

Introducción a LP con SWI/Prolog

CARLOS LORÍA-SÁENZ LORACARLOS@GMAIL.COM

OCTUBRE-NOVIEMBRE 2017

EIF/UNA



Objetivos

2

- ▶ Fundamentos paradigma lógico (LP)
- ▶ Lógica como modelo
- ▶ Inferencia/deducción
- ▶ Cláusulas de Horn
- ▶ Programas Prolog
- ▶ Unificación
- ▶ Resolución y Árbol de Prueba
- ▶ El caso SWI-Prolog

Herramientas

3



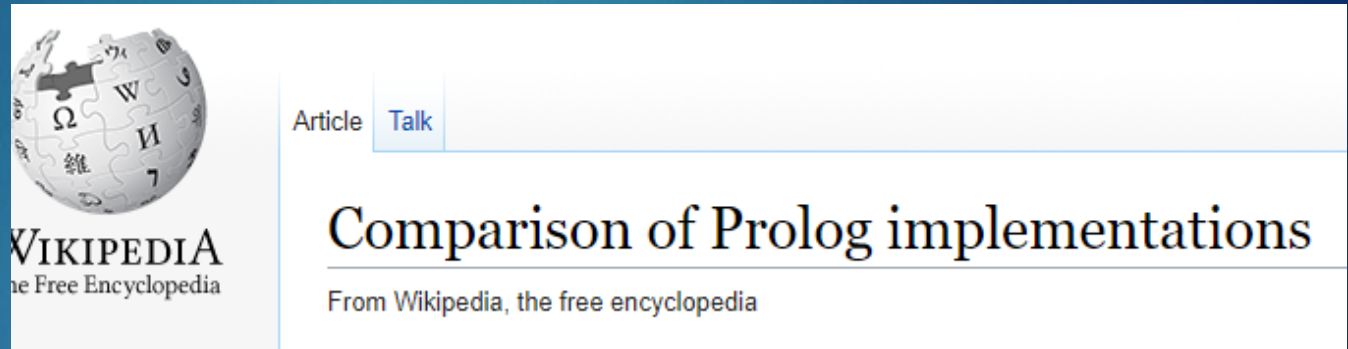
- ▶ SWI-Prolog (motor de Prolog)
- ▶ <http://www.swi-prolog.org/download/stable>
- ▶ Se refiere pdf con [ejercicios](#)

```
C:\> Símbolo del sistema  
  
SWIPL:swipl -v  
SWI-Prolog version 7.6.0-rc2 for x64-win64  
  
SWIPL:
```

Versiones de Prolog

4

► Ver [acá](#)



Platform			Features								Toolkit			Prolog Mechanics
Name	OS	Licence	Native Graphics	Compiled Code	Unicode	Object Oriented	Native OS Control	Stand Alone Executable	C Interface ^[3]	Java Interface ^[3]	Interactive Interpreter	Debugger	Code Profiler	Syntax
SWI-Prolog	Unix, Linux, Windows, Mac OS X	LGPL	Yes	Yes	Yes		Yes	Yes	Yes	Yes	Yes	Yes	Yes	ISO-Prolog, Edinburgh Prolog

Fuente Ejercicios

5

- ▶ Una excelente fuente de ejercicios
- ▶ http://www.cs.us.es/~jalonso/publicaciones/2006-ej_prog_declarativa.pdf

Ejercicios de programación declarativa con
Prolog

José A. Alonso Jiménez

Principios

6

- ▶ Programas =
- ▶ Lógica + Control + Presentación
- ▶ Lógica = Qué
- ▶ Control = Cómo
- ▶ Presentación= interfaz con “clientes”
- ▶ Programación lógica se concentra en el componente de “lógica”
- ▶ El componente de “control” lo mantiene el lenguaje en la forma de backtracking

Lógica como modelo

7

- ▶ Paradigma: representar relaciones entre objetos de dominio en el lenguaje
- ▶ Relaciones se “mezclan” por medio de reglas (combinan relaciones)
- ▶ Un programa es como un iterador que camina por una relación
- ▶ Usualmente un “query”
- ▶ Relaciones pueden ser complejas y recursivas

