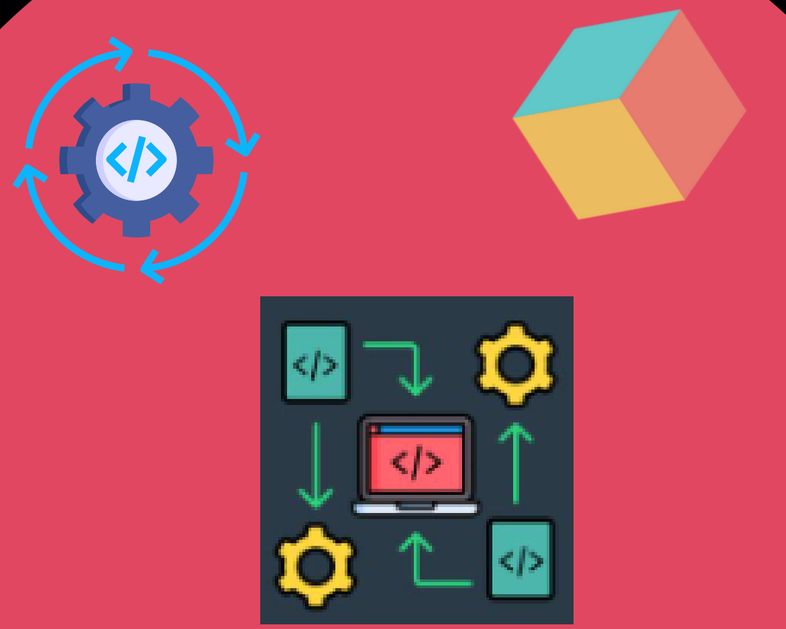
PROGRAMACIÓN LÓGICA

Alice Phung-Ngoc
Juan M. De La Torre
Nicolas Romero

¿QUÉ ES UN PARADIGMA?



"Es el conjunto de principios subyacentes que dan forma al estilo de un lenguaje de programación."

-Concepts, Techniques, and Models of Computer Programming.

