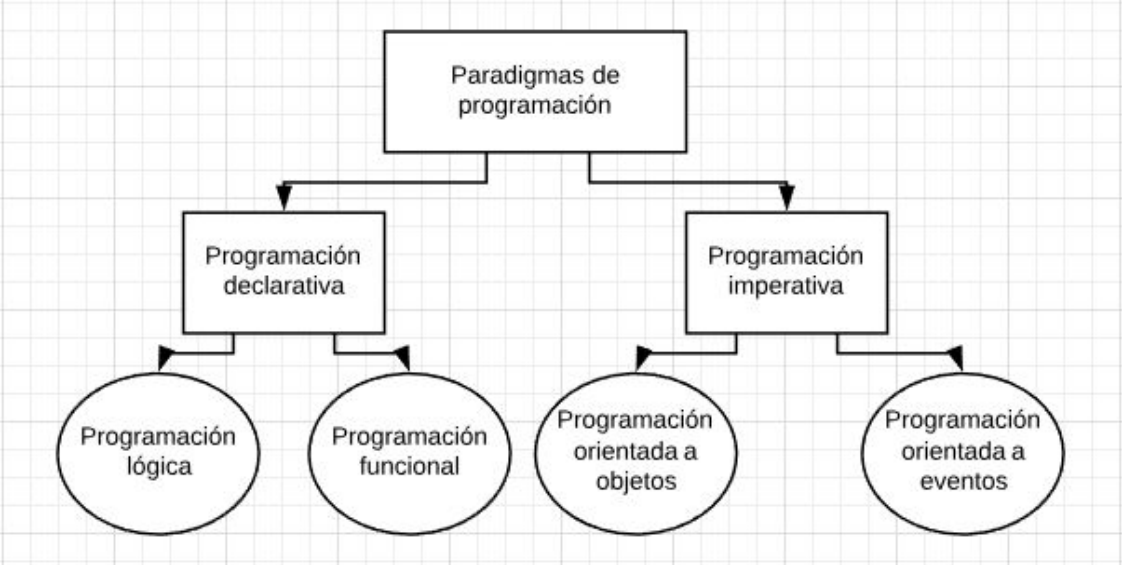
Programación Lógica

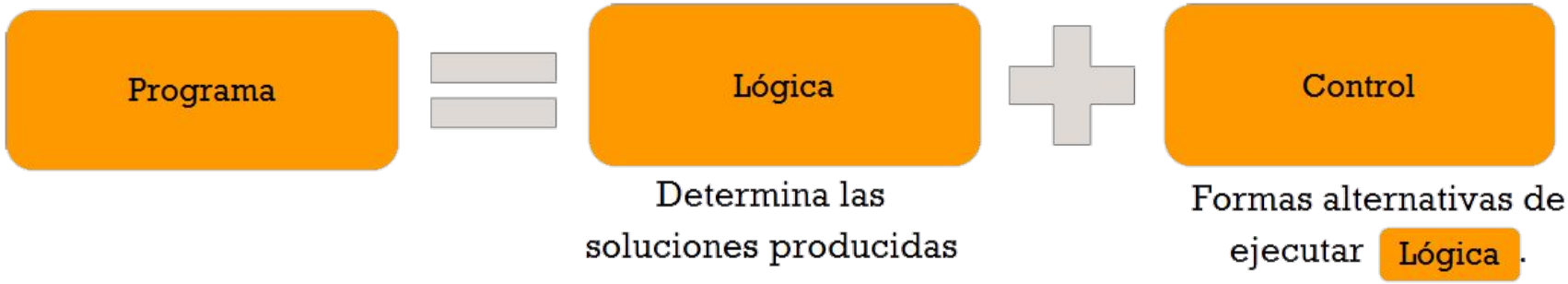
- Andrés Fernando Aranguren
- John Fredy Castro
- José Manuel Restrepo

Paradigmas de Programación

Un paradigma de programación representa un enfoque particular o filosofía para diseñar soluciones.

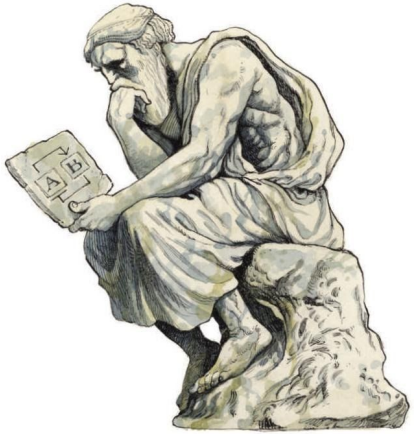


¿Qué es?


Programa
=
Lógica
Determina las soluciones producidas
+
Control
Formas alternativas de ejecutar Lógica.

Filosofía del Paradigma

Modelar problemas por medio de la abstracción, utilizando un sistema de lógica formal que permite llegar a una conclusión por medio de hechos y reglas.



