Programación Lógica

Luis Alfonso
Sergio Rivera
Pedro Valderrama

Contenido

1. Filosofía del paradigma
2. Historia
3. Conceptos claves
4. Ventajas y desventajas
5. Lenguajes de programación
6. Ejemplos en distintos lenguajes
7. Aplicaciones de este paradigma
8. Referencias/bibliografía

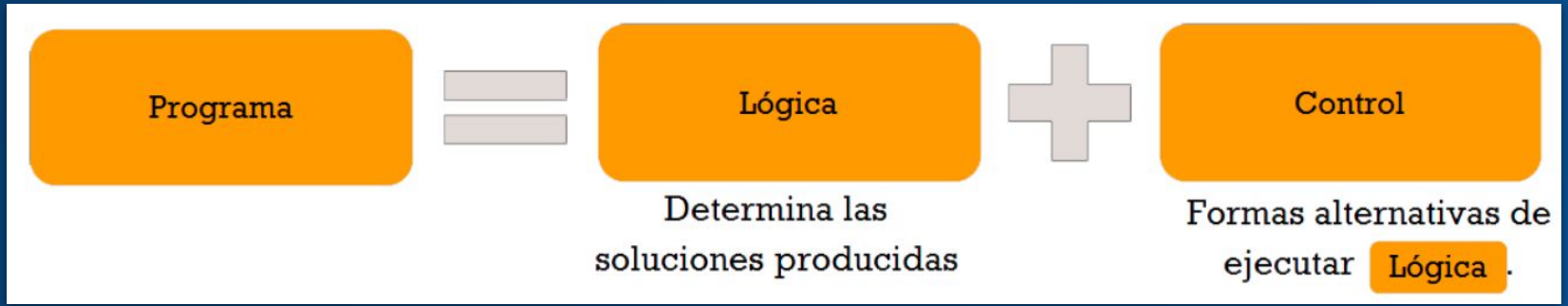
Filosofía del Paradigma

"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 Claves - ¿Qué es?

Paradigma de programación basado en la lógica de primer orden. La programación lógica estudia el uso de la lógica para el planteamiento de problemas y el control sobre las reglas de inferencia para alcanzar la solución automática.



Conceptos Claves - ¿Que trata de resolver?

Dado un problema S, saber si la afirmación A es solución o no del problema o en qué casos lo es. Además queremos que los métodos sean implantados en máquinas de forma que la resolución del problema se haga de forma automática

Conceptos Claves - Características del Paradigma

- Unificación de términos.
- Mecanismos de inferencia automática.
- Recursión como estructura de control básica.
- Visión lógica de la computación.

Conceptos Claves - Lógica Proposicional

Proposiciones: Elementos de una frase que constituyen por sí solos una unidad de comunicación de conocimientos y pueden ser considerados verdaderos o falsos.

Proposición Simple: “Pepito es humano”.

Proposición Compuesta: “Pepito es hombre y pepita es mujer”.

Conceptos Claves - Lógica Proposicional

Sentencia	⟶	*Sentencia Atómica* | *(Sentencia Conectiva Sentencia)* | *Cuantificador Variable … Sentencia* | ¬*Sentencia*
Sentencia Atómica	⟶	*Predicado (Término...)* | *Término = Término*
Término	⟶	*Función(Término)* | *Constante* | *Variable*
Conectiva	⟶	^ |v|⇒| ⇔
Cuantificador	⟶	¬*Sentencia*
Variable	⟶	*a* | *x* | *s* | …
Predicado	⟶	*TieneColor* | *EstáLloviendo* | ...
Función	⟶	*Hombre* | *Humano* | *Mujer* | ...

Más expresiva de la Lógica proposicional.

¿Qué se afirma? (predicado o relación)
¿De quién se afirma? (objeto)

Conceptos Claves - Cláusulas de Horn

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

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

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

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

Cláusula ‘definite’: Cláusula de Horn con exactamente un literal positivo.
Hecho: Cláusula ‘definite’ sin literales negativos.
Cláusula objetivo: Sin ningún literal positivo (consulta).