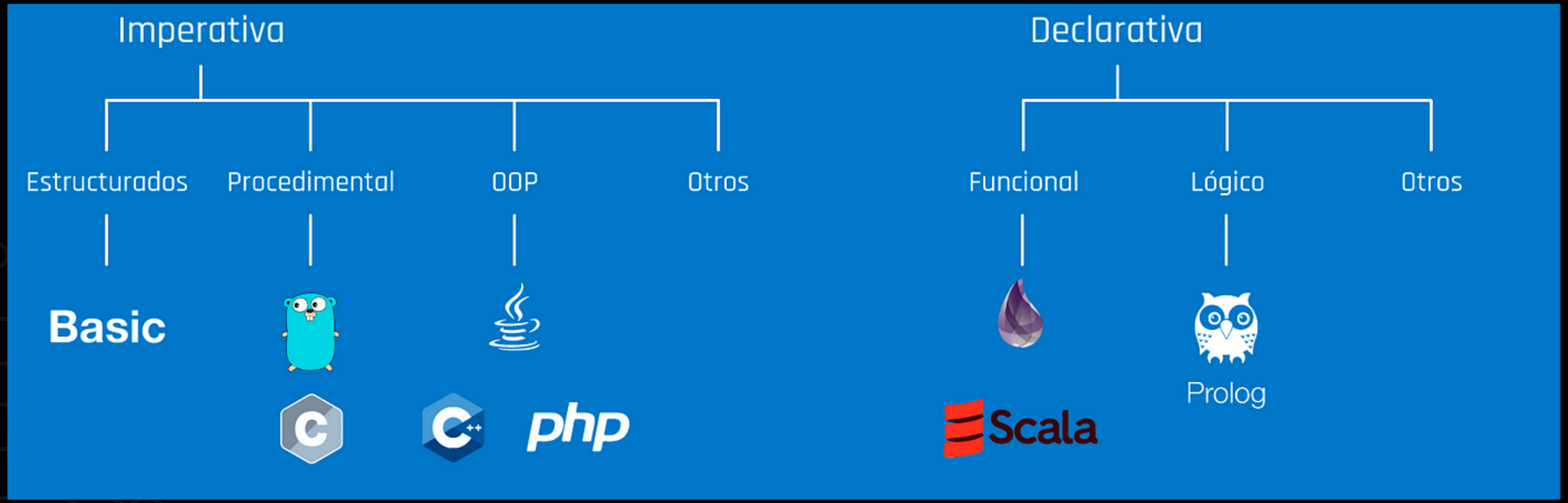
IMPERATIVO VS DECLARATIVO

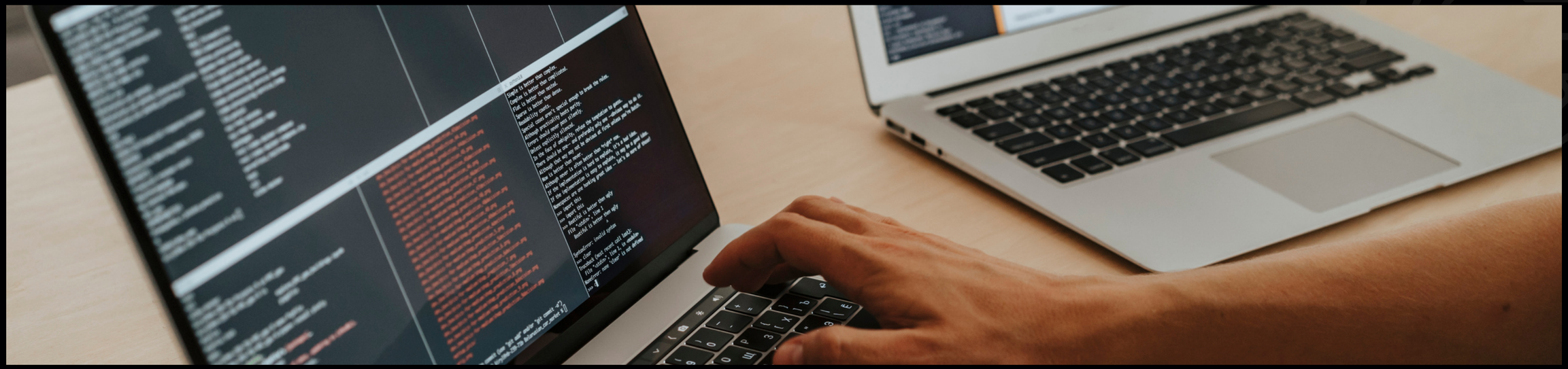
Imperativo

Cómo se deben hacer las cosas

Declarativo

Qué debe hacerse

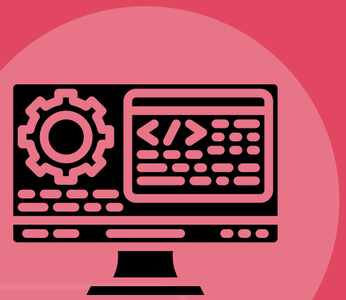




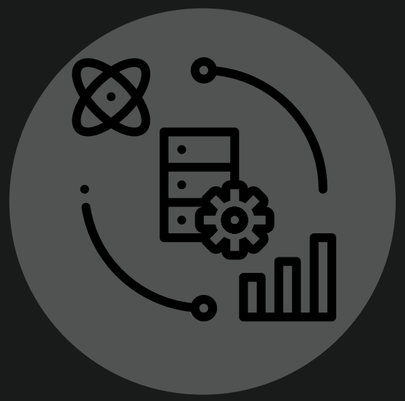
¿QUÉ ES LA PROGRAMACIÓN LÓGICA?

Es un enfoque en el que se definen las reglas y se solicita al programa que busque una solución lógica a un problema en particular

FILOSOFÍA DEL PARADIGMA

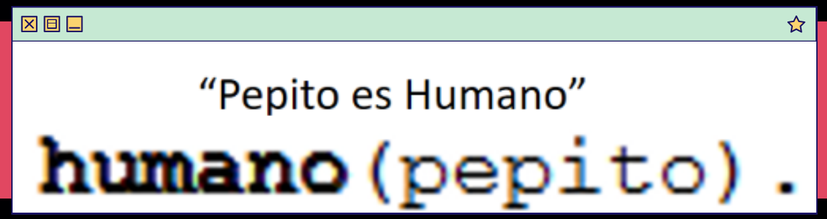


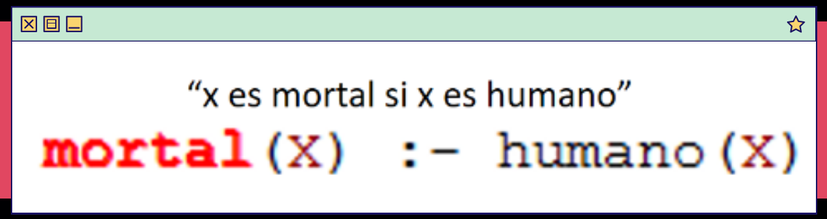
"Modelar problemas por medio de la abstracción, utilizando un sistema de lógica formal que permite llegar a una conclusión por medio de hechos y reglas"

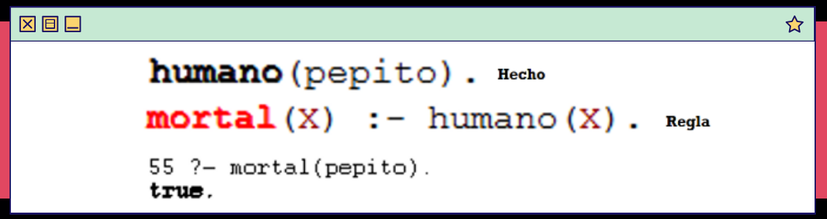


"Aplicación de reglas de la lógica para inferir conclusiones a partir de datos."

CONCEPTOS CLAVE DEL PARADIGMA







Hechos

Son afirmaciones que describen el estado del mundo en el que se está trabajando.

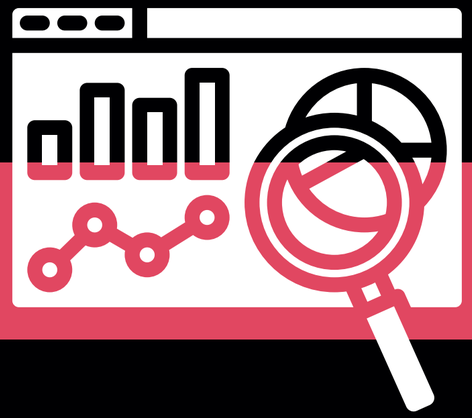
Reglas

Son afirmaciones que describen relaciones lógicas entre los hechos.

Consultas

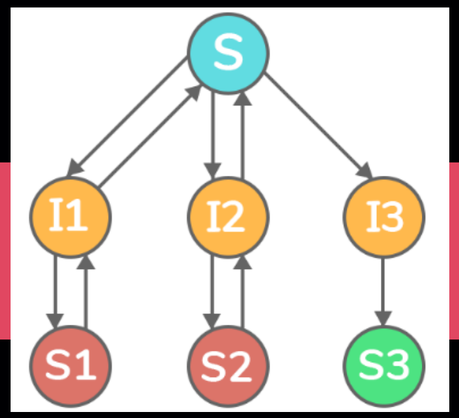
Son preguntas que se hacen al programa lógico para buscar soluciones a un problema específico.