Lógica Proposicional

También llamada lógica de predicados, es un sistema diseñado para estudiar la inferencia en los lenguajes de primer orden. Toma como elemento básico las frases declarativas simples o proposiciones.

Proposiciones: Elementos de una frase que constituyen por sí solos una unidad de comunicación de conocimientos y pueden ser considerados verdaderos o falsos, que puede ser atómica/simple o compuesta.

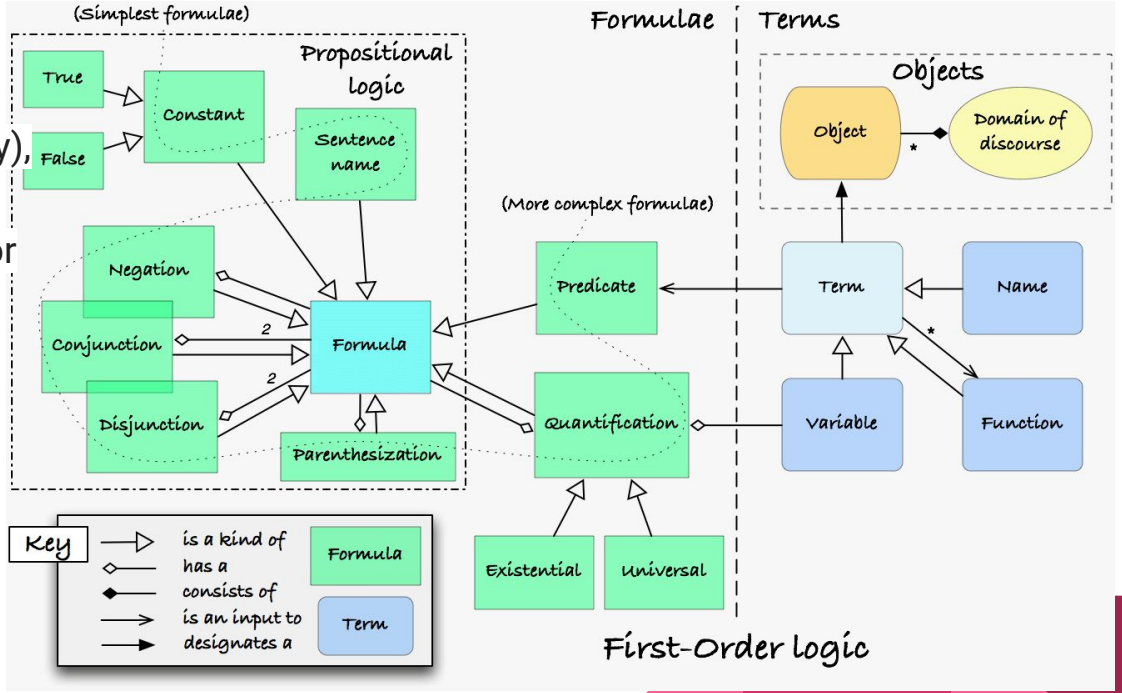
Proposición Simple: Estamos en paro.

Proposición compleja: Estamos en paro y no tenemos clase.

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*

Lógica de primer orden

Variables: u, v, w, x, y, z, foo.
Constantes: a, b, c, mary, verde.
Símbolos de función: padreDe (Mary), esAzul(cielo).
Símbolos de predicado: p, q, r, mayor (5,3), color (hierba, verde).
Conectivos: NO, Y, O, IMPLICA, SI Y SOLO SI., ↔ → ∨ ∧ ¬
Cuantificadores: Para todos (universal), existe (existencial). (universal) ∀ , (existential) ∃
Símbolos de puntuación: ')', '('



Lógica de primer orden

Aquí hay algunas definiciones de varias piezas de gramática informal utilizadas en conjunto con un alfabeto, al construir un lenguaje lógico de predicado de primer orden:

- Un término p.ej. a, X, f (b, f (b, Y, Z), Y)
- Una fórmula atómica p.ej. me gusta (X, sim), pide prestado (libro, sim)
- Una fórmula bien formada W, ¬ W, W1 y W2, W1 o W2, W1 IMPLICA W2, W1 SI Y SOLO SI W2
- Una sentencia es una fórmula bien formada en la que cada aparición de cada símbolo variable está vinculada.
- Un literal es una fórmula atómica o una fórmula atómica negada p.ej.¬likes(sim, pain)
- Un término básico es un término que no contiene símbolos variables.
- Una fórmula básica es una fórmula que no contiene símbolos variables.
- Las fórmulas atómicas son comúnmente llamadas predicados.

Cláusulas de Horn

Una fórmula lógica es cláusula de Horn si una secuencia de literales contiene a lo sumo uno de sus literales positivos.