Conceptos Claves - Cláusulas de Horn

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

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

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

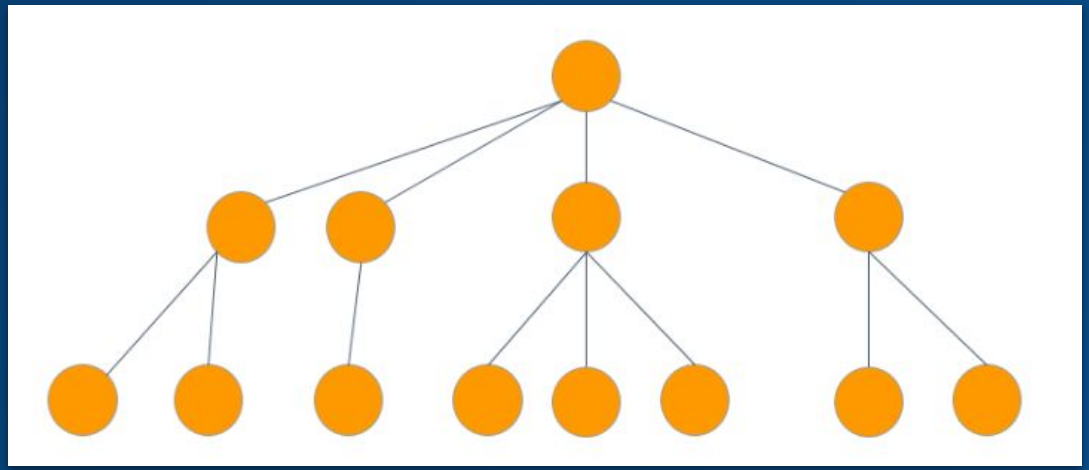
$$(mujer(A) \wedge padre(B, A)) \rightarrow hija(A, B)$$

Cláusula ‘definite’: Cláusula de Horn con exactamente un literal positivo.
Hecho: Cláusula ‘definite’ sin literales negativos.
Cláusula objetivo: Sin ningún literal positivo (consulta).

Conceptos Claves - Resolución SLD (Selective Linear Definite clause resolution)

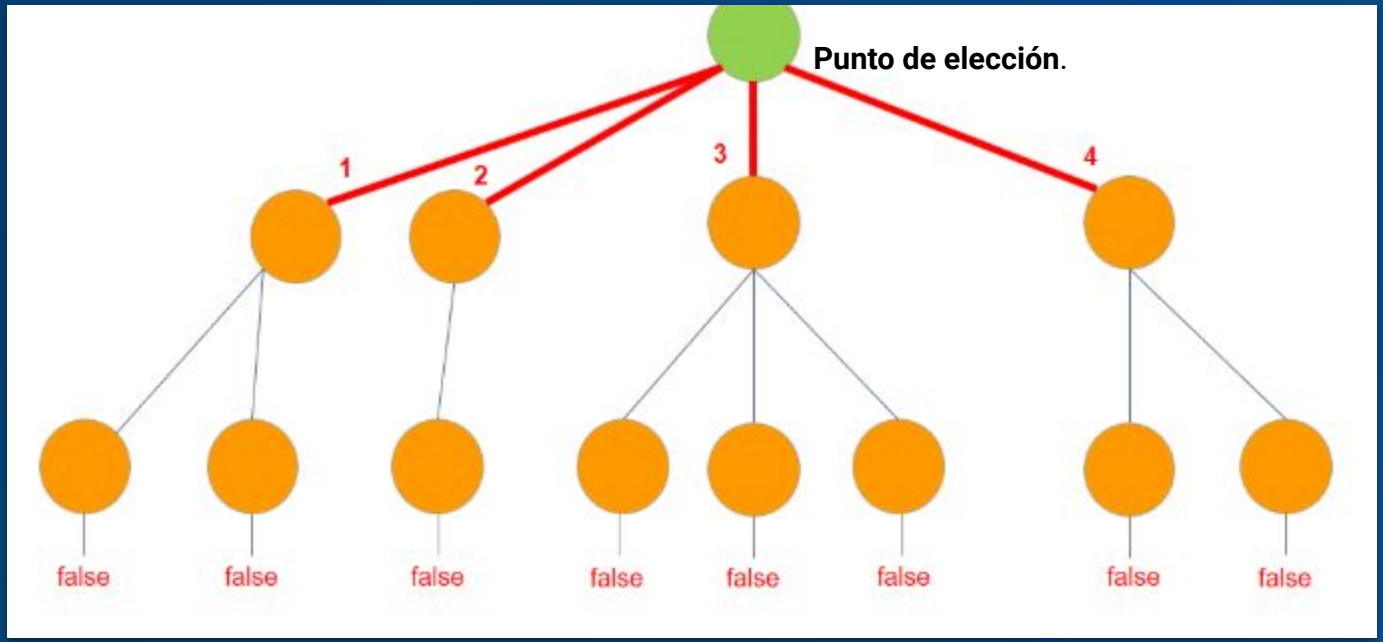
En SLD, todas las cláusulas son una secuencia cláusulas objetivo y el otro padre es una cláusula de entrada. En la resolución SL, el otro padre es una cláusula de entrada o una cláusula ancestral anterior en la secuencia.

