UNIVERSIDAD
NACIONAL
DE COLOMBIA
SEDE BOGOTÁ
FACULTAD DE INGENIERÍA

Programación lógica

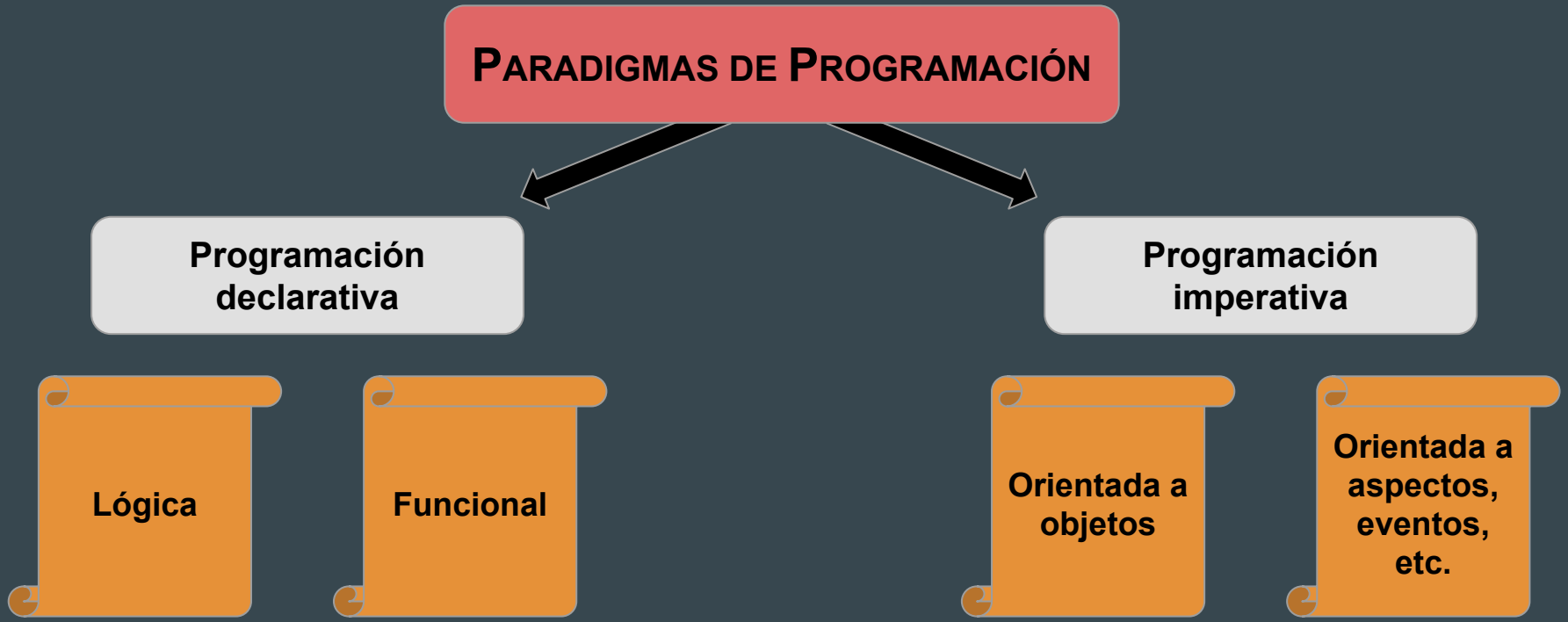
...

Cristian Andrés García Prieto
Andrés Felipe López Becerra
Miguel Ángel Borja Acevedo
Jorge Andrés Solano Avila