Un ejemplo de una cláusula de Horn puede ser esta ¬ p v ¬ q v . . . v ¬ t v u que puede ser reescrita como (p ∧ q ∧ . . . ∧ t) ➡ u

Cláusula ‘definite’: Una cláusula de Horn con exactamente un literal positivo.

Hecho: Cláusula ‘definite’ sin literales negativos.

Cláusula objetivo: Sin ningún literal positivo. (consulta)

Cláusulas de Horn - Ejemplo

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

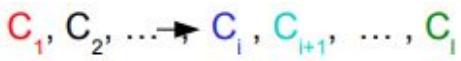
Estructura de cláusulas de Horn: (mujer(A) ∧ padre(B, A)) ➡ hija(A, B)

Prolog:

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

Resolución SLD

Es una resolución lineal con una función de selección para cláusulas ‘definite’, donde se parte del hecho de que una resolución de prueba puede ser restringida a una secuencia linear de cláusulas.



Donde la cláusula superior C1 , es una cláusula de entrada, y el resto de cláusulas Ci+1 es una solución de cuyos padres es la cláusula anterior Ci.

Recursión

Cuando una función se llama a sí misma, se llama recursión.
La recursión es útil para resolver problemas que pueden dividirse en problemas más pequeños del mismo tipo. Pero cuando se trata de resolver problemas usando recursión, hay varias cosas para tener en cuenta. Tomemos un ejemplo simple y tratemos de entenderlos.

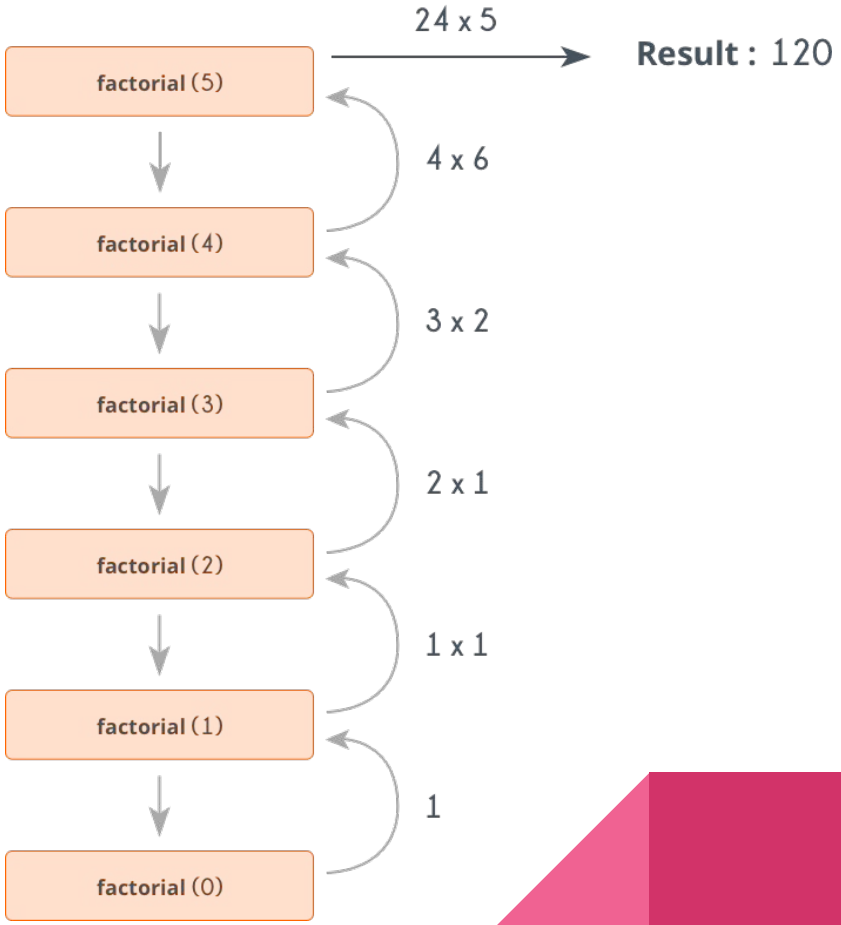
A continuación se muestra el pseudocódigo para hallar el factorial de un número dado mediante

```
function factorial(x)
    if x is 0                       // base case
        return 1
    return x*factorial(x-1)         // break into smaller problem(s)
```

