PROGRAMACIÓN LÓGICA

- David Felipe Rico Hernandez
- Gabriel Esteban Bejarano Delgado

UNIVERSIDAD NACIONAL DE COLOMBIA

SEDE BOGOTÁ

FACULTAD DE INGENIERÍA

CONTENIDO

- Introducción
- Programación Lógica
- Conceptos clave
- Lenguajes de programación
- Ejemplos
- Ventajas y desventajas
- Aplicaciones
- Referencias y bibliografía

INTRODUCCIÓN

- Paradigmas
- historia
- programación lógica

INTRODUCCIÓN

Programación Imperativa

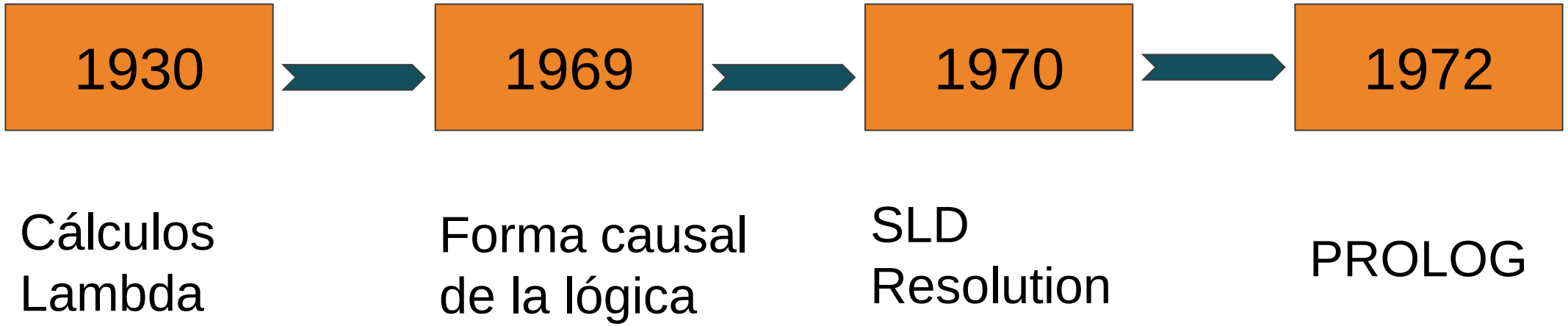
- Describe la programación en términos del estado del programa y sentencias que cambian dicho estado

Programación declarativa

- Se utilizan sentencias para describir el problema que se quiere solucionar

INTRODUCCIÓN

Historia



INTRODUCCION

Programación Lógica

Aplicación de reglas de la lógica para inferir conclusiones a partir de datos.

INTRODUCCIÓN

- **Lógica:** determina las soluciones producidas
- **Control:** son las formas alternativas de ejecutar la lógica

PROGRAMACION LOGICA

Que es?

construye base de conocimiento mediante reglas y hechos

Características

Unificación de términos

Mecanismos de inferencia Automáticos

Recursión como estructura de control Básica

Visión lógica de la computación

PROGRAMACION LOGICA

Logica de enunciados

$$
\begin{aligned}
\textit{Sentencia} &\longrightarrow \textit{Sentencia Atómica} \mid \textit{Sentencia Compleja} \\
\textit{Sentencia Atómica} &\longrightarrow \textbf{Verdadero} \mid \textbf{Falso} \mid \textit{Símbolo Proposicional} \\
\textit{Símbolo Proposicional} &\longrightarrow \mathbf{P} \mid \mathbf{Q} \mid \mathbf{R} \mid \ldots \\
\textit{Sentencia Compleja} &\longrightarrow \neg \textit{Sentencia} \\
&\mid (\textit{Sentencia} \wedge \textit{Sentencia}) \\
&\mid (\textit{Sentencia} \vee \textit{Sentencia}) \\
&\mid (\textit{Sentencia} \Rightarrow \textit{Sentencia}) \\
&\mid (\textit{Sentencia} \Leftrightarrow \textit{Sentencia})
\end{aligned}
$$

