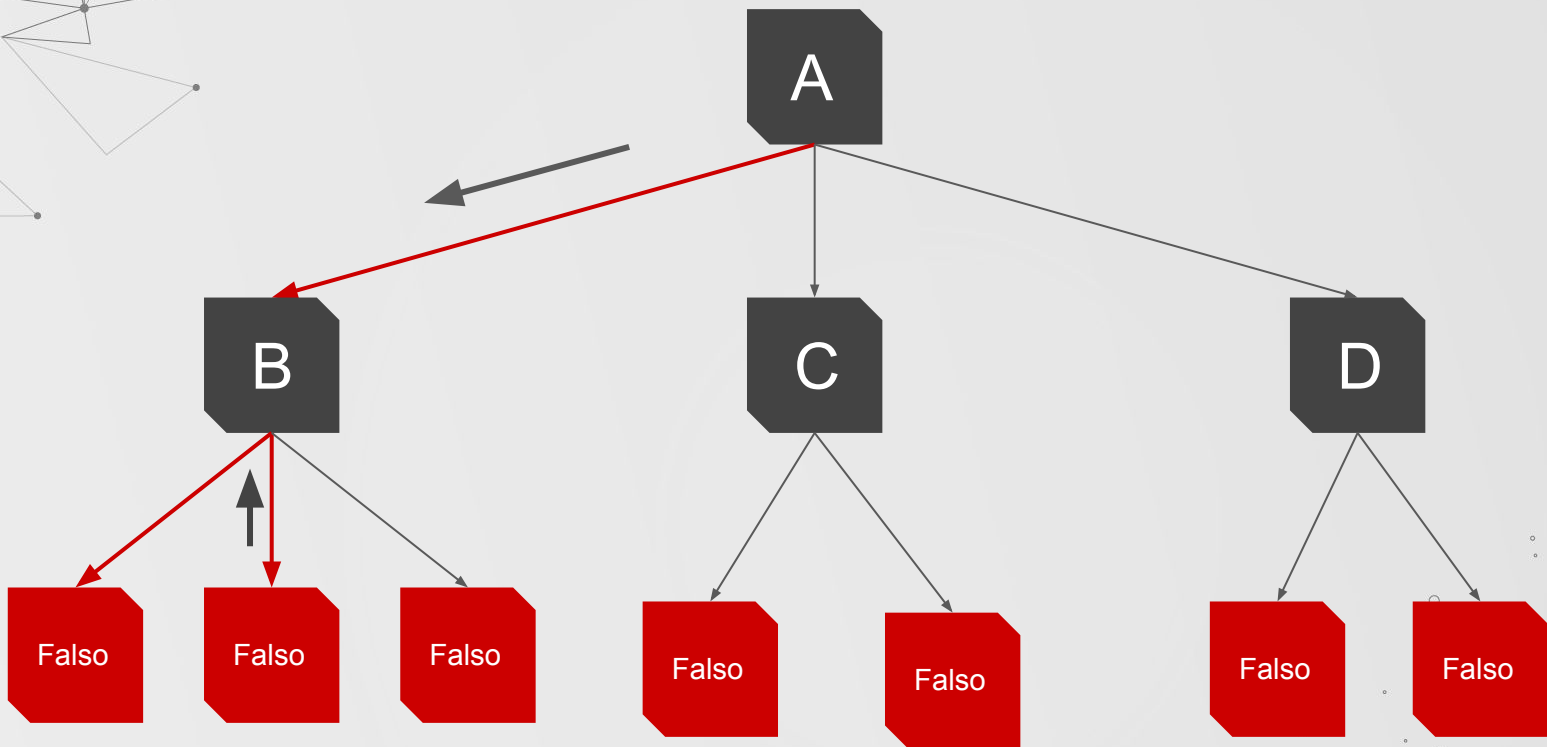


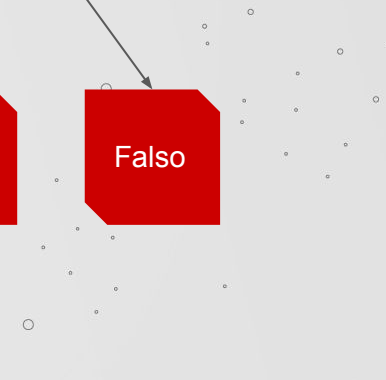
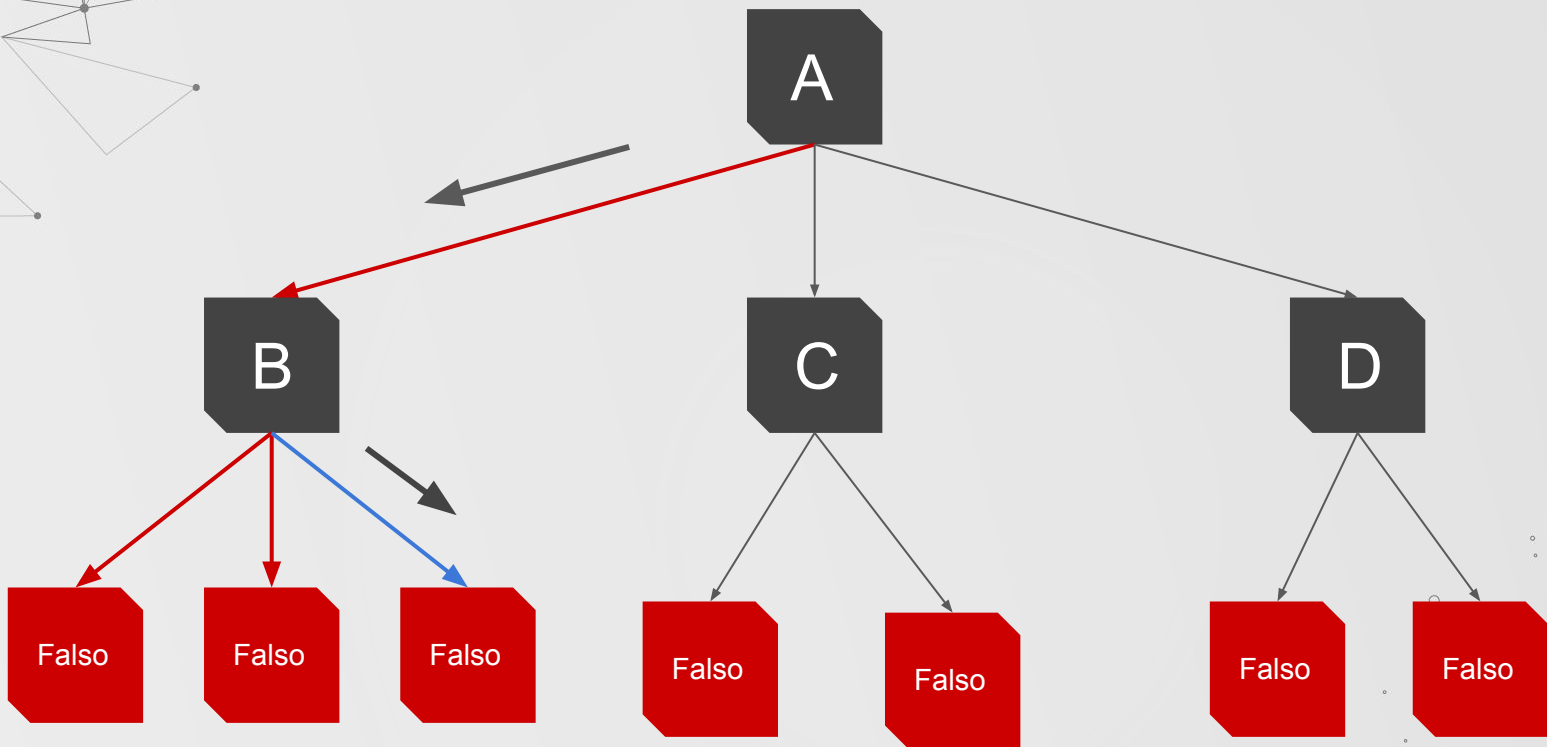
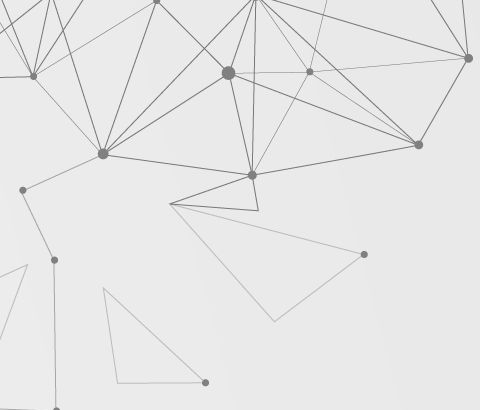


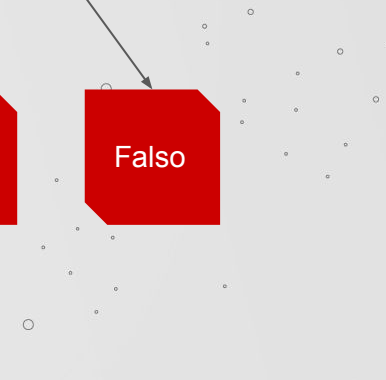
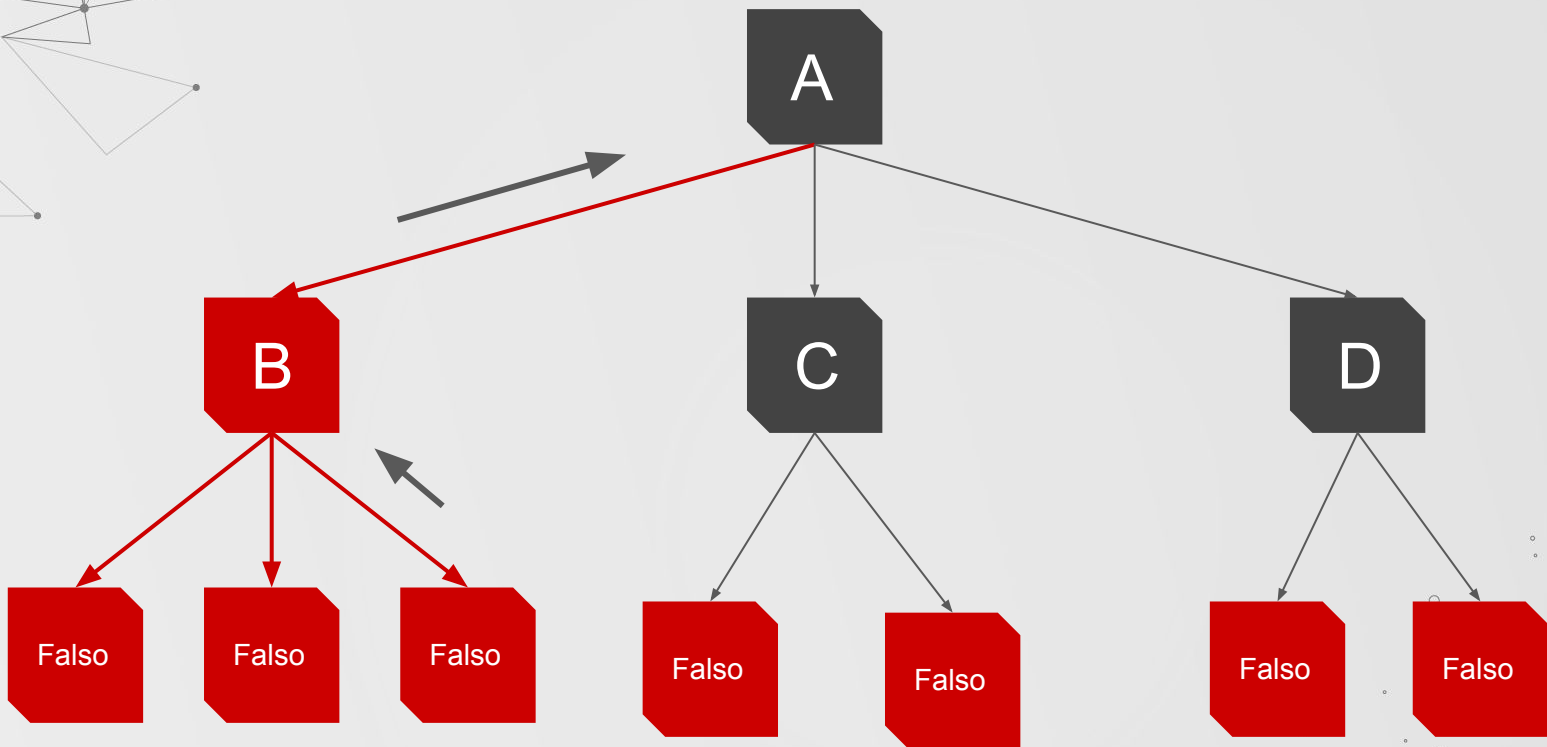


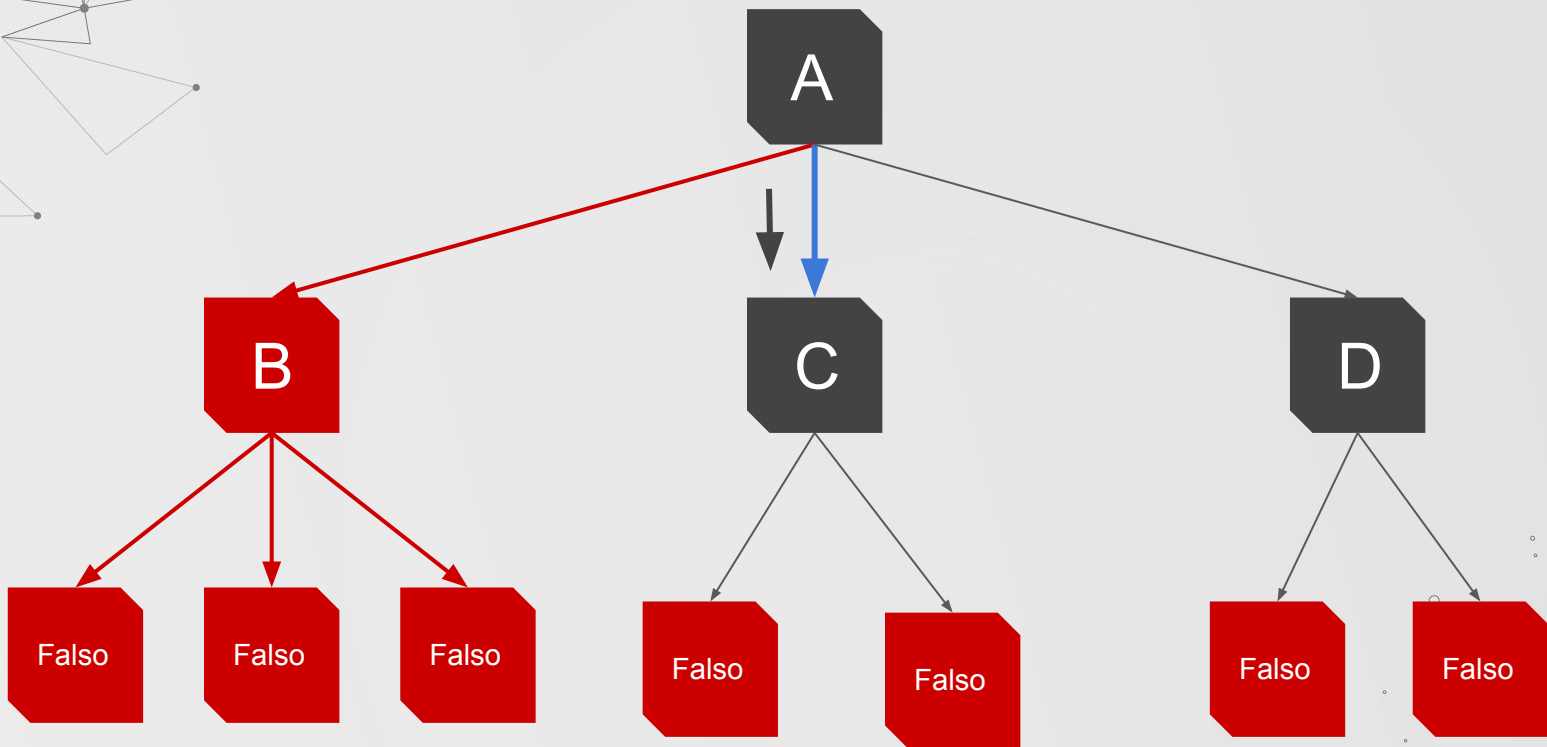


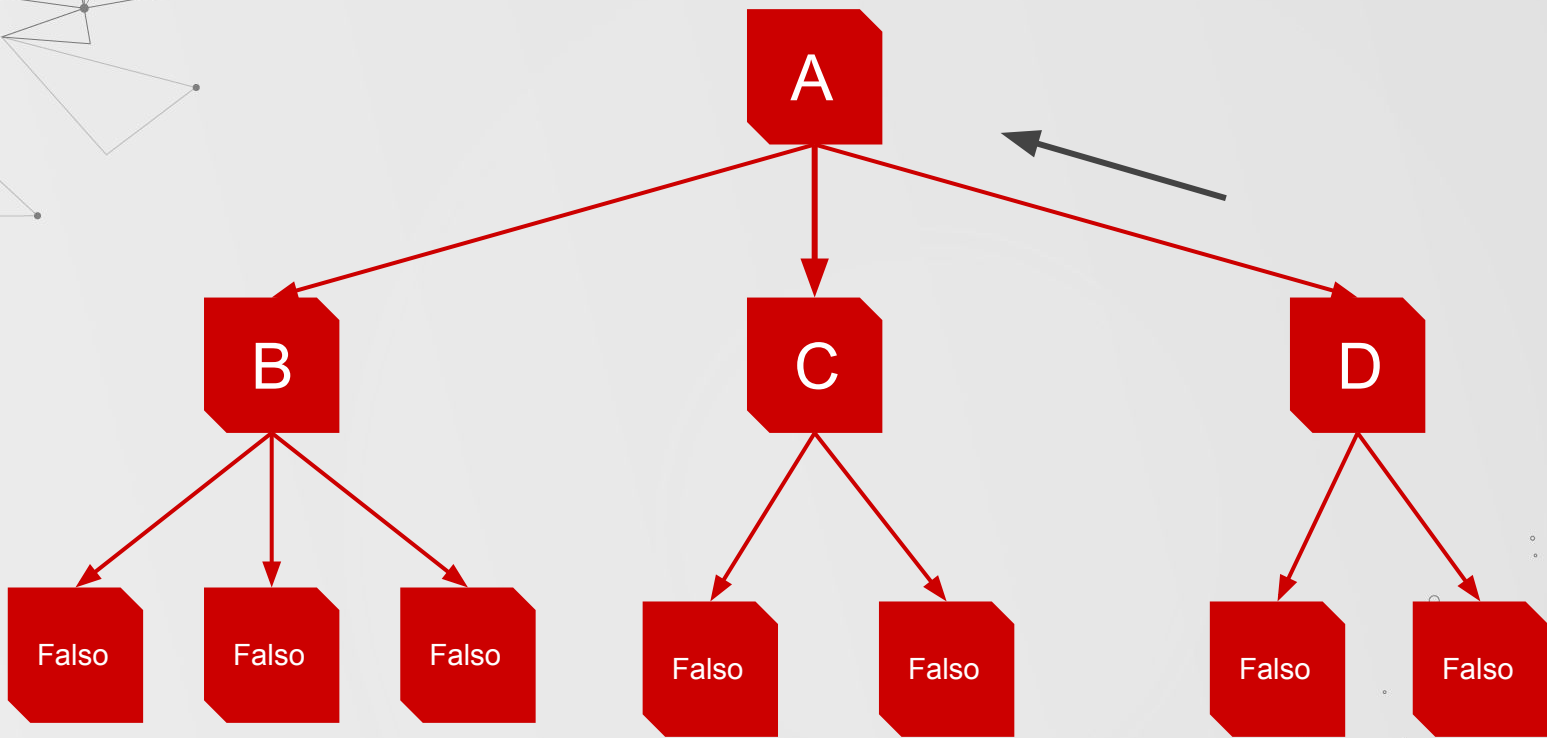


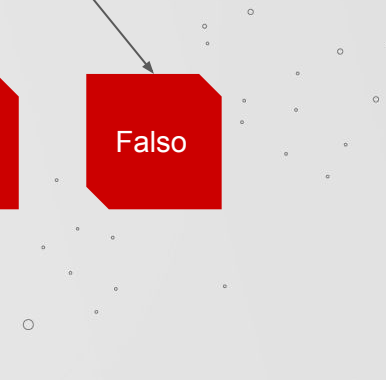
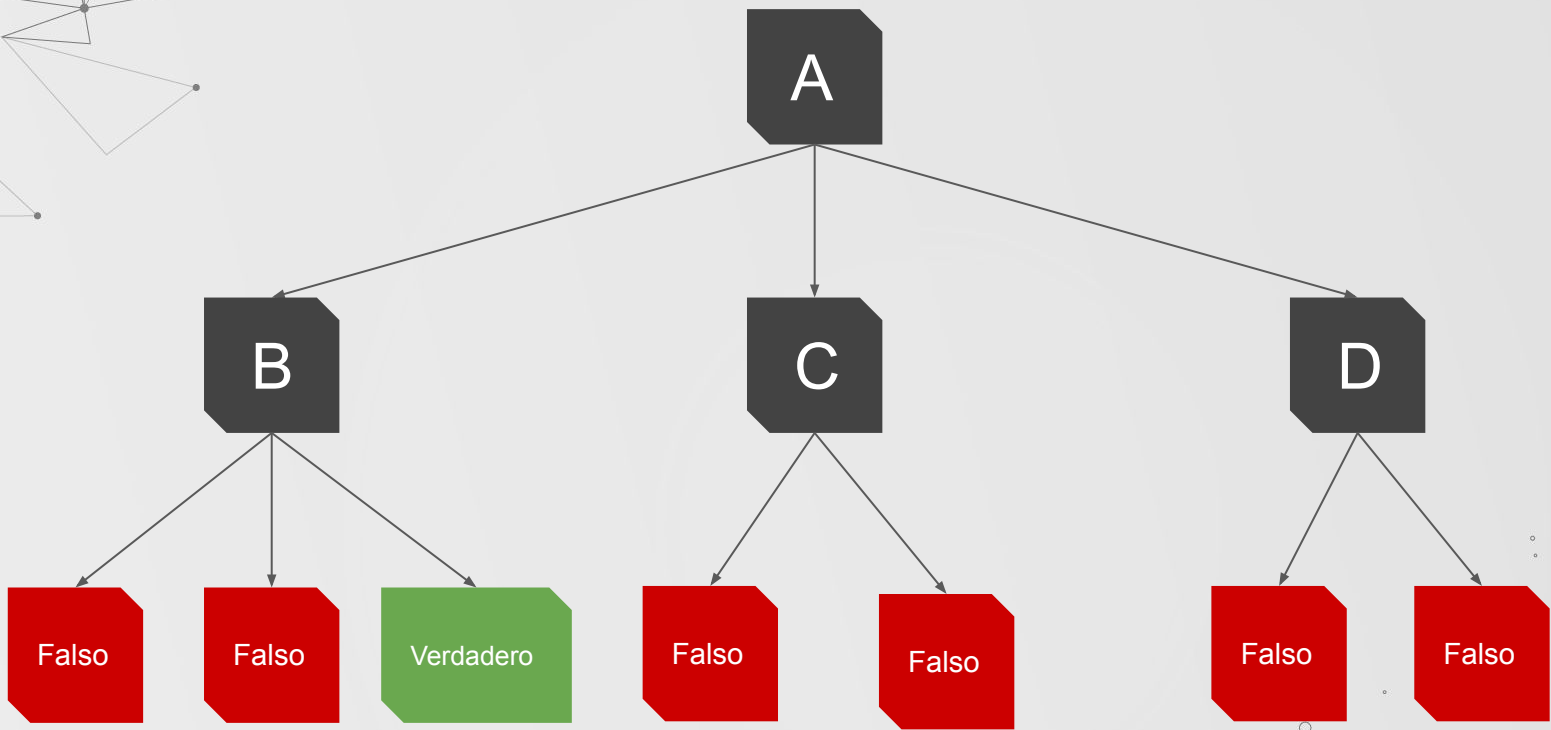