Recursión

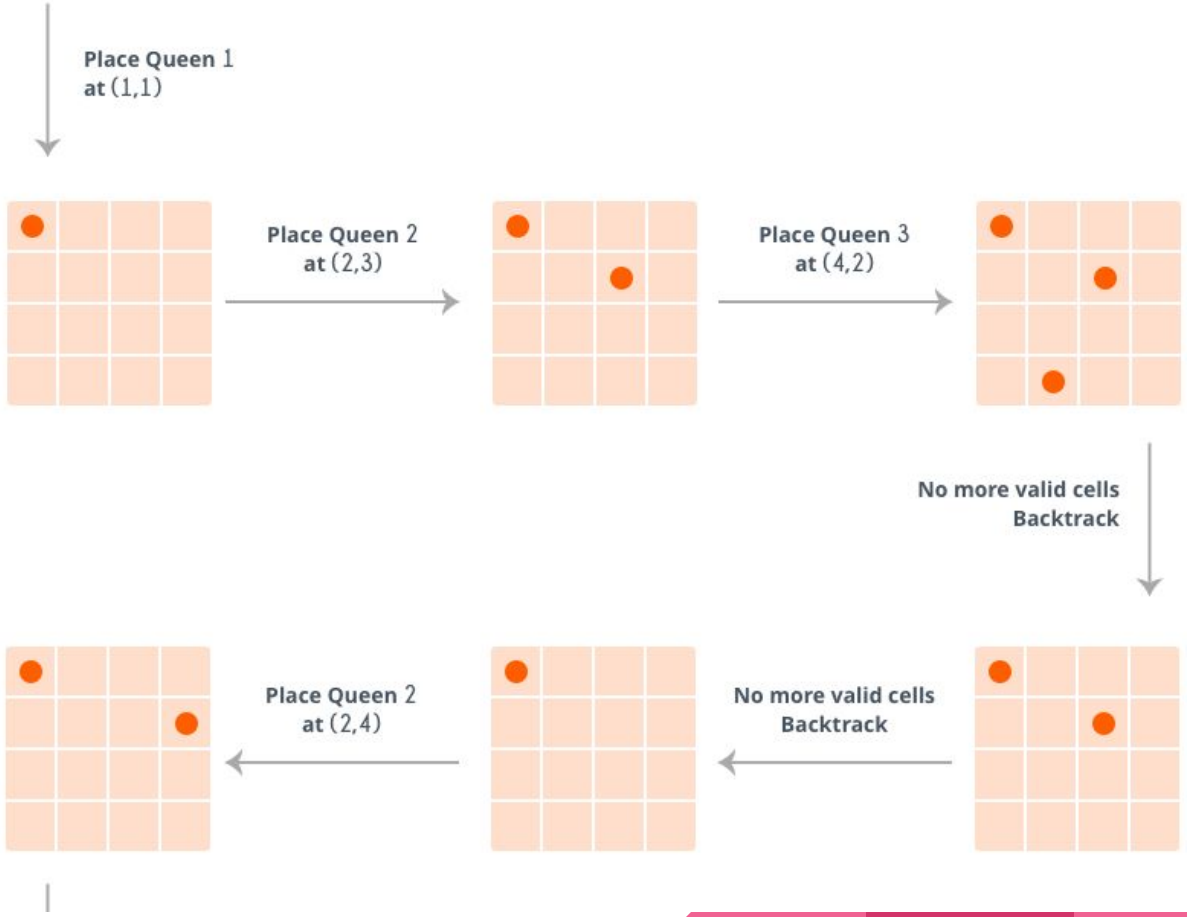
Solución gráfica al problema del factorial de 5 usando recursión.

Caso base: factorial (0) = 1



Backtracking

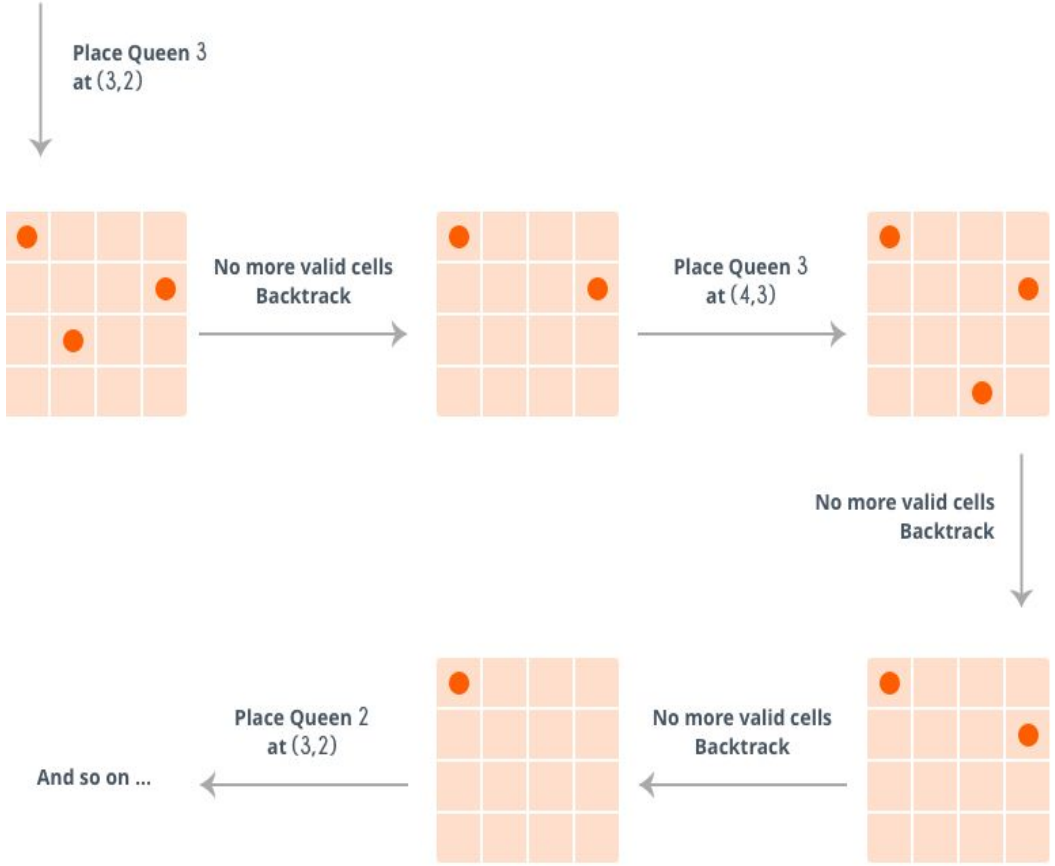
Tomemos un problema estándar.
Problema de las N-Reinas: dado un tablero de ajedrez que tiene NxN celdas, necesitamos colocar N reinas de tal manera que ninguna reina ataque a otra reina. Una reina puede atacar horizontalmente, verticalmente y diagonalmente.