Contraste con SQL

8

- ▶ La relaciones no son tablas.
- ▶ Existen en memoria. Se pueden salvar como texto
- ▶ El lenguaje de consulta y el de programación son el mismo.
- ▶ Son modelos relacionales tipo NoSQL
- ▶ Nota: SWI-Prolog tiene soporte para JSON

Forma clausal de Horn

- ▶ Es tipo de regla en la que cada cláusula tiene a lo más una literal positiva.
- ▶ Las variables están universalmente cuantificadas
- ▶ $\text{not}(P_1) \mid \text{not}(P_2) \mid \dots \mid \text{not}(P_n) \mid Q$
- ▶ Se escribe
- ▶ $Q \text{ :- } P_1, \dots, P_n$
- ▶ Las comas denotan & (un y lógico).
- ▶ Qué dice una cláusula de Horn:
- ▶ Q se tiene si se tienen todos P_1, \dots, P_n .

Usando SWI-prolog

10

- Usamos versión 7.6.0 windows 64 bits

```
Simbolo del sistema

SWIPL:swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.0-rc2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

1 ?- writeln('Hola Mundo!').
Hola Mundo!
true.

2 ?- halt.

SWIPL:
```



Salimos con
halt.

Consultando un archivo

11

- Se dice «consultar» un script

```

❏ Símbolo del sistema - swipl

SWIPL:
SWIPL:swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.0-rc2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

1 ?- ['./work/holamundo'].
true.

2 ?- holaMundo.
Hola Mundo!
true.

3 ?-

% Hola Mundo
/*
@author loriacarlos@gmail.com
@since EIF400-2017
*/

holaMundo :- write('Hola Mundo!'), nl.
```

Ejecutando un pl como script

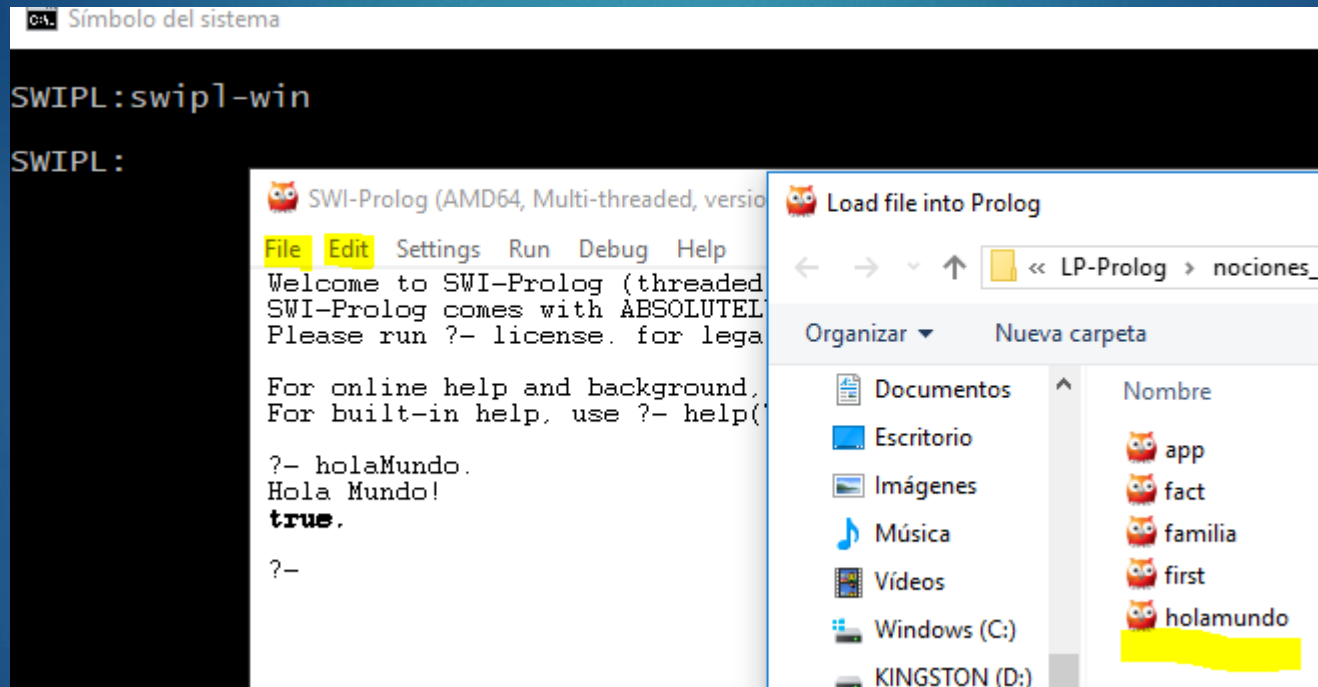
12

```
SWIPL:  
SWIPL:swipl -q -g halt -s work\fact.pl  
Fact(7)=5040  
SWIPL:
```