CONCEPTOS CLAVE DEL PARADIGMA



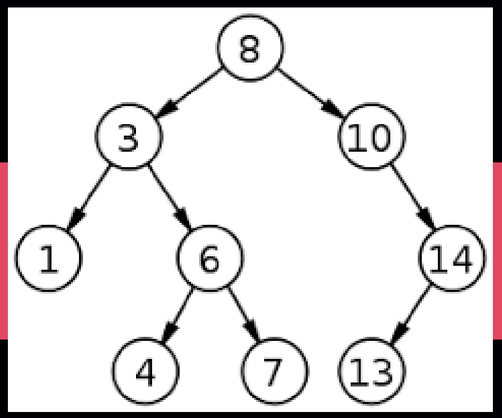
Unificación

Encontrar valores para las variables en una consulta que hacen que se cumplan las reglas del programa.



Backtracking

Es el proceso mediante el cual el programa lógico busca soluciones para un problema explorando todas las posibilidades



Árboles de búsqueda

Son estructuras de datos que se utilizan para representar todas las posibles soluciones a un problema

CONCEPTOS CLAVE DEL PARADIGMA

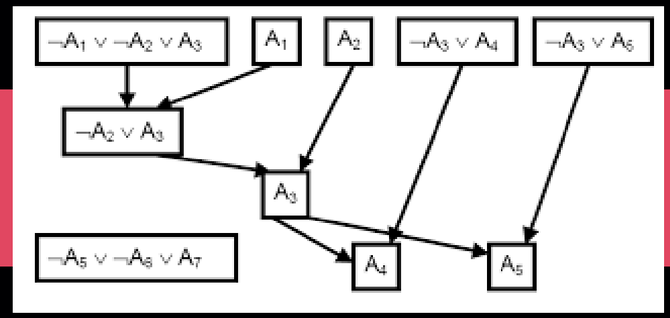
Cortes

Son restricciones que se aplican en un programa lógico para evitar que se generen soluciones que no son relevantes

~ (p ∧ q)	~ (p ∨ q)
F	F
V	F
V	F
V	V

Negación

La capacidad de negar una afirmación.



Resolución SLD

Es un algoritmo utilizado para buscar soluciones a través de una base de conocimientos en la programación lógica.

CARACTERÍSTICAS DEL PARADIGMA

01 Basada en la lógica

02 Declarativa

03 Basada en reglas

04 Inferencia

05 Resolución de problemas

06 No determinista

07 Recursiva

¿QUÉ TRATA DE RESOLVER?

Problemas que implican la deducción lógica a partir de una base de conocimientos y reglas

Se utiliza en áreas como la inteligencia artificial, la representación del conocimiento, la inferencia de datos, la planificación y la resolución de problemas en general.

VENTAJAS Y DESVENTAJAS

de la programación lógica

VENTAJAS

✓ Naturaleza declarativa

Programación hecha describiendo como es la solución, y no como resolver el problema :

- facilidad
- rapidez

en la escritura de programas y la comprensión

✓ Capacidad de inferencia

- Lenguajes hechos para realizar operaciones de inferencia lógica complejas
- Adecuado para procesamiento de lenguaje natural, representación del conocimiento

✓ Sistema de backtracking

- Puede buscar a través de un gran espacio de soluciones
- Poderoso para problemas que se representan en forma de árboles

✓ Sintaxis sencilla

- Pocas palabras reservadas
- Sin declaración de variables

Naturaleza declarativa

Es difícil de entender porque es una manera de pensar diferente, pero una vez que se entiende :

- Descripción de la solución en vez de como resolver el problema : más **sencillo**
- **Rapidez** en la escritura del programa (porque hay menos complejidad)

Ejemplo

```
girl(Mary).
girl(Lily).
boy(John).
boy(Peter).
mother(Mary, Lily).
mother(Mary, Peter).
father(John, Lily).
father(John, Peter);
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

Para preguntar cuales son las parejas padre/hijo, preguntamos *parent(X,Y)* al programa

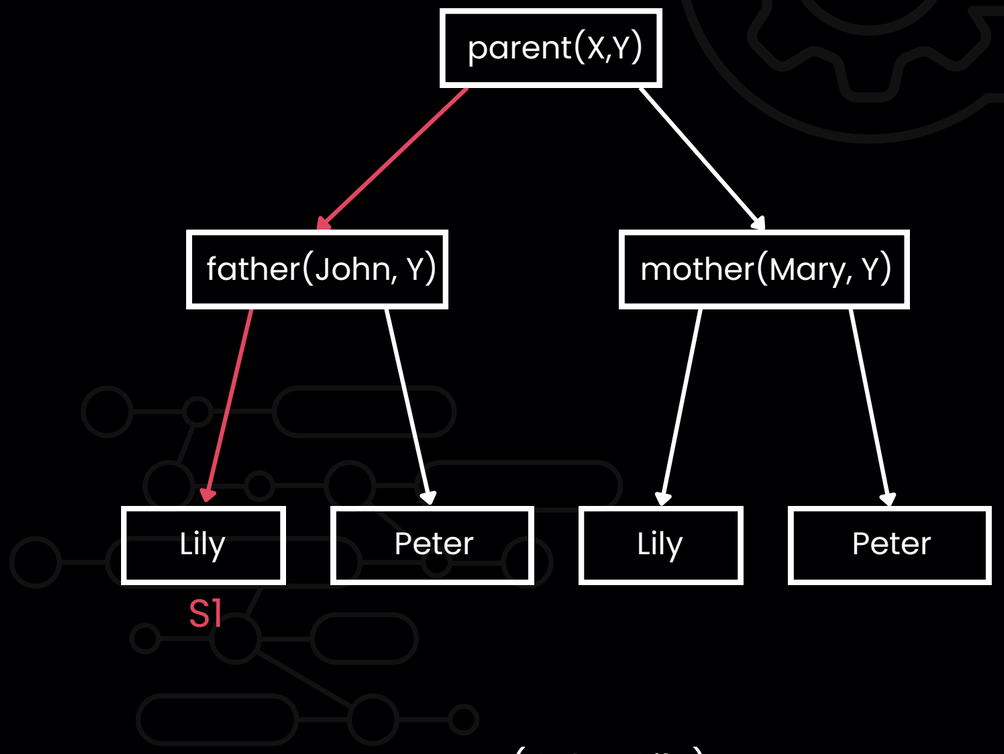
Sistema de backtracking

- Permite encontrar todas las soluciones de un problema, mientras le preguntamos por otras soluciones : búsqueda **exhaustiva**
- Funciona bien para árboles : **eficiente**
- Busca a través de un gran espacio de soluciones

Ejemplo