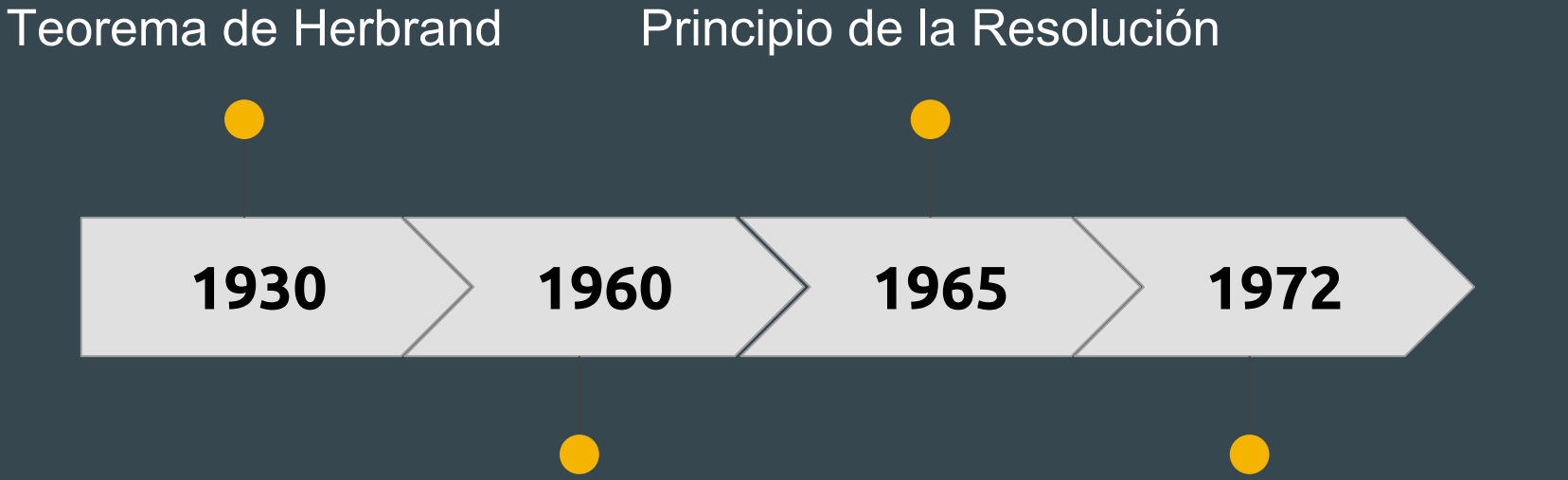
Agenda

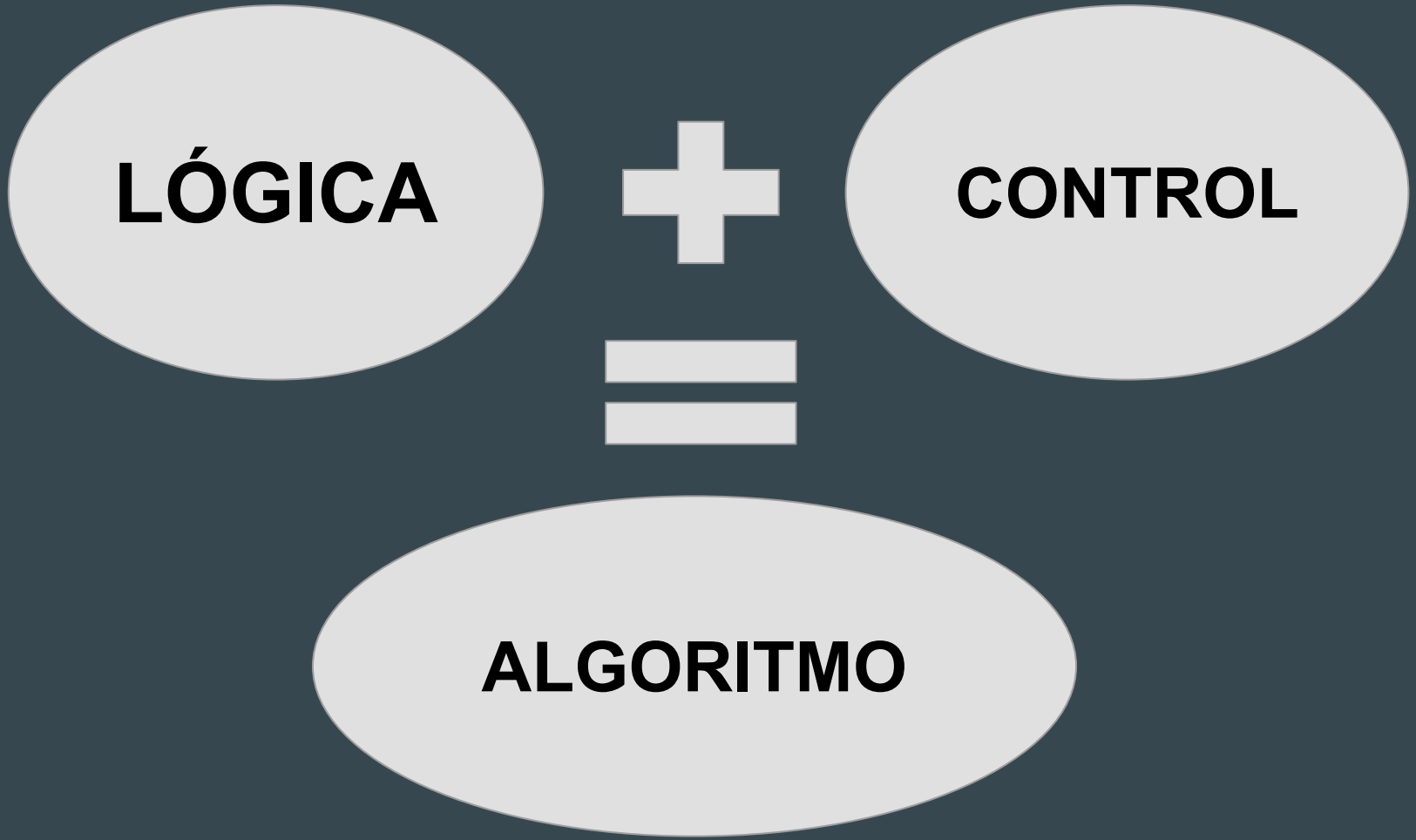
- Historia
- Conceptos claves
- Filosofía del paradigma
- Ventajas y desventajas
- Lenguajes de programación
- Ejemplos
- Aplicaciones
- Referencias

Paradigmas de programación



Historia


Teorema de Herbrand
Principio de la Resolución
1930
1960
1965
1972
Trabajos de demostración de teoremas asistidos por computador
PROLOG (PROgramming in LOGic)


LÓGICA
CONTROL
ALGORITMO

Conceptos clave

- Proposición

Expresión lingüística del razonamiento que se caracteriza por ser verdadera o falsa.