Conceptos Claves - Backtracking



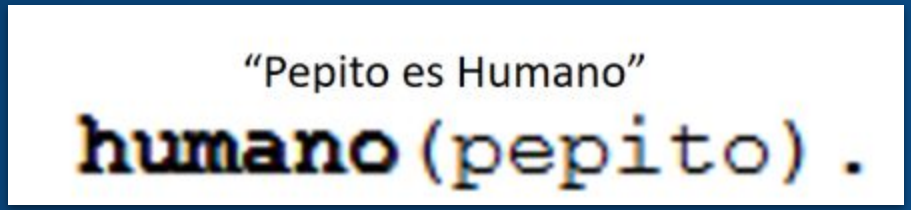
Unificación y backtracking

Conceptos Claves - Backtracking



Conceptos Claves - Hechos

Expresión atómica (declaración, cláusula o proposición) que se formula de la forma P(t1,...,tn), indicando que se verifica el predicado P sobre los objetos t1,...,tn



Conceptos Claves - Reglas

Conjunto de proposiciones lógicas escritas como cláusulas de Horn que permiten inferir el valor de verdad de nuevas proposiciones, permitiendo ampliar la base de conocimientos, a la vez que son utilizadas para definir el dominio del problema.

"x es mortal si x es humano"

```
mortal(X)  :-  humano(X).
```

Conceptos Claves - Consultas

Proposición construida con el propósito de ser demostrada/desmentida o de encontrar el conjunto de valores que la convierten verdadera. En las consultas se especifica el problema a resolver. Partiendo de que los humanos son mortales y de que Pepito es humano (ejemplos anteriores), deducimos que Pepito es mortal.

```
humano(pepito).          Hecho
mortal(X) :- humano(X).  Regla

55 ?- mortal(pepito).
true.
```

Conceptos Claves - Recursión

La recursividad puede ser tratada de una manera más eficaz si se piensa en que hace el algoritmo recursivo que se piensa aplicar, en vez de cómo hacerlo. Para esto se usará la modularidad, la cual se basa en separar el problema en otros más pequeños y hallar la solución a estos para luego unirlos, como es usual en la programación lógica.

Historia

- Aristóteles (384-322 a.C.) teoría silogística
- René Descartes y Gottfried Leibnitz. siglo XVII
- George Boole (1815-1864) relación entre la lógica y el álgebra
- Frege, Cantor, Peano, Russell, Whitehead siglo XIX y siglo XX
- Alan Turing (1912-1954) “máquinas de calcular”

Historia


Finales
1930
Cálculos Lambda
Alonzo Church
1969
Forma Causal de la Lógica
Cordell Green
1972
PROLOG
Alain Colmerauer
Philippe Roussel
1970
SLD – resolution
Robert Kowalski

Ventajas y Desventajas

Ventajas:

• La programación lógica se puede utilizar para expresar conocimiento de una manera que no dependa de implementación, haciendo que los programas sean más flexibles, comprimidos y comprensibles.

• Permite separar el conocimiento del uso, es decir, se puede cambiar la arquitectura de la máquina sin cambiar los programas o su código subyacente.

• Se puede modificar y ampliar de manera natural para apoyar otros paradigmas de desarrollo.

• Se puede usar en disciplinas no computacionales que dependen del razonamiento y medios precisos de expresión.

Desventajas:

- **Inicialmente, debido a una inversión insuficiente en tecnologías complementarias, los usuarios tenían problemas en acceso a los servicios**

- **Al principio, las definiciones deficientes para soportar aritmética, tipos, etc. tuvieron un efecto desalentador en la comunidad de desarrolladores.**

- **No hay una forma adecuada de representar los conceptos computacionales como las variables de estado (de la manera en la que se encuentra en lenguajes convencionales).**

- **Algunos programadores prefieren el paradigma de programas operados por máquina (tipo máquina de Turing), ya que prefieren el control activo que los elementos "móviles" de la programación lógica.**

Lenguajes de Programación

- ALF (Algebraic Logic Functional)
- Alma-0
- Curry
- Fril
- Prolog
- Mercury (se basa en Prolog)
- Visual Prolog(Se basa en Prolog)
- CLPR (Constraint Logic Programming Real)
- CSP
- Lambda Prolog
- Logtalk (se basa en Prolog)
- Gödel
- SequenceL