PROGRAMACION LOGICA

Lógica de Predicados

Sentencia	$\longrightarrow$	*Sentencia Atómica* | *(Sentencia Conectiva Sentencia)* | *Cuantificador Variable … Sentencia* | ¬*Sentencia*
Sentencia Atómica	$\longrightarrow$	*Predicado (Término...)* | *Término = Término*
Término	$\longrightarrow$	*Función(Término)* | *Constante* | *Variable*
Conectiva	$\longrightarrow$	^ |v|$\Rightarrow$| $\Leftrightarrow$
Cuantificador	$\longrightarrow$	¬*Sentencia*
Variable	$\longrightarrow$	*a* | *x* | *s* | …
Predicado	$\longrightarrow$	*TieneColor* | *EstáLloviendo* | …
Función	$\longrightarrow$	*Hombre* | *Humano* | *Mujer* | …

PROGRAMACION LOGICA

Cláusulas de Horn: es una cláusula (disyunción de literales) con, como máximo, un literal positivo.

$$\neg p \lor \neg q \lor \cdots \lor \neg t \lor u$$

$$(p \land q \land \cdots \land t) \rightarrow u$$

PROGRAMACION LOGICA

Ejemplo cláusulas de horn

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

$$\neg mujer(A) \lor \neg padre(B, A) \lor hija(A, B)$$

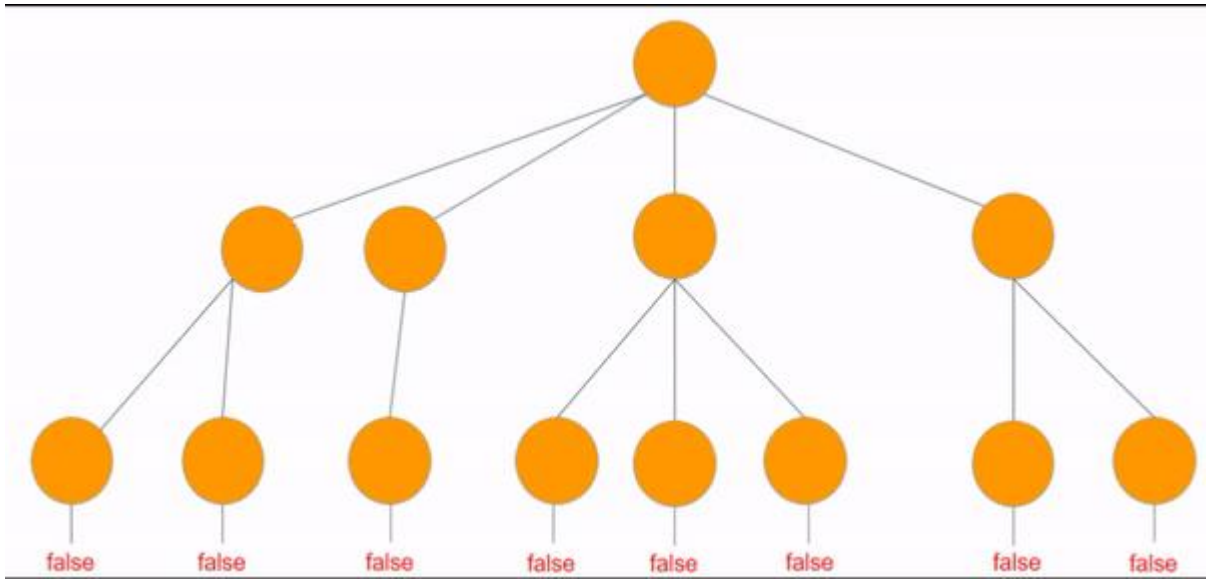
PROGRAMACION LOGICA

Resolución **SLD** (Selective Linear Definite clause resolution)

