

Metodología de Programación I

Introducción

Dr. Alejandro Guerra-Hernández

Departamento de Inteligencia Artificial
Facultad de Física e Inteligencia Artificial
aguerra@uv.mx
<http://www.uv.mx/aguerra>

Maestría en Inteligencia Artificial 2008



Universidad Veracruzana

Un trabajito en 1970

- ▶ ¿Cómo programar un sistema como el siguiente?

Sistema

```
> Los gatos matan ratones. Tom es un gato al que no le
  gustan los ratones que comen queso. Jerry es un ratón
  que come queso. Max no es un gato.
> ¿Qué hace Tom?
A Tom no le gustan los ratones que comen queso. Tom mata
  ratones.
```



Sistema de lenguaje natural de Alain Colmerauer et al. [1].

- ▶ ¿Con razonamientos como el siguiente?

Sistema

```
> ¿Qué come Jerry?
```

```
Queso.
```

```
> ¿Qué come Tom?
```

```
Lo que comen los gatos a los que no les gustan los ratones  
que comen queso.
```



¿Lógica?

- ▶ Observen que este **lenguaje** define símbolos para:
 - ▶ Designar elementos (Tom, Jerry, etc.).
 - ▶ Designar conjuntos (Gatos, Ratones, etc.).
 - ▶ Relaciones binarias (Comer, Matar, Gustar, etc.).
 - ▶ Funciones (The, Subset, True)
- ▶ El razonamiento se hacía mediante **resolución-SL**.
- ▶ Definiendo así una **lógica**.



À la française

- 1965. Alan Robinson [3] formula el **principio de resolución**. Reglas de inferencias menos humanas, pero más eficientes.
- 1970. Alain Colmerauer, Philippe Roussel y Robert Pasero [1] trabajan en traducción automática y **procesamiento de lenguaje natural**.
- 1971. Robert Kowalski [2] define la **resolución-SL**.
- 1974. El lenguaje **Prolog**.



Una versión más universal

- Siglo XIX.** En su segunda mitad, Gottlob Frege introduce la **lógica de primer orden**. Modificada a su forma actual por Giuseppe Peano y Bertrand Russell.
- 1930's.** Kurt Göedel y Jacques Herbrand estudiaron la noción de computabilidad basada en **derivaciones**.
- 1965.** La resolución y **unificación** por Alain Robinson.
- 1974.** La resolución-SL de Kowalski: fin del debate (en ese momento) sobre representaciones **declarativas** y **procedimentales**. Prolog.
- 1983.** La máquina abstracta de Warren (**WAM**) [1].
Compilación independiente al procesador usado.



Resumiendo

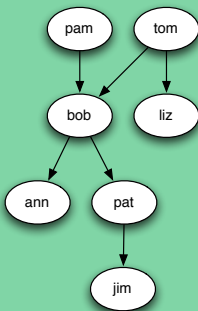
- ▶ La programación lógica es una **herramienta** y un **sujeto de estudio** de la inteligencia artificial.
- ▶ La **lógica de primer orden** es fundamental para entender este paradigma de programación.
- ▶ La programación lógica es un paradigma de programación, que **difiere** de otros paradigmas como la programación imperativa (Algol, C, Pascal, etc.), la orientada a objetos (Simula, Smalltalk, Eiffel, C++, Java, etc.), o la funcional (ML, Haskell, Lisp, etc.).
- ▶ Prolog \neq programación lógica, pero es su realización práctica más usada en la actualidad.



Objetos, Relaciones y Programas

- ▶ Supongamos que queremos razonar sobre la genealogía de una familia [2]:

$X \rightarrow Y$. X es progenitor de Y



Programa prolog

```
1 progenitor(pam,bob).
2 progenitor(tom,bob).
3 progenitor(tom,liz).
4 progenitor(bob,ann).
5 progenitor(bob,pat).
6 progenitor(pat,jim).
```



Cargando Prolog

- ▶ Prolog se ejecuta desde una terminal (usen `p1` en lugar de `swipl`):

Kyrie

```
> swipl
Welcome to SWI-Prolog (Multi-threaded, Version
      5.6.64)
Copyright (c) 1990-2009 University of Amsterdam.
Please visit http://www.swi-prolog.org for details
.

For help, use ?- help(Topic). or ?- apropos(Word).
?-
```



REPL

- El sistema está en un ciclo **Read-Eval-Print** (REPL).