Alain Colmerauer

Lenguajes de Programación - Prolog

- PROgraming in LOGic por el francés Philippe Roussel
- Creado por Philippe Roussel y Alain Colmerauer en 1972
- Basado en las interpretaciones de cláusulas de Horn de Robert Kowalski
- lenguaje más relacional que funcional

Lenguajes de Programación - ALF

- ALF (Algebraic Logic Functional)
- Creado por Michael Hanus en 1995
- Basado en las cláusula de Horn con igualdad que consisten en predicados y cláusulas de horn para programación lógica
- Usa una gramática independiente del contexto y la sintaxis es similar a la de Prolog

Ejemplos en distintos lenguajes - ALF

```
module nats.

   export 0, s, +, < .

   datatype nat = { 0 ; s(nat) }.

   func  + : nat, nat -> nat infixleft 500.

   pred  < : nat, nat  infix.

rules.

   N + 0    = N.
   N + s(M) = s(N + M).

   0    < s(M).
   s(N) < s(M) :- N < M.

end nats.
```

```
module list(elem).

   export [] , '.' , append , member.

   datatype list = { '.'(elem,list) ; [] }.

   func append: list, list -> list.

   pred member : elem, list.

rules.

   append([],L)     = L.
   append([H|T],L) = [H|append(T,L)].

   member(E,[E|_]).
   member(E,[_|L]) :- member(E,L).

end list.
```

Ejemplos en distintos lenguajes - ALF

```
module natord.

   export 0 , s , + , < , > , =/= , =< , >= .

   use nats.

   pred >     : nat, nat infix;
        =/=   : nat, nat infix;
        =<    : nat, nat infix;
        >=    : nat, nat infix.

rules.

   N > M   :- M < N.
   N =/= M :- N < M.
   N =/= M :- N > M.
   N =< N.
   N =< M  :- N < M.
   N >= N.
   N >= M  :- N > M.

end natord.
```

```
module isort.

   export isort.

   use natord;
       list(nat).

   func isort : list -> list;
        insert: nat, list -> list.

rules.

   isort([]) = [].
   isort([E|L]) = insert(E,isort(L)).

   insert(E,[])    = [E].
   insert(E,[F|L]) = [E,F|L]          :- E =< F.
   insert(E,[F|L]) = [F|insert(E,L)]  :- E > F.

end isort.

?- isort([3,1,5,4,1,3,2]) = L.
```

Lenguajes de Programación - Mercury

- Diseñado por Zoltan Somogyi en la Universidad de Melbourne
- Es de tipado fuerte
- Está basado en Prolog y tiene una sintaxis similar, pero en la práctica es más rápido que este
- Es un lenguaje puramente declarativo
- Diseñado para resolver “aplicaciones del mundo real” de forma robusta.
- compatible con el polimorfismo.

Ejemplos en distintos lenguajes - Mercury

```
:- module hello.

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

:- implementation.
main(IO0, IO1) :-
    io.write_string("Hello World!\n", IO0, IO1).
```

```
:- module fact.

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

:- implementation.
:- import_module int.
:- pred fact(int::in, int::out) is det.

fact(N, X) :-
  ( N = 1 -> X = 1 ; fact(N - 1, X0), X = N * X0 ).

main(!IO) :-
  fact(5, X),
  io.print("fact(5, ", !IO),
  io.print(X, !IO),
  io.print(")\n", !IO).
```

Lenguajes de Programación - Gödel

- Diseñado por John Lloyd y Patricia Hill
- también es de tipado fuerte
- Es un lenguaje puramente declarativo
- Soporta polimorfismo.
- Llamado así por el lógico y matemático Húngaro Kurt Gödel

Ejemplos en distintos lenguajes - 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.
```

Lenguajes de Programación - Logtalk

- Diseñado por Paulo Moura en 1998
- lenguaje multiparadigma
- Es un lenguaje cuyo objetivo es traer los beneficios de la programación orientada a objetos y la programación lógica
- la sintaxis está basada en Prolog, por lo cual si se conoce prolog su aprendizaje es más fácil

Ejemplos en distintos lenguajes - Logtalk

```
:- object(my_first_object).

    :- initialization((write('Hello world'), nl)).

    :- public(p1/0).
    p1 :- write('This is a public predicate'), nl.

    :- private(p2/0).
    p2 :- write('This is a private predicate'), nl.

:- end_object.
```

```
?- logtalk_load(my_first_object).
Hello world
true.