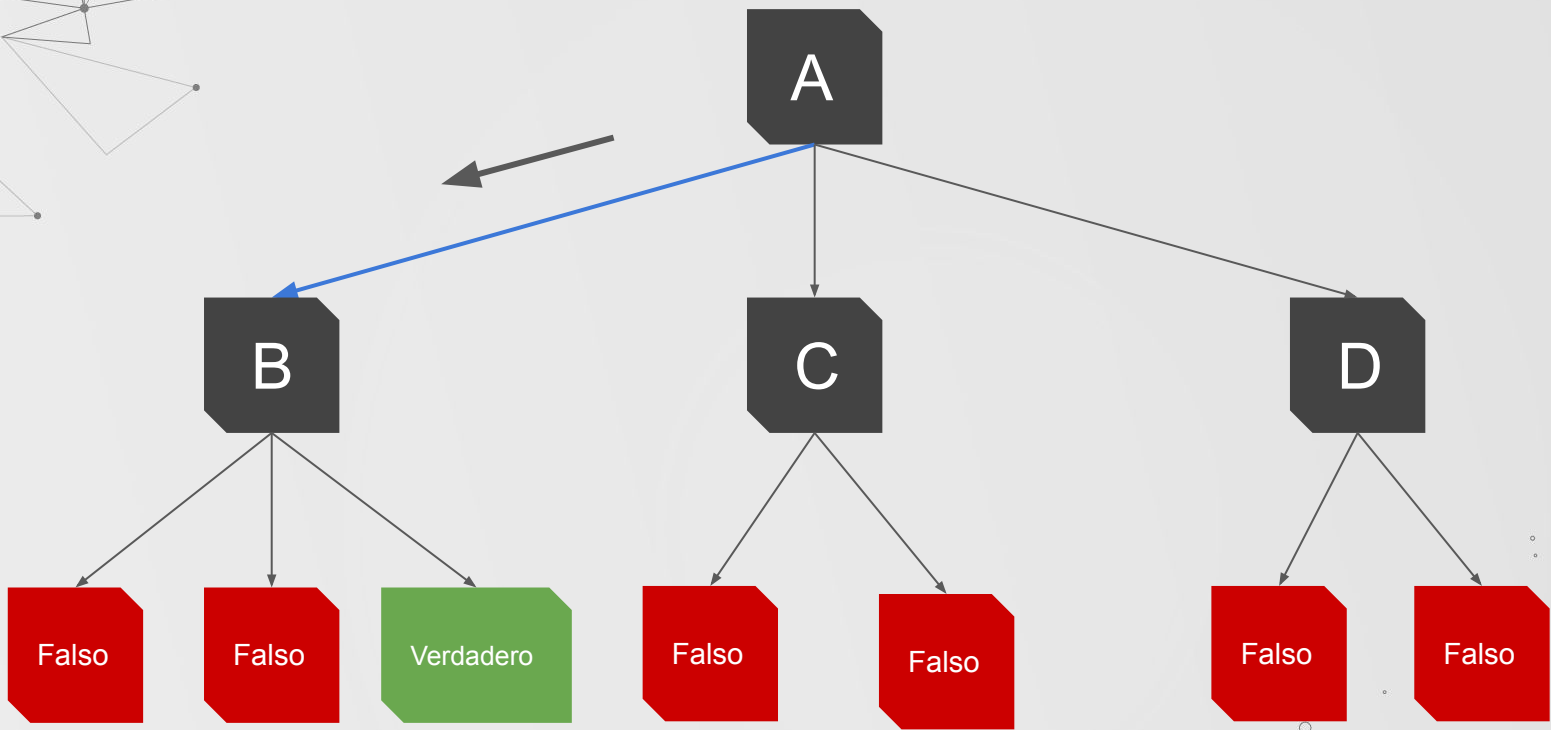


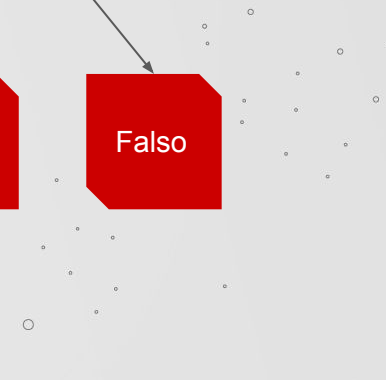
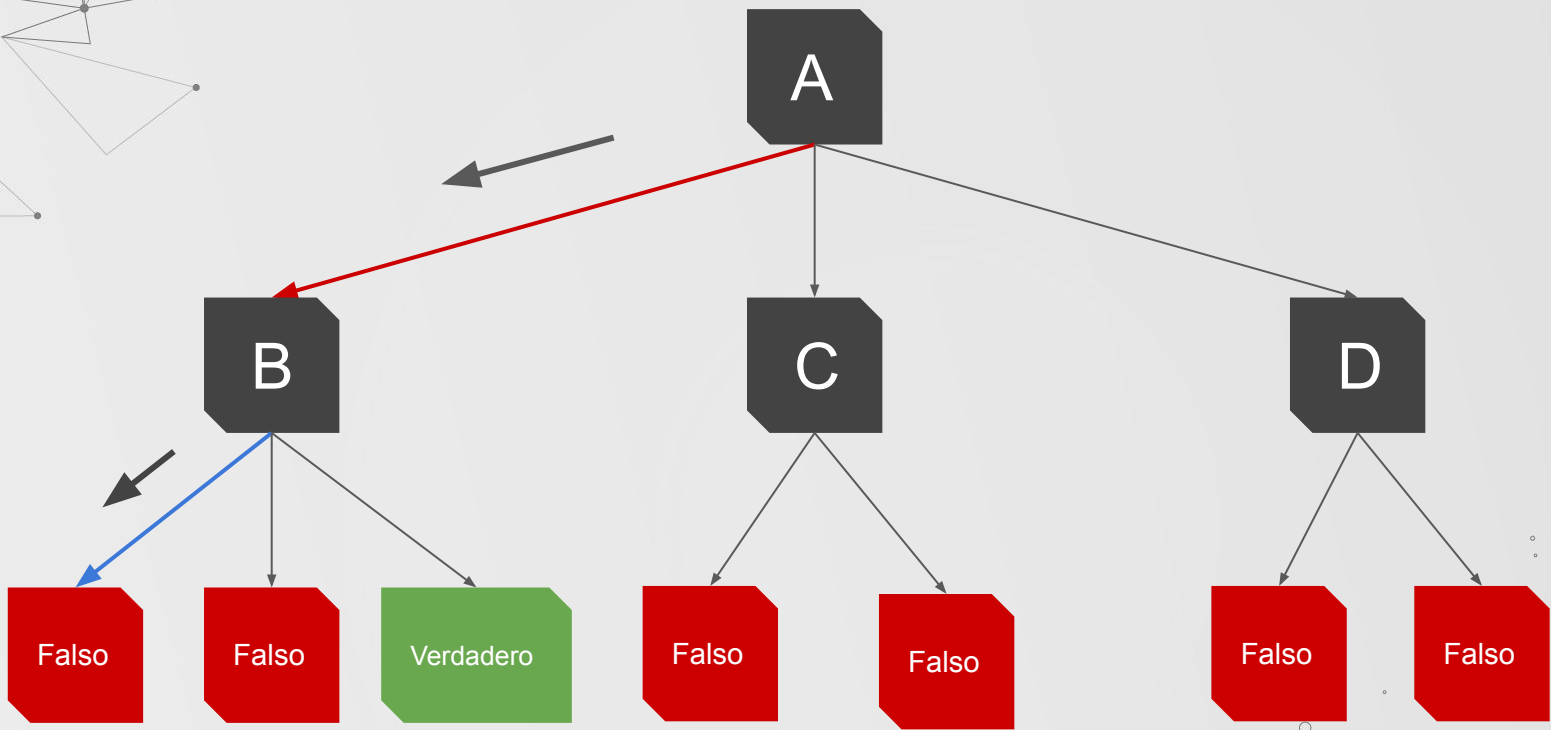


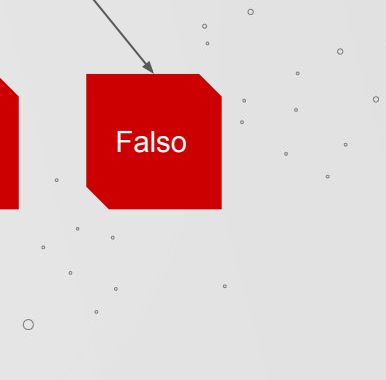
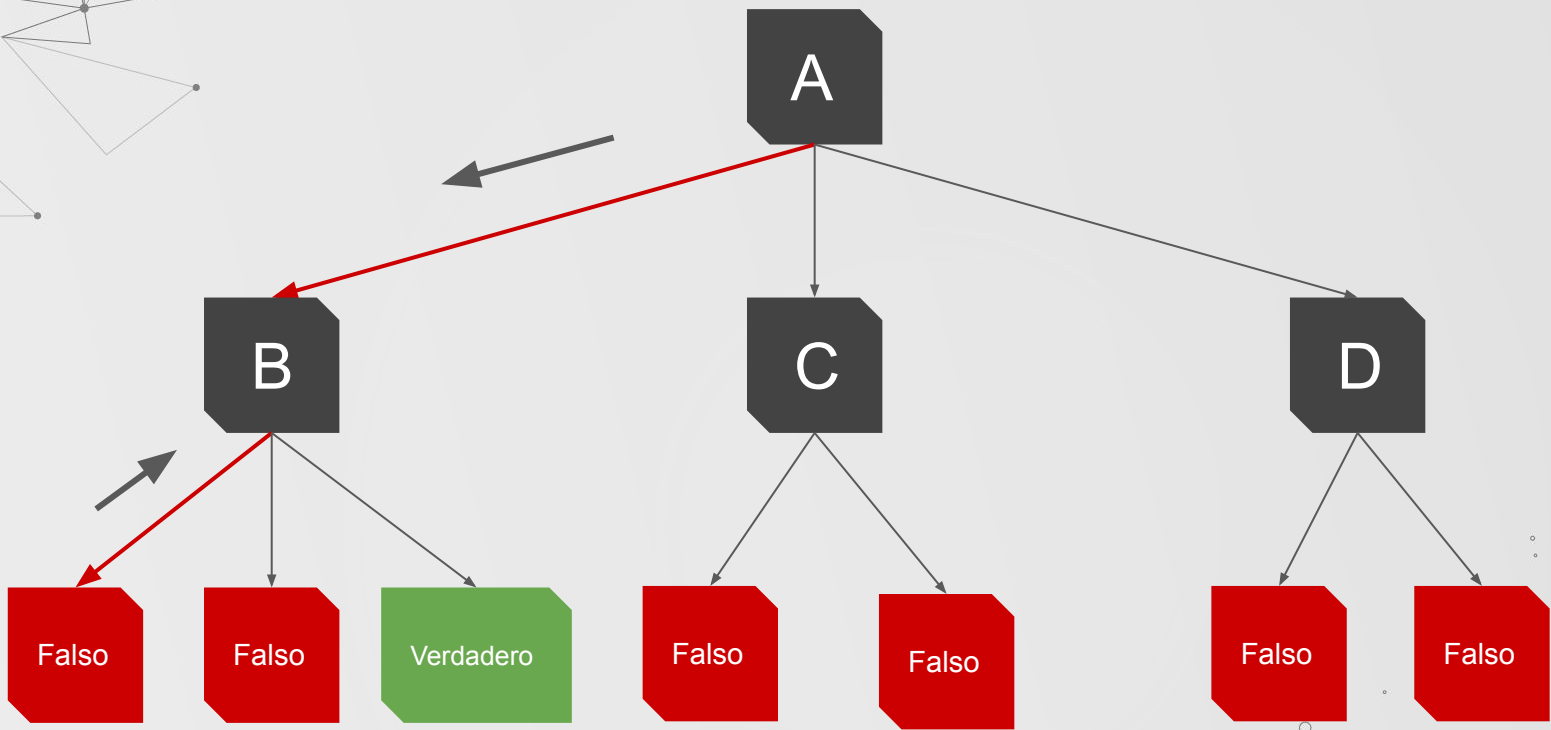


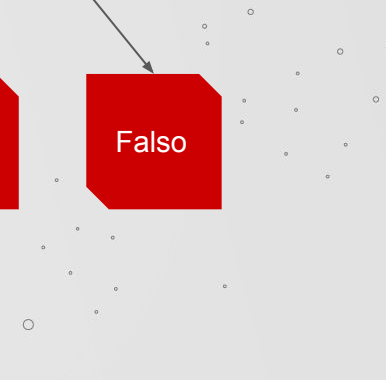
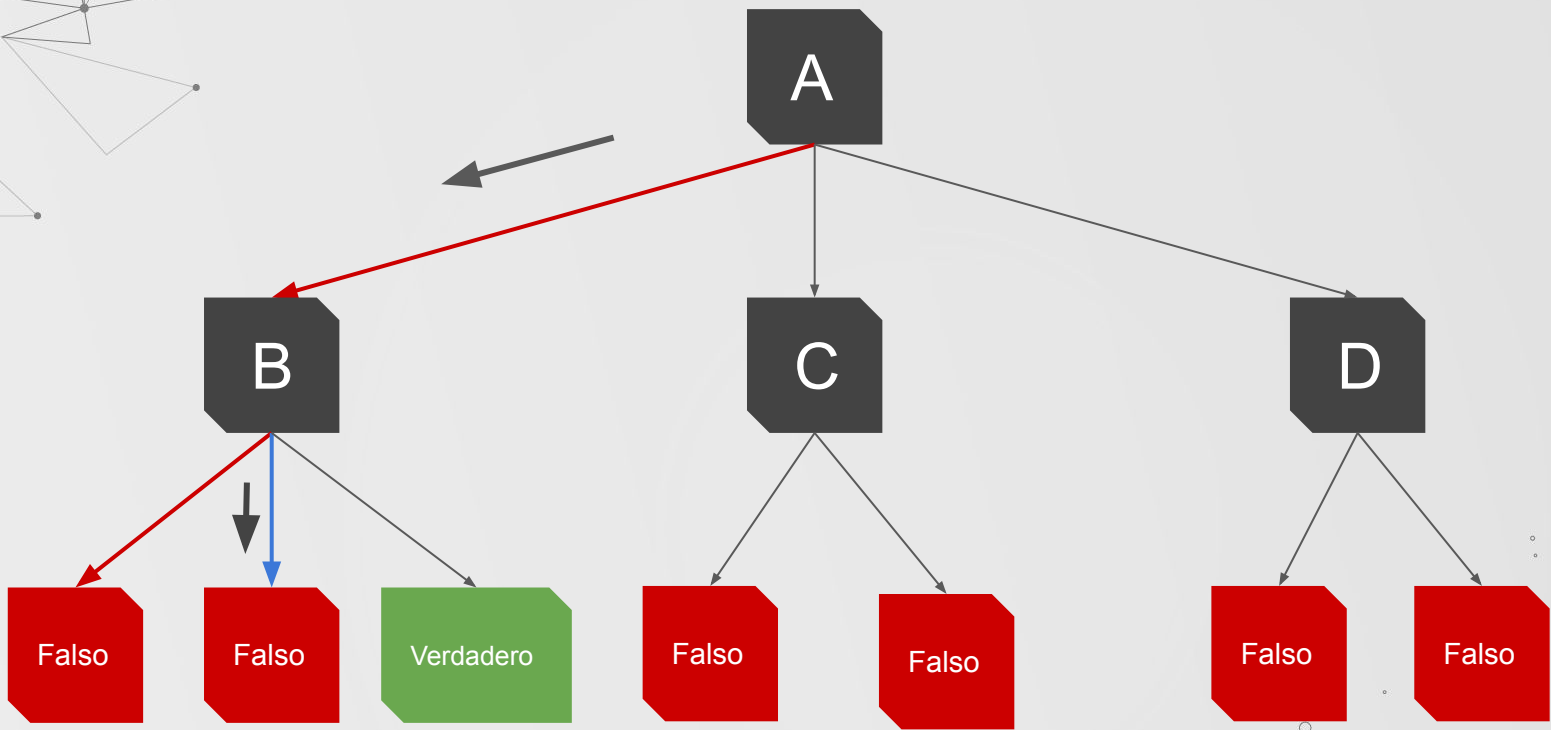
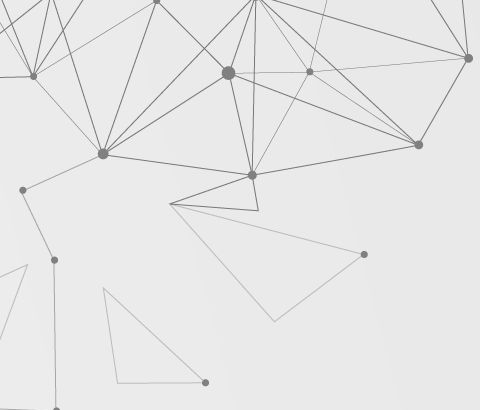


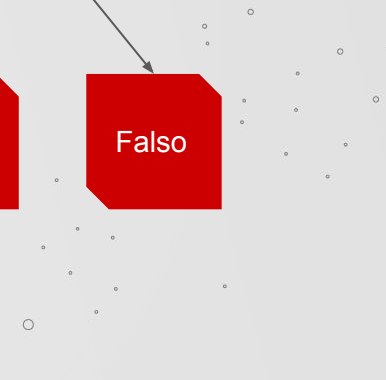
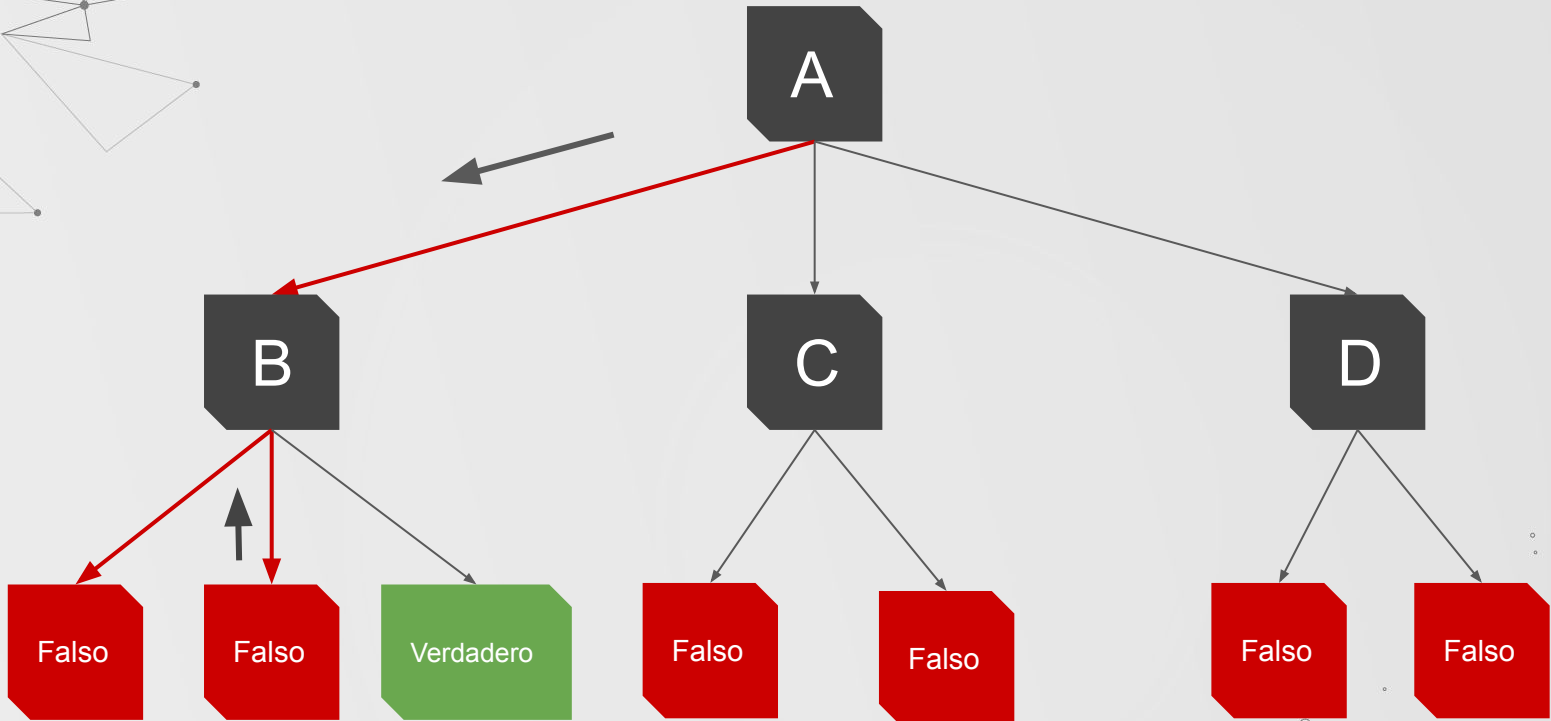
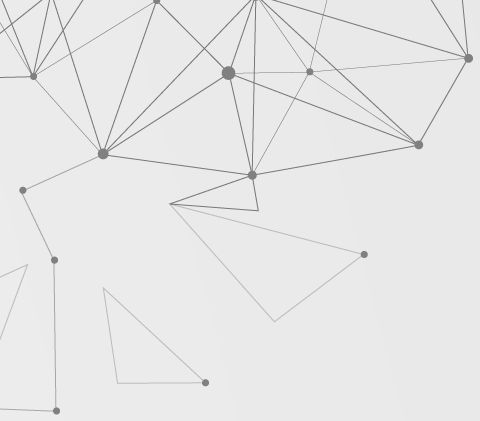