Consola Prolog

```
?- [prog01].
% prog01 compiled 0 sec, 168 bytes
Yes
?- progenitor(bob,pat).
Yes
?- progenitor(liz,pat).
No
?- progenitor(tom,ben).
No
?-
```

Editor

```
1 progenitor(pam,bob).
2 progenitor(tom,bob).
3 progenitor(tom,liz).
4 progenitor(bob,ann).
5 progenitor(bob,pat).
6 progenitor(pat,jim).
```



Variables

- ▶ Se pueden hacer preguntas más interesantes usando variables:

Interactuando con Prolog

```
1  ?- progenitor(X,liz).
2  X = tom
3  Yes
4  ?- progenitor(bob,X).
5  X = ann ;
6  X = pat ;
7  No
```

progenitor/2

```
1  progenitor(pam,bob).
2  progenitor(tom,bob).
3  progenitor(tom,liz).
4  progenitor(bob,ann).
5  progenitor(bob,pat).
6  progenitor(pat,jim).
```



¿Quién es el abuelo de Jim?

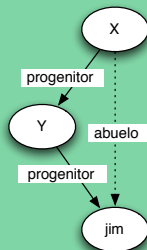
Interactuando con Prolog

```

1  ?- progenitor(Y,jim), progenitor(X,Y).
2  Y = pat
3  X = bob
4  Yes
5  ?- progenitor(X,Y), progenitor(Y,jim).
6  X = bob
7  Y = pat
8  Yes
9  ?- progenitor(tom,X), progenitor(X,Y).
10 X = bob
11 Y = ann ;
12 X = bob
13 Y = pat ;
14 No

```

abuelo/2



Resumiendo

- ▶ Es sencillo definir en Prolog una **relación** especificando las n -tuplas de objetos que la satisfacen. n es conocido como **aridad**.
- ▶ Un programa Prolog consiste de **cláusulas**.
- ▶ Los argumentos de una relación pueden ser: objetos concretos o **constantes** como `tom` y `ann`; objetos generales o **variables** como `X` e `Y`.
- ▶ Las preguntas planteadas a Prolog consisten en una o más metas. Una secuencia de metas significa **conjunción**.
- ▶ La respuesta a una pregunta puede ser positiva o negativa, dependiendo de si la meta se puede **satisfacer** o no.
- ▶ Si varias respuestas satisfacen una pregunta, Prolog encontrará **tantas** como el usuario quiera.



Reglas

- ▶ Las reglas tienen dos partes:
 - ▶ Una parte condicional (el lado derecho de la regla o **cuerpo** de la regla).
 - ▶ Una conclusión (el lado izquierdo de la regla o **cabeza** de la regla).

vastago/2

```
1 vastago(Y,X) :- progenitor(X,Y).
```

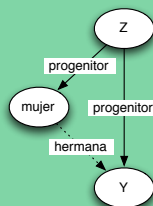


Extendiendo nuestro conocimiento

Nuevos hechos y reglas

```
1  mujer(pam).
2  mujer(liz).
3  mujer(pat).
4  mujer(ann).
5  hombre(tom).
6  hombre(bob).
7  hombre(jim).
8  hermana(X,Y) :- progenitor(Z,X),
9                  progenitor(Z,Y),
10                 mujer(X).
11  abuela(X,Y) :- progenitor(X,Z),
12                  progenitor(Z,Y),
13                 mujer(X).
14  madre(X,Y) :- progenitor(X,Y),
15                 mujer(X).
```

hermana/2



Probando el nuevo conocimiento

- ▶ La relación *hermana/2* presenta una anomalía:

Interactuando con Prolog

```
1 ?- hermana(ann,pat).
2 Yes
3 ?- hermana(X,pat).
4 X = ann ;
5 X = pat ;
6 No
```