Juanito está en clase de lenguajes

Proposición

Conceptos clave

- Predicado

Expresión lingüística que puede conectarse con una o varias expresiones para formar una oración.

Saturno es un planeta

Expresión

Predicado (Expresión)

Conceptos clave

- Cuantificadores

Operador sobre un conjunto de individuos permitiendo construir proposiciones sobre conjuntos.

$$x > 3$$

Conceptos clave

- Cuantificadores

Operador sobre un conjunto de individuos permitiendo construir proposiciones sobre conjuntos.

$$\forall x \in Z: x > 3$$

$$Z=\{..., -2, -1, 0, 1, 2, ...\}$$

Conceptos clave

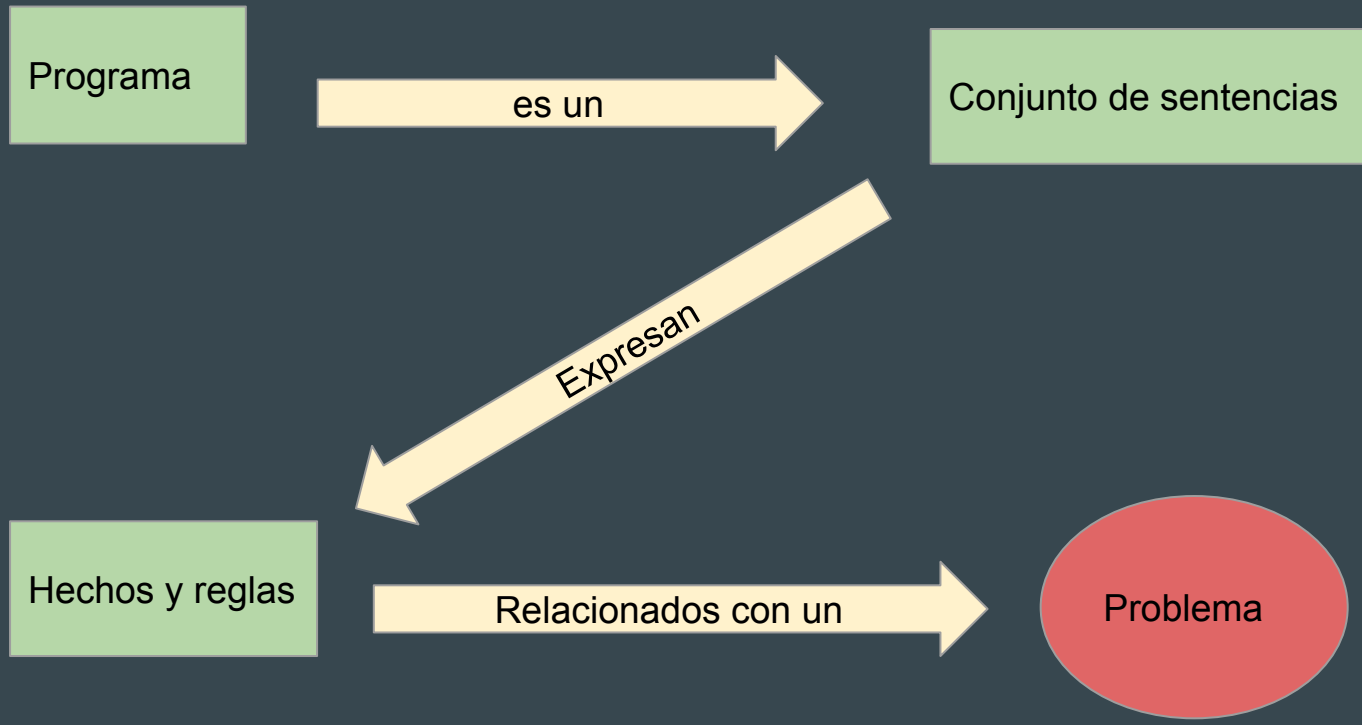
- Cuantificadores

Operador sobre un conjunto de individuos permitiendo construir proposiciones sobre conjuntos.

$$\exists x \in Z: x > 3$$

$$Z=\{..., -2, -1, 0, 1, 2, ...\}$$

Filosofía del paradigma



Filosofía del paradigma

- Se basa en fragmentos de la lógica de predicados (cláusulas de Horn).