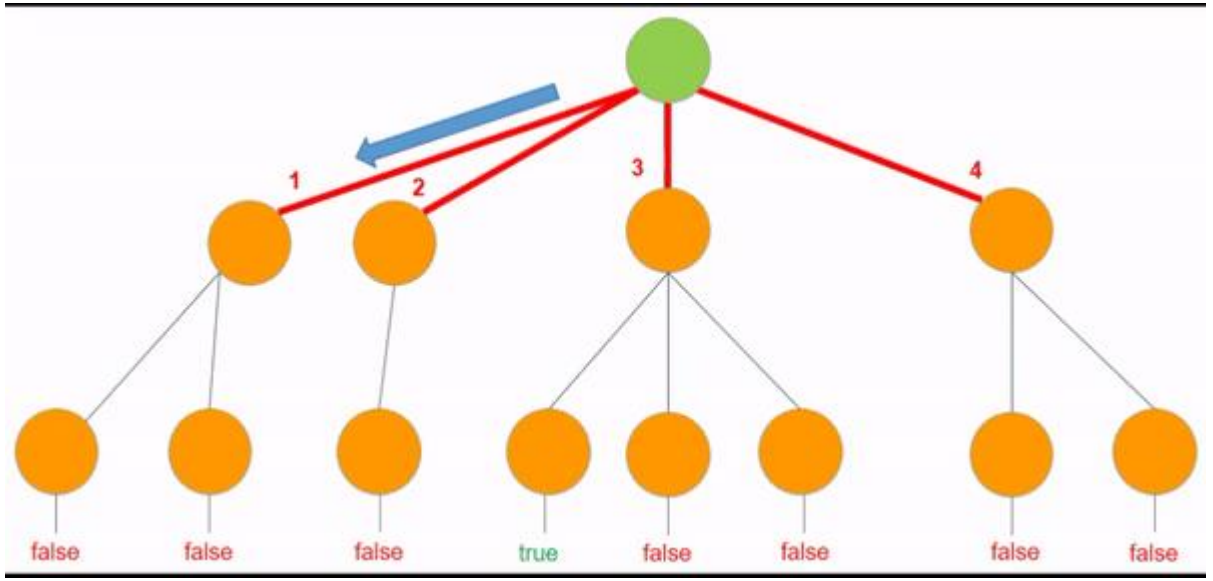
Es un caso particular de la resolucion general, donde:

- Los resolventes son siempre objetivos (clausulas sin cabeza)
- Los programas son conjuntos de clausulas (de Horn) definidas, hechos y reglas.
- Hay una funcion de seleccion que selecciona un atomo del resolvente a quien aplicar resolucion.

PROGRAMACION LOGICA



PROGRAMACION LOGICA



CONCEPTOS CLAVE

- Hecho(Fact)
- Reglas
- Consultas
- Recursion

CONCEPTOS CLAVE

Hechos: Declaración, cláusula o proposición cierta o falsa, el hecho establece una propiedad o relación entre objetos y es la forma más sencilla de sentencia.

```
esHumano(gabriel). %gabriel es humano
esHumano(david). %david es humano
```

Reglas: Implicación o inferencia lógica que deduce nuevo conocimiento, establece dependencias entre hechos, la regla permite definir nuevas relaciones a partir de otras ya existentes.

```
esMortal(X):- esHumano(X). %Un elemento X es mortal, siempre y cuando ese X sea humano
```

CONCEPTOS CLAVE

Consultas: Se especifica el problema, la proposición a demostrar o el objetivo. La consulta se deduce de la base de conocimiento (hechos + reglas).

esHumano(david). esHumano(gabriel). esMortal(X):- esHumano(X). ?- esMortal(gabriel). %pregunta si gabriel es mortal	esHumano(gabriel). esMortal(X):- esHumano(X). ?- esMortal(david). %pregunta si david es mortal
true	false