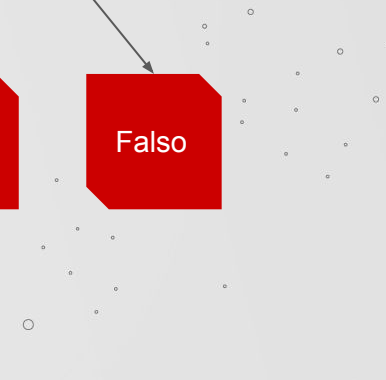
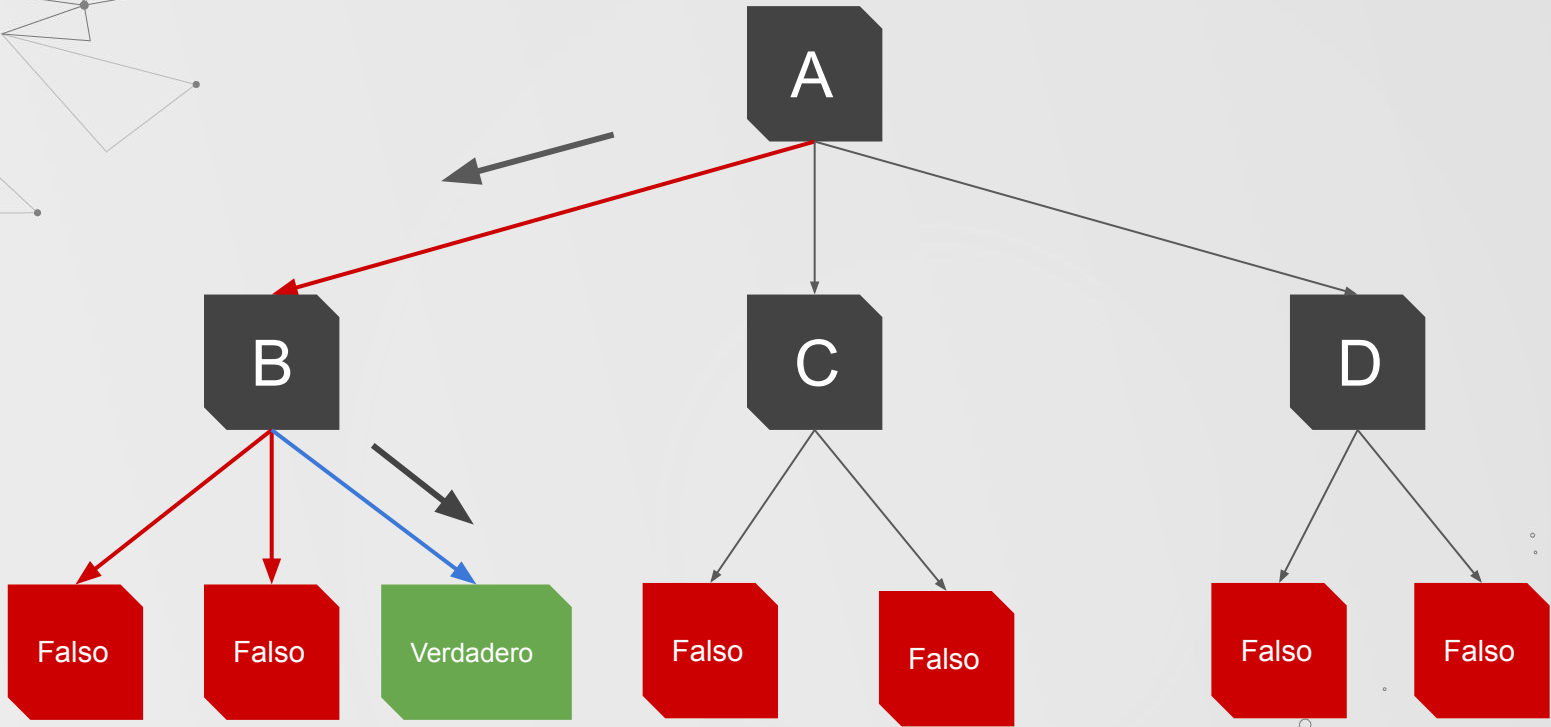
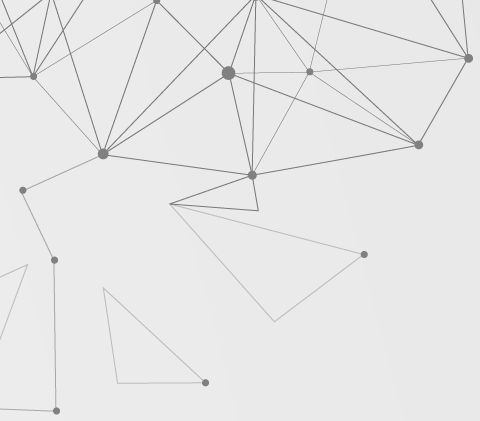


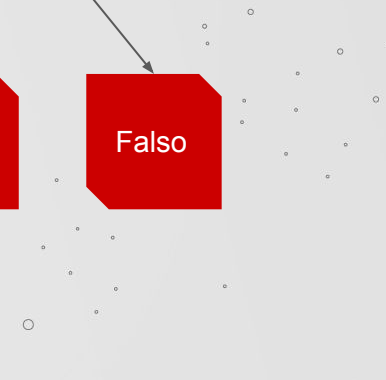
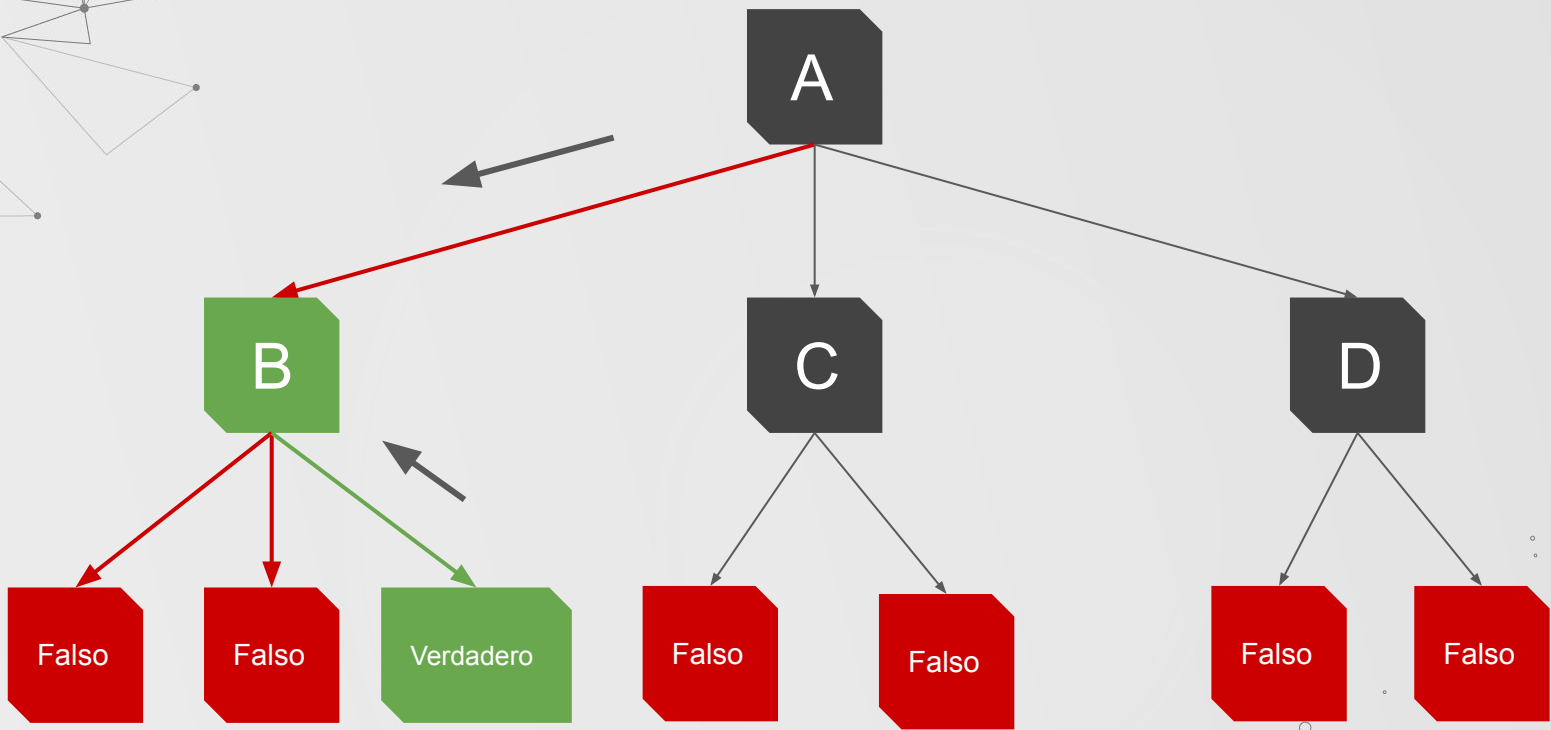


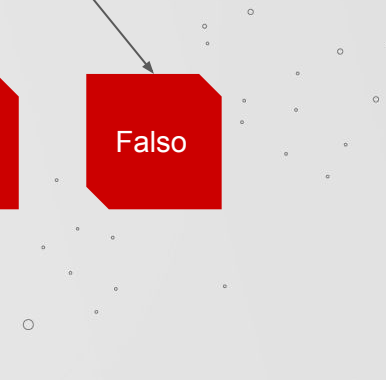
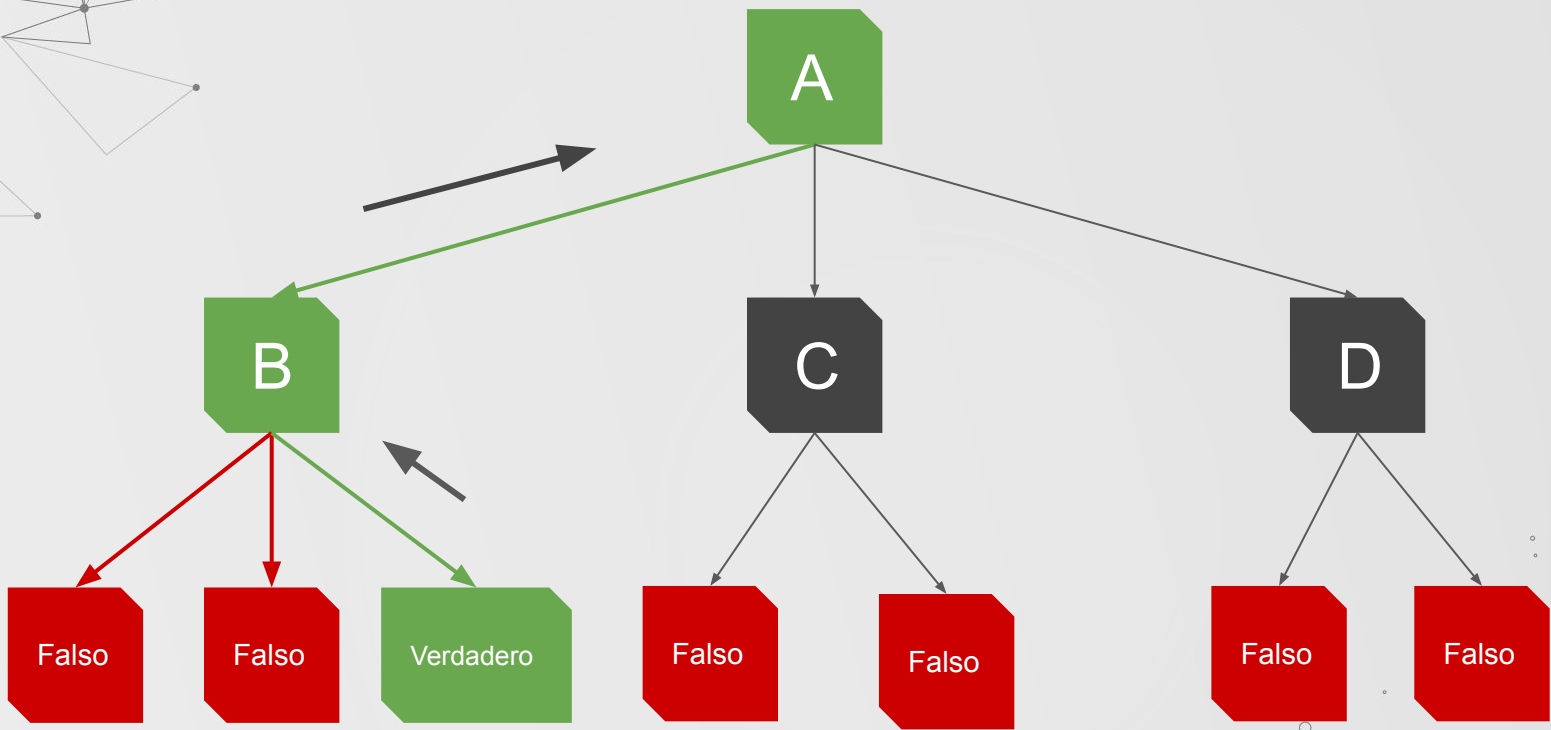








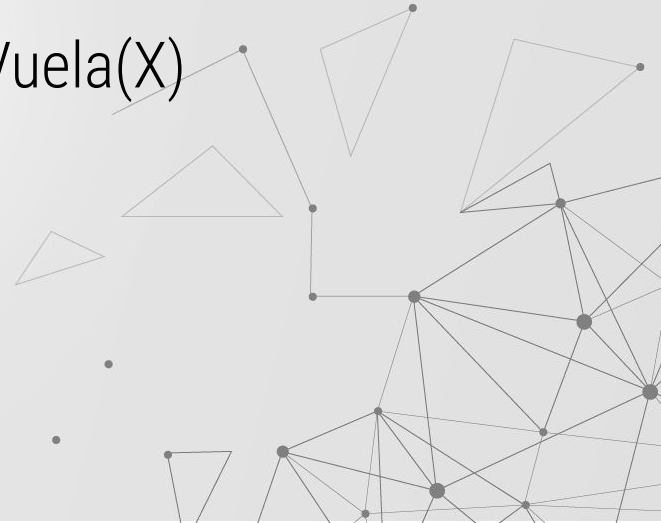


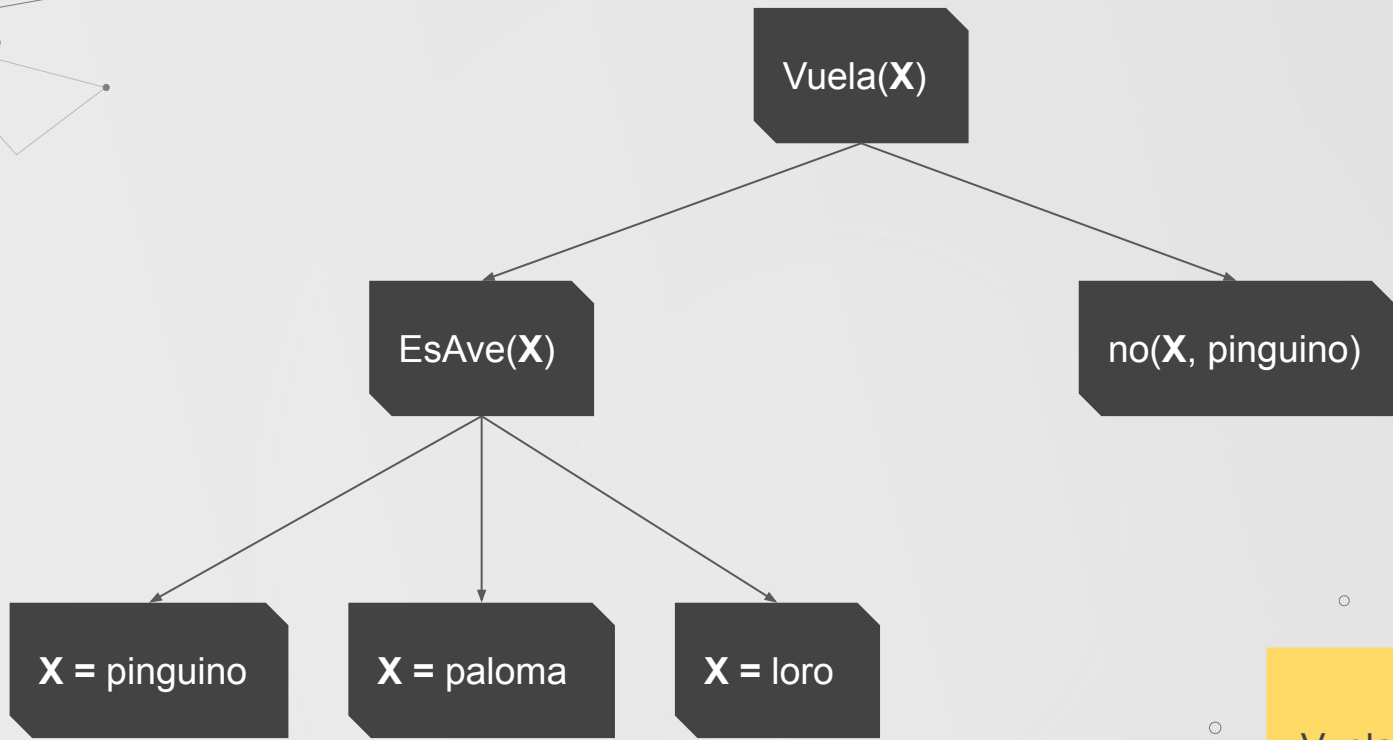


MOTOR DE INFERENCIA

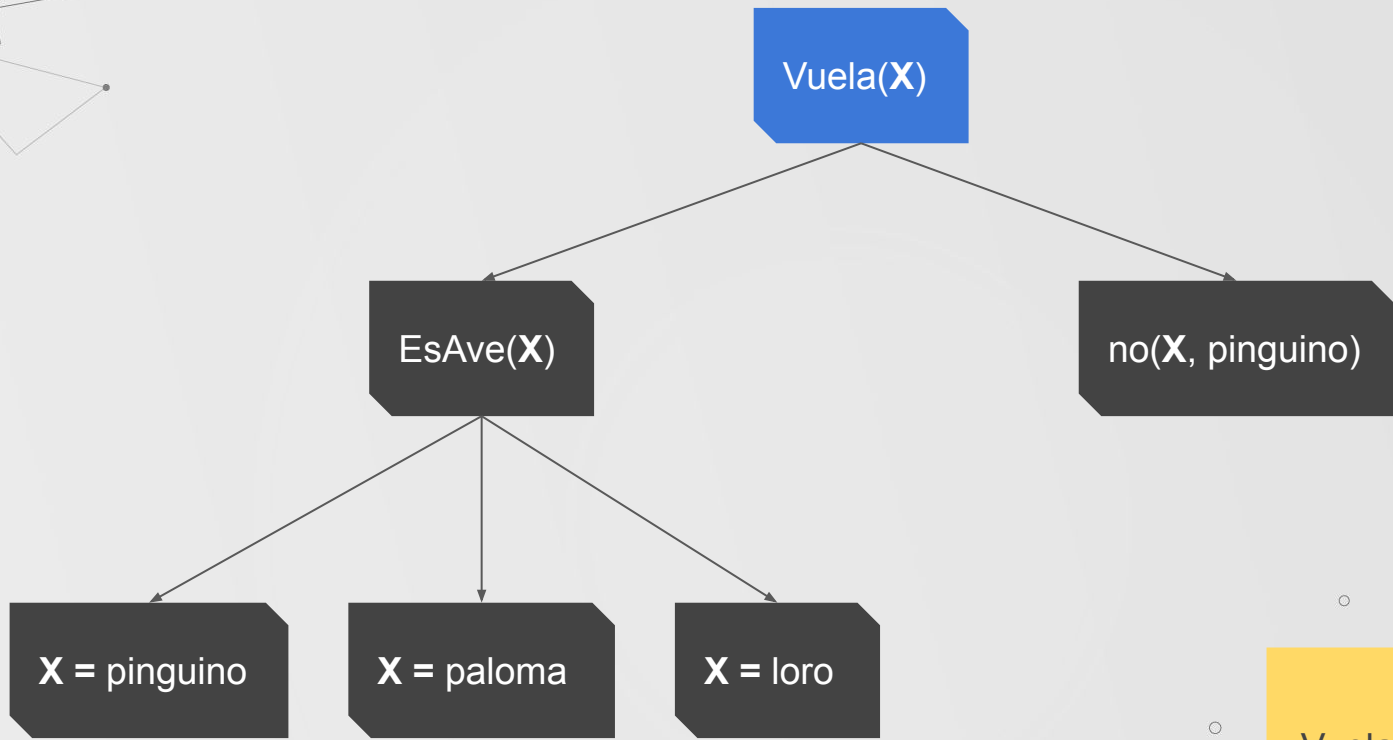
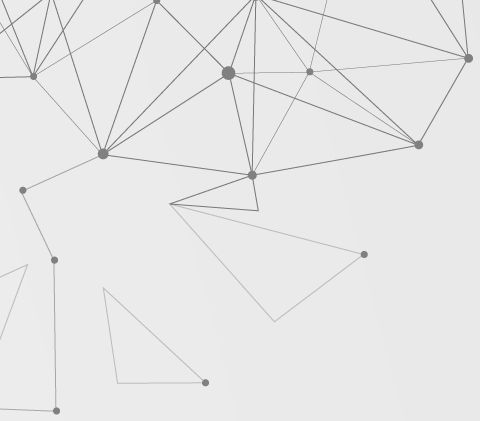
Utilizado para inferir los resultados a partir de hechos y reglas

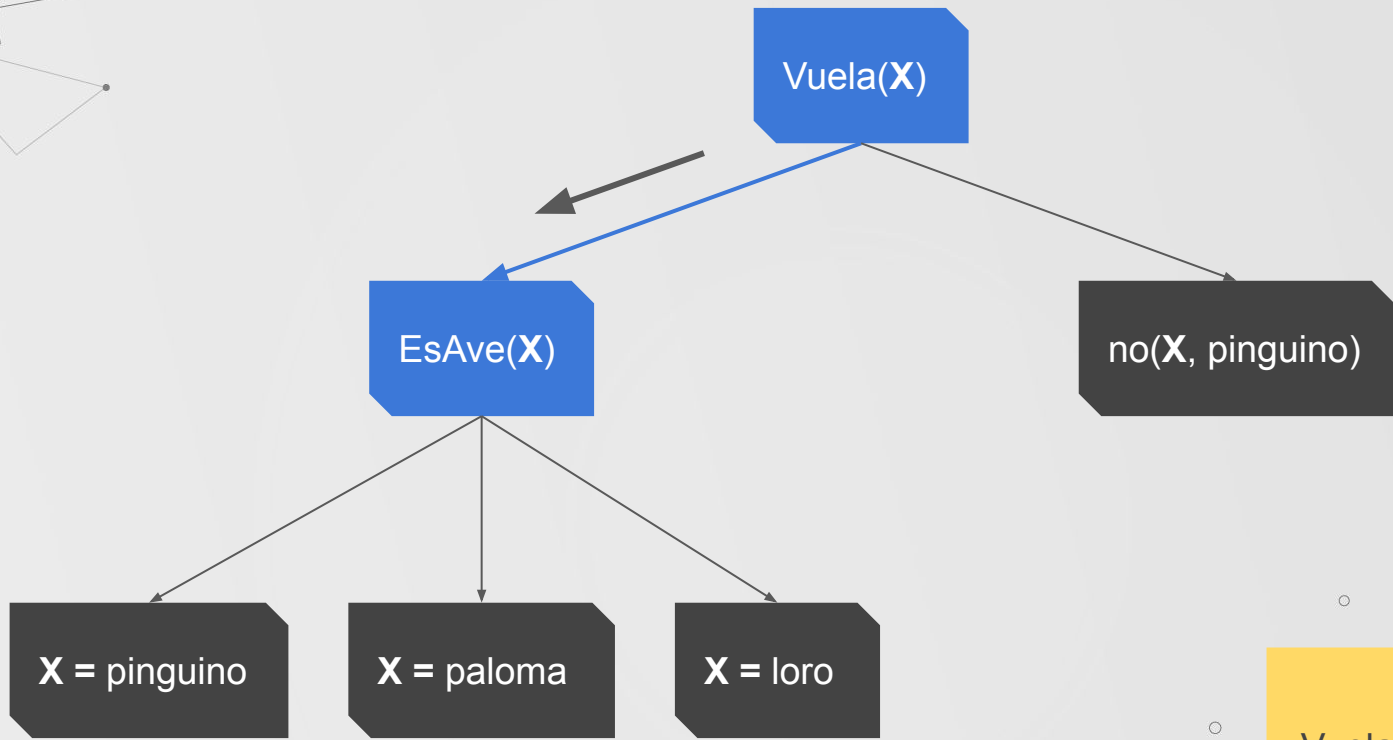
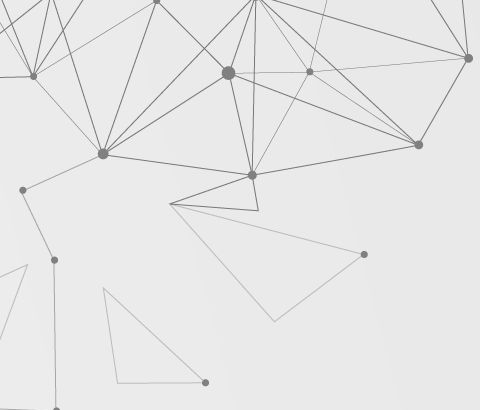
$EsAve(X) \wedge no(X, pingüino) \rightarrow Vuela(X)$