Lógica de predicados: estudia frases declarativas a mayor detalle, considerando la estructura interna de las proposiciones. Se toman como elementos básicos los objetos (¿De quién se afirma?) y relaciones o predicados (¿Qué se afirma?). Además de otros elementos:

Variables	Constantes
Función	Conectivas(negación, ‘y’, ‘o’, ...)
Cuantificadores(∀, ∃)	Signos de puntuación.

Cláusulas de Horn: secuencia de literales que contiene a lo sumo uno de sus literales positivos (disyunción de literales).

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

ó

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

Ejemplo (Prolog):

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

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

Lectura de predicados: (mujer(A) $\wedge$ padre(B, A)) $\rightarrow$ hija(A, B)

- Como semántica declarativa (interpreta/considera a un programa lógico como lo que es) se usa la semántica por teoría de modelos: *el universo de Herbrand.*

Universo de Herbrand: toma como dominio de interpretación un universo de discurso puramente sintáctico.

“Dado un lenguaje de primer orden L, el universo de Herbrand U_L de L es el conjunto de todos los términos básicos de L (i.e., los términos bien formados sin variables que se pueden construir con los símbolos del alfabeto de L)”

Universo de Herbrand.

Ejemplo 1. Si C (conjunto de constantes de L) = $\emptyset$ y F un conjunto de funciones unarias: F = { f/1 }

UL = { a, f(a), f(f(a)), . . . }

Ejemplo 2. Sea L que contiene a la constante a, la función binaria g, entonces:

UL = { a, g(a, a), g(a, g(a, a)), g(g(a, a), a), . . . }

Universo de Herbrand.

Ejemplo 3. Sea C = { a, b } y F = { f/1, g/1 }

UL_0 = { a, b }

UL_1 = { a, b, f(a), f(b), g(a), g(b) }

UL_2 = { a, b, f(a), f(b), g(a), g(b),

f(f(a)), f(f(b)), f(g(a)), f(g(b)),

g(f(a)), g(f(b)), g(g(a)), g(g(b)) }

.

.

.

Universo de Herbrand.

Ejemplo 4. Demostrar que si $A \Rightarrow B$ y $B \Rightarrow C$, entonces $A \Rightarrow C$

(1)	$A \Rightarrow B$	Hipótesis
(2)	$B \Rightarrow C$	Hipótesis
(3)	A	Hipótesis
(4)	B	Modus ponens entre (1) y (3)
(5)	C	Modus ponens entre (2) y (4)

NOTA: Modus ponens:

$$\frac{A \Rightarrow B \qquad A}{\therefore \quad B}$$

- Resolución SLD (Selective Linear Definite clause): es un método de prueba por refutación que emplea el algoritmo de unificación como mecanismo de base y permite la extracción de respuestas.

unificación: dos o más expresiones se vuelven idénticas, por medio de una sustitución(unificadora). Se expresa: {X/a, Y/b} o {a/X, b/Y}

Ejemplos correctos	Ejemplos incorrectos
✓ {a/X, f(b)/Y}	X {a/X, b/X}
✓ {Y/X, b/Z}	X {a/X, Y/Y, f(a)/Z}
✓ {b/X, b/Y, f(a)/Z}	X {a/b, a/Y, f(a)/Z}

Ejemplo de unificación:

conjunto finito de expresiones: $C = \{ P(f(x), a), P(y, a) \}$

substitución: $\omega = \{ x/a, y/f(a), z/a \}$

ω es un unificador de C, porque $\omega(E_1) = \omega(E_2)$

Pero,

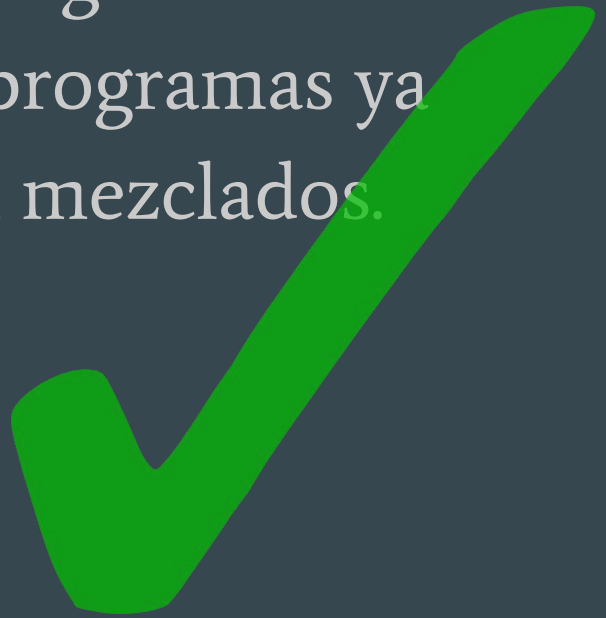
el conjunto $C = \{ P(f(x), a), P(y, f(w)) \}$, no es unificable porque los segundos argumentos a y f(w), no se pueden unificar.