Backtracking

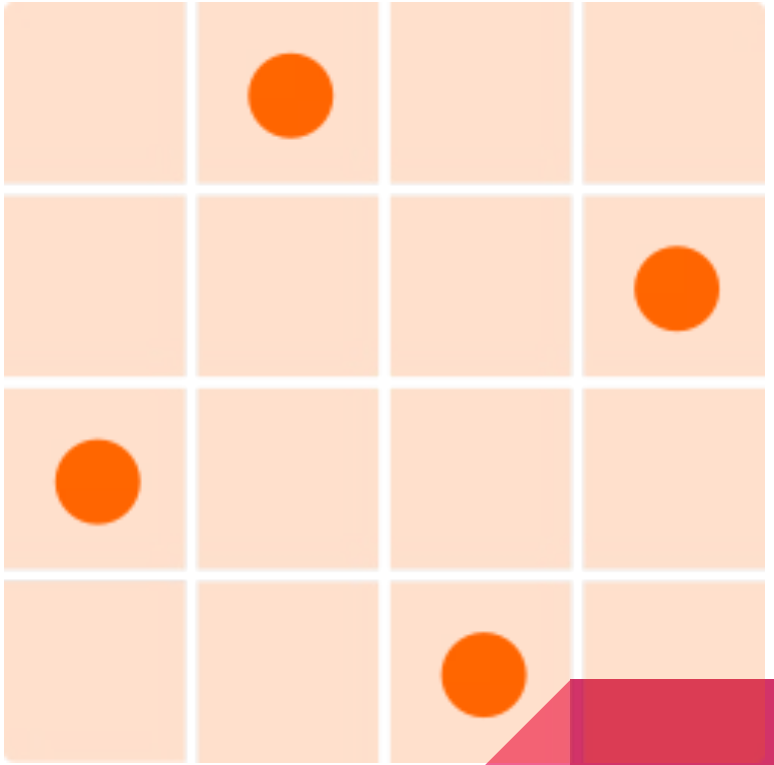
Así que inicialmente tenemos NxN celdas sin ataques donde necesitamos colocar N reinas. Pongamos la primera reina en una celda (i, j), de modo que ahora el número de celdas no atacadas se reduzca, y la cantidad de reinas que se ubicarán es N-1. Coloca la próxima reina en alguna celda no atacada. De nuevo, esto reduce el número de celdas sin ataque y el número de reinas que se colocarán se convierte en N-2. Continúa haciendo esto, siempre y cuando se cumplan las siguientes condiciones.

*El número de celdas sin ataque no es 0.
*El número de reinas a colocar no es 0



Backtracking

Si el número de reinas que se colocan se convierte en 0, entonces se terminó, encontramos una solución. Pero si el número de celdas desatendidas se convierte en 0, entonces debemos retroceder, es decir, eliminar la última reina colocada de su celda actual, y colocarla en alguna otra celda. Hacemos esto recursivamente.



Hechos

Son las sentencias más sencillas. Un hecho es una fórmula atómica o átomo: p(t1,...,tn)p(t1,...,tn) e indica que se verifica la relación (predicado) **p** sobre los objetos (términos) t1,...,tnt1,...,tn.