Otra opción: swipl-win

13

- Versión más amigable



Prolog como lenguaje

14

- ▶ Dinámico
- ▶ No hay análisis de tipos
- ▶ Compilado a una máquina virtual llamada WAM (Warren Abstract Machine)
- ▶ SWI-Prolog está en C
- ▶ Interfaz con Java

Programa Prolog

15

- ▶ Un programa es un conjunto de cláusulas
- ▶ Una cláusula (regla) es de la forma
 - ▶ $p \text{ :- } q_1, \dots, q_n.$
- ▶ Si n es cero se llama un “fact”.
- ▶ Si p no viene es un “goal”.
- ▶ ¡No olvidar el punto al final!

Goals y Directivas

16

- Las cláusulas sin “cabeza” se entienden como directivas o evaluaciones

```
fact(0, 1) :- !.  
fact(N, F) :- N1 is N - 1,  
              fact(N1, F1),  
              F is N * F1.  
%Goal  
:- fact(7, F), format("fact(~d)=~d", [7, F]).
```

```
:- dynamic persona/3.
```

Declara persona
como dinámico de
3 parámetros

Ejemplo (en work)

17

```
% fact(N+, F-) F (de salida) es el factorial de N (de entrada)

fact(0, 1). % FACT: 1 es el factorial de 0
fact(N, F) :- N > 0,                % N es mayor que cero
              N1 is N - 1,          % y N1 es N menos 1
              fact(N1, F1),         % y F1 es el factorial de N1
              F is N * F1.          % F es N veces F1
%Goal
:- fact(7, F), format("fact(~d)=~d", [7, F]).
```

```
SWIPL:swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.0-rc2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

1 ?- ['./work/fact'].
fact(7)=5040
true.

2 ?-
```

Notación

18

- ▶ + parámetro de entrada
- ▶ – parámetro de salida
- ▶ ? parámetro de entrada o salida
- ▶ : de entrada usualmente siendo un `goal`

append(?List1, ?List2, ?List1AndList2)

List1AndList2 is the concatenation of *List1* and *List2*

Availability: *built-in*

is_list(+Term)

True if *Term* is bound to the empty list (`[]`) or a term with functor `'[]'`¹¹² and arity 2 and the second argument is a list.¹¹³ This predicate acts as if defined by the definition below on *acyclic* terms. The implementation *fails* safely if *Term* represents a cyclic list.

maplist(:Goal, ?List1, ?List2)

As [maplist/2](#), operating on pairs of elements from two lists.

Ejercicio

19

- Implemente `fact` recursiva de cola

Ejemplo: familia

- ▶ Revise el archivo familia.pl
- ▶ Átomo: empieza con minúscula o entre '...'
- ▶ Variable: empieza en mayúscula
- ▶ `[a, b, 1, [c,d]]` es una lista

```
PL:swipl
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 7.2.3)
Copyright (c) 1990-2015 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

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

1 ?- [familia].
true.

2 ?- padre(X, Y).
X = catalina,
Y = beto ;
X = catalina,
Y = chico ;
X = catalina,
Y = raquel ;
X = pedro,
Y = beto ;
X = pedro,
Y = chico ;
X = pedro,
Y = raquel
```

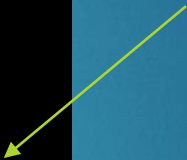
Use ; para next
Use a (enter)
para abortar

Spy, debug, nospy, nodebug, trace, notrace

```
4 ?- spy(papa)
.
% Spy point on papa/2
true.

[debug] 5 ?- papa(X,Y).
* Call: (7) papa(_G2730, _G2731) ? creep
  Call: (8) hombre(_G2730) ? creep
  Exit: (8) hombre(beto) ? creep
  Call: (8) padre(beto, _G2731) ? creep
  Exit: (8) padre(beto, ana) ? creep
* Exit: (7) papa(beto, ana) ? creep
X = beto,
Y = ana ;
  Redo: (8) padre(beto, _G2731) ? creep
  Exit: (8) padre(beto, juan) ? creep
* Exit: (7) papa(beto, juan) ? creep
X = beto,
Y = juan
```