Programación lógica vs Programación funcional

Programa: Conjunto de cláusulas que definen relaciones	Programa: Conjunto de ecuaciones que definen funciones
Semántica operacional: Resolución SLD (unificación)	Semántica operacional: Reducción(ajuste de patrones)
Primer orden	Orden superior
Uso múltiple de un mismo procedimiento	Uso único de cada función
Indeterminismo	Determinismo
Variables lógicas	Sin variables lógicas
Sin estructuras infinitas	Estructuras potencialmente infinitas
Sin evaluación perezosa	Evaluación perezosa

Ventajas

- Nivel de abstracción elevado, que facilita la claridad y el mantenimiento de programas.
- Facilita la verificación formal de programas ya que la lógica y el control no están mezclados.



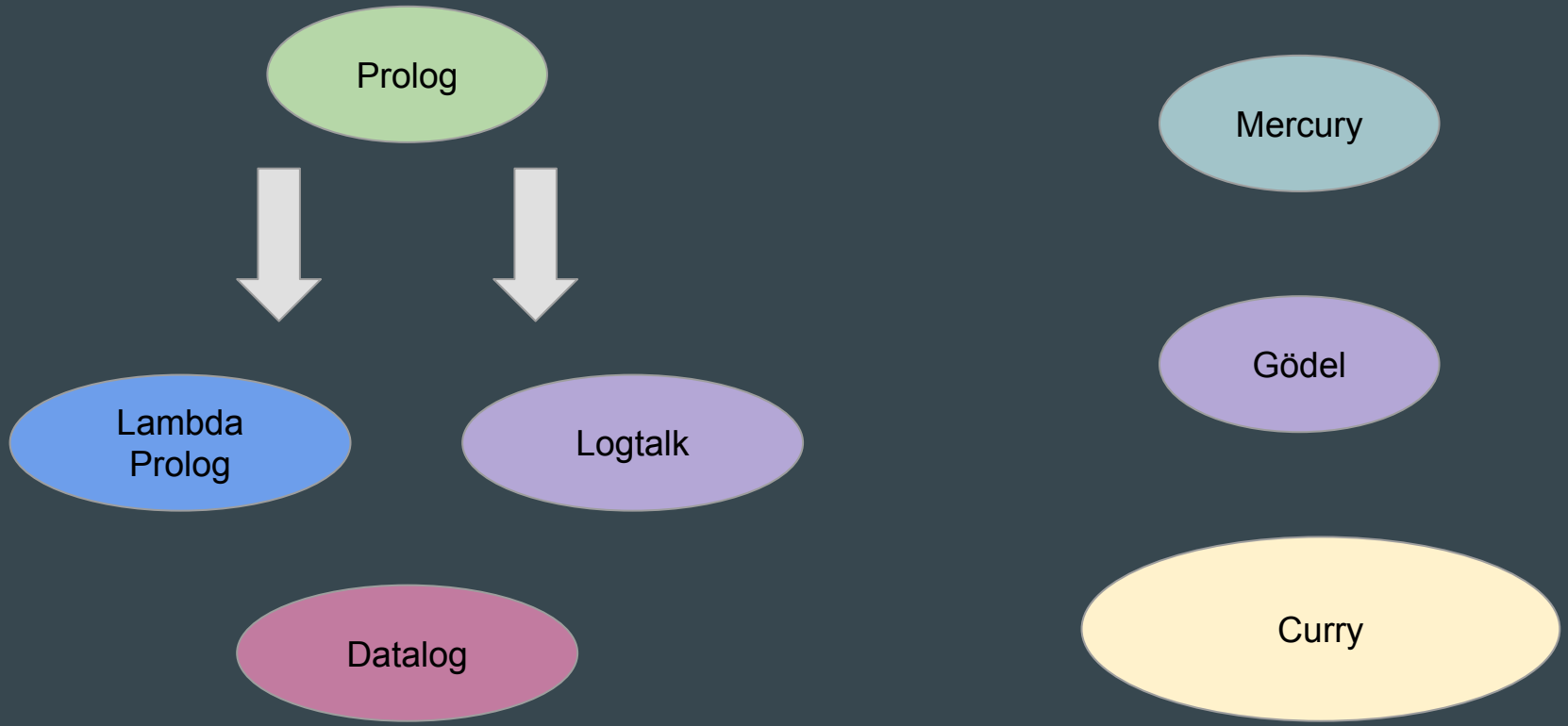
Ventajas

- Puede mejorarse la eficiencia modificando el componente de control sin tener que modificar la lógica del algoritmo.
- Fácil entendimiento.
- Simplicidad.
- Sencillez en la implementación de estructuras complejas.

Desventajas

- Poca eficiencia.
- No existen herramientas de depuración efectivas.
- En problemas reales, es poco utilizado.