```
girl(Mary).
girl(Lily).
boy(John).
boy(Peter).
mother(Mary, Lily).
mother(Mary, Peter).
father(John, Lily).
father(John, Peter);
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

parent(X,Y) ?



parent(John, Lily)

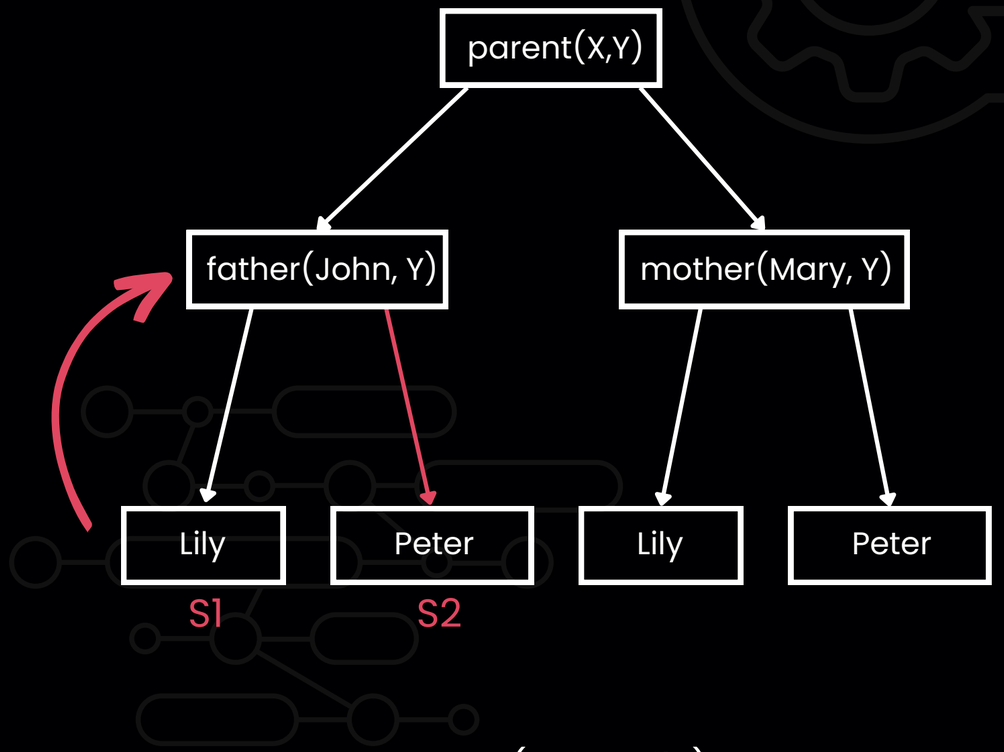
Sistema de backtracking

- Permite encontrar todas las soluciones de un problema, mientras le preguntamos por otras soluciones : busqueda **exhaustiva**
- Funciona bien para arboles : **eficiente**
- Busca a través de un gran espacio de soluciones