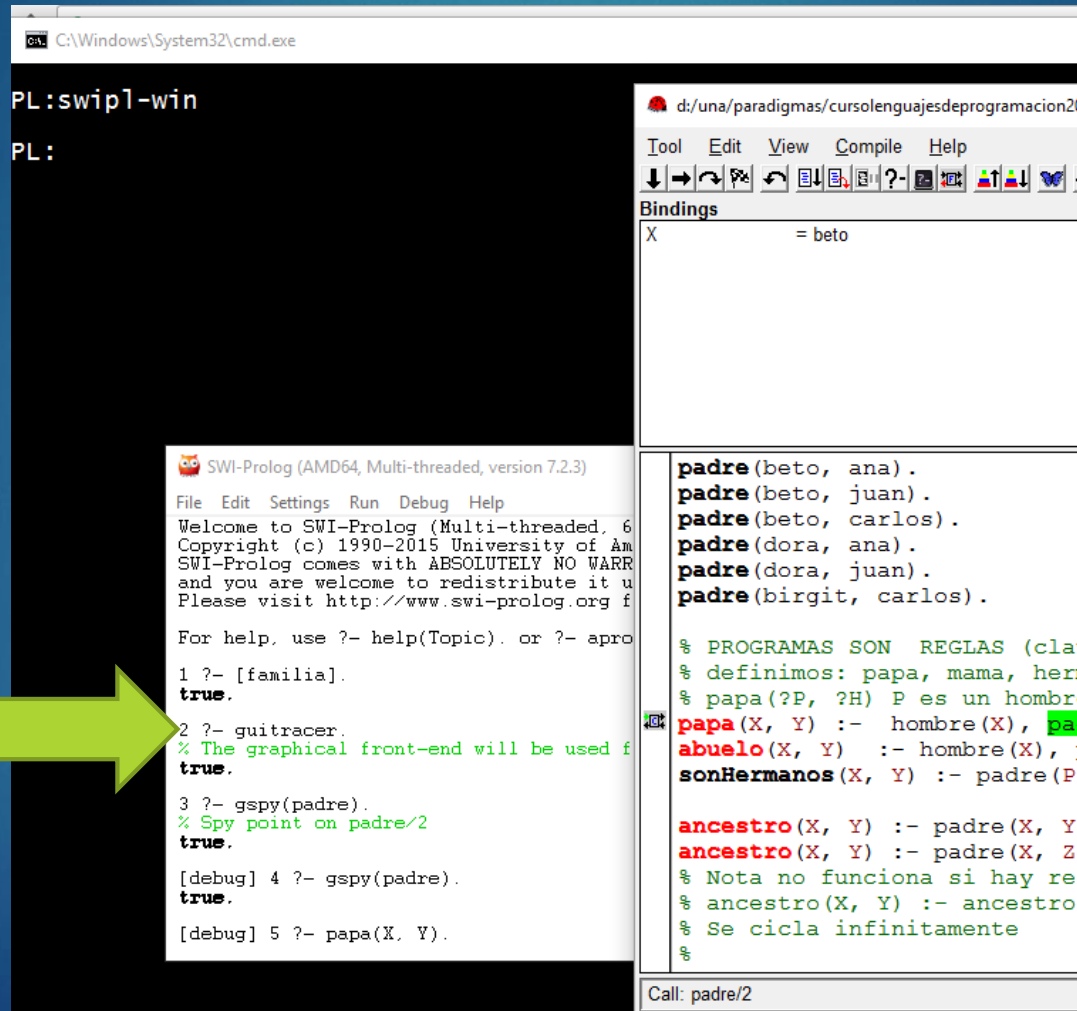
c: creep
a: abort



```
[trace] 6 ?- nodebug.
true.
```