Lenguajes de Programación



Prolog

- Está basado en el lenguaje de la Lógica de Primer Orden

```
es_padre_de(felix,clara).
```

```
regala(jorge, flores, clara).
```

- Se compone de cláusulas de Horn

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

Prolog

- Su ejecución se basa en:

Unificación

```
X is 3+5.
X = 8

X = 3+5.
X = 3+5
```

Backtracking

```
hermanode(A,B) :-
    padrede(C,A) ,
    padrede(C,B).
```

Ejemplo Prolog

```
padrede('juan', 'maria'). % juan es padre de maria
padrede('pablo', 'juan'). %pablo es padre de juan
padrede('pablo', 'marcela'). % pablo es padre de marcela
padrede('carlos', 'debora'). % carlos es padre de debora

% A es hijo de B si B es padre de A
hijode(A,B) :- padrede(B,A).
% A es abuelo de B si A es padre de C y C es padre B
abuelode(A,B) :-
   padrede(A,C),
   padrede(C,B).
% A y B son hermanos si el padre de A es también el padre de B y si A y B no son lo mismo
hermanode(A,B) :-
   padrede(C,A) ,
   padrede(C,B),
   A \== B.

% A y B son familiares si A es padre de B o A es hijo de B o A es hermano de B
familiarde(A,B) :-
   padrede(A,B).
familiarde(A,B) :-
   hijode(A,B).
familiarde(A,B) :-
   hermanode(A,B).
```

Ejemplo Prolog

```
% juan es hermano de marcela?
?- hermanode('juan', 'marcela').
yes

% carlos es hermano de juan?
?- hermanode('carlos', 'juan').
no

% pablo es abuelo de maria?
?- abuelode('pablo', 'maria').
yes

% maria es abuela de pablo?
?- abuelode('maria', 'pablo').
no
```

Gödel

- Existe el polimorfismo

- Tareas de meta-programación:
  - Compilación
  - Depuración
  - Verificación o transformación de programas

- Es más declarativo que Prolog

Máximo Común Divisor

```
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.
```

Mercury

- Es lógico y funcional
- Cuenta con un sistema modular
  - Interface
  - Implementación
- Se puede predeterminar el número de veces que se va a llamar a un predicado

```
:- pred factorial(int::in, int::out) is det.
```

- det
- semidet
- multi
- nondet

Ejemplos Mercury

```
:- module hello.


:- interface.
:- import_module io.
:- pred main(io::di, io::uo) is det.

:- implementation.

main(!IO) :-
   io.write_string("Hello world from Mercury!\n",!IO).
```

```
:- module fib.
:- interface.
:- import_module io.

:- pred main(io::di, io::uo) is det.

:- implementation.
:- import_module int.

:- pred fib(int::in, int::out) is det.

fib(N, X) :-
    (  if    N =< 2
       then  X = 1
       else  fib(N - 1, A), fib(N - 2, B), X = A + B
    ).

main(!IO) :-
    fib(17, X),
    io.write_string("fib(17, ", !IO),
    io.write_int(X, !IO),
    io.write_string(")\n", !IO).
```

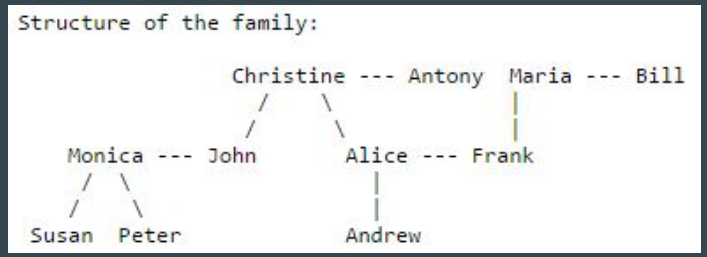
Curry

- Es funcional y lógico

- Programación Lógica:
  - Variables lógicas
  - Estructuras de datos parciales
  - Una función de búsqueda

- Adicionales:
  - Evaluación determinista
  - Demanda de funciones

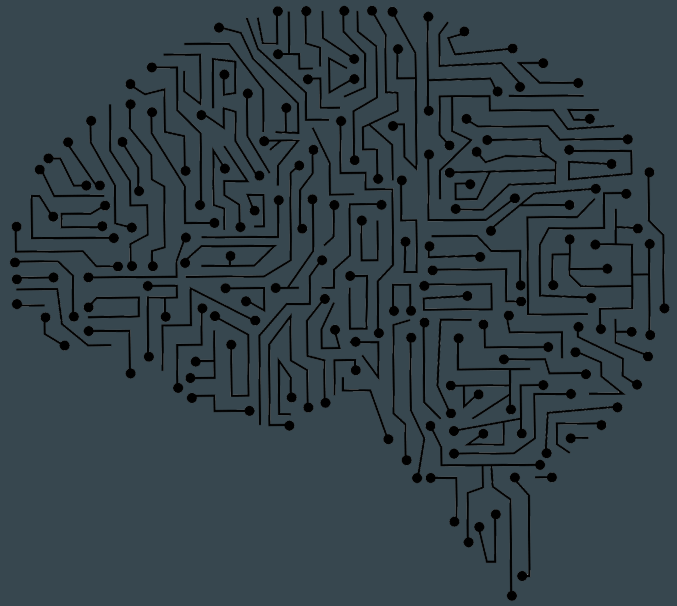
Ejemplo Curry




```
data Person = Christine | Maria | Monica | Alice | Susan |
              Antony | Bill | John | Frank | Peter | Andrew

female Christine  = True
female Maria      = True
female Monica     = True
female Alice      = True
female Susan      = True


male Antony  = True
male Bill    = True
male John    = True
male Frank   = True
male Peter   = True
male Andrew  = True


married Christine Antony  = True
married Maria Bill        = True
married Monica John       = True
married Alice Frank       = True


mother Christine John   = True
mother Christine Alice  = True
mother Maria Frank      = True
mother Monica Susan     = True
mother Monica Peter     = True
mother Alice Andrew     = True


father f c | let m free in (married m f && mother m c) =:= True  = True


grandfather g c | let f free in (father g f && father f c) =:= True  = True
grandfather g c | let m free in (father g m && mother m c) =:= True  = True
```