?- my_first_object::p1.
This is a public predicate
true.
```

Aplicaciones de este paradigma

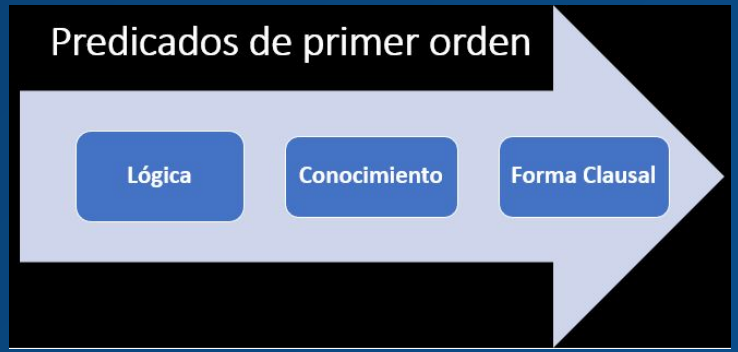
- Inteligencia artificial
- Sistemas Expertos
- Demostración automática de problemas
- Reconocimiento de lenguaje natural
- Sistemas de administración de bases de datos relacionales
- Procesamiento de lenguaje natural
- Resolución de ecuaciones simbólicas
- Simulación
- Creación de prototipos
- Machine Learning

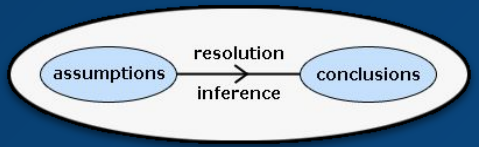
Inteligencia artificial

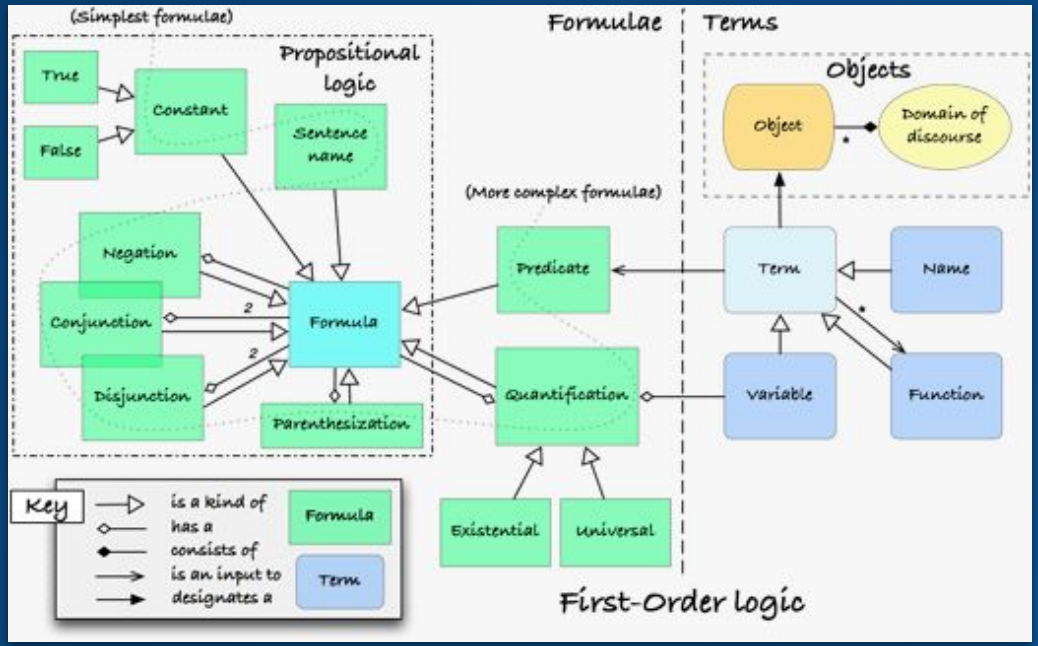
La Inteligencia Artificial (IA) es la capacidad de una máquina artificial para actuar de manera “inteligente”

La programación lógica es un método para tratar de permitir que las máquinas razonen.

La lógica se usa para representar el conocimiento y la inferencia se usa para manipularlo.




assumptions
resolution
inference
conclusions


(Simplest formulae)
Propositional logic
Formulae
Terms
Objects
True
False
Constant
Sentence name
Object
Domain of discourse
(More complex formulae)
Negation
Conjunction
Disjunction
2
2
Formula
Parenthesization
Predicate
Term
Name
Quantification
Variable
Function
Existential
Universal
Key
is a kind of
has a
consists of
is an input to
designates a
Formula
Term
First-Order logic

Sistemas Expertos

Los sistemas expertos son un subconjunto de los sistemas de toma de decisiones.

Usualmente los Sistemas expertos por medio de programación Lógica se desarrollan en ProLog

Un concepto fundamental en Prolog en la *Tabla de Factores*

Una tabla de factores es una tabla simple que se puede hacer en Excel y es parte de la Representación del conocimiento.

La representación del conocimiento es una breve nota del sistema experto.

En la tabla de factores solo representamos el conocimiento que tenemos y esto nos facilita escribir reglas y hechos.

Tabla de factores para ti

- ENTJ, INTJ, ENTP, INTP
- INFJ, INFP, ENFJ, ENFP
- ISTJ, ISFJ, ESTJ, ESFJ
- ISTP, ISFP, ESTP, ESFP

For the purpose of this tutorial I'm only considering 2 types of moods. We are categorizing moods generally in these two types:

- Happy
- Sad

So, we just need to ask the person these questions:

1. Are you an **Introvert or Extrovert**?
   Are you **Intuitive or Observant**?
   Are you **Thinking or Feeling**?
   Are you **Judging or Prospecting**?

Personality 1	Personality 2	Personality 3	Personality 4	Personality Type	Genre 1	Genre 2	Genre 3
Extroverts (E)	Intuitive (N)	Thinking (T)	Judging (J)	ENTJ	Jazz (64%)	Classical (79%)	Electronica (70%)
Introverts (I)	Intuitive (N)	Thinking (T)	Judging (J)	INTJ	Classical (78%)	Metal (42%)	Alternate Rock
Extroverts (E)	Intuitive (N)	Thinking (T)	Prospecting (P)	ENTP	Classical (76%)	Rock (84%)	Alternate Rock (88%)
Introverts (I)	Intuitive (N)	Thinking (T)	Prospecting (P)	INTP	Punk (51%)	Rock (82%)	Metal (48%)
Introverts (I)	Intuitive (N)	Feeling (F)	Judging (J)	INFJ	Alternate Rock (84%)	World (46%)	Classical
Introverts (I)	Intuitive (N)	Feeling (F)	Prospecting (P)	INFP	Punk (49%)	Rock (82%)	Alternate Rock (86%)
Extroverts (E)	Intuitive (N)	Feeling (F)	Judging (J)	ENFJ	Jazz (64%)	World (52%)	Blues (52%)
Extroverts (E)	Intuitive (N)	Feeling (F)	Prospecting (P)	ENFP	Jazz (62%)	Electronica (75%)	Ambient (65%)
Introverts (I)	Observant (S)	Thinking (T)	Judging (J)	ISTJ	Rock	Alternative Rock	Pop
Introverts (I)	Observant (S)	Feeling (F)	Judging (J)	ISFJ	Religious (42%)	Rock	Alternative Rock
Extroverts (E)	Observant (S)	Thinking (T)	Judging (J)	ESTJ	Hip-Hop (57%)	Religious (48%)	Pop