Versión gráfica: solo con swipl-win

22



The screenshot shows the SWI-Prolog graphical front-end (swipl-win) running in a Windows environment. The main window is titled "C:\Windows\System32\cmd.exe" and displays the command prompt. The user has entered "PL:swipl-win" and "PL:". The graphical editor window is titled "d:/una/paradigmas/cursolenguajesdeprogramacion20" and shows the Prolog code. A green arrow points from the command prompt to the graphical editor.

```
PL:swipl-win
PL:
```

SWI-Prolog (AMD64, Multi-threaded, version 7.2.3)

File Edit Settings Run Debug Help

Welcome to SWI-Prolog (Multi-threaded, 6
Copyright (c) 1990-2015 University of Am
SWI-Prolog comes with ABSOLUTELY NO WARR
and you are welcome to redistribute it u
Please visit <http://www.swi-prolog.org> f

For help, use ?- help(Topic). or ?- apro

```
1 ?- [familia].
true.
2 ?- guitracar.
% The graphical front-end will be used f
true.
3 ?- gspy(padre).
% Spy point on padre/2
true.
[debug] 4 ?- gspy(padre).
true.
[debug] 5 ?- papa(X, Y).
```

Bindings

X	= beto
---	--------

```
padre(beto, ana).
padre(beto, juan).
padre(beto, carlos).
padre(dora, ana).
padre(dora, juan).
padre(birgit, carlos).

% PROGRAMAS SON REGLAS (clau
% definimos: papa, mama, herm
% papa(?P, ?H) P es un hombre
papa(X, Y) :- hombre(X), pac
abuelo(X, Y) :- hombre(X), p
sonHermanos(X, Y) :- padre(P,

ancestro(X, Y) :- padre(X, Y)
ancestro(X, Y) :- padre(X, Z)
% Nota no funciona si hay rec
% ancestro(X, Y) :- ancestro
% Se cicla infinitamente
%
```

Call: padre/2

Ejemplos

23

- ▶ Regla de papá: es un hombre que es padre
- ▶ Regla Proposicional (sin variables): un salado si está casado y vive con la suegra

```
papa(X, Y) :- hombre(X), padre(X, Y).  
salado :- casado, viveConSuegra.
```

Tipos de Objetos

24

- ▶ Hay átomos: `'juan sin miedo'`, `'csh'`
- ▶ Pueden ser números
- ▶ Se pueden omitir las comillas si no hay caracteres blancos o especiales
- ▶ Hay variables: `X`, `Y` (primera letra en mayúscula)
- ▶ Hay términos: `campeon('Costa Rica', csh)`
- ▶ `campeon` se llama un functor.
- ▶ Hay funtores con significado especial `, is, =, \+, \=, =:=, =\=...`, Se llaman predicados
- ▶ Hay funciones aritméticas: `+`, `-`, `*`, `/`, `mod`,...

Términos evaluados y árboles (AST)

25

- ▶ Los términos son ASTs. No se evalúan por defecto
- ▶ Para evaluar los aritméticos use `is` (no `=`)

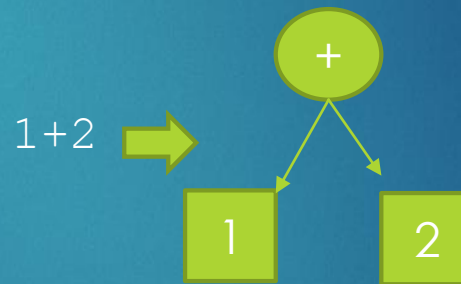
```
SWIPL:swipl
Welcome to SWI-Prolog
SWI-Prolog comes with
Please run ?- license

For online help and
For built-in help,

1 ?- X = 1 + 2.
X = 1+2.

2 ?- X is 1 + 2.
X = 3.

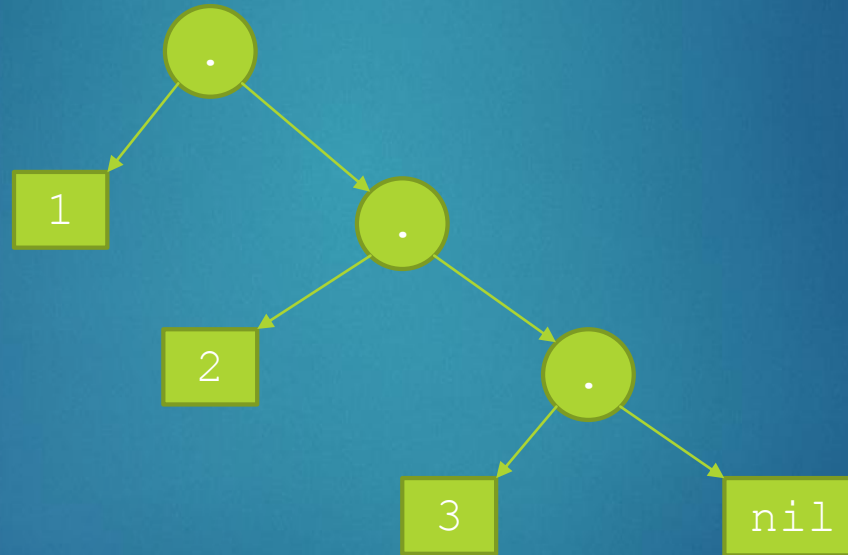
3 ?-
```