Aplicaciones

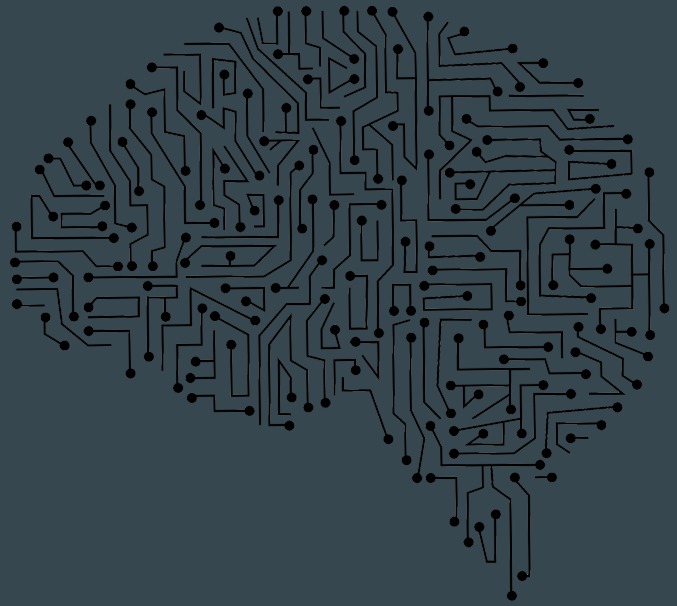
Inteligencia Artificial:



Este es el uso por excelencia de la programación lógica, se puede encontrar la mejor respuesta para un juego o darle solución a este, también se puede analizar lenguaje natural o encontrar teoremas sobre una teoría ya existente.

Aplicaciones

Inteligencia Artificial:



```
hanoi(0,_,_,_).
hanoi(N,Origen,Auxiliar,Destino):- N1 is N-1,
    hanoi(N1,Origen,Destino,Auxiliar),
    def_pasos(N,Origen,Destino),
    hanoi(N1,Auxiliar,Origen,Destino).

def_pasos(N,Origen,Destino):-write(' ficha '),write(N),  write
(' | '),
write(Origen),write(' - '),write(Destino),write('  |'),writeln(' ').

?- hanoi(4,"Inicio","medio","final").
```

Aplicaciones

Sistemas expertos:

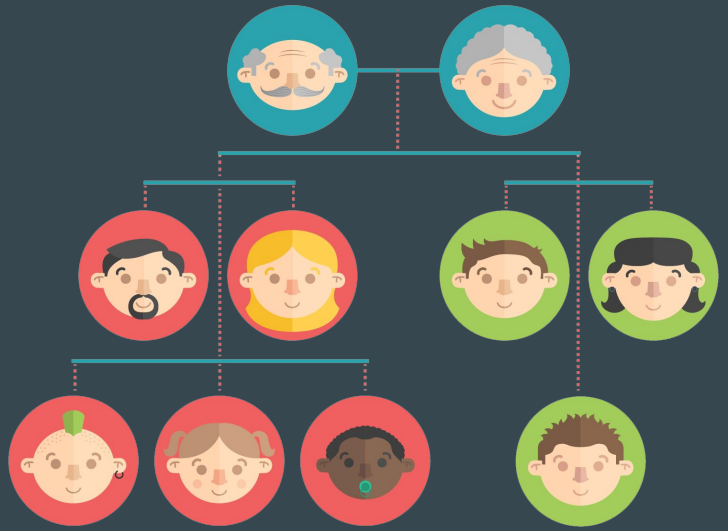
No se comportan como otras clases de programas, en lugar de realizar una serie de tareas, deben tener un cuerpo de conocimiento y deben ser capaces de manipular dicho conocimiento para obtener conclusiones (mediante algún método de resolución de problemas).