Ejemplo

```
girl(Mary).
girl(Lily).
boy(John).
boy(Peter).
mother(Mary, Lily).
mother(Mary, Peter).
father(John, Lily).
father(John, Peter);
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

parent(X,Y) ?



parent(John, Lily)

parent(John, Peter)

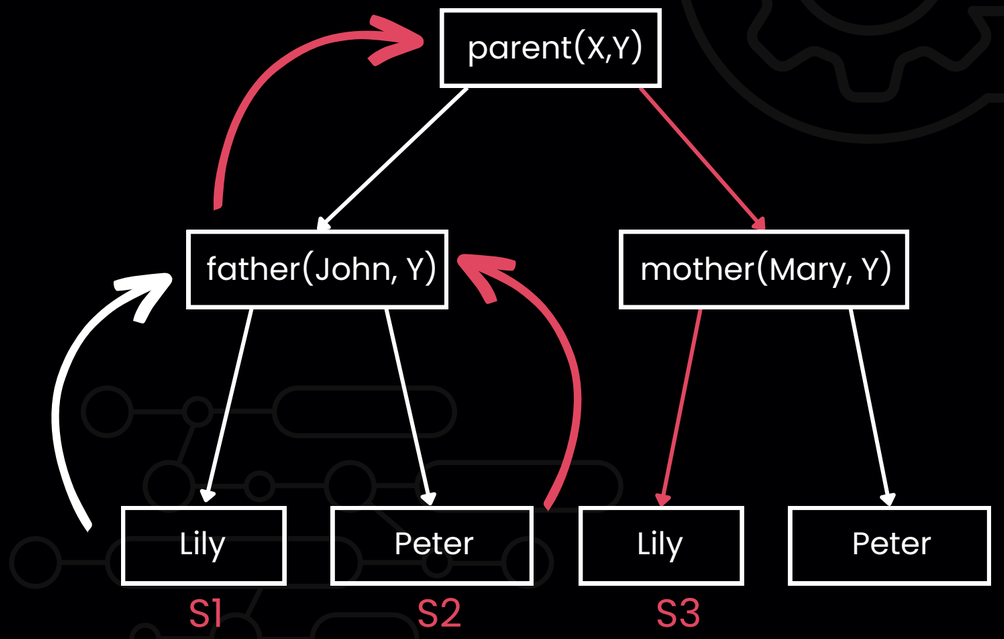
Sistema de backtracking

- Permite encontrar todas las soluciones de un problema, mientras le preguntamos por otras soluciones : busqueda **exhaustiva**
- Funciona bien para arboles : **eficiente**
- Busca a través de un gran espacio de soluciones

Ejemplo

```
girl(Mary).
girl(Lily).
boy(John).
boy(Peter).
mother(Mary, Lily).
mother(Mary, Peter).
father(John, Lily).
father(John, Peter);
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

parent(X,Y) ?



parent(John, Lily)

parent(John, Peter)

parent(Mary, Lily)

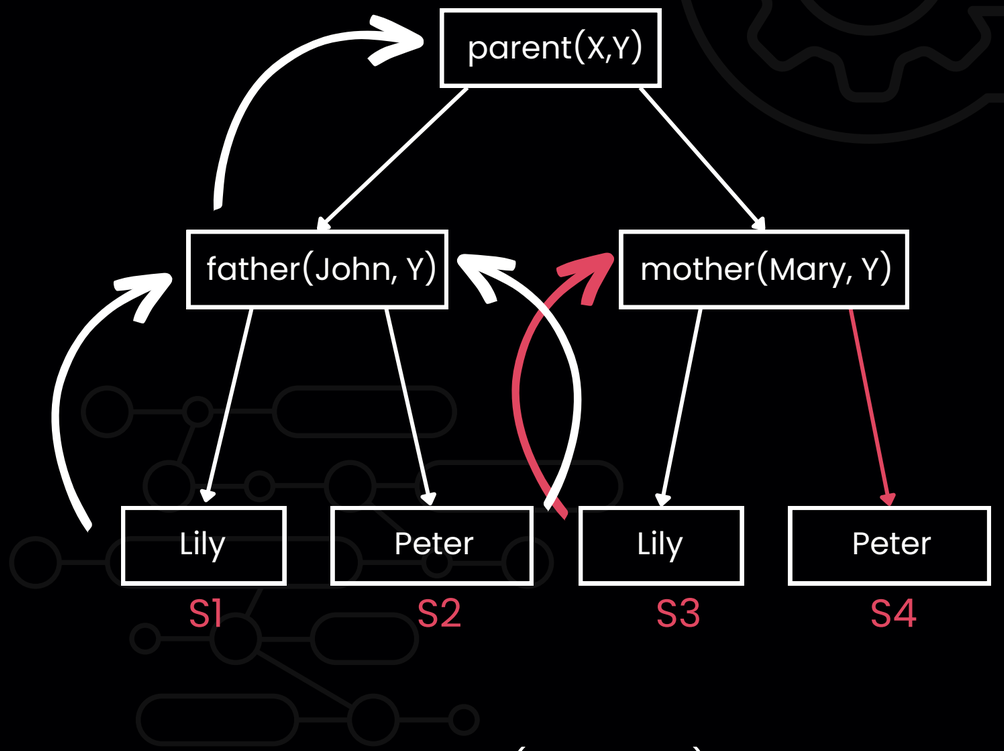
Sistema de backtracking

- Permite encontrar todas las soluciones de un problema, mientras le preguntamos por otras soluciones : busqueda **exhaustiva**
- Funciona bien para arboles : **eficiente**
- Busca a través de un gran espacio de soluciones

Ejemplo

```
girl(Mary).
girl(Lily).
boy(John).
boy(Peter).
mother(Mary, Lily).
mother(Mary, Peter).
father(John, Lily).
father(John, Peter);
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

parent(X,Y) ?



parent(John, Lily)

parent(John, Peter)

parent(Mary, Lily)

parent(Mary, Peter)

Capacidad de inferencia

- Utiliza el sistema de backtracking
- Razonamiento **automático**, hecho por el lenguaje
- Puede manejar consultas complejas y ambiguas (gracias al proceso de unificación)

Ejemplo

```
father(John, Paul).
father(John, Peter).
father(John, Tom).
father(Peter, Tim).
father(Paul, Bob).
grandfather(X,Z) :- father(X,Y),
                    father(Y,Z).
```

grandfather(X,Bob) ?

father(X,Y)

X = {John, Peter, Paul},
Y = {Paul, Peter, Tom, Tim, Bob}

Capacidad de inferencia

- Utiliza el sistema de backtracking
- Razonamiento **automático**, hecho por el lenguaje
- Puede manejar consultas complejas y ambiguas (gracias al proceso de unificación)

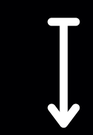
Ejemplo