Términos especiales: listas

26

- ▶ $[1, 2, 3]$ es una lista
- ▶ $[]$ lista vacía (nil)



Ejercicio

27

- ▶ Dibuje el término de $[[1], 2, [3, 4]]$

Modelo de “control”

28

- Recursión
- Backtracking

```
SWIPL:swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.0-rc2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

1 ?- member(2, [1,2,3]).
true .

2 ?- member(X, [1,2,3]).
X = 1 ;
X = 2 ;
X = 3.

3 ?-
```

Backtracking



- Deducir a partir de las reglas y los hechos

```
mortal(X) :- humano(X). % clausula (regla)
humano(socrates).      % fact

% Inferencia mortal(socrates)
```

Inferencia en cláusulas de Horn

30

- ▶ Siempre es posible formular LP usando cláusulas de Horn
- ▶ Permiten inferencia por resolución
- ▶ Es una forma eficiente de controlar la búsqueda
- ▶ Ejemplo proposicional (sin variables)
 - ▶ $p :- q, r \quad = p \mid \text{not}(q) \mid \text{not}(r)$
 - ▶ $r :- s \quad = r \mid \text{not}(s)$
 - ▶ $q :- s \quad = q \mid \text{not}(s)$
 - ▶ $s. \quad = s$

Resolución (proposiciones)

- ▶ Dadas dos cláusulas C_1 y C_2
- ▶ Si C_1 tiene p y C_2 tiene $\text{not}(p)$ entonces
- ▶ Forme una nueva cláusula uniendo C_1 con C_2 pero borrando ambos : p y $\text{not}(p)$
- ▶ Ejemplo
- ▶ $C_1 = p \mid \text{not}(q) \mid \text{not}(r)$
- ▶ $C_2 = \underline{q} \mid \text{not}(s)$
- ▶ Resolución = $p \mid \text{not}(s) \mid \text{not}(r)$

Resolución: inferencia

32

Premisas

- | | | |
|----|---|----------|
| 1. | $p \mid \text{not}(q) \mid \text{not}(r)$ | |
| 2. | $r \mid \text{not}(s)$ | |
| 3. | $q \mid \text{not}(s)$ | |
| 4. | s | |
| 5. | $p \mid \text{not}(s) \mid \text{not}(r)$ | por 1,3 |
| 6. | $p \mid \text{not}(r)$ | por 5, 4 |
| 7. | $p \mid \text{not}(s)$ | por 6, 2 |
| 8. | p | por 7, 4 |

Ejercicio

33

- ▶ Reescriba el ejemplo anterior en Prolog y pruebe

Resolución: búsqueda

Ejemplo
familia.pl

padre(beto, Y)

fail

Y=ana

Y=juan

```
% padre(?P, ?H): P es el padre/madre de H
padre(catalina, beto).
padre(catalina, chico).
padre(catalina, raquel).
padre(pedro, beto).
padre(pedro, chico).
padre(pedro, raquel).
padre(beto, ana).
padre(beto, juan).
padre(beto, carlos).
padre(dora, ana).
padre(dora, juan).
padre(birgit, carlos).
```

- Búsqueda “dfs”
- Se puede ver como un “árbol de prueba” o de búsqueda

Resolución: Dibuje el árbol de prueba

padre(beto, Y), mujer(Y)