Aplicaciones

I. **Bases de conocimiento:** Herramienta usada para describir y recuperar conocimiento.
II. **Resolución de problemas:** Solución de problemas específicos, basado en un conocimiento previo.
III. **Análisis de lenguaje natural:** Generar intérpretes y compiladores, traducción automática de lenguaje natural, entre otros.

Aplicaciones

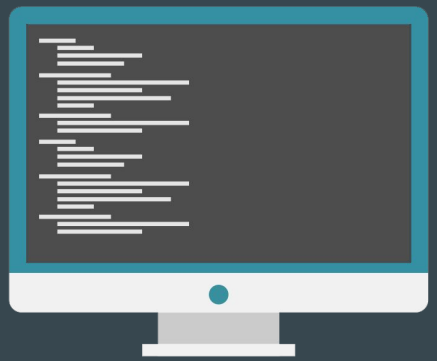
Bases de conocimiento:



```
hermano("juan", "pedro").      %juan y pedro son hermanos
hermano("pedro", "jose").      % pedro y jose son hermanos
hijo("juancito", "juan").      %juancito es hijo de juan
hijo("ana", "juan").           %ana es hija de juan
hijo("natalia", "pedro").      %natalia es hija de pedro
hijo("andres", "jose").        %andres es hija de jose

tio(X,Y):- (hermano(Z,X);hermano(X,Z)),hijo(Y,Z).

?- tio(A,D)  %¿A de quien es tío? o ¿Cuales D son sobrino de A?
```

Aplicaciones

Resolución de problemas:

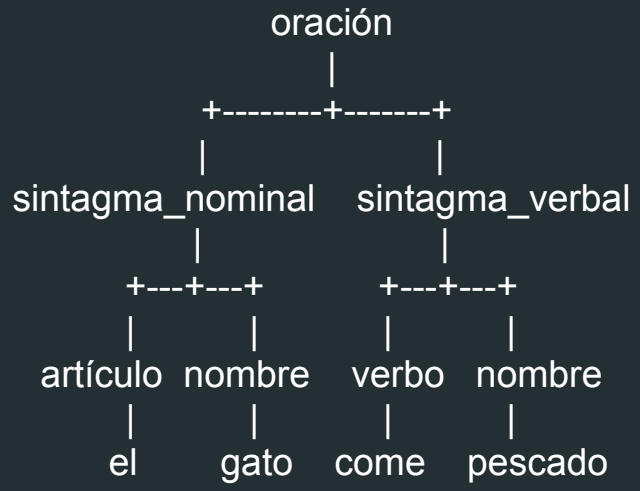


```
factorial(0,1).                    %Caso base
factorial(Num,Result):-            %Recursion
    Num>0,
    Num1 is Num-1,
    factorial(Num1,Result1),
    Result is Num*Result1.

?- factorial(5,120)                %¿5! es igual a 120?
```

Aplicaciones

Análisis de lenguaje natural:

Frases: El gato come pescado - El perro come carne



Aplicaciones

Análisis de lenguaje natural:

```
oracion(O) :- sintagma_nominal(SN),
sintagma_verbal(SV),
append(SN,SV,O).

sintagma_nominal(SN) :- nombre(SN).

sintagma_nominal(SN) :- articulo(A),
nombre(N),
append(A,N,SN).

sintagma_verbal(SV) :- verbo(V),
sintagma_nominal(SN),
append(V,SN,SV).
```

```
articulo([el]).
nombre([gato]).
nombre([perro]).
nombre([pescado]).
nombre([carne]).
verbo([come]).

?- oracion([perro,come,pescado]).
```

Referencias

- Estudios sobre la programación lógica y sus aplicaciones, Universidad de Santiago de Compostela - 1996
- Programación Lógica. Teoría y Práctica, Editorial PEARSON EDUCACIÓN S.A. - 2007
- Foundations of Logic Programming, Editorial Springer-Verlag - 1991
- Cláusulas de Horn. Resolución SLD (enlace: http://gpd.sip.ucm.es/jaime/pl/sld.pdf)
- Labra, J. E. y Fernández, D. Lógica de predicados. Universidad de Oviedo.
- Semántica Declarativa para la Programación en Lógica. 2009 (enlace: http://cs.uns.edu.ar/~grs/Logica/007-2009.Semantica%20Declarativa.ByN.pdf)
- Alonso, J., Cordón, A. y Hidalgo, M. Lógica informática Tema 11: Modelos de Herbrand. 2012-2013.

Referencias

Enlaces web:

- https://prezi.com/wplool-wnjgy/programacion-logica-con-clausulas-de-horn/
- http://www.taringa.net/post/ciencia-educacion/18313207/Jacques-Herbrand-un-genio-de-la-logica-matematica.html
- Programming with Logic and Objects, Michael Kifer, Stony Brook University.
- Curry, A Tutorial Introduction. Sergio Antoy, Michael Hanus. Portland State University, U.S.A.
- http://www.logicprogramming.org/
- http://www.scs.leeds.ac.uk/hill/GOEDEL/expgoedel.html

Gracias.