```
father(John, Paul).
father(John, Peter).
father(John, Tom).
father(Peter, Tim).
father(Paul, Bob).
grandfather(X,Z) :- father(X,Y),
                    father(Y,Z).
```

grandfather(X,Bob) ?

father(X,Y)

X = {John, Peter, Paul},
Y = {Paul, Peter, Tom, Tim, Bob}



father(Y,Z)

X = {John, Peter, Paul},
Y = {Paul, Peter, Tom},
Z = {Tim, Bob}

Capacidad de inferencia

- Utiliza el sistema de backtracking
- Razonamiento **automático**, hecho por el lenguaje
- Puede manejar consultas complejas y ambiguas (gracias al proceso de unificación)

Ejemplo

```
father(John, Paul).
father(John, Peter).
father(John, Tom).
father(Peter, Tim).
father(Paul, Bob).
grandfather(X,Z) :- father(X,Y),
                    father(Y,Z).
```

grandfather(X,Bob) ?

father(X,Y)

X = {John, Peter, Paul},
Y = {Paul, Peter, Tom, Tim, Bob}

father(Y,Z)

X = {John, Peter, Paul},
Y = {Paul, Peter, Tom},
Z = {Tim, Bob}

Z = Bob

X = {John},
Y = {Paul},
Z = {Bob}

DESVENTAJAS

✗ Códigos largos/difíciles de depurar

- Descripción completa del espacio de soluciones : mucho código
- Pocas herramientas de depuración, y poco eficientes

✗ Pocas aplicaciones

- Problemas de desempeño : problemas con procesamiento en tiempo real / aplicaciones de alto rendimiento

✗ Ineficiencia

- Por el sistema de backtracking o la sobrecarga del motor de inferencia, los lenguajes lógicos pueden ser muy lentos

✗ Poca comunidad

- Poco mejoramiento de los lenguajes, como no son muy utilizados
- Soporte de herramienta y recursos limitados

LENGUAJES DE PROGRAMACIÓN

en programación lógica

PRINCIPALES LENGUAJES DE PROGRAMACIÓN LÓGICA

PROLOG

Uno de los mas antiguos y famosos lenguajes de programación lógica. Viene de **Pro**gramación **Lóg**ica.

+ Interoperabilidad
+ Fácil de entender

– Manipulación de datos limitada
– Limitado en programación orientada a objetos

Aplicaciones principales

- Inteligencia artificial
- Procesamiento del lenguaje
- Sistemas expertos

MERCURY

Lenguaje de programación lógica y funcional, basado en Prolog (sintaxis, funcionamiento)

+ Rendimiento
+ Modularidad

– Complejidad del lenguaje
– No interoperabilidad

Aplicaciones principales

- Inteligencia artificial
- Procesamiento del lenguaje
- Desarrollo de juegos

OTROS LENGUAJES

DATALOG

- Basado en Prolog
- Cada formula es una clausula de Horn
- Utilizado para la representación del conocimiento y las bases de datos deductivas

OZ

- Lenguaje combinando ideas de programación lógica, orientada a objetos y de restricciones
- Utilizado en procesamiento del lenguaje natural, programación distribuida, IA

ASP

- ASP = Answer Set Programming
- Conjunto de reglas definiendo las relaciones entre objetos
- Utilizado en procesamiento del lenguaje natural, bioinformática, análisis de las redes sociales

EJEMPLOS CON PROLOG Y MERCURY

EJEMPLO SENCILLO DE PROLOG

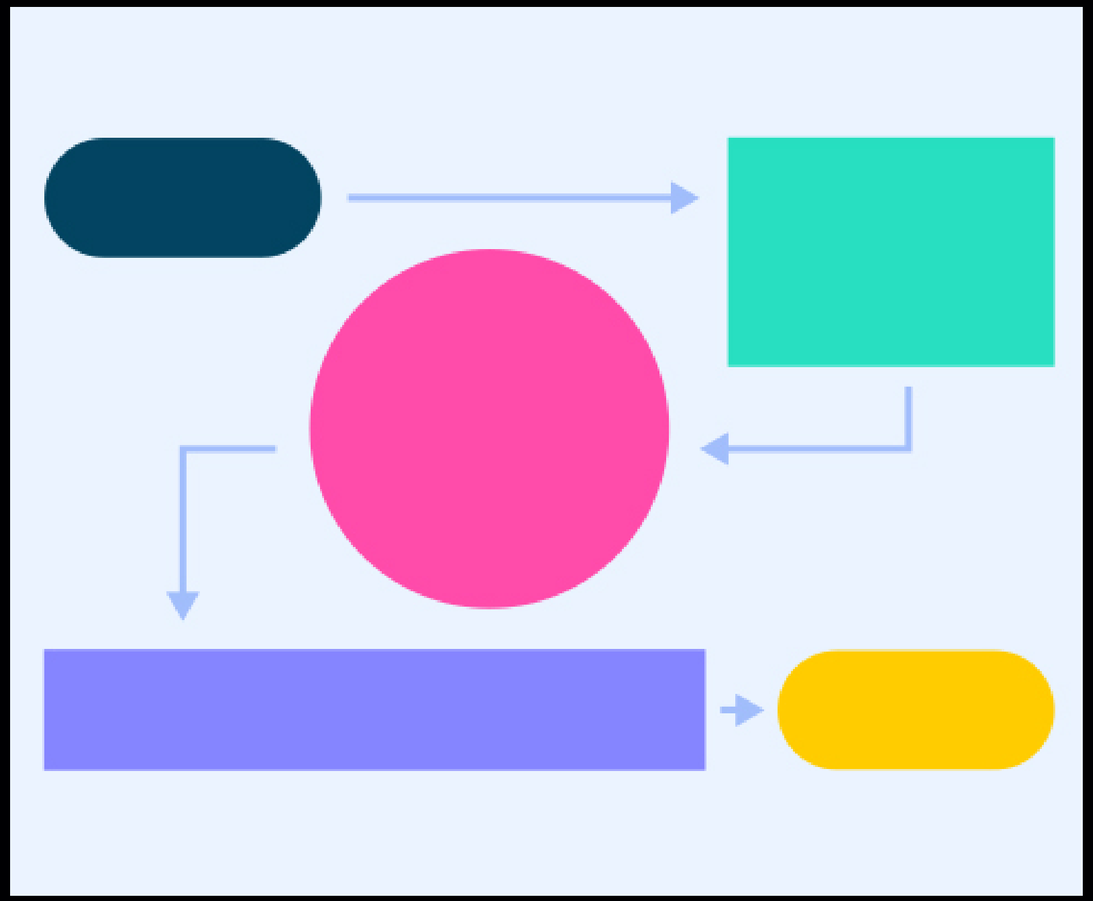
Notebooks

Es posible usar Notebooks con las siguientes herramientas

https://github.com/Calysto/calysto_prolog

https://github.com/madmax2012/SWI-Prolog-Kernel

Lo ideal es usarlo en local y usar "bridges" para conectarlo con otros lenguajes



https://shorturl.at/gpxU1

TORRES DE HANOI