Ejercicio

36

resolution.pl

```
p(X, f(b)) :- r(X). %1
q(h(_)).          %2
q(b).             %3
r(a).             %4
r(b).             %5
```

```
1 ?- [resolution].
true.

2 ?- p(X, f(Y)), q(Y).
X = a,
Y = b ;
X = Y, Y = b.

3 ?-
```

Usando un árbol de prueba explique el resultado observado

Control del backtracking

37

- ▶ `fail` predicado que siempre falla y provoca backtracking si hay alguno
- ▶ `!` (se lee `cut`) sólo se cumple una vez y no deja devolverse al backtracking

```
fact(0, 1) :- !.  
fact(N, F) :- N1 is N - 1,  
              fact(N1, F1),  
              F is N * F1.  
%Goal  
:- fact(7, F), format("fact(~d)=~d", [7, F]).
```

Resolución con variables

38

- ▶ $C_1 = p(X) \text{ :- } q(X, f(X)), r(X).$
- ▶ $C_2 = q(1, Y)$
- ▶ Acá $q(X, f(X))$ no es igual a $q(1, Y)$.
- ▶ No hay resolución exacta
- ▶ Pero si hacemos $X=1, Y= f(1)$ se vuelven iguales
- ▶ Ese proceso se llama unificación
- ▶ Es buscar un cambio (sustitución) sobre las variables que haga los términos iguales

Unificación Ejemplos

39

- ▶ Ejemplo: $f(a, g(X, b)) = f(Y, g(c, b))$
- ▶ **Regla funtores:**
- ▶ $\{f(s_1, \dots, s_m) = g(t_1, \dots, t_n)\} \cup S, B \rightarrow \{s_1 = t_1, \dots, s_n = t_n\} \cup S, B$ si $f = g$ y $n = m$.
FAIL otro caso
- ▶ **Regla Asociar** $\{X = t\} \cup S, B \rightarrow S, B \cup (X \rightarrow t)$ si X no atada en B y X no ocurre en t .
- ▶ Si X atada y $B(X) = t \rightarrow S, B$. Otro caso
Fail
- ▶ **Regla Éxito:** $\{\}, B \rightarrow B$

Ejercicios

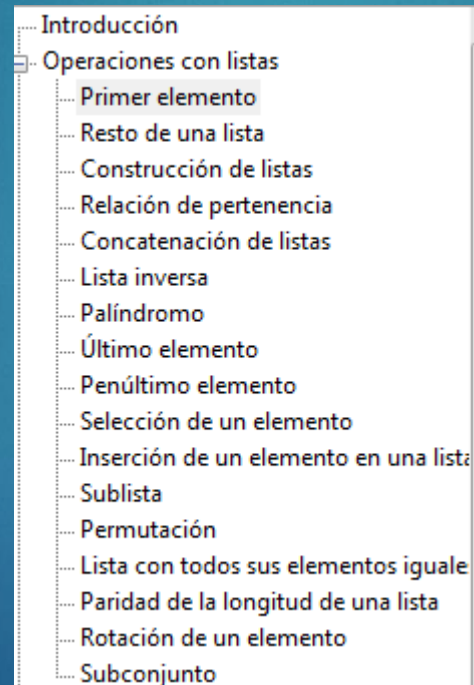
40

- ▶ Prediga la unificación de los siguientes términos y verifique con el REPL de (SWI)-Prolog
- ▶ $f(a, g(X, b)) = f(Y, g(c, b))$
- ▶ $[X, Y|[3,4]] = [1,2|R]$
- ▶ $X = f(a, X)$

Ejercicios

41

- Teclee en SWI-Prolog y estudie los ejercicios del libro de José Jiménez, la parte de listas



A screenshot of a table of contents for a book. The items are listed with dotted lines to the left of each entry. The 'Operaciones con listas' section is expanded, showing a list of sub-topics. The 'Primer elemento' item is highlighted with a light blue background.

.....	Introducción
.....	Operaciones con listas
.....	Primer elemento
.....	Resto de una lista
.....	Construcción de listas
.....	Relación de pertenencia
.....	Concatenación de listas
.....	Lista inversa
.....	Palíndromo
.....	Último elemento
.....	Penúltimo elemento
.....	Selección de un elemento
.....	Inserción de un elemento en una lista
.....	Sublista
.....	Permutación
.....	Lista con todos sus elementos iguales
.....	Paridad de la longitud de una lista
.....	Rotación de un elemento
.....	Subconjunto

Assert/Retract(all)

42

- ▶ Sirven para añadir y remover términos dinámicamente
- ▶ `assert`: agrega al final
- ▶ `assertz` lo mismo
- ▶ `asserta` inserta de primero
- ▶ `retract` borra según el término (puede borrar varios en backtracking). Puede dar `fail`.
- ▶ `retractall` borrar sin forzar backtracking. Nunca da `fail`.