Los hechos definen relaciones incondicionales:

“Pikachu está sorprendido”

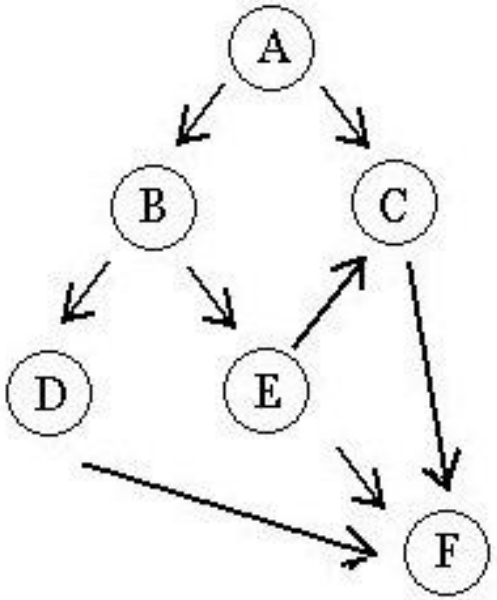
Sorprendido(pikachu)

Reglas

Una regla en programación lógica es un mecanismo que permite, utilizando un grupo de proposiciones base verdaderas, inferir otro grupo de nuevas proposiciones verdaderas.

L1. likes(x, programming) => loves(milhouse,x).

Esta regla puede verse como: “si a x le gusta programar entonces milhouse ama a x”



Consultas

Una consulta es una proposición que busca ser demostrada por medio de las reglas, utilizando unas proposiciones base.

- **likes(lisa, programming) ->** el hecho (h1).
- **likes(x,programming)=> loves(milhouse,x)**. sería la regla (L1)
- **loves(milhouse, lisa)? ->** consulta
- Tendríamos como conclusión que milhouse ama a lisa.

Algunos lenguajes para programación lógica

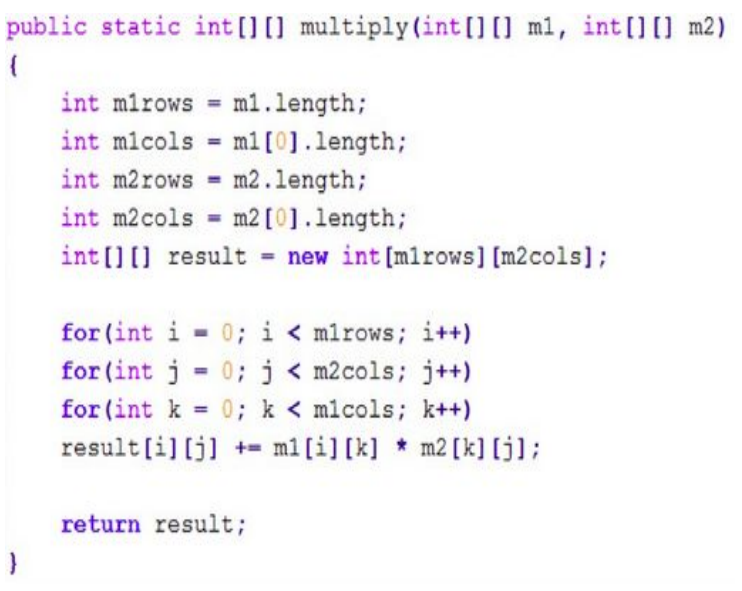
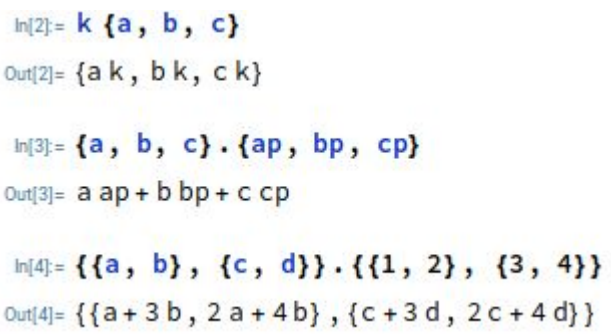
Prolog	SequenceL	Wolfram
• Basado en Lógica de primer orden • Usa cláusulas de Horn • Lenguaje de programación declarativo • Utilizado principalmente en inteligencia artificial y lingüística computacional	• Facilita para los usuarios la programación para la optimización en hardware de múltiples cores. • No necesita que el programador especifique que quiere paralelizar • Utilizado principalmente para computación paralela	• Permite utilizar y crear estructuras. • librerías para crear máquinas de turing, gráficos y audio, resolución de ecuaciones diferenciales. etc. • Utilizado principalmente en computación simbólica.