Nueva definición

```
1 hermana(X,Y) :-
2     progenitor(Z,X),
3     progenitor(Z,Y),
4     mujer(X),
5     dif(X,Y).
```



Resumiendo

- ▶ Los programas Prolog pueden **extenderse** fácilmente agregando nuevas cláusulas.
- ▶ Las cláusulas en Prolog son de tres tipos: **hechos**, **reglas** y **metas**.
- ▶ Los hechos declaran cosas que son verdaderas siempre, **incondicionalmente**.
- ▶ Las reglas declaran cosas que son verdaderas dependiendo de ciertas **condiciones**.
- ▶ Por medio de las preguntas el usuario puede **computar** qué cosas son verdaderas.



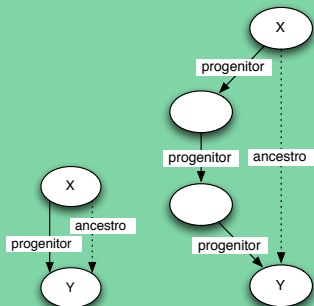
Resumiendo

- ▶ Las cláusulas de Prolog tienen **cabeza** y **cuerpo**. El cuerpo es una lista de metas separadas por comas. Las comas implican **conjunción**.
- ▶ Los hechos son cláusulas con el **cuerpo vacío**; las preguntas tienen la **cabeza vacía**; y las reglas tienen **cabeza y cuerpo**.
- ▶ En el curso de una computación, las variables pueden ser **substituidas** por otros objetos.
- ▶ Las variables se asumen **cuantificadas universalmente**. La cuantificación **existencial** sólo es posible en las variables que aparecen en el cuerpo de una cláusula.



Extensional vs Intensional

ancestro/2



Código

```

1  ancestro(X,Z) :-
2     progenitor(X,Z).
3  ancestro(X,Z) :-
4     progenitor(X,Y),
5     progenitor(Y,Z).
6  ancestro(X,Z) :-
7     progenitor(X,Y0),
8     progenitor(Y0,Y1),
9     progenitor(Y1,Z).
10 ...
  
```



Definición recursiva

- ▶ Ancestro definido en términos de ancestro:

Interactuando con Prolog

```
1  ?- ancestro(pam,X).
2  X = bob ;
3  X = ann ;
4  X = pat ;
5  X = jim ;
6  No
7  ?- ancestro(X,jim).
8  X = pat ;
9  X = pam ;
10 X = tom ;
11 X = bob ;
12 No
```

Definición *ancestro/2*

```
1  ancestro(X,Z) :-
2      progenitor(X,Z).
3
4  ancestro(X,Z) :-
5      progenitor(X,Y),
6      ancestro(Y,Z).
```



Resumiendo

- ▶ Las reglas recursivas definen conceptos en términos de **ellos mismos**.
- ▶ Están definidas por al menos dos casos: uno **terminal** (no recursivo) y la llamada **recursiva**.
- ▶ Una relación recursiva define **intencionalmente** un concepto.
- ▶ **intencional** \neq **intencional**.



Demostración como cómputo

- ▶ Satisfacer una meta implica **demostrar** que la meta es verdadera, asumiendo que las relaciones en el programa lógico son verdaderas.
- ▶ Satisfacer una meta significa entonces demostrar que la meta es una **consecuencia lógica** De los hechos y reglas definidas en un programa.
- ▶ Si la pregunta contiene variables, Prolog necesita también **computar** cuales son los objetos particulares (que remplazaran a las variables) para los cuales la meta se satisface.



Programas lógicos como matemáticas

- ▶ Prolog acepta hechos y reglas como un conjunto de **axiomas**.
- ▶ El usuario plantea preguntas **teoremas**.
- ▶ Prolog trata de probar este teorema, es decir, **demostrar** que el teorema se sigue lógicamente de los axiomas.



Ejemplos

Programa lógico

```
1 falible(X) :- hombre(X).  
2 hombre(socrates).
```

Interactuado con Prolog

```
?- falible(socrates)  
Yes
```

Lógica

Axioma 1 Todos los hombres son falibles.