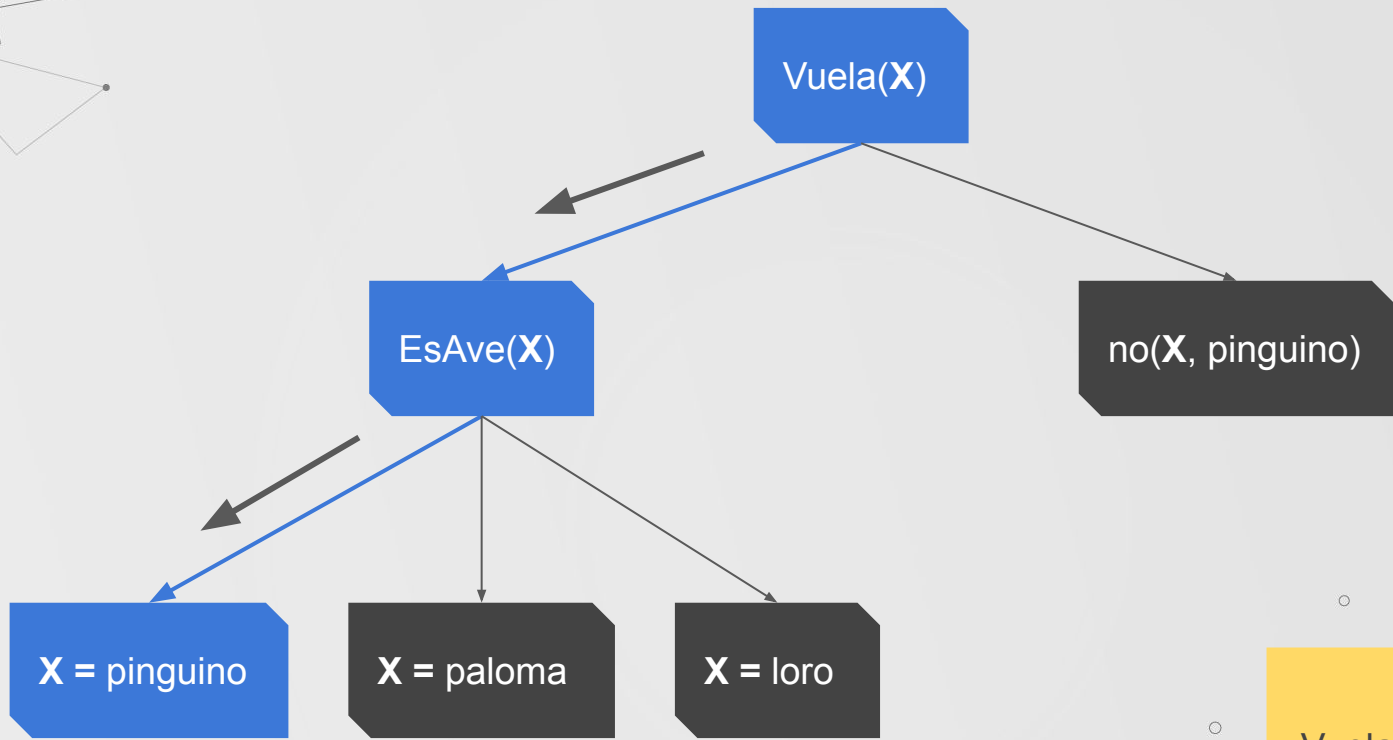


Vuela(paloma)

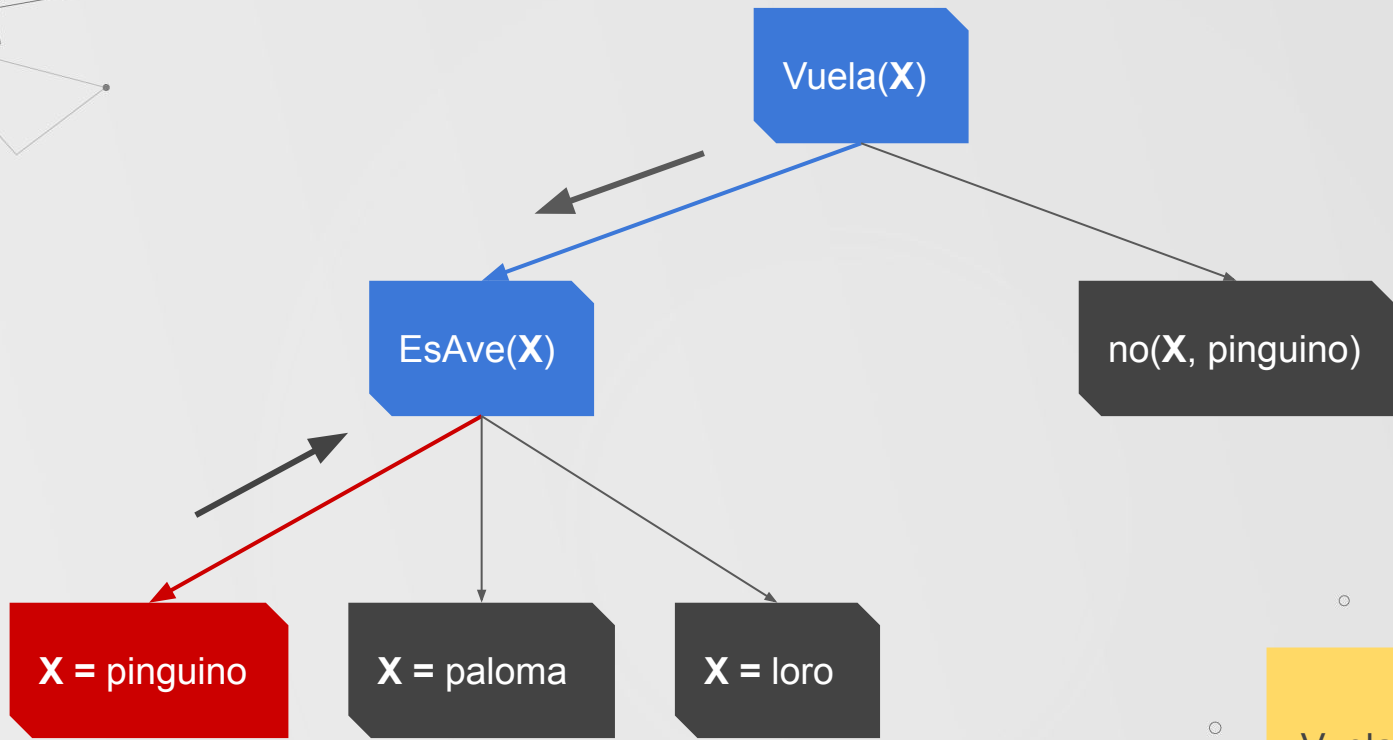




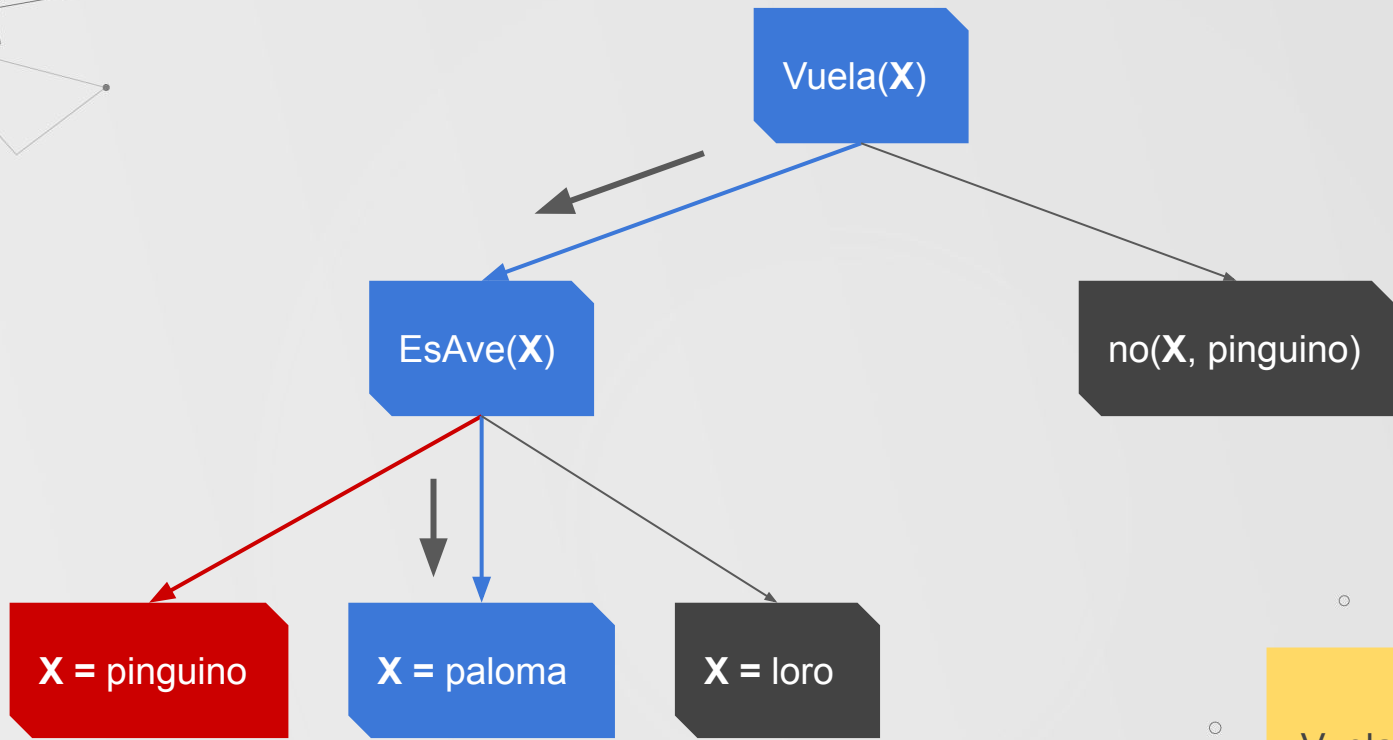
Vuela(paloma)



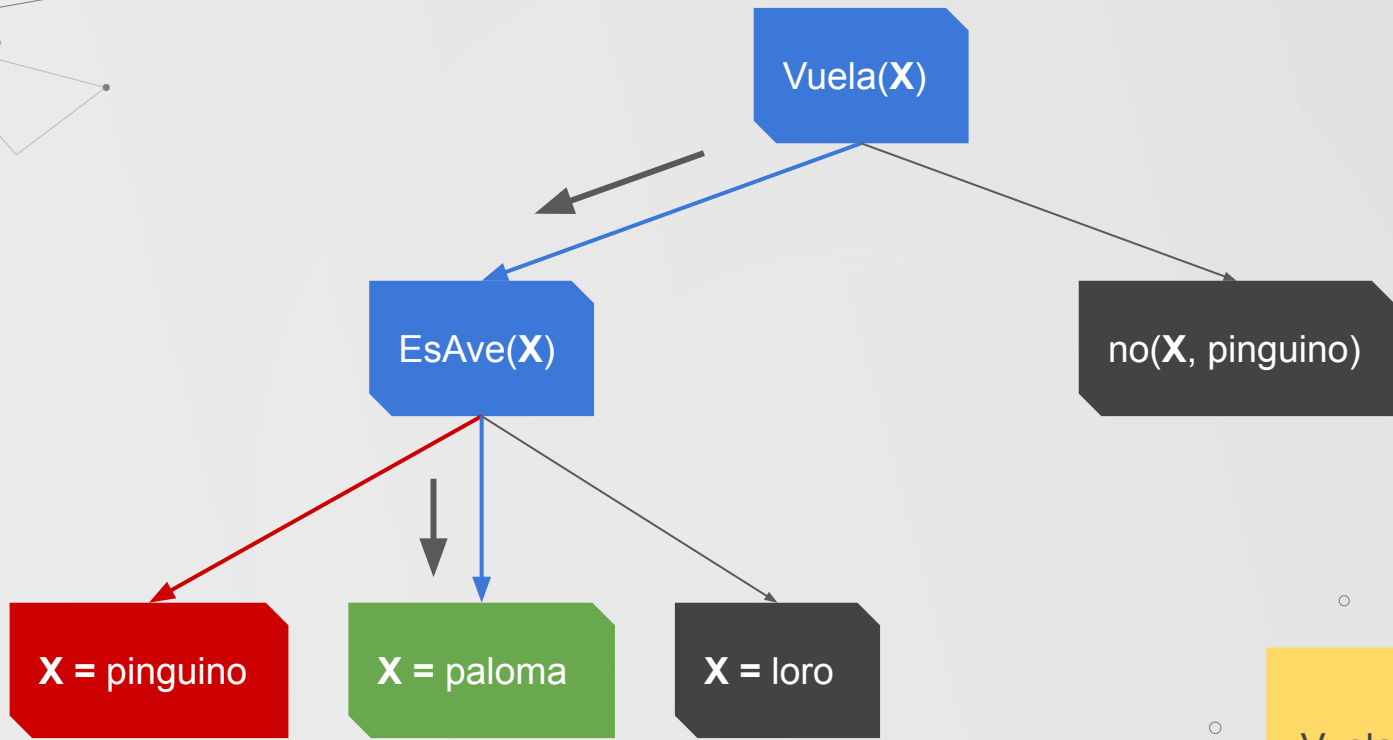
Vuela(**paloma**)



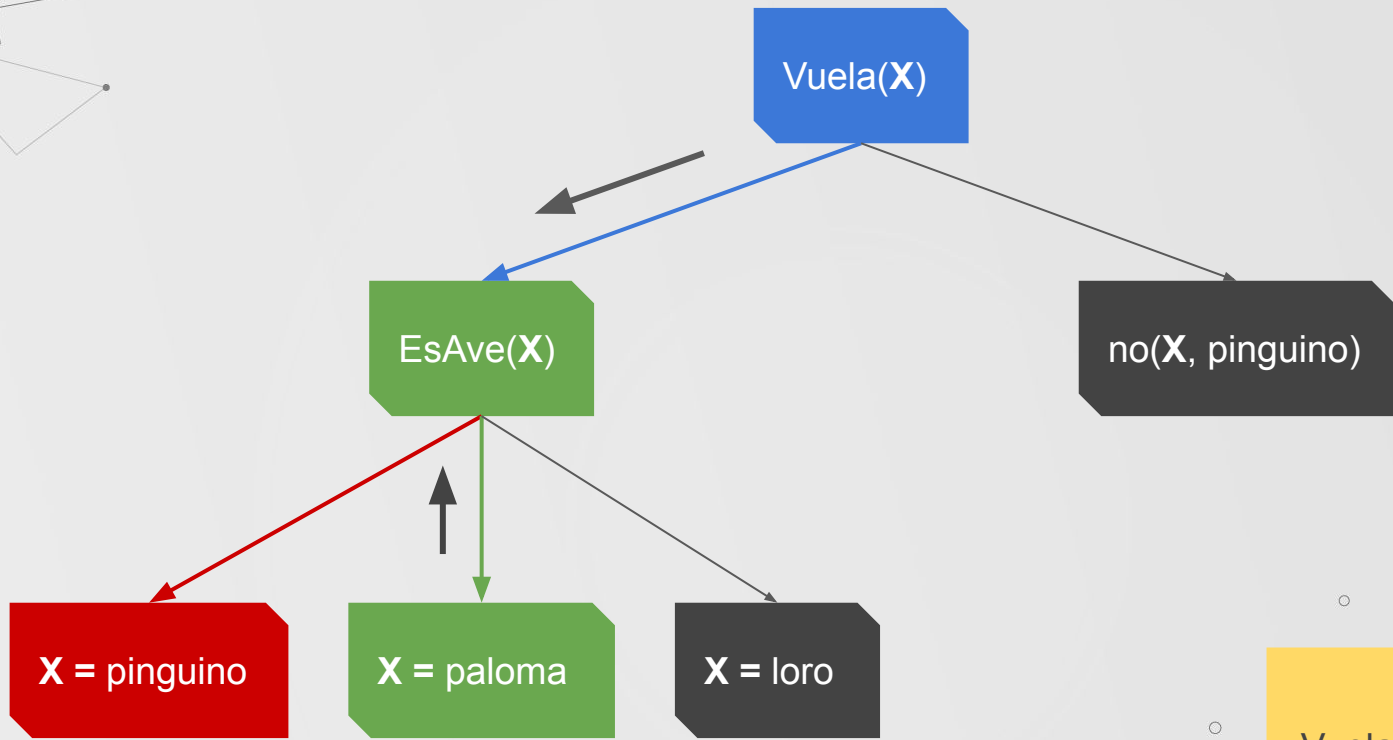
Vuela(**paloma**)



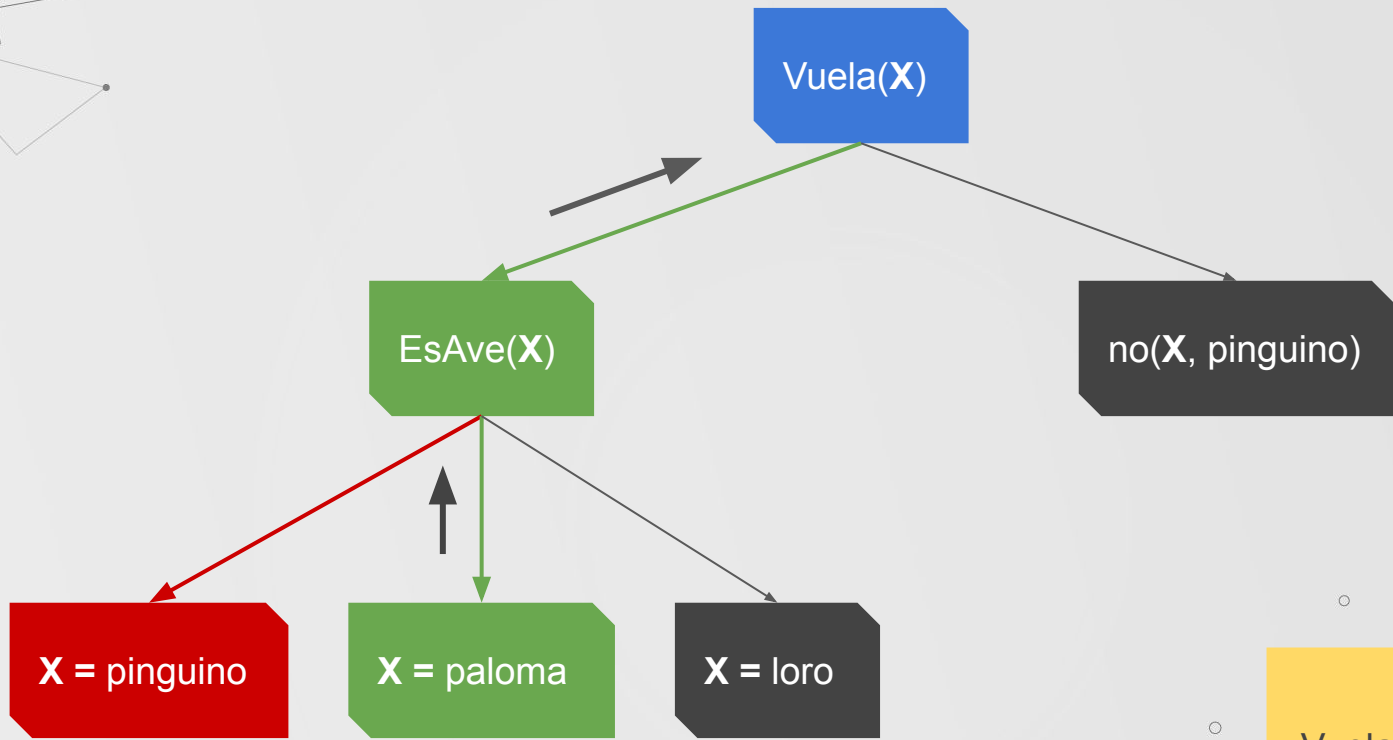
Vuela(paloma)

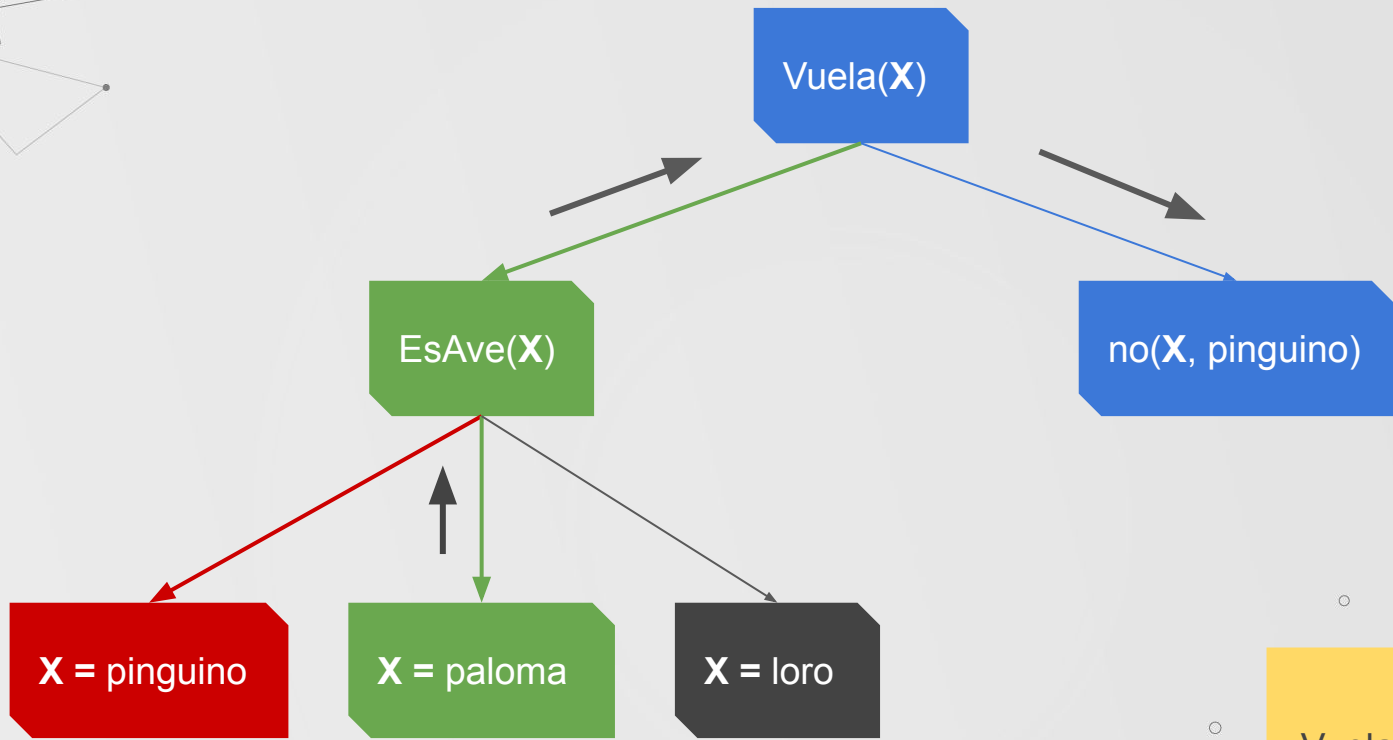


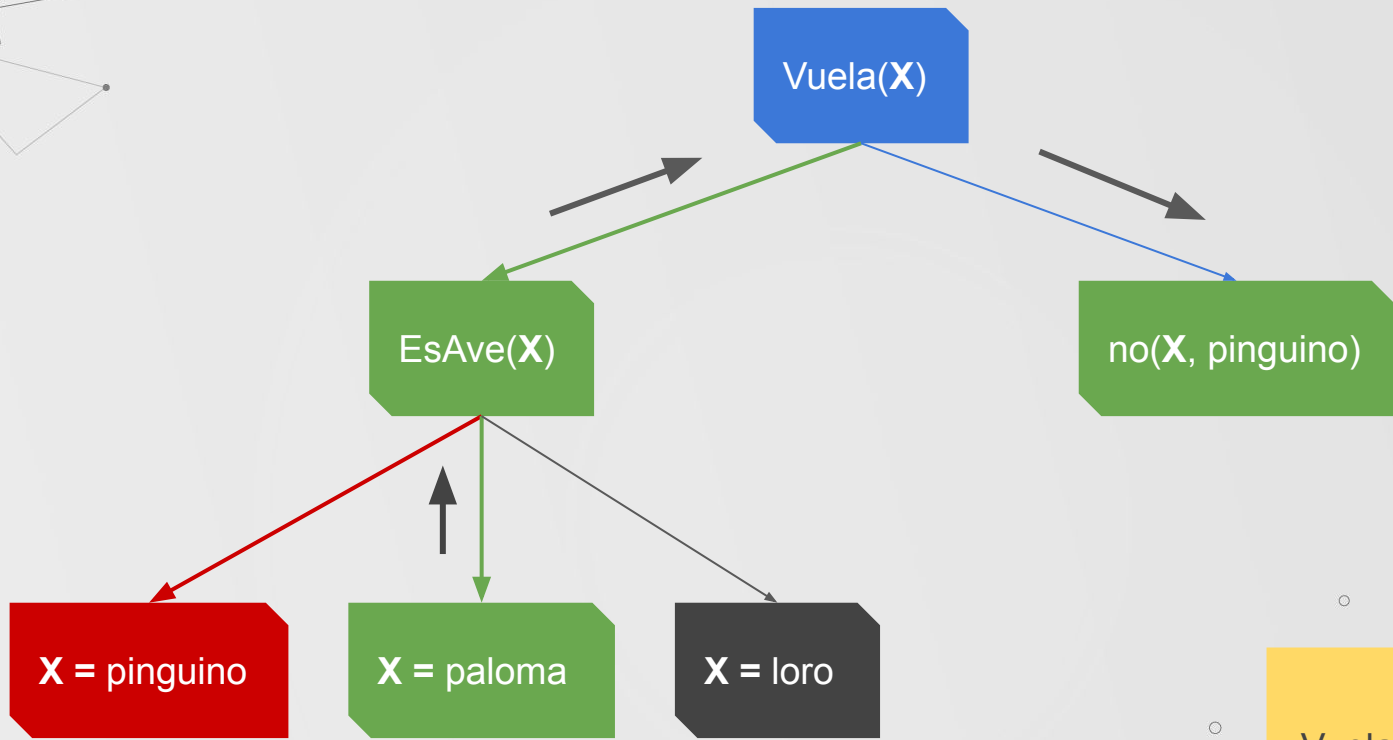
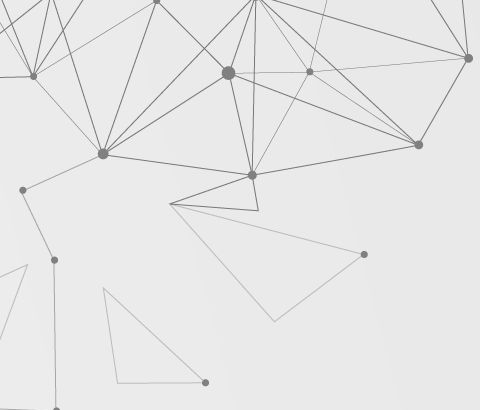
Vuela(paloma)