Prolog

Java	Prolog

`public static int[][] multiply(int[][] m1, int[][] m2)`
`{`
`  int m1rows = m1.length;`
`  int m1cols = m1[0].length;`
`  int m2rows = m2.length;`
`  int m2cols = m2[0].length;`
`  int[][] result = new int[m1rows][m2cols];`
 	
`  for(int i = 0; i < m1rows; i++)`
`  for(int j = 0; j < m2cols; j++)`
`  for(int k = 0; k < m1cols; k++)`
`  result[i][j] += m1[i][k] * m2[k][j];`
 	
`  return result;`
`}`
	`:- module(matrix_multiply,`
	`  [matrix_multiply/3`
	`  ,dot_product/3`
	`  ]).`
	`:- use_module(library(clpfd), [transpose/2]).`
	
	`matrix_multiply(X,Y,M) :-`
	`  transpose(Y,T),`
	`  maplist(row_multiply(T),X,M).`
	
	`row_multiply(T,X,M) :-`
	`  maplist(dot_product(X),T,M).`
	
	`dot_product([X|Xs],[T|Ts],M) :-`
	`  foldl(mul,Xs,Ts,X*T,M).`
	
	`mul(X,T,M,M+X*T).`

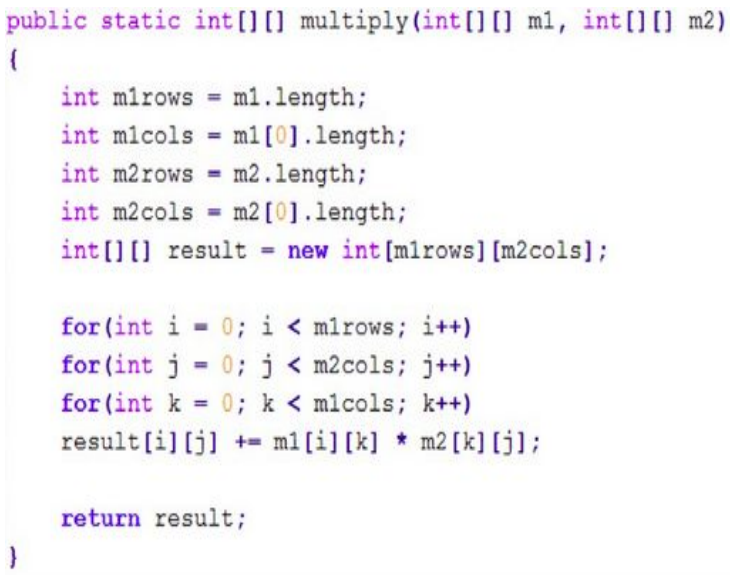
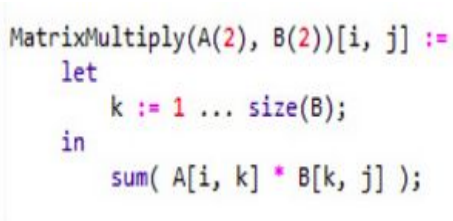
Wolfram

Java	Wolfram
	

```
public static int[][] multiply(int[][] m1, int[][] m2)
{
    int m1rows = m1.length;
    int m1cols = m1[0].length;
    int m2rows = m2.length;
    int m2cols = m2[0].length;
    int[][] result = new int[m1rows][m2cols];

    for(int i = 0; i < m1rows; i++)
    for(int j = 0; j < m2cols; j++)
    for(int k = 0; k < m1cols; k++)
    result[i][j] += m1[i][k] * m2[k][j];

    return result;
}
```

```
In[2]:= k {a, b, c}
Out[2]= {a k, b k, c k}

In[3]:= {a, b, c} . {ap, bp, cp}
Out[3]= a ap + b bp + c cp

In[4]:= {{a, b}, {c, d}} . {{1, 2}, {3, 4}}
Out[4]= {{a + 3 b, 2 a + 4 b}, {c + 3 d, 2 c + 4 d}}
```