Extroverts (E)	Observant (S)	Feeling (F)	Judging (J)	ESFJ	Pop (80%)	Soul (57%)	Country (53%)
Introverts (I)	Observant (S)	Thinking (T)	Prospecting (P)	ISTP	Punk (48%)	Metal	Alternate Rock
Introverts (I)	Observant (S)	Feeling (F)	Prospecting (P)	ISFP	Reggae (46%)	Ambient (64%)	Pop (78%)
Extroverts (E)	Observant (S)	Thinking (T)	Prospecting (P)	ESTP	Electronica (79%)	Metal (50%)	Hip-Hop (58%)
Extroverts (E)	Observant (S)	Feeling (F)	Prospecting (P)	ESFP	Ambient (62%)	Pop (88%)	Hip-Hop (57%)

Factor Table from Knowledge that we have

- ENTJ is a personality
  $P(x) \rightarrow$ x is a personality
- **Happy/Sad is a mood.**
  $M(x) \rightarrow$ x is a mood.
- https://www.youtube.com/watch?v=c8YIlU_30Kk song is of Jazz Genre.
  $S(x,y) \rightarrow$ song x is of y genre

In predicate logic, it'll be:

$\forall x: S(x,y) \rightarrow M(x) \wedge P(x) \vee P(x) \vee P(x)$

That is the only rule that we'll need as it's a small system.

https://medium.com/analytics-vidhya/learn-prolog-language-by-creating-an-expert-system-d46a811c58e8

Bases de datos relacionales

Sistemas lógicos, como Prolog, se basan en una evaluación orientada a tuplas que utiliza la unificación para unir variables con valores atómicos (tupla).

Por otra parte, el modelo relacional utiliza un mecanismo de evaluación orientado a conjuntos.

Se busca una arquitectura que conecta los dos sistemas, para permitir consultas de alto nivel y que sean eficientes.

¿CÓMO LOGRARLO?

1. Con el acoplamiento de una implementación de un sistema lógico existente a un sistema de gestión de base de datos relacional existente
2. Extender un sistema lógico existente con algunas funcionalidades de gestión de bases de datos relacionales.
3. Extender un sistema de bases de datos relacionales existente con algunas funcionalidades de sistemas lógicos
4. Integrando estrechamente las técnicas de programación lógica con la de sistemas de gestión de bases de datos relacionales

¿Cómo relacionar SQL y Prolog?

Relational Algebra	SQL	Prolog
Π_{SUBJ}(Student)	SELECT SubjCode FROM Student	student(_,_,SubjCode).
Student $\bowtie_{SubjCode=SubjCode}$ Subject	SELECT * FROM Student, SubjCode WHERE Student.SubjCode= Subject.SubjCode	student(SNO,SName,SubjCode), subject(SubjCode,Title).
Student $\sigma_{Name="Wong" \wedge SNO>"1000" \wedge SubjCode=SubjCode}$ Subject	SELECT * FROM Student, Subject WHERE Student.SName="Wong" and Student.SNO>1000 and Student.SubjCode= Subject.SubjCode	student(SNO,"Wong",SubjCode), SNO>1000, subject(SubjCode, Title).
Student × Subject	SELECT * FROM Student, Subject WHERE Student.SNO=1000 and Subject.SubjCode="cs317" and Student.SubjCode="cs317"	student(1000,SName,cs317), subject(cs317,Title).
Intersection	SELECT * FROM Student, Lecturer WHERE Student.SNO=Lecturer.LNO and Student.SName= Lecturer.SName and Student.SubjCode= Lecturer.SubjCode	student(SNO,Sname,SCode), lecturer(SNO,Sname,SCode).
Union	SELECT * FROM Student, Lecturer	student(SNO1,Sname1,SCode1), lecturer(SNO2,Sname2,SCode2).

Reconocimiento de lenguaje natural

La primera aplicación de procesamiento de lenguaje fue escrita por Alain Colmerauer (Grupo de inteligencia Artificial - Universidad de Marsella - 1973). Fue además la primera aplicación de programación lógica.

Colmerauer desarrolló un formalismo gramatical que podría compilarse en Prolog, llamado "Gramáticas de metamorfosis".

Las gramáticas de metamorfosis, , admiten reglas que permiten reescribir símbolos(terminales y no terminales) que son términos lógicos.