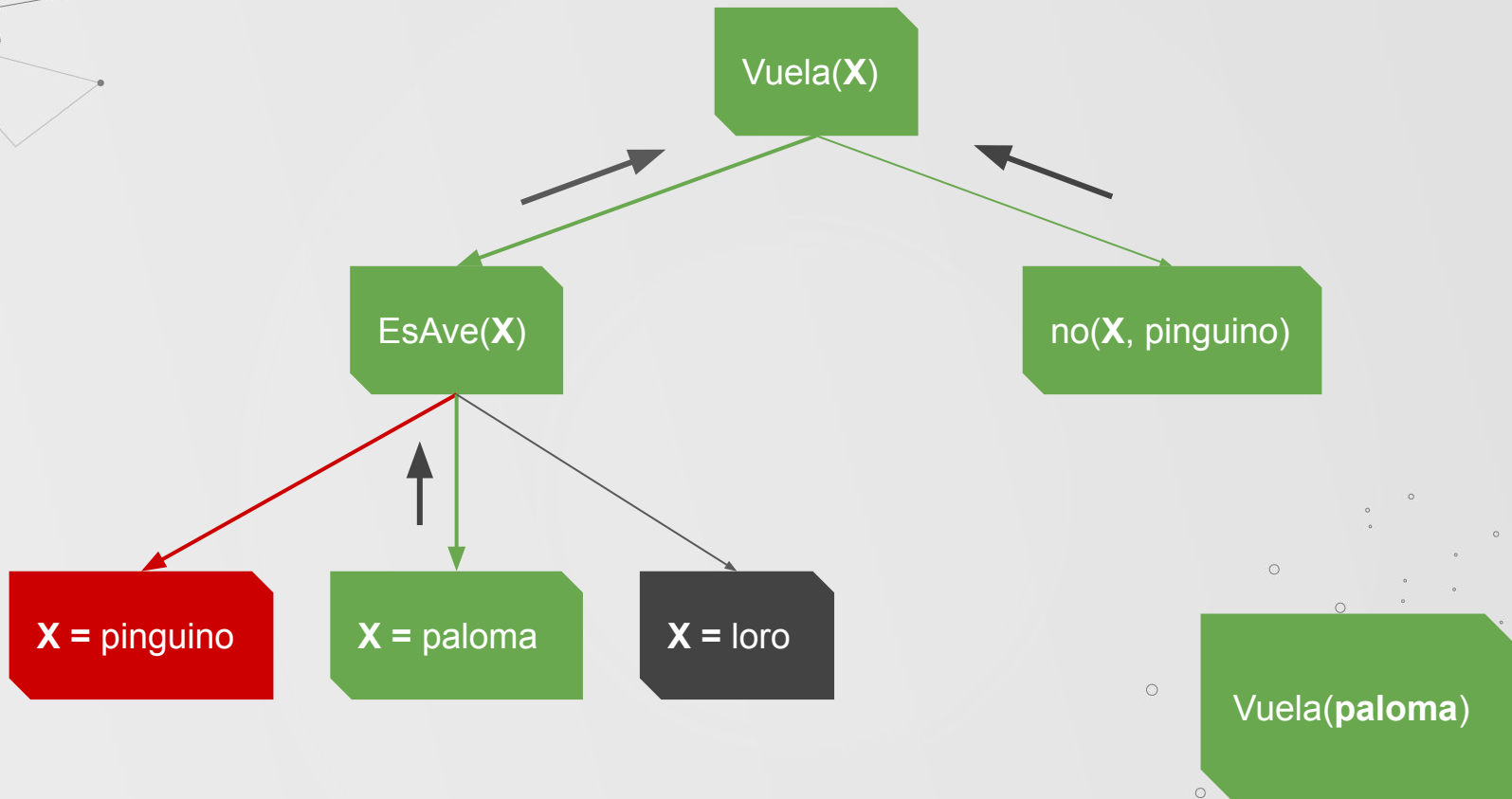
Vuela(paloma)

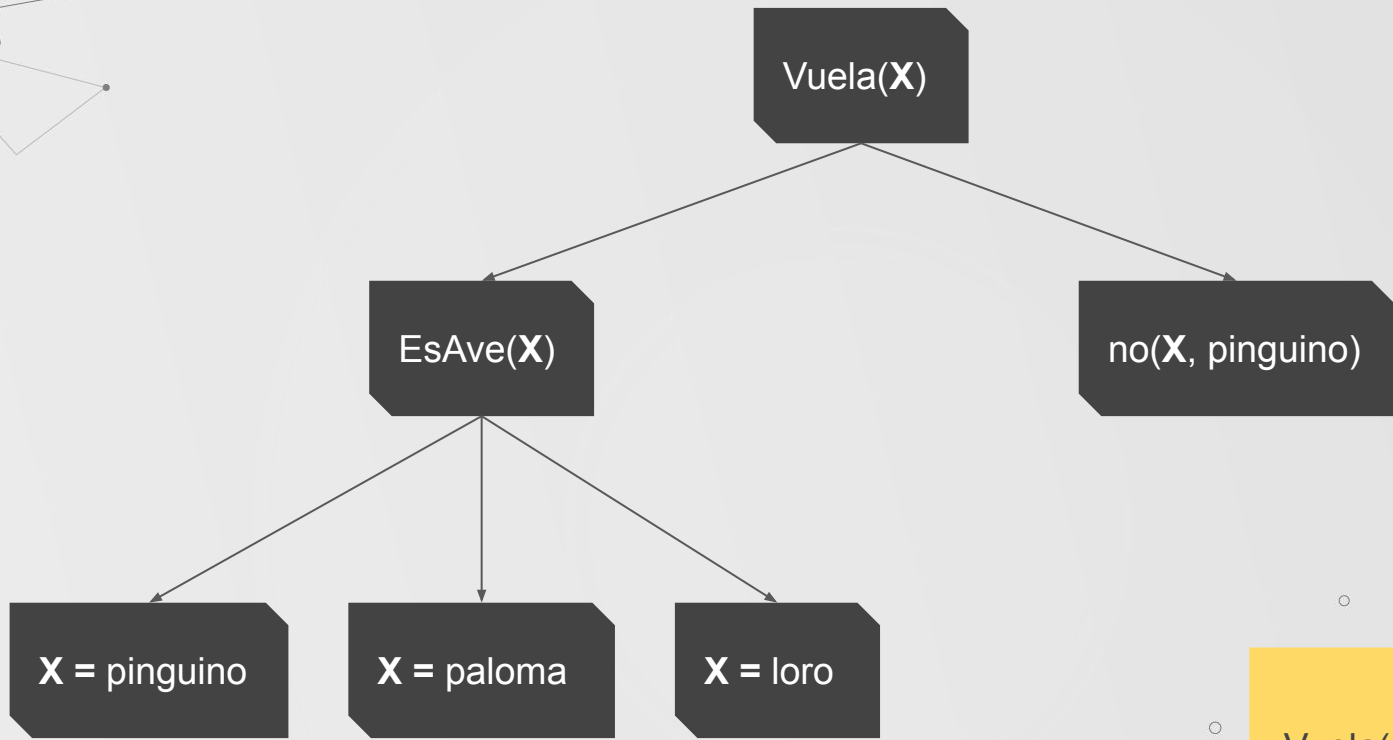




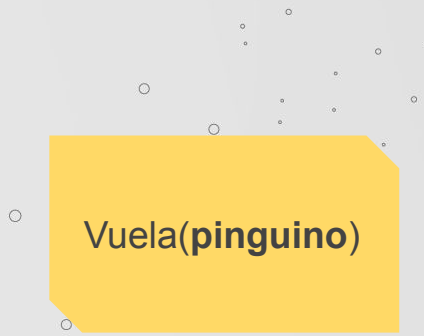
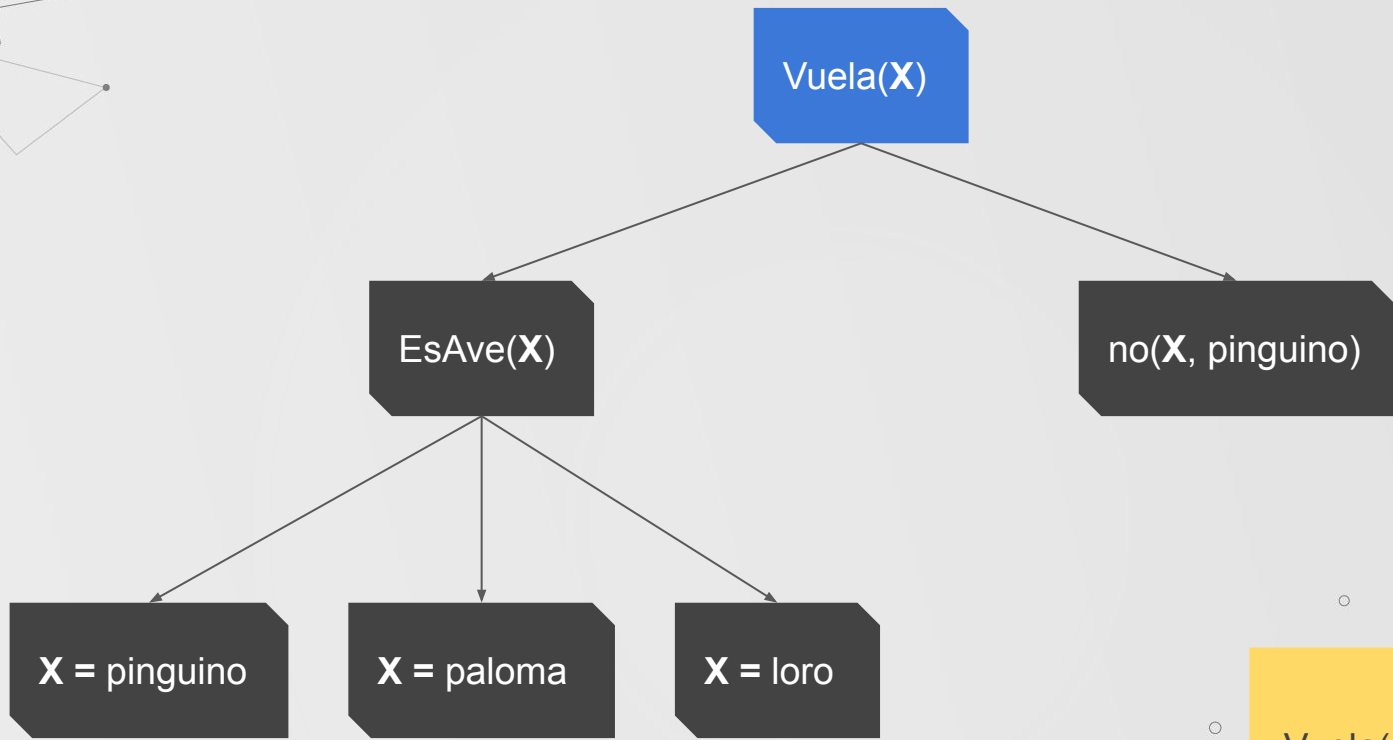


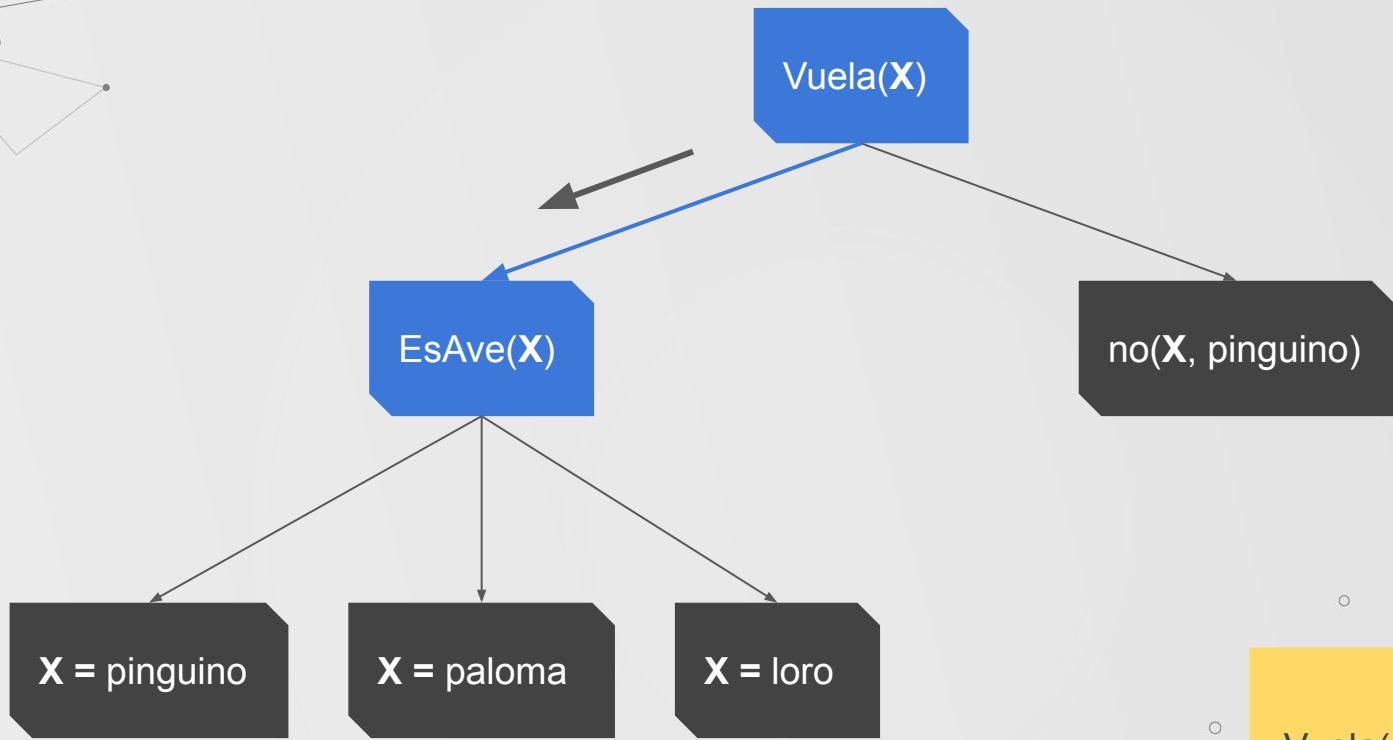
Vuela(paloma)



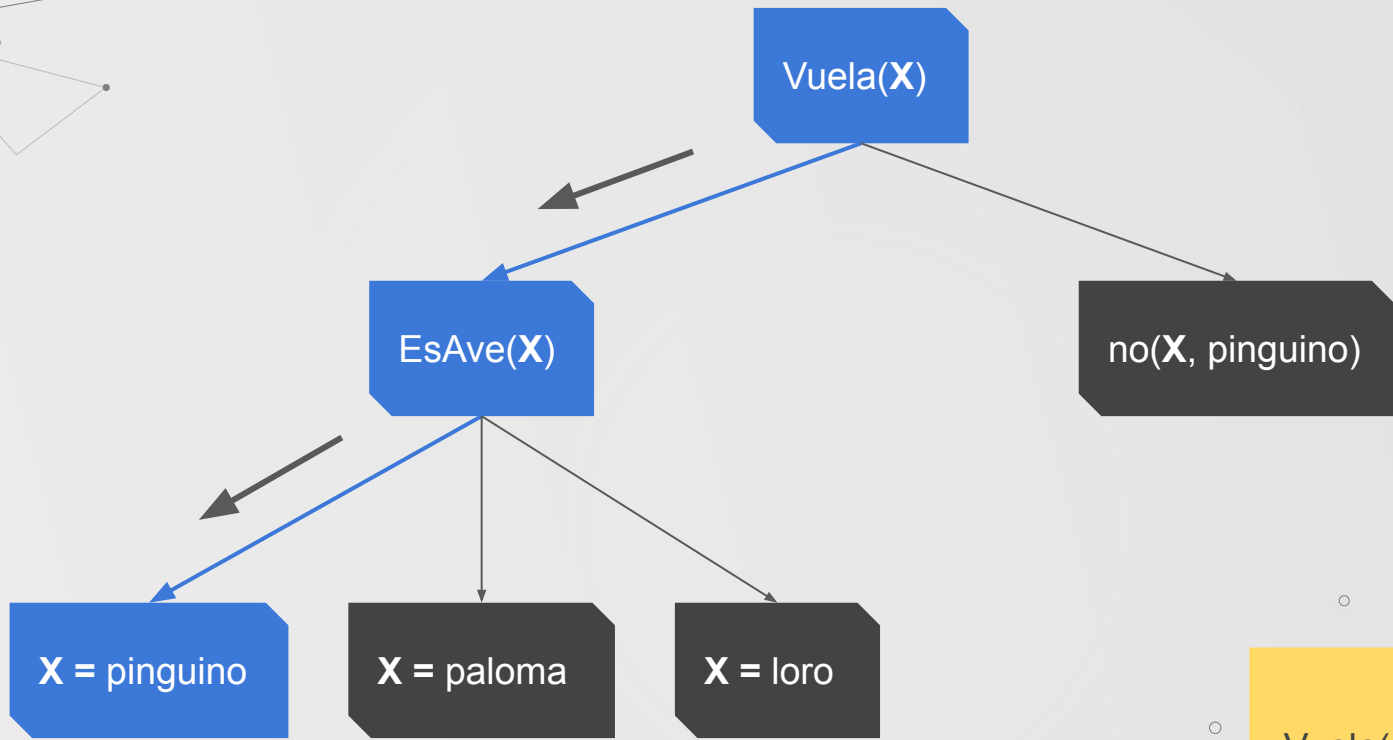


Vuela(pinguino)

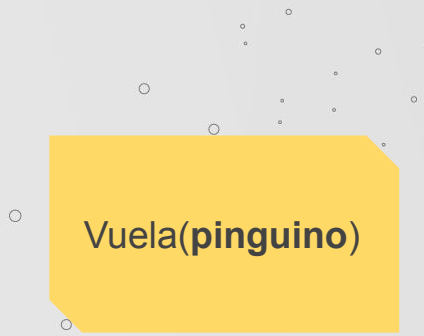
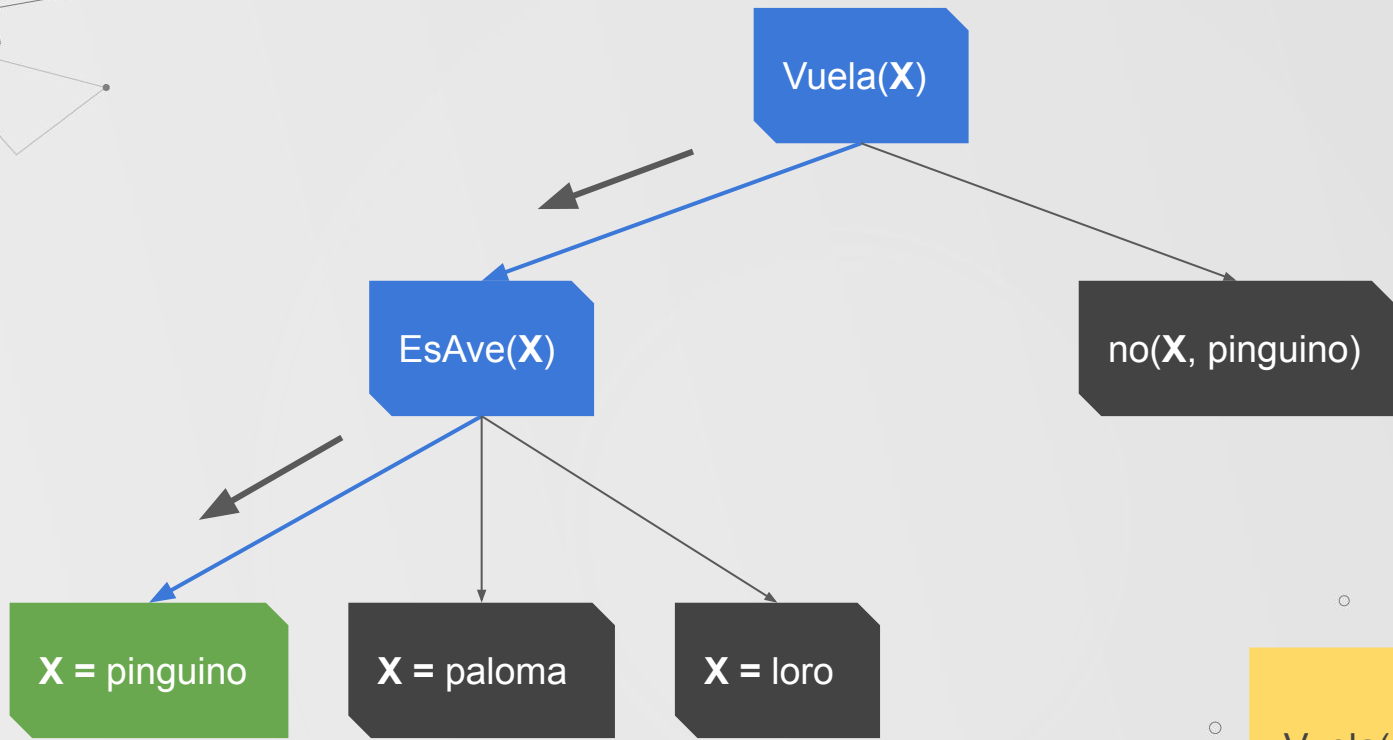