```
1  hanoi(1, X, Y, _):-  % base case
2      write('Move top disk from '),
3      write(X),
4      write(' to '),
5      write(Y),
6      nl.
7  hanoi(N, X, Y, Z):-  % recursion
8      N>1,
9      M is N-1,
10     hanoi(M, X, Z, Y),
11     hanoi(1, X, Y, _),
12     hanoi(M, Z, Y, X).
```

```
?- hanoi(3, left,right,center).
```

```
hanoi(0,_,_,_).

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


def_pasos(Origen,Destino):-
    write(' desde '),
    write(Origen),
    write(' hasta '),
    write(Destino),
    write('\n').
```

```
hanoi(N):-move(N,'A','C','B').

move(0,_,_,_):-!.
move(N,Src,Dest,Spare):-M is N-1,
                        move(M,Src,Spare,Dest),
                        primitive(Src,Dest),
                        move(M,Spare,Dest,Src).

primitive(X,Y):-writelist([move, a, disk, from, X, to, Y]),
                nl.

writelist([]).
writelist([H|T]):-write(H),
                  write(' '),
                  writelist(T).
```

Greta Mae Evans

IT Programming

PROBLEMA DE LOS 4 COLORES

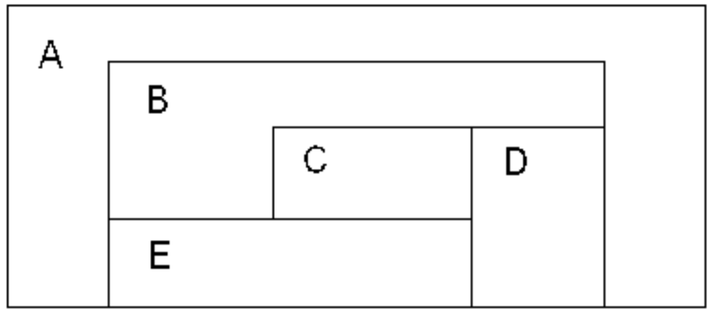
Buscar una manera en que los recuadros adyacentes, tengan un color diferente, teniendo unicamente 4 colores

2.2 The 4-Color Problem

```
map( A, B, C, D, E) :-
    adjacent( A, B), adjacent( A, D), adjacent( A, E),
    adjacent( B, C), adjacent( B, D), adjacent( B, E),
    adjacent( C, D), adjacent( C, E),
    adjacent( D, E).


color( red).
color( bue).
color( yellow).
color( green).


adjacent( X, Y) :- color( X), color (Y), X \= Y.
```



EJEMPLO EN MERCURY

01

Como podemos ver en los ejemplos la sintaxis de prolog y de mercury es muy parecida

02

En mercury se prioriza la escalabiilidad, escribir codigo para futuro

```
:- type list(T)
    --->    [ T | list(T) ]
    ;       [].
```

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

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

```
mother(paul,  faye).
mother(james, faye).

parent(C, P) :-
    mother(C, P).
parent(C, P) :-
    father(C, P).

sibling(A, B) :-
    parent(A, P),
    parent(B, P).
```

```
:- module daytype.
:- interface.
:- import_module io.
:- pred main(io::di, io::uo) is det.
:- implementation.
:- import_module string, list.

:- type days
    --->    sunday
    ;       monday
    ;       tuesday
    ;       wednesday
    ;       thursday
    ;       friday
    ;       saturday.