```
sentence --> noun_phrase, verb_phrase.

    noun_phrase --> name.

    verb_phrase --> verb.

    name  --> [maria].
    verb  --> [laughs].
```

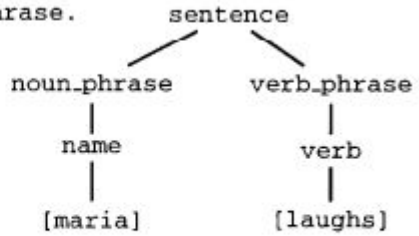


FIGURE 1. A sample grammar and parse tree.

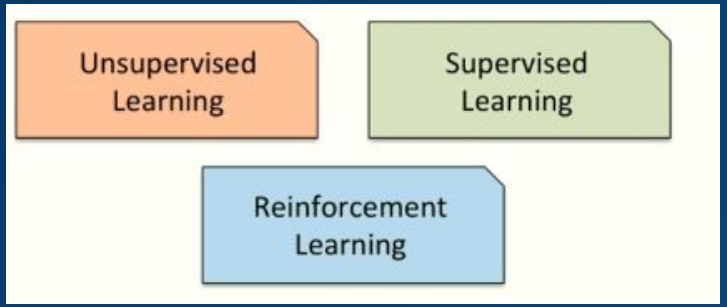
```
sentence(Sem) -->
        noun_Phrase(X), verb_phrase(X,Sem).

noun_phrase(X) --> name(X).

verb_phrase(X,P) --> verb(X,P).