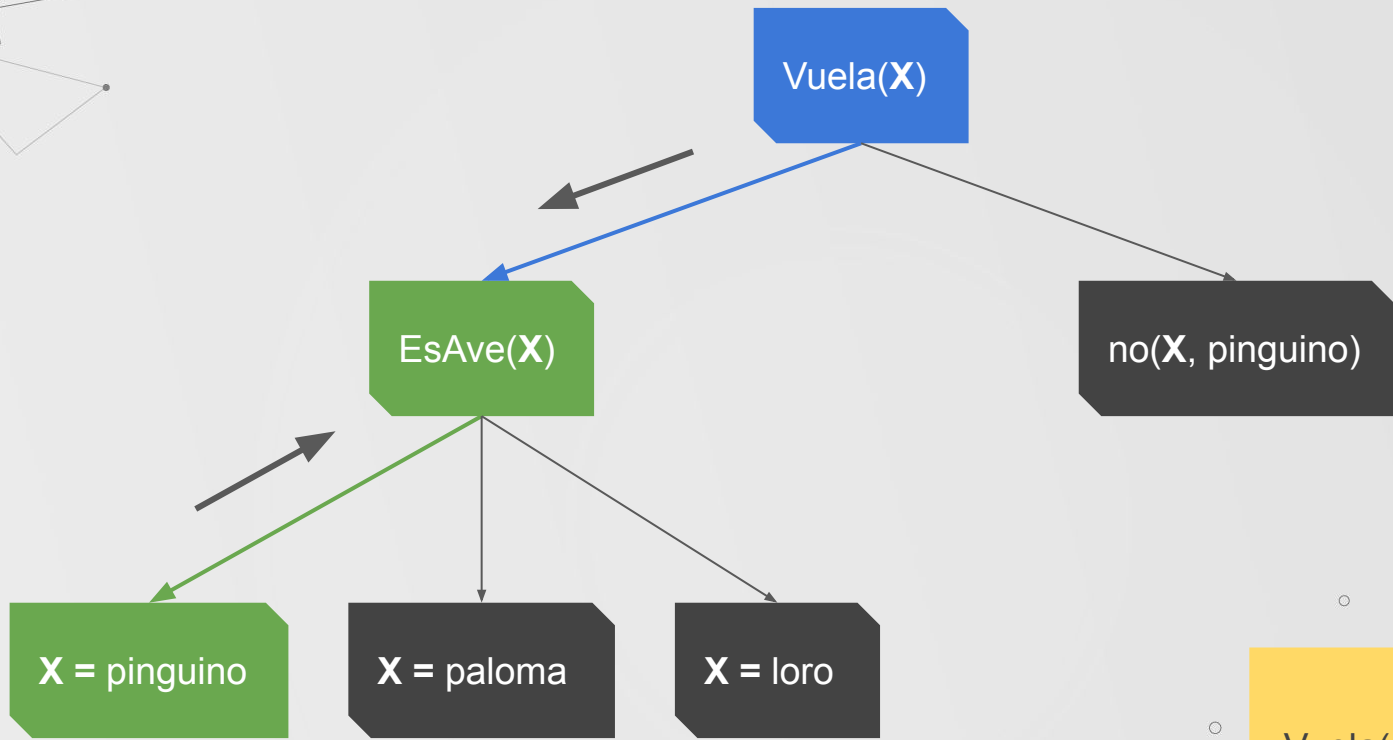


Vuela(pinguino)

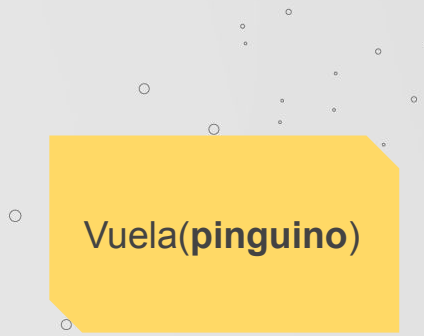
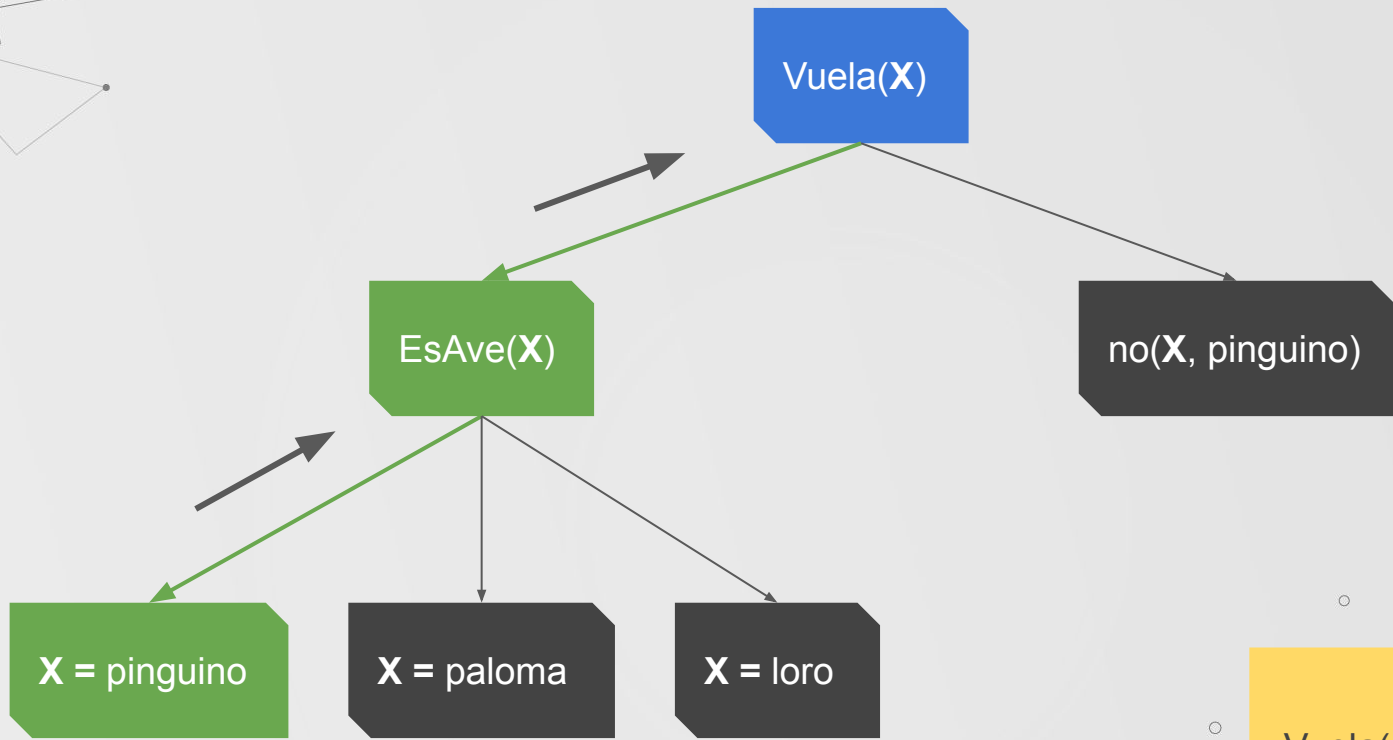
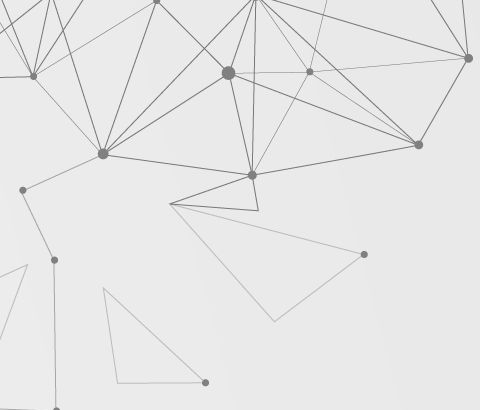


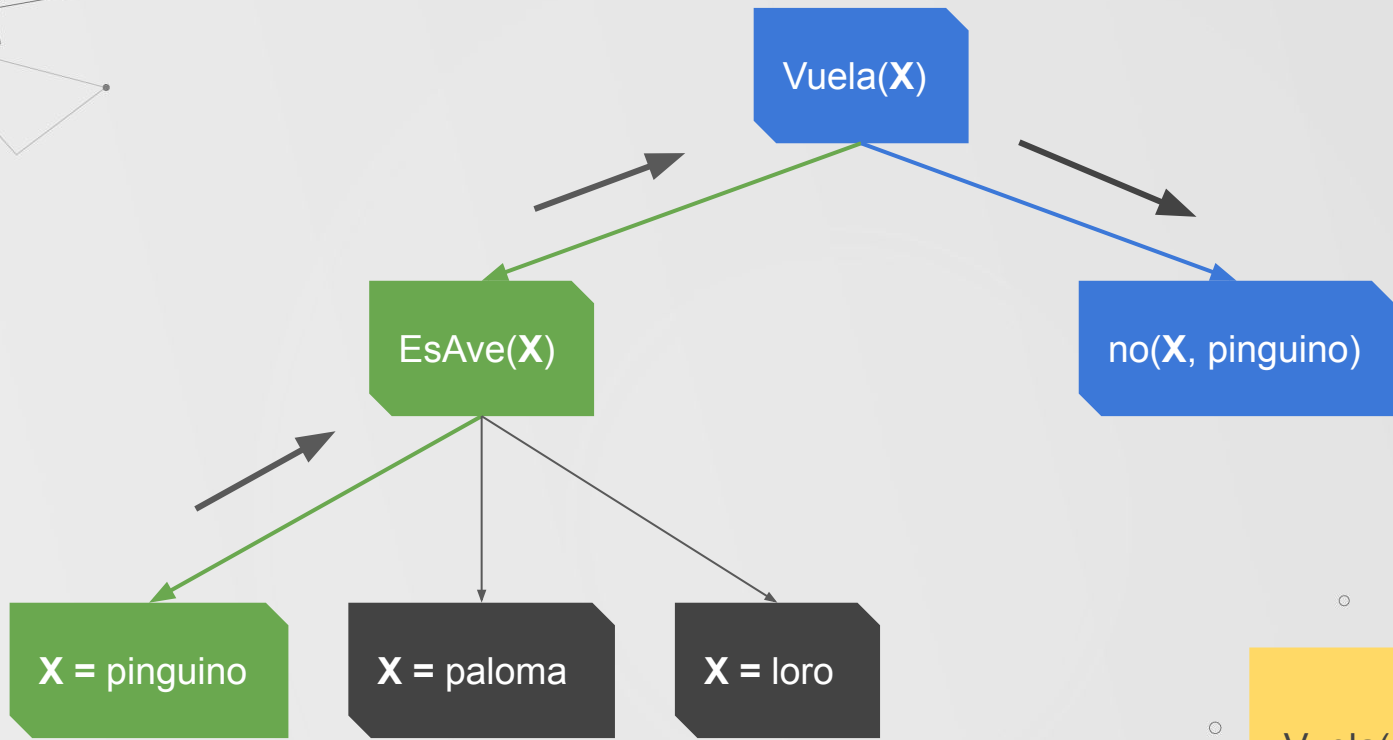
Vuela(pinguino)



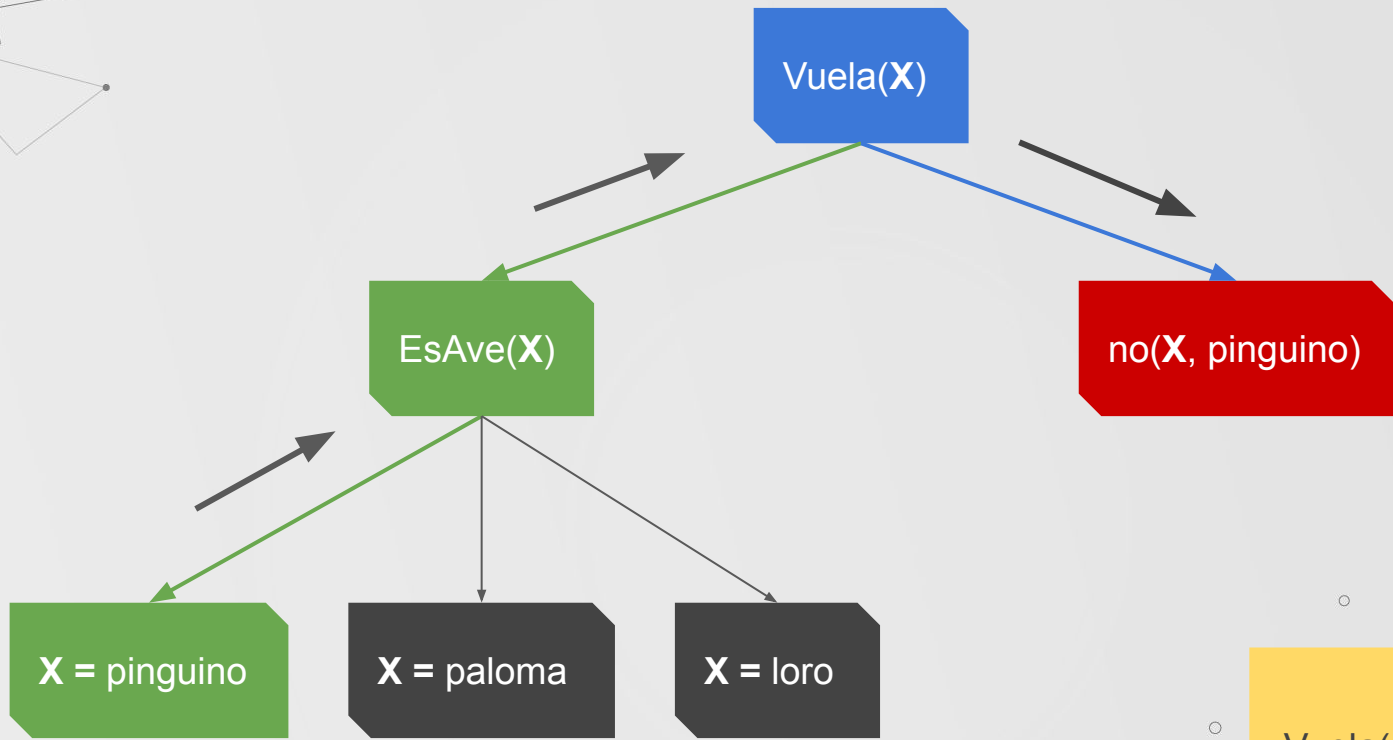


Vuela(pinguino)

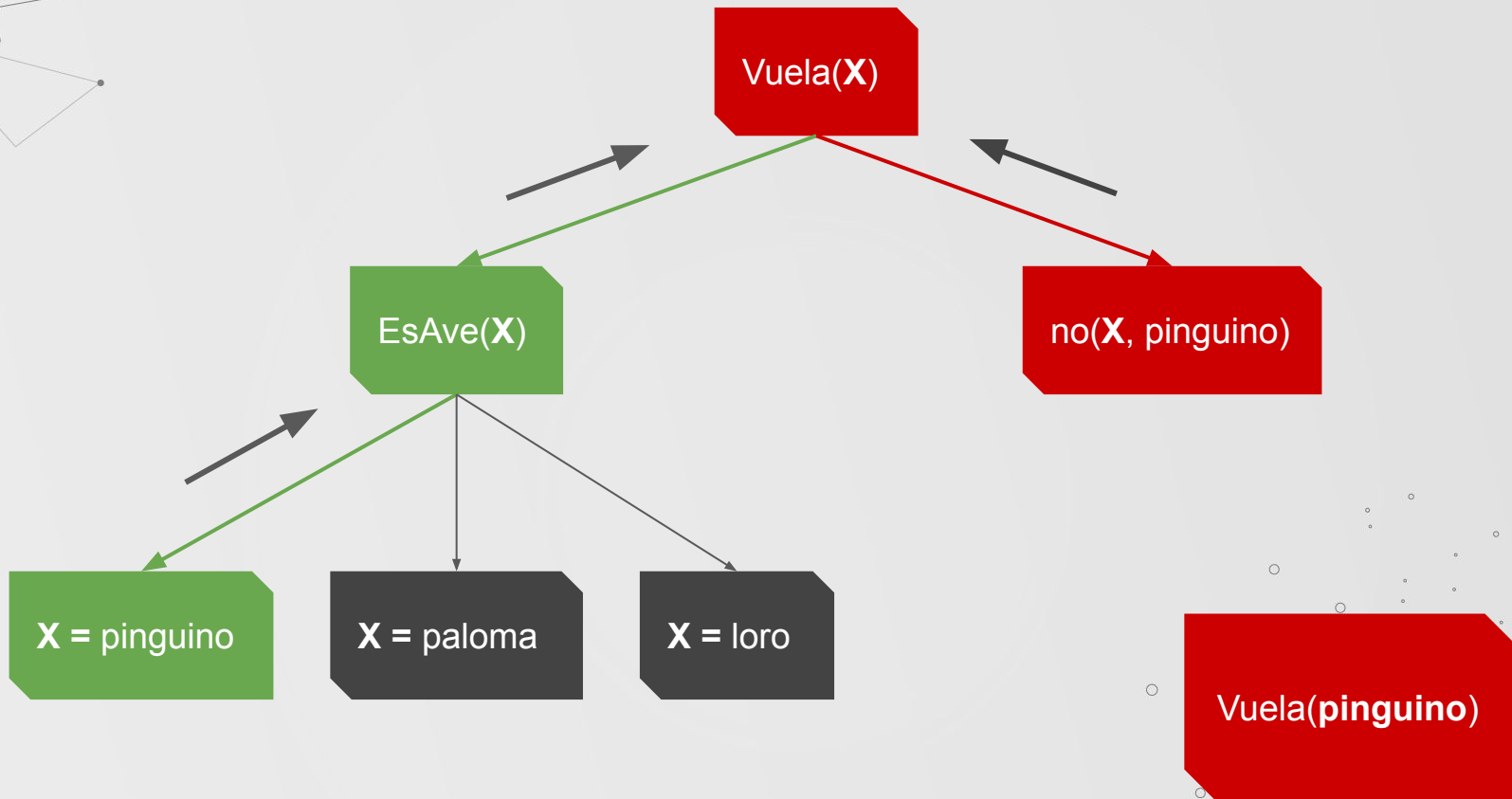
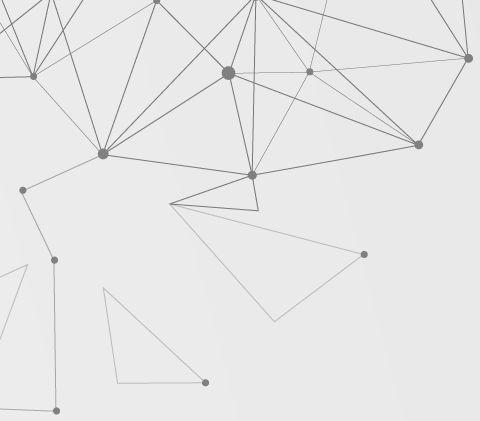


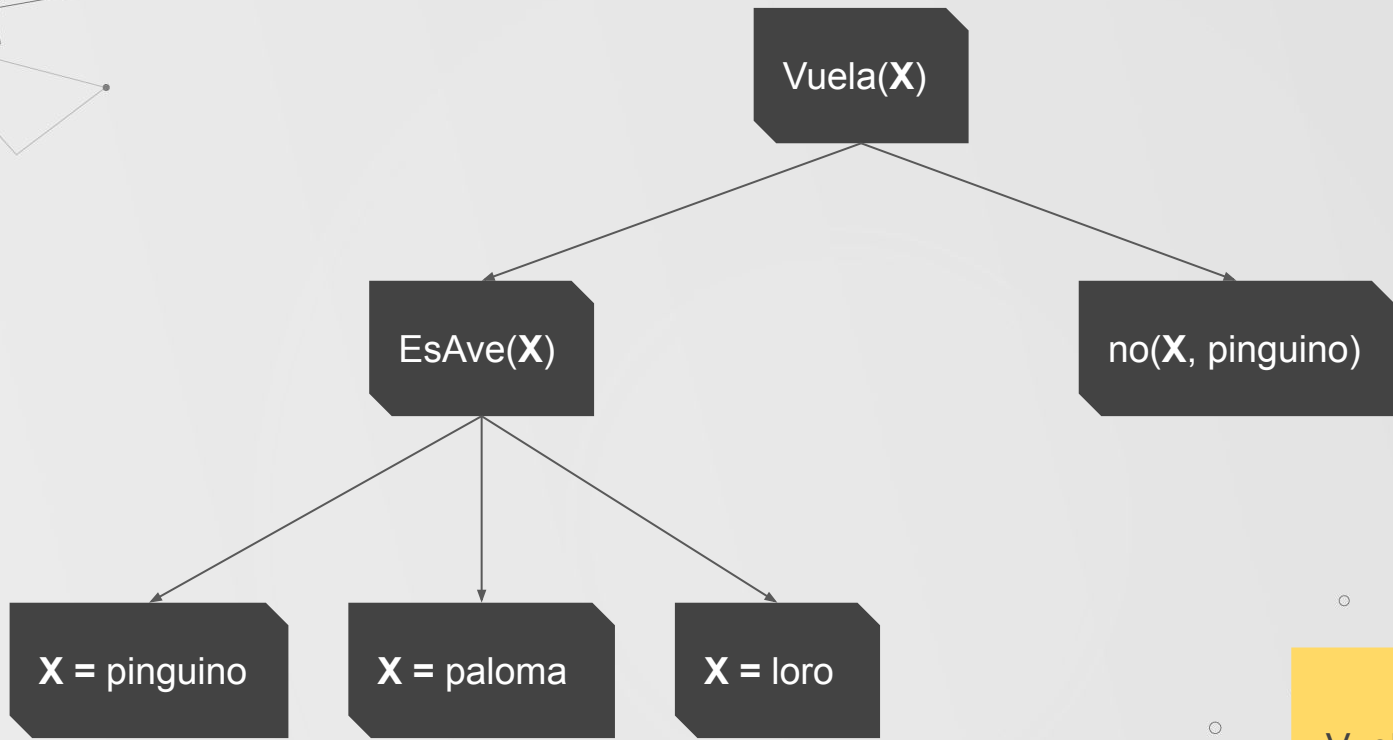


Vuela(pinguino)