Ejemplo

43

```
SWIPL:swipl
Welcome to SWI-Prolog (t
SWI-Prolog comes with AB
Please run ?- license. f

For online help and back
For built-in help, use ?

1 ?- assert(f(1)).
true.

2 ?- assert(f(2)).
true.

3 ?- assert(f(3)).
true.

4 ?- f(X).
X = 1 ;
X = 2 ;
X = 3.

5 ?- asserta(f(3)).
true.

6 ?- f(X).
X = 3 ;
X = 1 ;
X = 2 ;
X = 3.

7 ?-
```

```
7 ?- retract(f(1)).
true.

8 ?- f(X).
X = 3 ;
X = 2 ;
X = 3.

9 ?- retractall(f(3)).
true.

10 ?- f(X).
X = 2.

11 ?-
```

Predicados generadores

44

- ▶ Sirven para “generar” o “agrupar” valores en listas o rangos
- ▶ Ejemplos
- ▶ `member`
- ▶ `numlist`
- ▶ Otra opción son los metapredicados

Metapredicados

45

- ▶ Reciben variables que son predicados
- ▶ Predicados “Higher-order” como en FP
- ▶ Ejemplos (hay más)
- ▶ `\+ (not)`
- ▶ `call`
- ▶ `;` (or) `y ->;` (if-then-else)
- ▶ `findall`
- ▶ `forall`
- ▶ `bagof, setof`
- ▶ `maplist`
- ▶ `foldl (reduce)`

Ejercicio

46

- ▶ Investigue sobre metapredicados que no se explican acá

Ejemplos

47

```
SWIPL:swipl  
Welcome to SWI-Prolog (threaded, 64  
SWI-Prolog comes with ABSOLUTELY NO  
Please run ?- license. for legal de  
  
For online help and background, vis  
For built-in help, use ?- help(Topi  
  
1 ?- ['work/max'].  
true.  
  
2 ?- max(10, 20, M).  
M = 20.  
  
3 ?-
```

```
max(X, Y, M) :- X>Y -> M = X ; M = Y.
```

Ejemplos

48

```
SWIPL:swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 7
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Topic).

1 ?- numlist(1, 3, N), member(X, N), S is X*X.
N = [1, 2, 3],
X = 1, S = 1;
N = [1, 2, 3],
X = 2,
S = 4;
N = [1, 2, 3],
X = 3,
S = 9.

2 ?-
```

Ejemplos

49

```
SWIPL:swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.6.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Subject).

1 ?- member(X, [1,2,3,4]), assert(f(X)), fail.
false.

2 ?- findall(X, f(X), A).
A = [1, 2, 3, 4].

3 ?- forall(f(X), X > 0).
true.

4 ?- forall(f(X), X > 2).
false.

5 ?- forall(f(X), \+ X > 2).
false.

6 ?- forall(f(X), \+ X > 10).
true.

7 ?-
```

Ejemplo (nein)

50

```
SWIPL:swipl
Welcome to SWI-Prolog (t
SWI-Prolog comes with AB
Please run ?- license. f

For online help and back
For built-in help, use ?

1 ?- ['work/not'].
true.

2 ?- nein(true).
false.

3 ?- nein(false).
true.

4 ?- nein(fail).
true.

5 ?-
```

```
nein(P) :- call(P), !, fail.
nein(_).
```

Ejercicio

51

a) Indique de manera clara y concisa qué retorna el siguiente predicado `goo` en su variable `R`. Asuma que `L` es una lista de entrada.

```
:- dynamic foo/1.  
% L es solo de entrada y R de entrada/salida  
goo(L, R) :- retractall(foo(_)),  
             forall((member(X, L),  
                    \+ foo(X)),  
                   assert(foo(X))),  
             findall(X, retract(foo(X)), R).
```

b) Reescriba `goo` en forma recursiva.