CONCEPTOS CLAVE

Recursión: Permite que un bloque de instrucciones se ejecute un cierto número de veces.

```
fib(0,0).
fib(1,1).
fib(X,Y):- X1 is X-1,X2 is X-2,
                        fib(X1,Y1),
                        fib(X2,Y2),
                        Y is Y1+Y2.
?- fib(6, X).
```

```
X = 8
```

```
factorial(0, 1).
factorial(N, R) :- N > 0,
        N1 is N - 1,
        factorial(N1, R1),
        R is N*R1.
?- factorial(6,X).
```

```
X = 720
```

LENGUAJES DE PROGRAMACIÓN

Prolog

- Está basado en “cláusulas de Horn”.
- Se basa en lógica de primer orden.
- Es el más conocido y usado.
- Es declarativo.

ALF (Another Logical Framework).

- Combina la programación lógica con la programación funcional.
- Está basado en “cláusulas de Horn”.
- Se puede mezclar expresiones y ecuaciones.

Gödel

- Sentencias lógicas llevan un orden.
- Existe el polimorfismo.
- Buen lenguaje para tareas de meta-programación
- Más declarativo que prolog.
- Fuertemente tipado.
- Está basado en módulos.

LENGUAJES DE PROGRAMACIÓN

Mercury

- Es un lenguaje de alto nivel derivado de Prolog.
- Es un lenguaje compilado.
- Está basado en módulos.
- Es puramente declarativo.

MONA

- “Traduce” los programas (fórmulas) a autómatas de estados finitos.

EJEMPLOS

Prolog

```
partirI([],[],[]).
partirI([H|T],[H|I],D):- partirD(T,I,D).
partirD([],[],[]).
partirD([H|T],I,[H|D]):- partirI(T,I,D).
partir(A,B,C):- partirI(A,B,C).

fusion(L,[],L).
fusion([],L,L).
fusion([H|T],[H2|T2],[H|L]):- H <H2,fusion(T,[H2|T2],L).
fusion([H|T],[H2|T2],[H2|L]):- H2 =< H, fusion([H|T],T2,L).

merge([],[]).
merge([H],[H]).
merge(L,O):- partir(L,I,D), merge(I,IO), merge(D,DO),fusion(IO,DO,O).
```

ALF (Another Logical Framework).

```
vec
 Fin ∈ (Nat) Set
    0Fin ∈ (n ∈ Nat) Fin(s(n))
    sFin ∈ (↓n ∈ Nat; Fin(n)) Fin(s(n))
 finToNat ∈ (↓n ∈ Nat; Fin(n)) Nat
 emb ∈ (↓n ∈ Nat; Fin(n)) Fin(s(n))
 addFin ∈ (↓m, ↓n ∈ Nat; Fin(m); Fin(n)) Fin(add(m, n))
 Vec ∈ (Set; Nat) Set
    nilVec ∈ (A ∈ Set) Vec(A, 0)
    consVec ∈ (↓A ∈ Set; ↓n ∈ Nat; A; Vec(A, n)) Vec(A, s(n))
 nth ∈ (↓A ∈ Set; ↓n ∈ Nat; Vec(A, n); Fin(n)) A
 update ∈ (↓A ∈ Set; ↓n ∈ Nat; Vec(A, n); Fin(n); A) Vec(A, n)
 insert ∈ (↓A ∈ Set; ↓n ∈ Nat; Vec(A, n); Fin(n); A) Vec(A, s(n))
 delete ∈ (↓A ∈ Set; ↓n ∈ Nat; Vec(A, s(n)); Fin(n)) Vec(A, n)
```

EJEMPLOS

Gödel

```
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

```
:- type maybe(T) --->  yes(T) ; no.
```

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

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

:- implementation.
:- import_module int.

:- func fib(int) = int.
fib(N) = (if N =< 2 then 1 else fib(N - 1) + fib(N - 2)).

main(!IO) :-
       io.write_string("fib(10) = ", !IO),
       io.write_int(fib(10), !IO),
       io.nl(!IO).
       % Could instead use io.format("fib(10) = %d\n", [i(fib(10))], !IO).
```