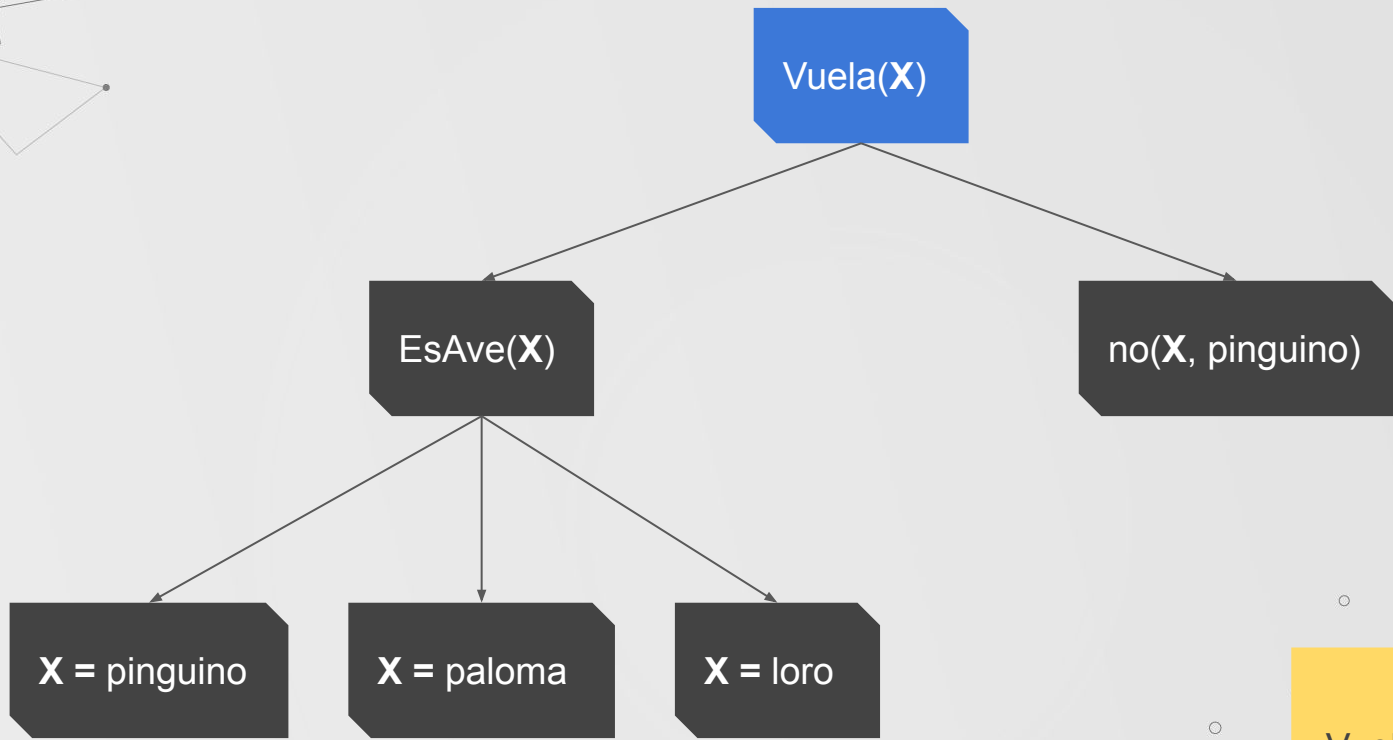


Vuela(pinguino)

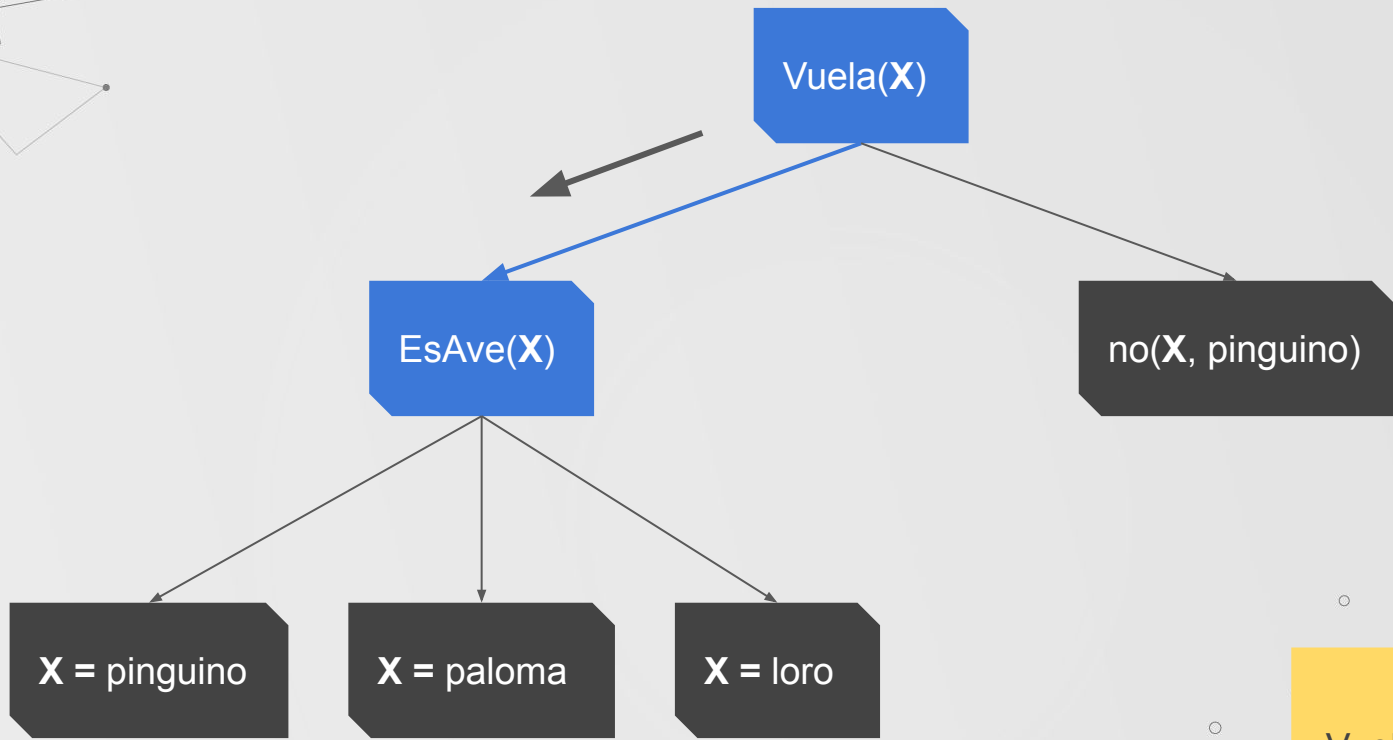


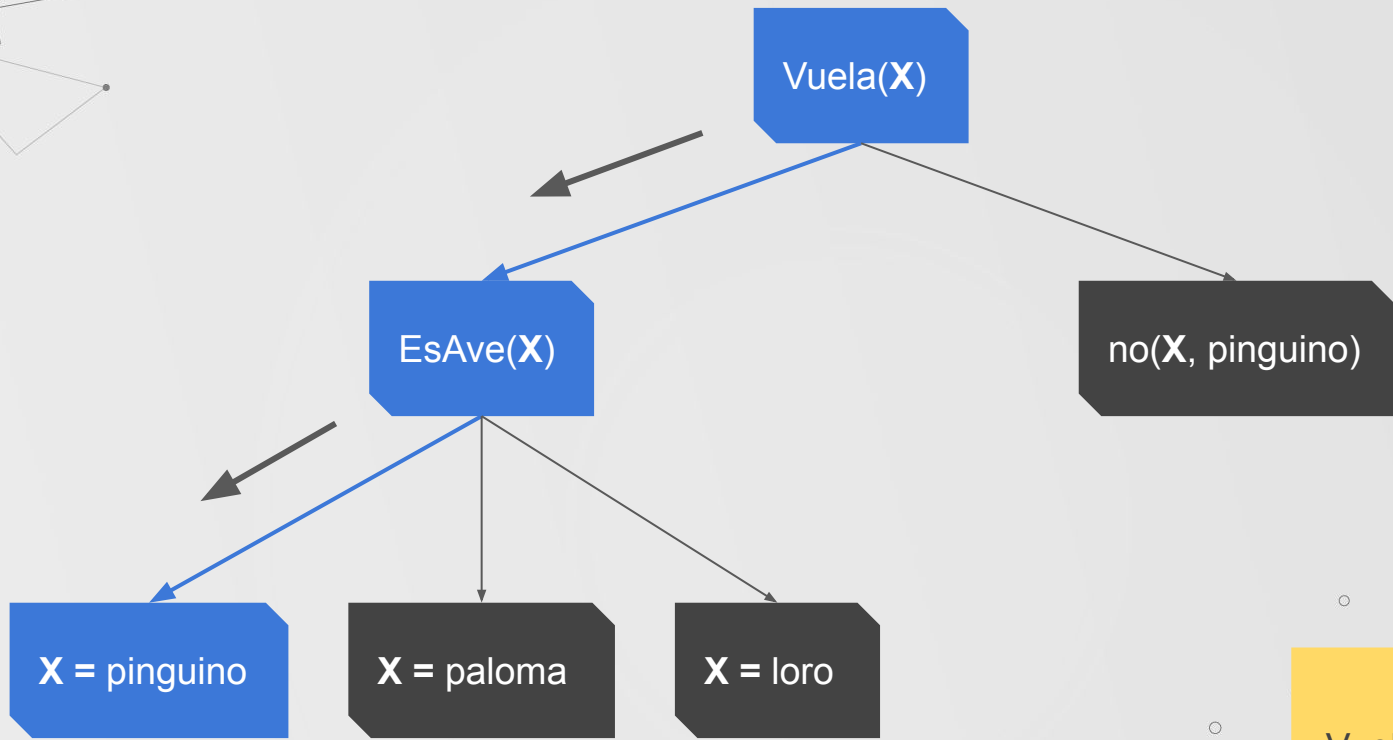


Vuela(agUILA)

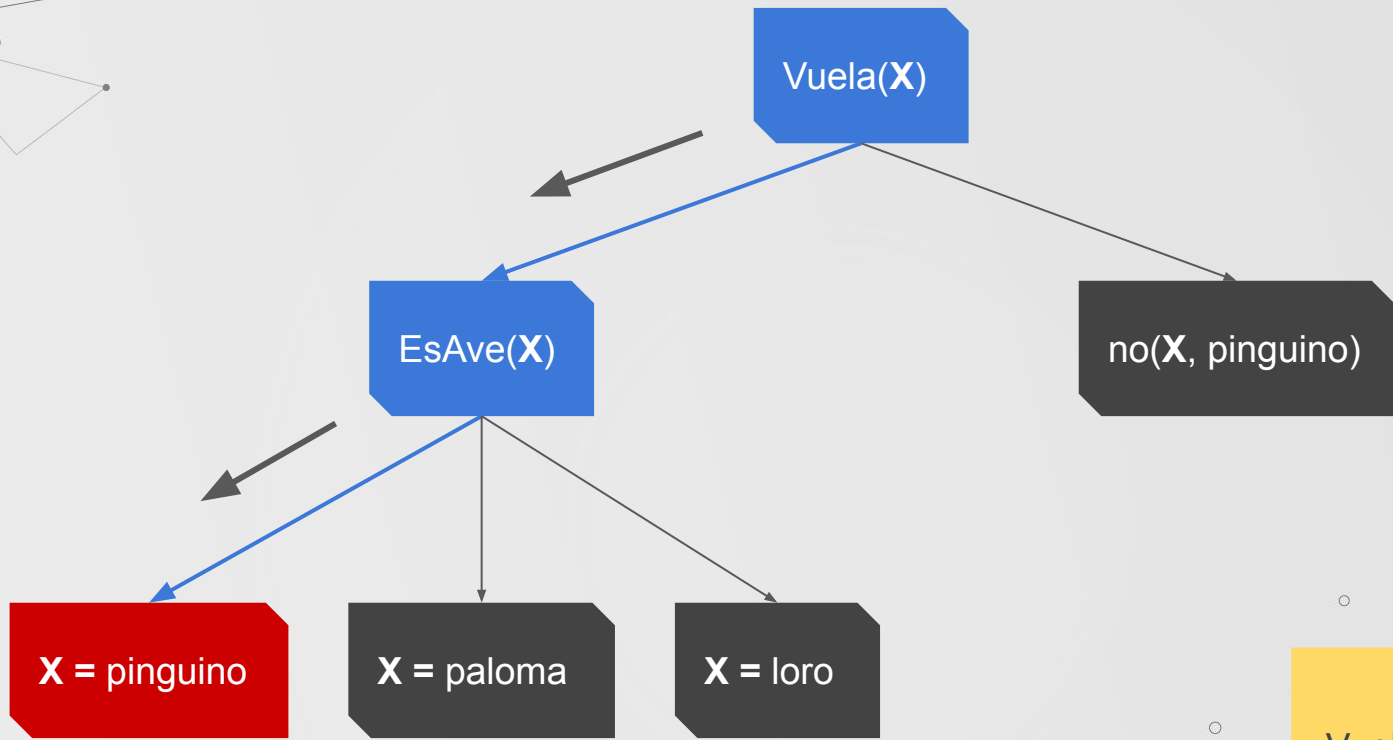
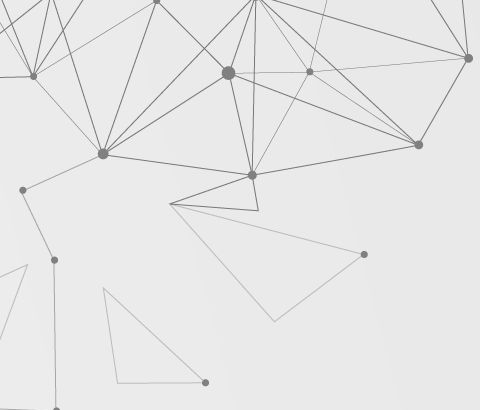


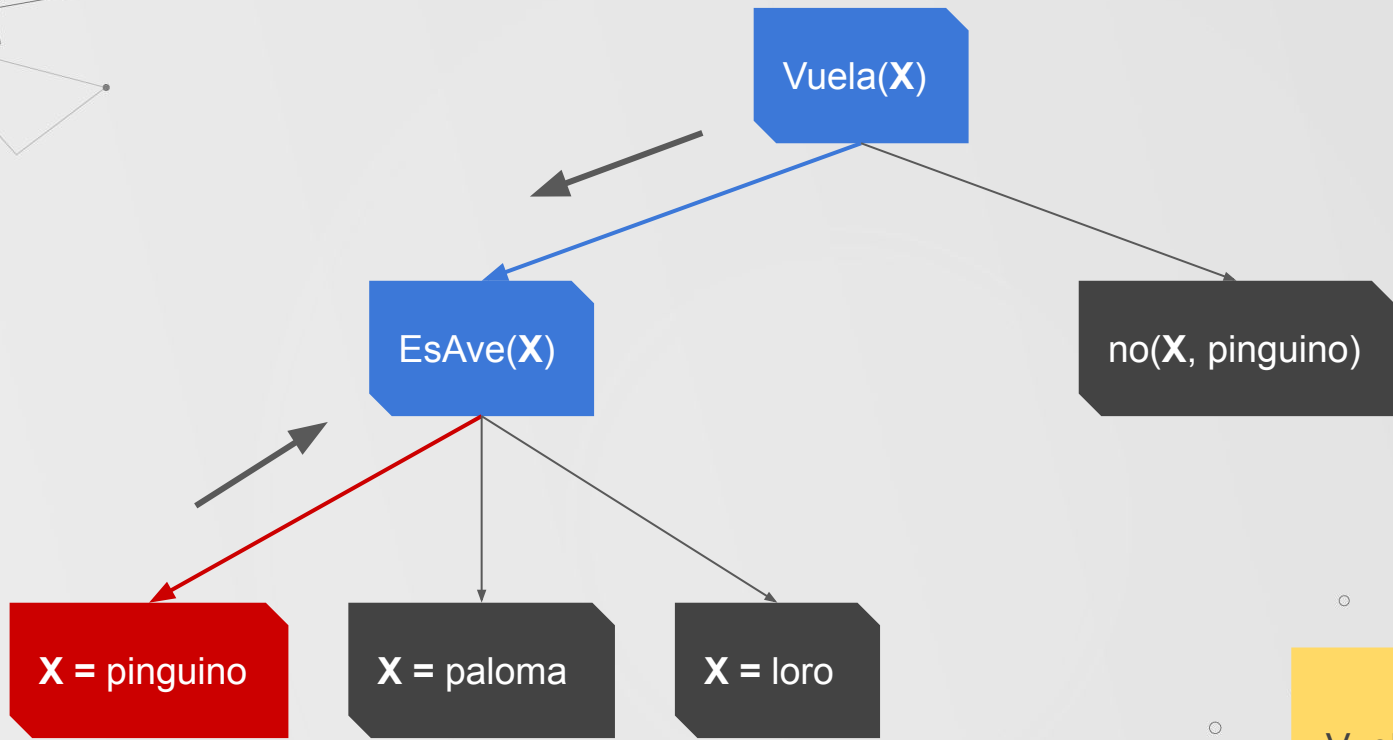
Vuela(agUILA)

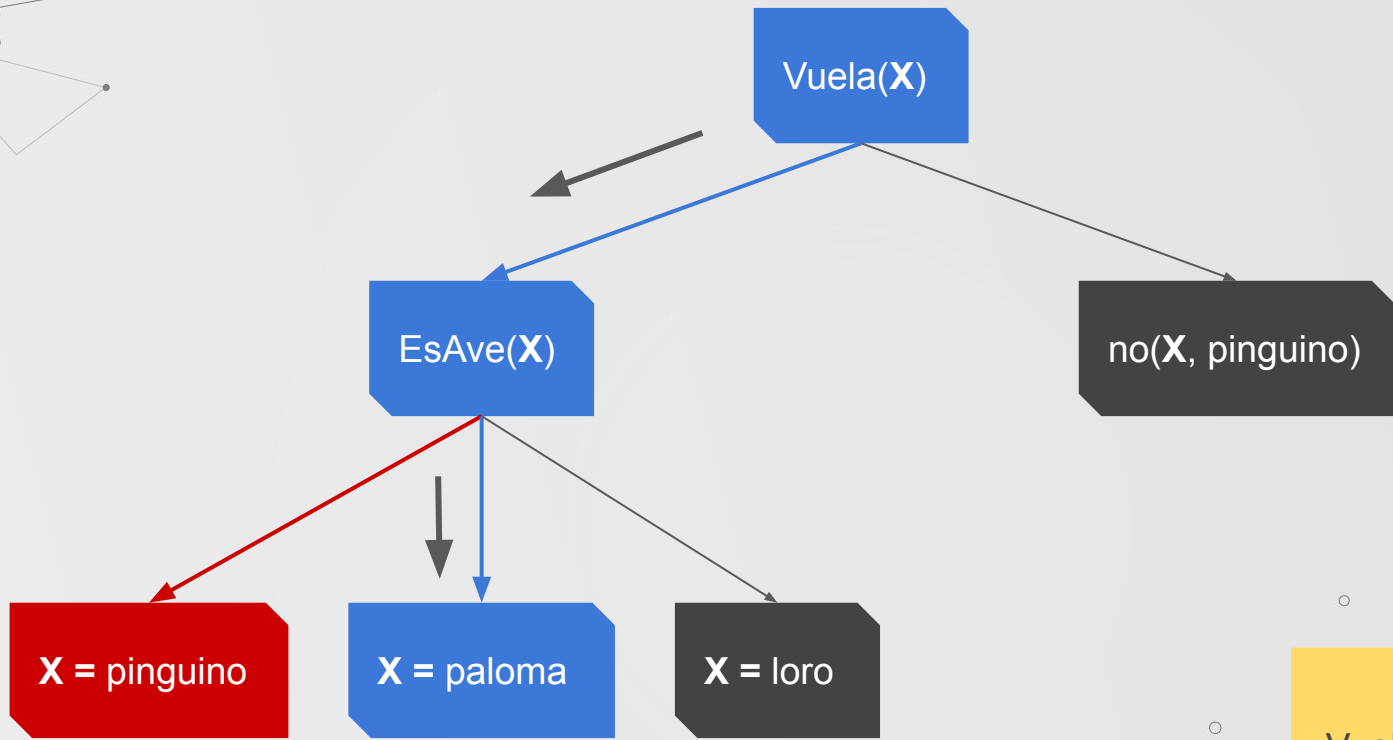


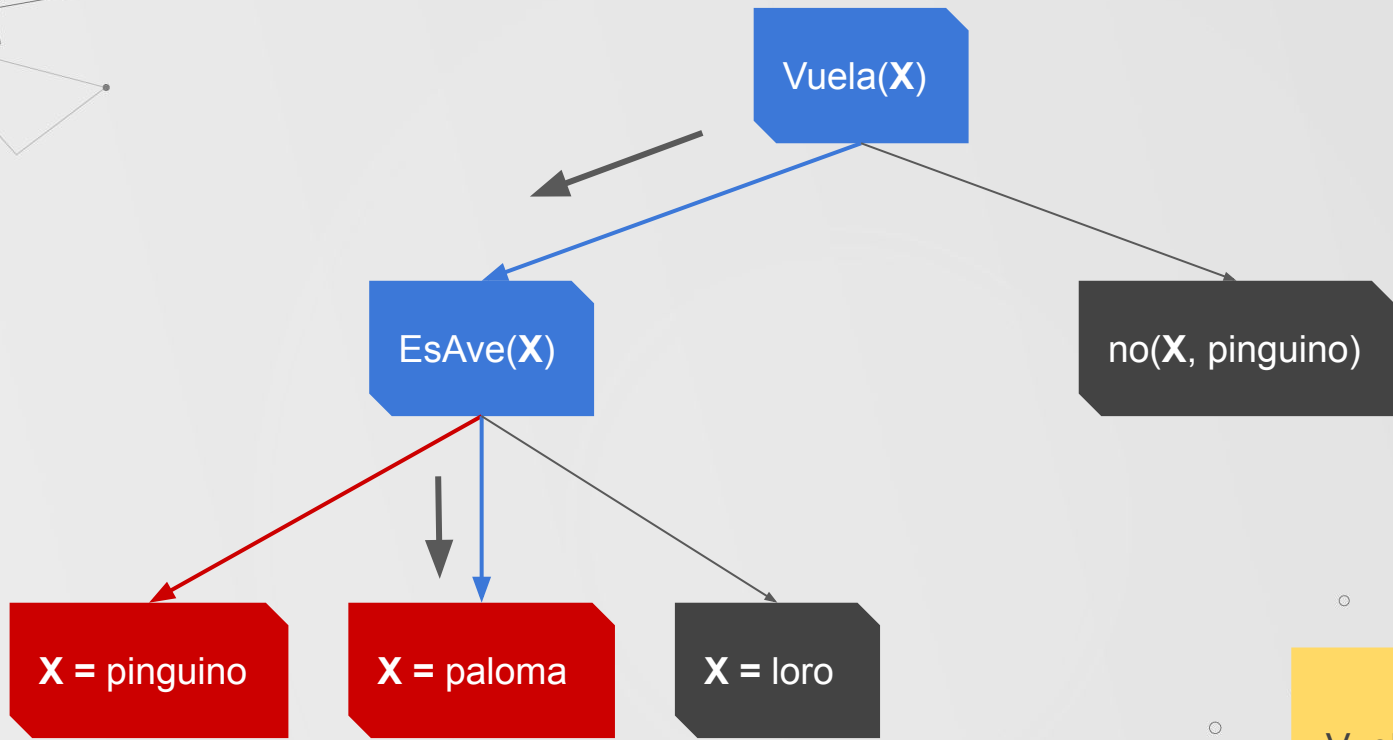


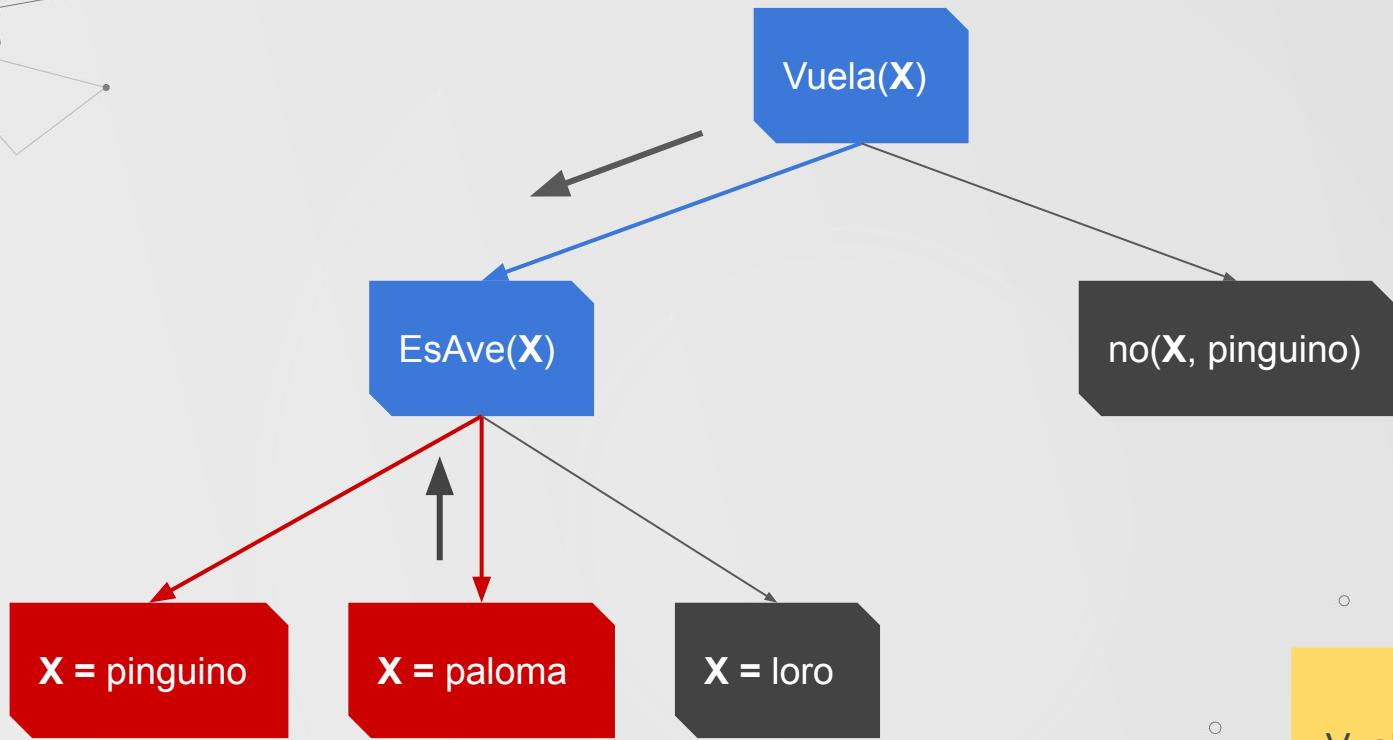
Vuela(agUILA)