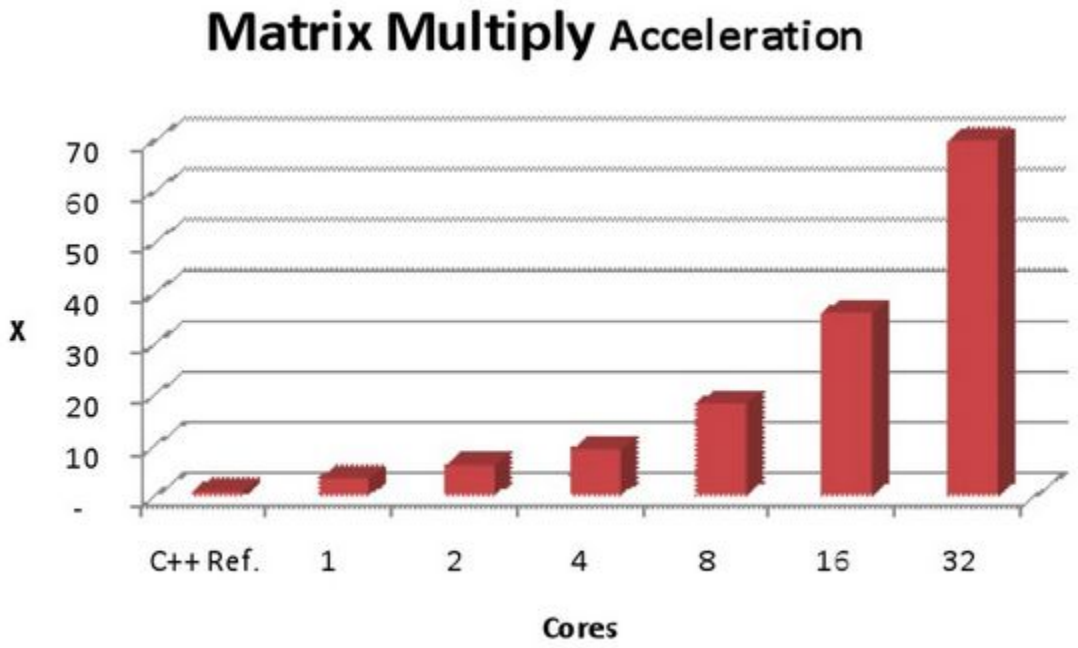
SequenceL

Java	SequenceL
	

```java
public static int[][] multiply(int[][] m1, int[][] m2)
{
    int m1rows = m1.length;
    int m1cols = m1[0].length;
    int m2rows = m2.length;
    int m2cols = m2[0].length;
    int[][] result = new int[m1rows][m2cols];

    for(int i = 0; i < m1rows; i++)
    for(int j = 0; j < m2cols; j++)
    for(int k = 0; k < m1cols; k++)
    result[i][j] += m1[i][k] * m2[k][j];

    return result;
}
```

```
MatrixMultiply(A(2), B(2))[i, j] :=
    let
        k := 1 ... size(B);
    in
        sum( A[i, k] * B[k, j] );
```

SequencL


Matrix Multiply Acceleration
X
70
60
50
40
30
20
10
-
C++ Ref.
1
2
4
8
16
32
Cores

Ventajas

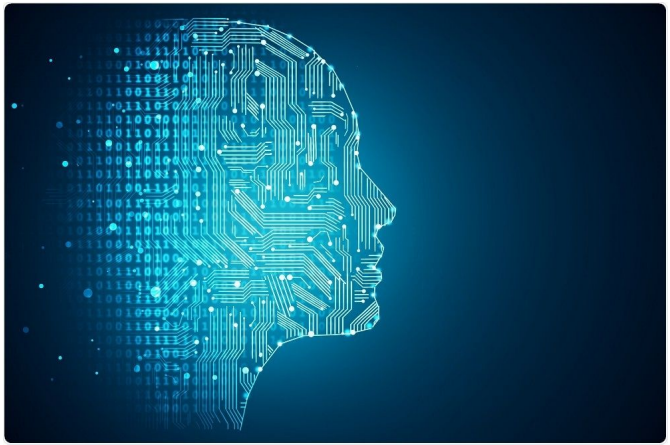
- La programación lógica se puede utilizar para expresar conocimiento de una manera que no depende de la
  Implementación, haciendo que los programas sean más flexibles, comprimidos y comprensibles.
- Permite separar el conocimiento del uso, por ejemplo, se puede cambiar la arquitectura de la máquina
  sin cambiar los programas o su código subyacente.
- Puede modificarse y extenderse de manera natural para apoyar formas especiales de conocimiento, tales como meta-nivel o conocimiento de orden superior.
- Se puede utilizar en disciplinas no computacionales basadas en el razonamiento y en medios precisos de expresión.

Desventajas

- No hay una forma adecuada de representar conceptos computacionales encontrados en los mecanismos incorporados de las variables de estado (como se suele encontrar en los lenguajes convencionales).
- Inicialmente, debido a una inversión insuficiente en tecnologías complementarias, los usuarios estaban mal atendidos.
- Al principio, instalaciones deficientes para soportar aritmética, tipos, etc. tenían un efecto desalentador en la comunidad de programación.
- Algunos programadores siempre tienen, y siempre preferirán la naturaleza abiertamente operativa de los programas operados por máquinas, ya que prefieren el control activo sobre las "partes móviles".

Aplicaciones

- Procesamiento de lenguaje natural.
- Sistemas expertos basados en producción.
- Inteligencia artificial.



Referencias

http://soft.vub.ac.be/~cderoove/declarative_programming/decprog3_sld_cut_naf.pdf

http://www.doc.ic.ac.uk/~cclw05/topics1/first.html

http://www.doc.ic.ac.uk/~cclw05/topics1/folCM.gif

https://www.hackerearth.com/practice/basic-programming/recursion/recursion-and-backtracking/tutorial/

http://www.j-paine.org/students/practicals/ps2/node8.html

https://reference.wolfram.com/language/

http://ferestrepoca.github.io/paradigmas-de-programacion/proglogica/logica_teoria/lang.html

https://www.bdti.com/InsideDSP/2014/11/18/TMT