name(maria) --> [maria].
verb(X,laughs(X)) --> [laughs].
```

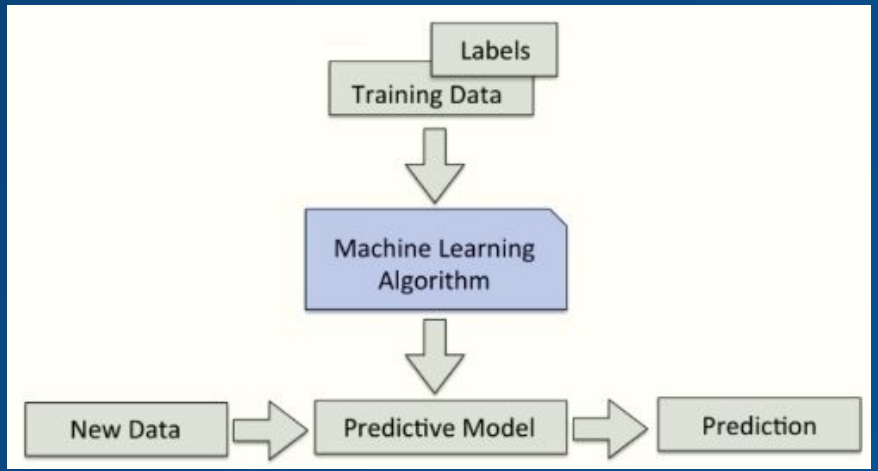
FIGURE 2. A DCG that builds semantic structures.

Machine Learning

Se basa en la Programación Lógica Inductiva

Comprende la construcción de programas lógicos a partir de ejemplos y conocimiento previo


Unsupervised Learning
Supervised Learning
Reinforcement Learning


Labels
Training Data
Machine Learning Algorithm
New Data
Predictive Model
Prediction

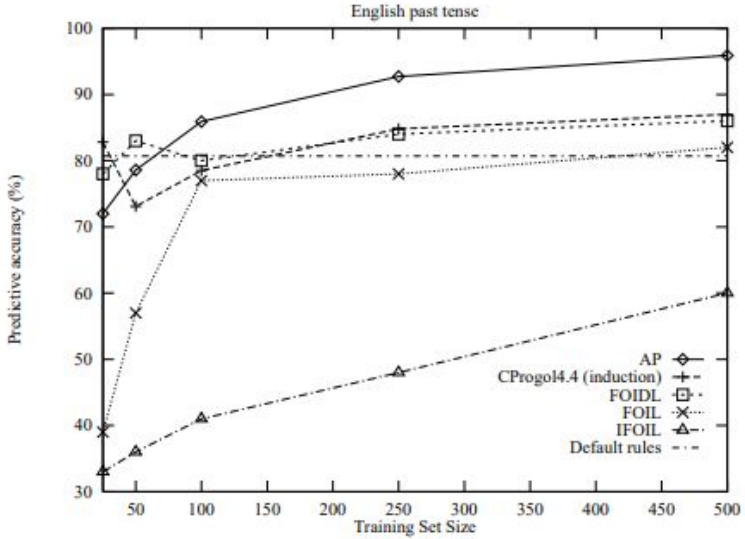
Procesamiento de lenguaje natural

En el aprendizaje de la gramática, la teoría lógica a sintetizar consiste reglas junto con representaciones semánticas.

Los ejemplos son oraciones del idioma que se aprende, complementados con restricciones en las posibles de reglas

Examples	Hypotheses
past([w,o,r,r,y],[w,o,r,r,i,e,d]). past([w,h,i,z],[w,h,i,z,z,e,d]). past([g,r,i,n,d],[g,r,o,u,n,d]).	past(A,B) :- split(A,C,[r,r,y]), split(B,C,[r,r,i,e,d]).

Figure 14.7 Form of examples and hypotheses for past tense domain



Ejemplos e hipótesis de aprendizaje de pasado simple

AP: Procesamiento análogo basado en programación lógica inductiva

Referencias/Bibliografía

- Keller B. Lecture 8: Logic Programming Languages, https://courses.cs.vt.edu/~cs3304/Spring02/lectures/lect08.pdf ,Virginia Tech
- Hanus M. ALF's user Manual, https://www.informatik.uni-kiel.de/~mh/systems/ALF/manual.pdf,
- Lista de lenguajes de programacion logicos https://en.wikipedia.org/wiki/Category:Logic_programming_languages
- Restrepo F. Prog. Logica, http://ferestrepoca.github.io/paradigmas-de-programacion/proglogica/logica_teoria/lenguajes.html
- Double C. Getting started with Mercury https://bluishcoder.co.nz/2019/06/23/getting-started-with-mercury.html
- Logtalk Manual https://logtalk.org/manuals/userman/features.html#integration-of-logic-and-object-oriented-programming