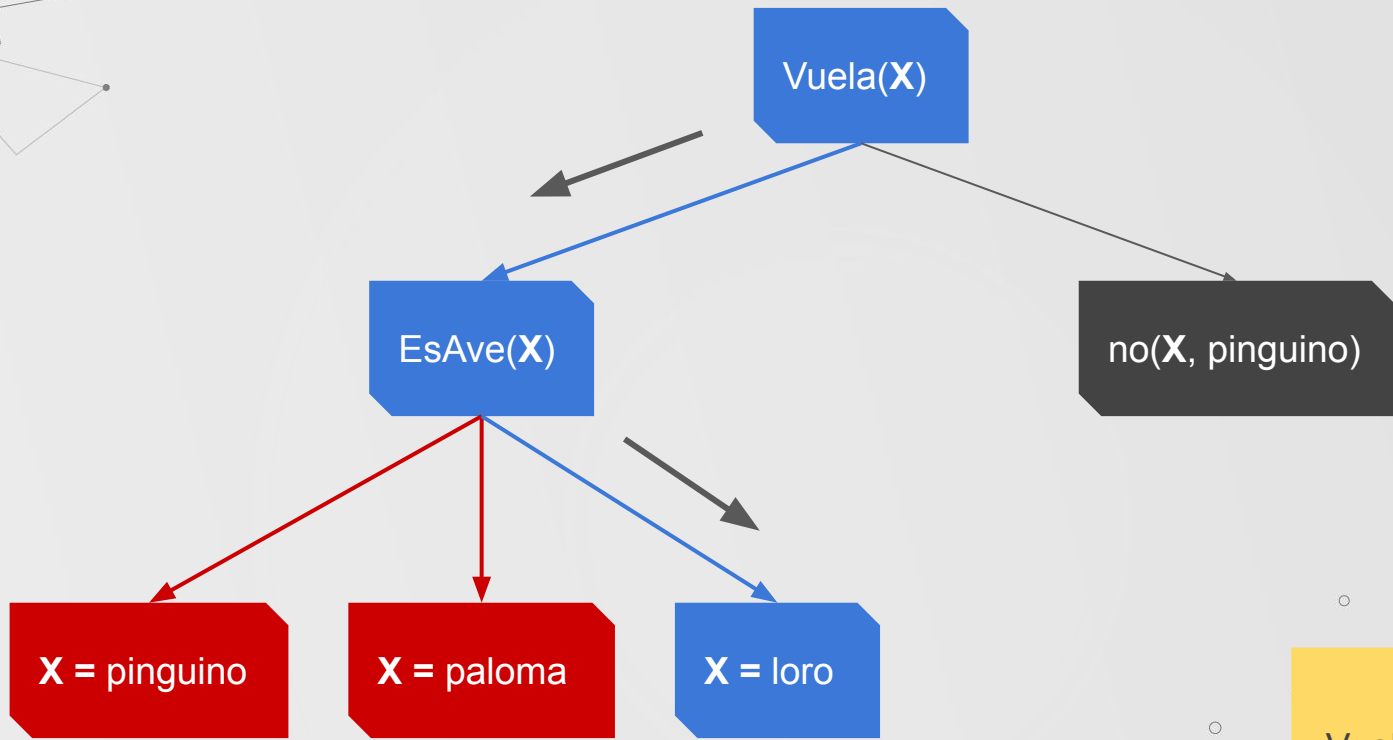
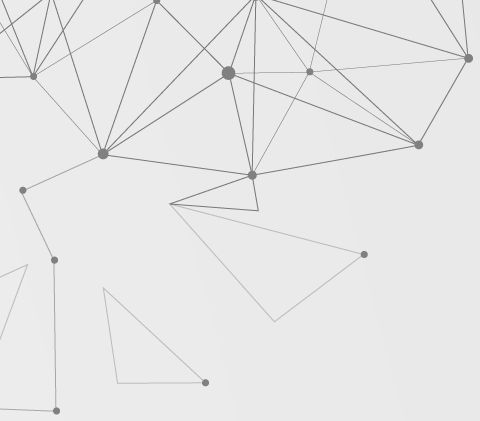


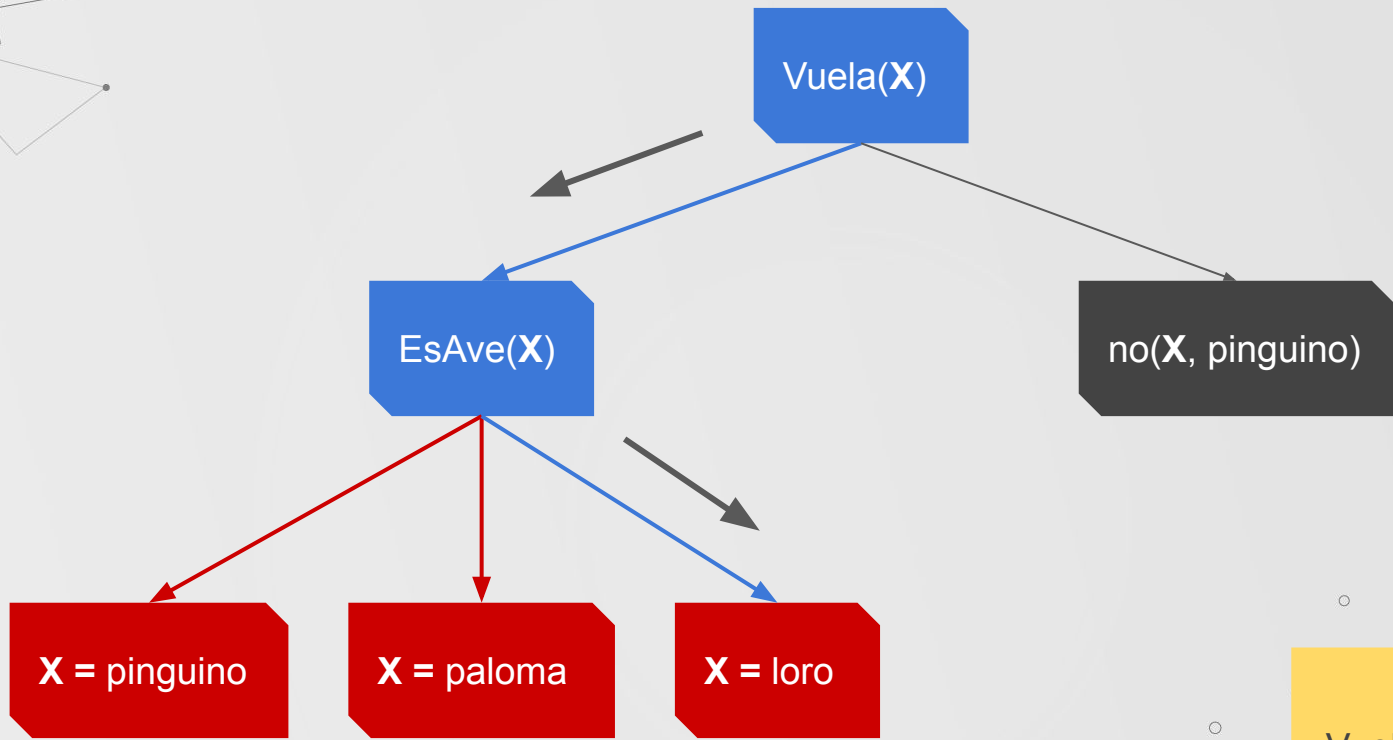


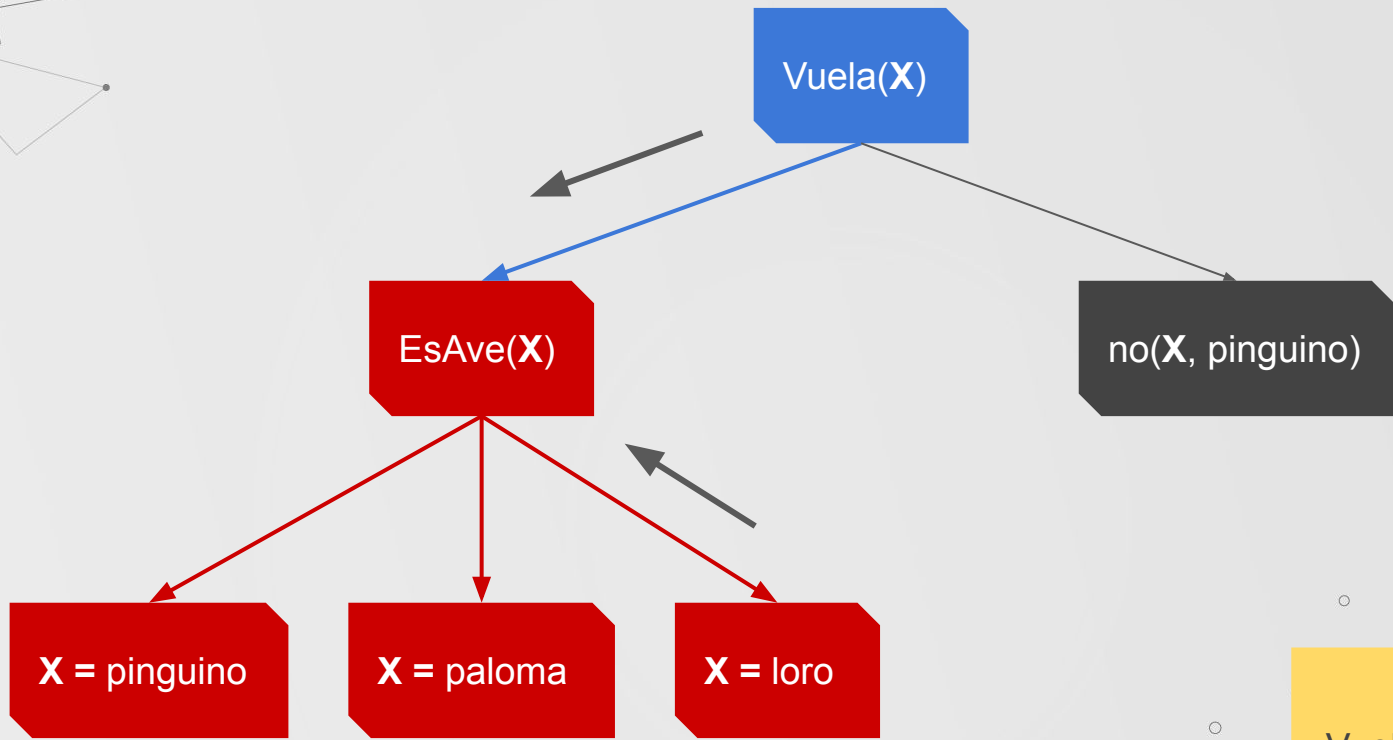


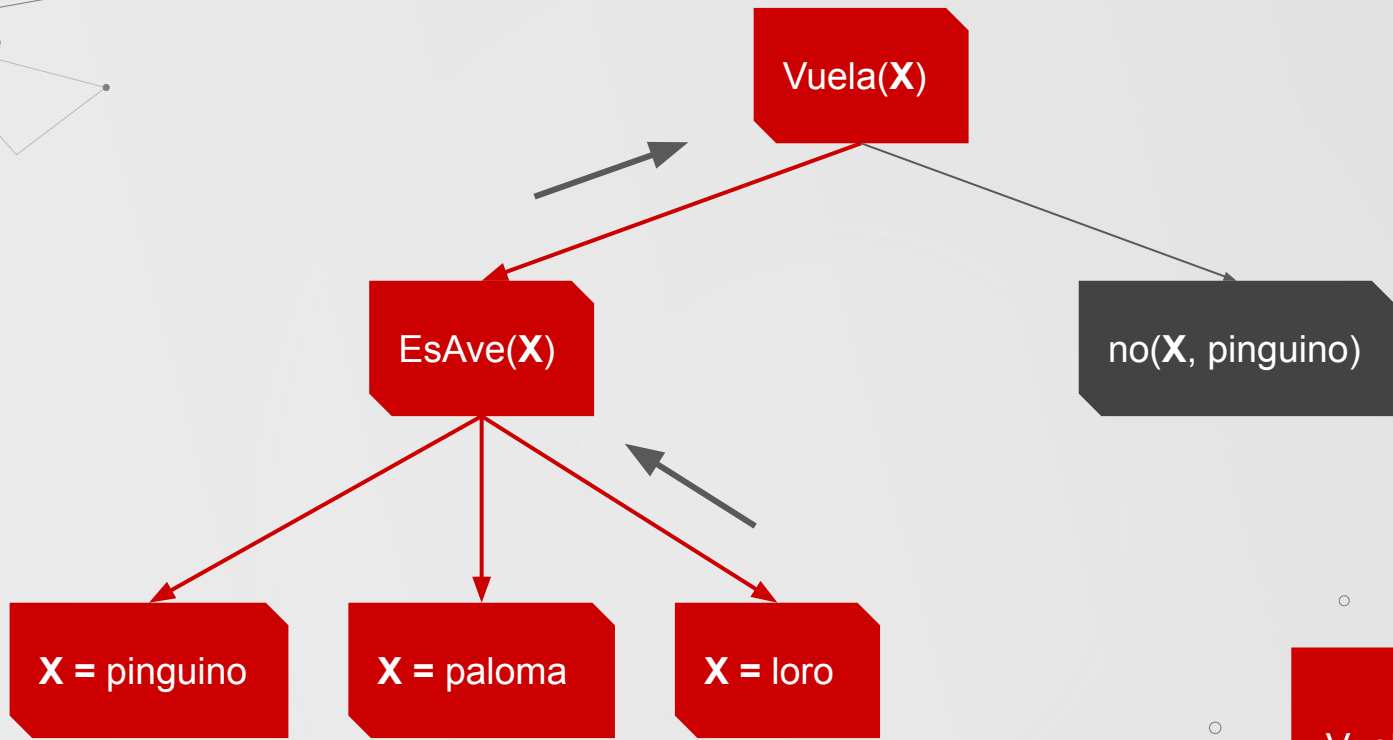
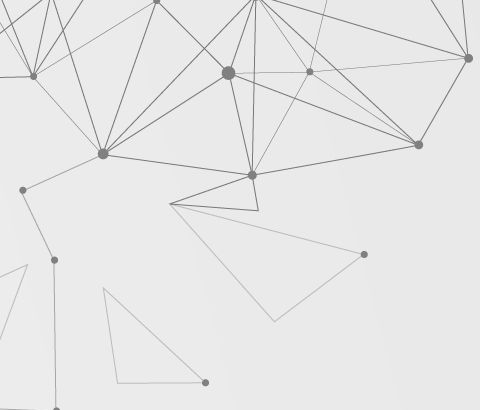












RECURSION

Especificación de un ALGORITMO basado en su propia definición.

Permite obtener resultados a partir de la definición de un caso base y llamadas recursivas

$$\text{factorial}(0) = 1$$
$$\text{factorial}(N) = N * \text{factorial}(N-1)$$



04

VENTAJAS Y DESVENTAJAS



VENTAJAS



Expresiones simples y precisas de los problemas



Optimización al modificar el componente de control sin modificar la lógica del algoritmo



La base de conocimiento es fácilmente escalable.



Sencillez en la implementación de estructuras complejas



DESVENTAJAS



Su Inferencia está limitada totalmente por su base de conocimiento.



No existen herramientas de depuración efectivas.



Áreas de aplicación muy específicas



No resulta muy adecuada para representar ciertos problemas del mundo real

The background features a complex network of thin grey lines connecting various-sized dark grey circular nodes. The nodes are scattered across the page, with a higher density on the right side. Some nodes are larger than others, and the lines form a web-like structure. There are also some faint, larger geometric shapes like triangles and a circle in the background.

05

**LENGUAJES DE
PROGRAMACIÓN
LÓGICA**

PROLOG

- Es el lenguaje más representativo de la programación Lógica
- Se basa en lógica de primer orden
- Está basado en cláusulas de Horn
- Es un lenguaje de programación declarativo



PROLOG

```
1 %---Axiomas y predicados
2 padre(antonio,jesus).
3 padre(jesus,mauricio).
4 padre(jesus,david).
5 padre(mauricio,cristian).
6 padre(david,esteban).
7 padre(david,gustavo).
8 %---Reglas
9 hijo(X,Y) :- padre(Y,X).
10 abuelo(X,Y) :- padre(X,Z), padre(Z,Y).
11 hermano(X,Y) :- padre(Z,X), padre(Z,Y), X \== Y.
12 tio(X,Y) :- padre(Z,Y), hermano(Z,X).
13 sobrino(X,Y) :- tio(Y,X).
14
```

```
≡ ?- hermano(mauricio,david).
```

```
true
```

```
≡ ?- sobrino(esteban,mauricio).
```

```
true
```

Notebook : <https://swish.swi-prolog.org/p/EjemploProLog.swinb>

EJEMPLO 2

```
1  % Hechos:
2  es_español("Manolo").
3  es_italiano("Marco").
4  es_colombiano("Marcelo").
5
6  % Reglas:
7  es_europeo(A) :- es_español(A).
8  es_europeo(A) :- es_italiano(A).
9  es_americano(A) :- es_colombiano(A).
10 es_terricola(A) :- es_europeo(A).
11 es_terricola(A) :- es_americano(A).
12 son_del_mismo_continente(A,B) :- es_europeo(A), es_europeo(B).
13 son_del_mismo_continente(A,B) :- es_americano(A), es_americano(B).
```

EJEMPLO 2

```
1 | ?- son_del_mismo_continente("Manolo", "Marco").  
2 | yes
```

```
1 | ?- es_europeo(A).  
2 | A = Manolo  
3 | A = Marco
```

DATALOG (PYTHON)

- Puede ejecutar consultas lógicas en bases de datos u objetos de Python, y usar cláusulas lógicas para definir clases de python
- Las declaraciones de Datalog se pueden especificar en cualquier orden
- Es ideal para implementar un sistema experto.



DATALOG (PYTHON)

```
1 from pyDatalog import pyDatalog
2 from pyDatalog.pyDatalog import create_terms as terms
3 from pyDatalog.pyDatalog import ask
4
5 pyDatalog.create_terms('scale') # the long way of doing it
6 terms('A, B, C, V')
7
8 scale['meter', 'inch'] = 39.3700787
9 scale['mile', 'inch'] = 63360.0
10 scale['feet', 'inch'] = 12.0
11
12 scale[A, B] = 1/scale[B, A]
13 scale[A,B] = scale[A,C] * scale[C, B]
14
15 print(scale['inch', 'meter'] == V)
16 print(scale['mile', 'meter'] == V)
17
18 terms('conv')
19 conv[V, A, B] = V * scale[A, B]
20 print(conv[3, 'mile', 'meter'] == V)
21 print(conv[1, 'meter', 'feet'] == V)
```

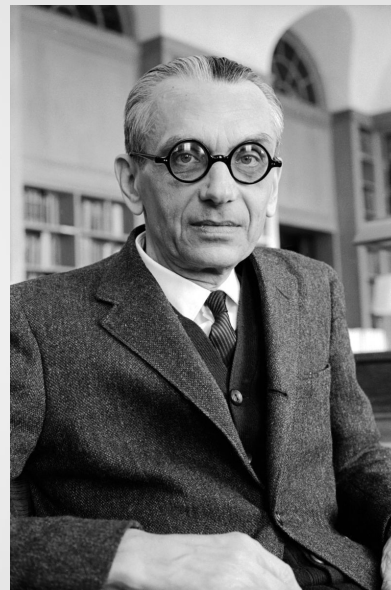
```
In [5]: print(scale['inch', 'meter'] == V)
V
-----
0.025400000025908
```

```
In [6]: print(scale['mile', 'meter'] == V)
V
-----
1609.3440016415307
```

```
In [9]: print(conv[3, 'mile', 'meter'] == V)
V
-----
4828.032004924591
```

GÖDEL (programming language)

- Es un lenguaje de programación declarativo (Aún más que Prolog).
- Es fuertemente tipado.
- Las sentencias lógicas llevan un orden.
- Existe el polimorfismo.
- Es un buen lenguaje para tareas de cómo compilación, depuración, análisis, verificación o transformación de programas
- No funciona en un entorno Windows



Kurt Gödel

GÖDEL (programming language)

```
MODULE      GCD.
IMPORT      Integers.

PREDICATE   Gcd : Integer * Integer * Integer.
Gcd(i,j,d) <-
  CommonDivisor(i,j,d) &
  ~ SOME [e] (CommonDivisor(i,j,e) & e > d).

PREDICATE   CommonDivisor : Integer * Integer * Integer.
CommonDivisor(i,j,d) <-
  IF (i = 0 \ / j = 0)
  THEN
    d = Max(Abs(i),Abs(j))
  ELSE
    1 <= d <= Min(Abs(i),Abs(j)) &
    i Mod d = 0 &
    j Mod d = 0.
```


06

APLICACIONES DE LA PROGRAMACIÓN LÓGICA



APLICACIONES

Sistemas Expertos

Inteligencia Artificial

Comprobación
automática de teoremas

Reconocimiento de
lenguaje natural.



GRACIAS!

¿Alguien tiene preguntas?

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

Please keep this slide for attribution.

BIBLIOGRAFIA

- <http://blog.koalite.com/2013/08/que-es-la-programacion-logica/>
- <http://diposit.ub.edu/dspace/bitstream/2445/64643/1/memoria.pdf>
- https://ferestrepoca.github.io/paradigmas-de-programacion/proglogica/logica_teorja/lang.html
- <https://www.monografias.com/trabajos106/logica-primer-orden/logica-primer-orden.shtml>
- http://www.it.uc3m.es/jvillena/irc/practicas/estudios/Lenguajes_Logicos.pdf
- <https://swish.swi-prolog.org/p/EjemploProLog.swinb>
- [https://en.wikipedia.org/wiki/G%C3%B6del_\(programming_language\)](https://en.wikipedia.org/wiki/G%C3%B6del_(programming_language))