EJEMPLOS (minikanren)

Python:
- pykanren
- pythological
- LogPy
- microkanren.py

Java:
- mk.java
- java8kanren

C#:
- csharp_ukanren
- uKanren.NET

Common Lisp:
- cl-kanren-trs
- Kanren-Fset

http://minikanren.org/

- **miniKanren es un lenguaje de dominio específico incorporado para la programación lógica.**
- **El lenguaje central miniKanren es muy simple, con solo tres operadores lógicos y un operador de interfaz.**
- **miniKanren se ha implementado en un número creciente de lenguajes, incluidos Scheme, Racket, Clojure, Haskell, Python, JavaScript, Scala, Ruby, OCaml y PHP, entre muchos otros lenguajes.**
- **miniKanren está diseñado para ser modificado y ampliado fácilmente; las extensiones incluyen la Programación Lógica de Restricciones, programación de lógica probabilística, programación lógica nominal y presentación.**

EJEMPLOS (Ancestros)

Prolog	Python (con LogPy)
parent(nathan, steve). parent(nathan,edith). parent(david, john). parent(jim, david). parent(steve, jim). parent(david,adrian). parent(jim,homero). parent(homero,bart). parent(homero,lisa). parent(homero,magie). parent(edith,ramon). parent(edith,felipe).	from kanren import * parent = Relation() facts(parent, ("nathan", "steve"), ... ("nathan", "edith"), ... ("david", "john"), ... ("nathan", "edith"), ... ("david", "john"), ... ("jim", "david"), ... ("steve", "jim"), ... ("david", "adrian"), ... ("jim", "homero"), ... ("homero", "bart"), ... ("homero", "lisa"), ... ("homero", "magie"), ... ("edith", "ramon"), ... ("edith", "felipe"))

EJEMPLOS (Ancestros)

Prolog	Python (con LogPy)

grandparent(A, B) :- parent(A, X), parent(X, B).
brother(A,B) :- parent(A,X),parent(B,Y), Y==X.
 	
ancestor(A, B) :- parent(A, B).
ancestor(A, B) :- parent(A, X), ancestor(X, B).
	def grandparent(A, B):
	... X = var()
	... return conde((parent(A, X), parent(X, B)))
	def brother(A, B):
	... X = var()
	... Y = var()
	... return conde((parent(A, X), parent(B, Y),eq(X,Y)))
	
	def ancestor(A, B):
	... X = var()
	... If conde(eq(parent(A, X), ancestor(X, B))):
	... ... Return conde(parent(A, X))
	... Else:
	... ... return conde((parent(A, X), ancestor(X, B)))

EJEMPLOS (Ancestros)

Prolog	Python (con LogPy)
?- ancestor(X, john).	`run(4, X, ancestor(X, "john"))`
X = david ; X = jim ; X = steve ; X = nathan ;	`('david', 'jim', 'steve', 'nathan')`

VENTAJAS Y DESVENTAJAS

Ventajas:

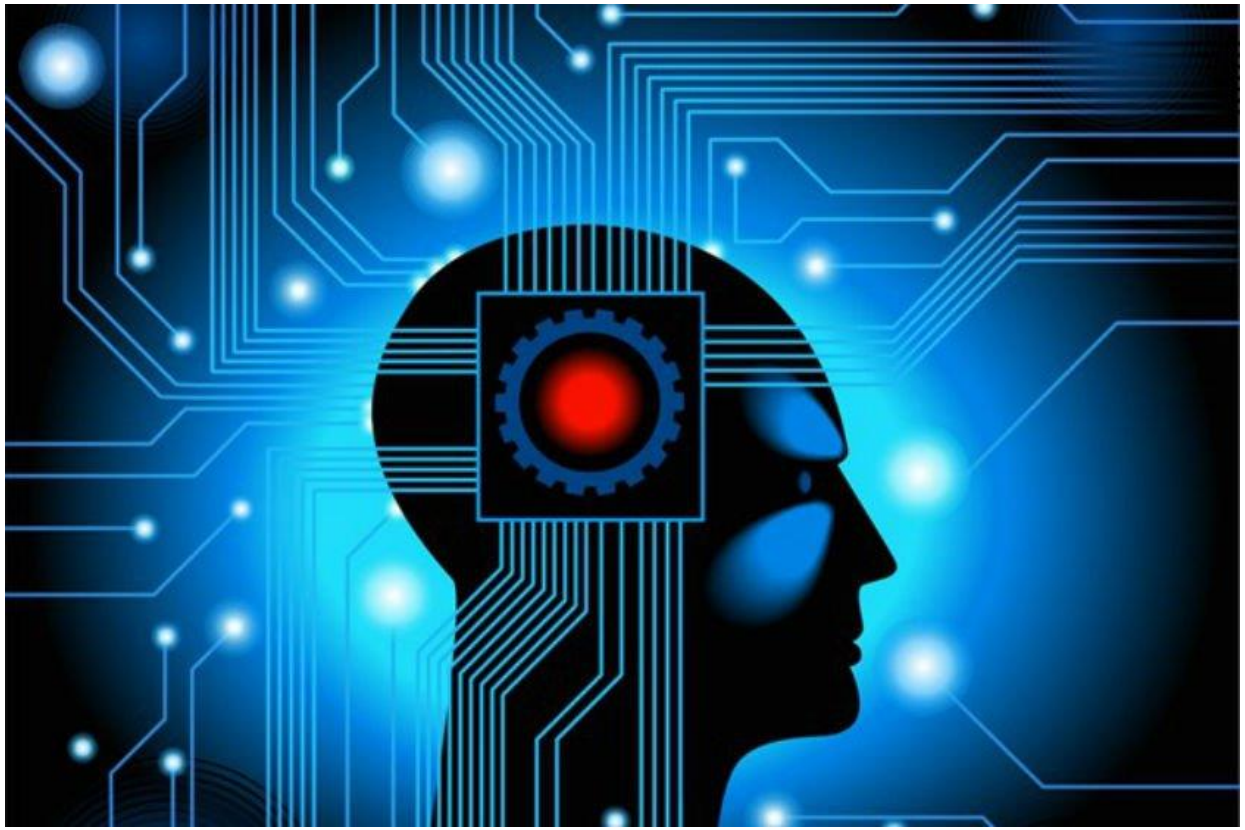
- Simplicidad
- Cercanía a las especificaciones del problema realizada con lenguajes formales
- Se puede modificar el componente de control sin modificar la lógica del algoritmo.
- Sencillez en la implementación de estructuras complejas

Desventajas:

- Poco eficientes.
- Poco utilizado en aplicaciones reales.
- Con poca información retorna false.
- No existen herramientas de depuración efectivas.

Aplicaciones

- Inteligencia artificial.
- Sistemas expertos.
- Demostración automática de teoremas.
- Reconocimiento de lenguaje natural.
- Aplicaciones o problemas reales.



BIBLIOGRAFIA

- http://programacion-programacionlogica.blogspot.com.co/
- https://www.infor.uva.es/~calonso/IAI/PracticasProlog/Tema1/T1_Introduccion_Prolog.PDF
- https://en.wikibooks.org/wiki/Prolog/Recursive_Rules
- http://cala.unex.es/cala/epistemowikia/index.php/Programaci%C3%B3n_L%C3%B3gica_y_Funcional#Programaci.C3.B3n_L.C3.B3gica
- http://www.it.uc3m.es/jvillena/irc/practicas/estudios/Lenguajes_Logicos.pdf
- https://es.wikipedia.org/wiki/Programaci%C3%B3n_l%C3%B3gica
- http://www.unizar.es/pde/fjgaspar/cap1.pdf