Axioma 2 Sócrates es un hombre.

Conclusión Sócrates es falible.



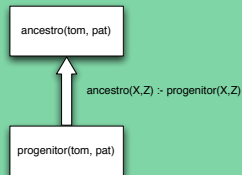
?- ancestro(tom,pat).

- Un proceso en un paso:

Programa

```
1 ancestro(X,Z) :-  
2   progenitor(X,Z).  
3  
4 ancestro(X,Z) :-  
5   progenitor(X,Y),  
6   ancestro(Y,Z).
```

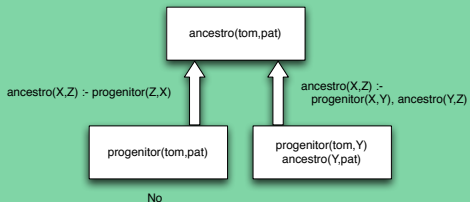
Caso base. Gráfico



?- ancestro(tom,pat).

- Un proceso en dos pasos:

Caso recursivo. Gráfico



Programa

```

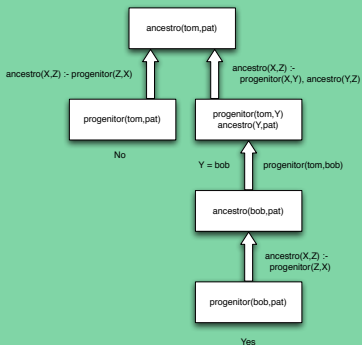
1  ancestro(X,Z) :-
2      progenitor(X,Z).
3
4  ancestro(X,Z) :-
5      progenitor(X,Y),
6      ancestro(Y,Z).
```



?- ancestro(tom,pat).

► Segundo paso:

Caso recursivo. Gráfico



Programa

```

1  ancestro(X,Z) :-
2     progenitor(X,Z) .
3
4  ancestro(X,Z) :-
5     progenitor(X,Y) ,
6     ancestro(Y,Z) .
    
```



Traza de un programa

- Pueden observar lo que el programa hace usando trace:

```
trace
1  ?- trace.
2  Yes
3  [trace] ?- ancestro(tom,pat).
4      Call: (7) ancestro(tom, pat) ? creep
5      Call: (8) progenitor(tom, pat) ? creep
6      Fail: (8) progenitor(tom, pat) ? creep
7      Redo: (7) ancestro(tom, pat) ? creep
8      Call: (8) progenitor(tom, _L345) ? creep
9      Exit: (8) progenitor(tom, bob) ? creep
10     Call: (8) ancestro(bob, pat) ? creep
11     Call: (9) progenitor(bob, pat) ? creep
12     Exit: (9) progenitor(bob, pat) ? creep
13     Exit: (8) ancestro(bob, pat) ? creep
14     Exit: (7) ancestro(tom, pat) ? creep
15  Yes
```



Retomando el temario

1. Fundamentos teóricos
 - 1.1 Lógica de primer orden.
 - 1.2 Cláusula y programa definitivos.
 - 1.3 Principio de resolución.
 - 1.4 Negación.
 - 1.5 Corte y la aritmética
2. Prolog
 - 2.1 Introducción a Prolog.
 - 2.2 Búsquedas en espacios de soluciones (A^*).
 - 2.3 Sistemas expertos (MYCIN).
 - 2.4 Aprendizaje automático (ID3).
 - 2.5 Planeación (STRIPS).
 - 2.6 Temas selectos.



Bibliografía



A. Colmerauer and P. Roussel.

The birth of Prolog.

ACM SIGPLAN Notices, 28(3):1–31, 1993.



R. Kowalski and D. Kuehner.

Linear resolution with selection function.

Artificial Intelligence, 2(3):227–260, 1971.



J. A. Robinson.

A machine-oriented logic based on the resolution principle.

Journal of the ACM, 12(1):23–41, 1965.



Bibliografía



D. H. D. Warren.

An abstract Prolog instruction set.
Technical Report 309, SRI, 1983.



I. Bratko.

Prolog programming for Artificial Intelligence.
Addison-Wesley, 1986.