:- func daytype(days) = string.
daytype(D) = R :-
    (
        (D = sunday ; D = saturday),
        R = "weekend"
    ;
        (D = monday ; D = tuesday ; D = thursday ; D = friday),
        R = "workday"
    ;
        D = wednesday,
        R = "humpday"
    ).

main(!IO) :-
    io.command_line_arguments(Args, !IO),
    ( if
        Args = [DayString],
        Lowered = to_lower(DayString),
        Term = Lowered ++ ".",
        io.read_from_string("args", Term, length(Term),
            ok(Day), io.posn(0, 0, 0), _)
    then
        io.format("daytype(%s) = ""%s"".\n",
            [s(Lowered), s(daytype(Day))], !IO)
    else
        io.progname_base("daytype", Program, !IO),
        io.format(io.stderr_stream, "usage: %s <weekday>\n", [s(Program)], !IO),
        io.set_exit_status(1, !IO)
    ).
```

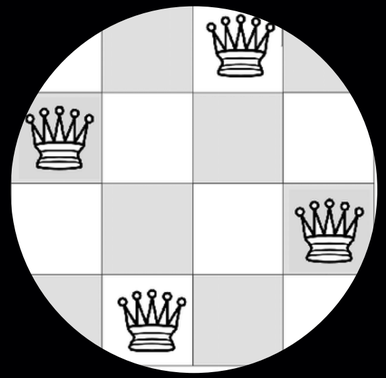
APLICACIONES REALES Y DOMINIOS

APLICACIONES

Revisión de Codigo

Lingüistica computacional

Problemas de lógica matemática



Inteligencia Artifical

CASOS REALES

Siemens

Programación lógica en el proceso de automatización de tareas
Y creación de maquinaria médica

CYC

Proyecto (ahora empresa) de desarrollo de Modelos de IA, más que todo para equipo

Nació con prolog tienen su propio lenguaje que también se basa en declaraciones de primer orden lógico

MÁS CASOS

Datalog y aplicaciones

Usada por ibm en su sitema de base de datos en DB2, así mismo se suele usar en bases de datos deductivas. es su principal uso.

Un paper que muestra el uso de estas bases de datos para analizar estructuras de proteínas.

https://academic.oup.com/bioinformatics/article/9/3/259/225149?login=false

Así mismo los motores de recomandación como los de amazon también lo usan.

MYCIN

Sistema de inferencia medico, un sistema experto para diagnosticar enfermedades infecciosas.

Prolog NLP

Hay muchas implementaciones para la manipulación de y procesado de lenguaje natural, suele usarse como paso intermedio.


GRACIAS

REFERENCIAS

- Mercury. Mercurylang. Tomado de: https://mercurylang.org/about.html
- The Gödel Programming Language. Tomado de: https://dennisdarland.com/my_sw_projects/goedel/v1_3_27/doc/book.pdf
- McCarthy, J. (n.d.). Datalog: Deductive Database Programming. Tomado de: https://docs.racket-lang.org/datalog/
- Ramírez, L. ¿Qué es una plataforma low code o programación de bajo código? IEBS. Tomado de: https://www.iebschool.com/blog/que-es-low-code-big-data/
- Genexus.Tomado de: https://www.genexus.com/es/
- Programación lógica. Paradigmas de programación. Tomado de : http://ferestrepoca.github.io/paradigmas-de-programacion/proglogica/logica_teoria/index.html
- "Programming in Prolog" de William Clocksin y Christopher Mellish
- "Artificial Intelligence: A Modern Approach" de Stuart Russell y Peter Norvig
- "The Logic Programming Paradigm: A 25-Year Perspective" de Michael Kifer, Wlodek Drabent y Agnieszka Lawrynowicz
- "Logic for Computer Science: Foundations of Automatic Theorem Proving" de Jean Gallier
- Nosco. (s.f.). Ejercicio 2: Torres de Hanoi. Recuperado de https://www.nosco.ch/ai/en/exercise_02.php#2.2
- Angelfire. (s.f.). Las Torres de Hanoi. Recuperado de https://www.angelfire.com/space/leosan/torres.html
- JC-Info. (2009, 26 de enero). Torres de Hanoi en Prolog (código). Recuperado de http://jc-info.blogspot.com/2009/01/torres-de-hanoi-en-prolog-cdigo.html
- Universidad de Huelva. (s.f.). Tema 4: Programación en Prolog. Recuperado de http://www.uhu.es/nieves.pavon/pprogramacion/temario/tema4/tema4.html
- Mercury-in.space. (s.f.). Crash Course in Mercury Programming. Recuperado de https://mercury-in.space/crash.html#orgd292251
- Mercury. (2005). Mercury: A Logic Based Approach to Functional Programming. Recuperado de https://mercurylang.org/documentation/papers/mfug_talk.pdf
- CoCalc. (s.f.). Examples in Prolog. Recuperado de https://cocalc.com/share/public_paths/401

REFERENCIAS

- "The family of concurrent logic programming languages", Ehud y. Shapiro, 1989
- "Predicate logic as programming language", Robert Kowalski, 1974
- "Logic Programming in Oz with Mozart", Peter Van Roy, 1999
- "Définition et explication de la programmation logique", tomado de techno-science.net
- "Avantages et inconvénients de Prolog", tomado de développez.net
- "Logic Programming", tomado de wikipedia
- "Programación declarativa: cuando el qué es más importante que el cómo", tomado de ionos.es
- "Prolog : backtracking", tomado de tutorialspoint.net
- "Using Prolog's inference engine", tomado de https://www.amzi.com/ExpertSystemsInProlog/02usingprolog.htm
- "The Gödel Programming Language", P.M. Hill and J.W. Lloyd, 1992
- "The Mozart Programming system", tomado de http://mozart2.org/
- "Programmation logique : présentation de Prolog", L. Rozé, P. Sebillot, INSA Rennes
- "Programmation logique : cahier de TP", B. Coüasnon, P. Sebillot, L. Rozé, P. Garcia, Y. Ricquebourg
- "The Art of Prolog", Leon Sterling and Ehud Shapiro, 2nd edition, The MIT Press, 1994
- "The Craft of Prolog", Richard A. O'Keefe, The MIT Press, 1990
- "Programming in Prolog", William F. Clocksin and Chris S. Mellish, 5th edition, Springer Verlag, 2003