Prolog

Contenido

- Introducción
- Hello word
- La familia Buendía
- Recursión
- Listas

¿Qué es Prolog?

- Lenguaje de programación
  - Declarativo
  - Programación lógica
- Está compuesto de:
  - Hechos
  - Reglas

Declarativo vs Imperativo

- Se describe qué hacer en lugar de cómo
  - abuelo(A,N) si padre(A,P) y padre(P,N)

Declarativo vs Imperativo

```
- def abuelo(A,N):
    For P in A.hijos():
      For N' in P.hijos():
       If N' is N:
          return true
    return false
```


Programación lógica

- Lógica de primer orden con Horn-clauses
  - C_1 & C_2 …$C_N \rightarrow$ G con N=0...
  - Con N=0 es un hecho
  - Con N>0 es una regla
- Back chaining para demostrar que G es satisfacible
- En particular prolog usa: Selective Linear Definite clause resolution (SLD).

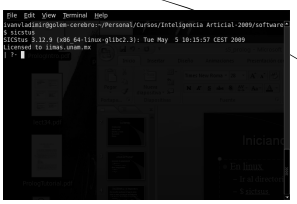
Muchas implementaciones

- Usaremos Sicstus Prolog
  - Software propietario
  - OAA lo soporta
- Versiones Open Source
  - SWI-prolog
  - Gnuprolog
  - Etc…

Iniciando Sicstus prolog

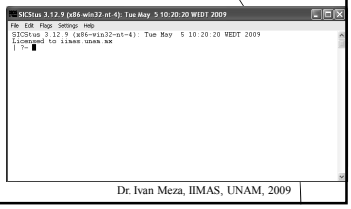
- En linux
  - En una terminal
  - Ir al directorio de trabajo e invocar:
  - $ sictsus
- En windows
  - Inicio -> Todos los programs -> Sicstus
- Se despliega información general y aparece el cursor

```
| ?-
```

Linux

windows



Finalizando Sicstus

- Linux
  - Teclear Ctrl+d
- Windows
  - File -> Halt

Hello word (Imperativo)

- Como una ‘query’
  - `print(‘Hello word’).`
- Poner atención a:
  - Comillas sencillas para la ‘cadena’/átomo.
  - Punto al final.
  - Mayúsculas y minúsculas son importantes.

Hello word (cont. 2)

- Creando un ‘programa’
  - `[user].`
  - `hello_word:-`
    `print(‘Hello word’).`
  - `Ctrl+d`
- Ahora teclear
  - `hello_word.`

Hello word (cont 3.)

- Guardar nuestro programa en un archivo
  - Crear archivo hw1.pl con:
  - **hello_word:-**
    print('Hello word').
  - Teclear dentro de prolog:
  - **consult(hw1).**
  - **hello_world.**
  - Intentar las variantes:
  - **consult('hw1.pl').**
  - **consult(hw1.pl).**

Hello world (cont. 4)

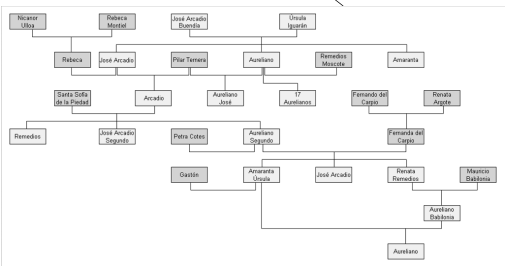
- Crear hw2.pl con:
  - **hello_word:-**

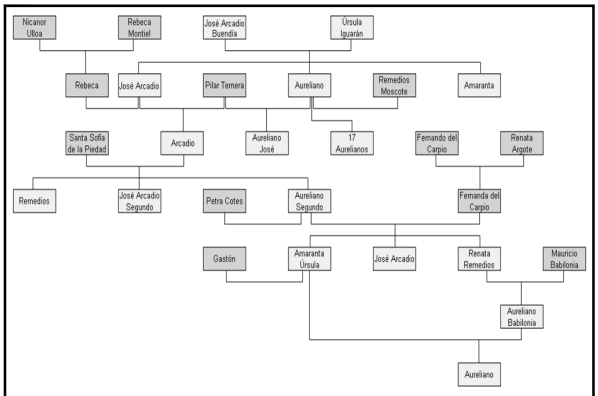
```
print(`¿Cúal es tu nombre?),
read(Name),
print(`Hola `),
print(Name),
nl,
print(`Hello word').
```


Sintaxis

- Términos
  - Átomos: Constantes: hello, 'hello world'
  - Variables (en mayúsculas): X, Name
  - Predicados: átomos + (+Args+), print/2, hello_word/0.
  - Listas: [+terminos+]
- Reglas
  - Predicado :- Termino,…,Termino

Los Buendía





Ahora sí un programa declarativo

- Relación abuelo (programa/reglas)
  - **abuelo(A,N):-**
    padre(A,P),
    padre(P,N).
- Más hechos
  - buendia.pl
- Preguntamos:
  - abuelo('Jose Arcadio Buendia','Arcadio').
  - abuelo('Arcadio',Remedios').

Disyunción

- ¿Qué pasa con?
  - `abuelo('Fernando…','Jose Arcadio')`
- ¿El padre de la madre?
  - `abuelo(A,N):-`
    `padre(A,M),`
    `madre(M,N).`

Más diversión

- ¿De quien es padre Aureliano?
  - `padre('Aureliano',X)`
- ¿Quién es la hijo de Pilar Ternera?
  - `madre('Pilar Ternera',X)`
- ¿Quiénes tienen la relación de abuelo?
  - `Abuelo(X,Y)`

'Queries'

- Una vez que se envía una query a Prolog
  - Prolog busca encontrar una asignación de las variables que la satisfaga
  - Una vez encontrada prolog pregunta sí esa asignación es adecuada
    - Con ENTER o 'yes' prolog termina de buscar y determina que el query es satisfecho
    - Con 'no' busca por otra asignación.

Recapitulando

- Entrar y salir de prolog
- Consultar programas
- Hechos y reglas
- "Queries"
- Conjunción y disyunción

Comandos

- Basta de leer buendia.pl cada vez que cargamos un programa.
  - Agregar el comando al archivo
  - `:- consult(buendia).`

Ancestros

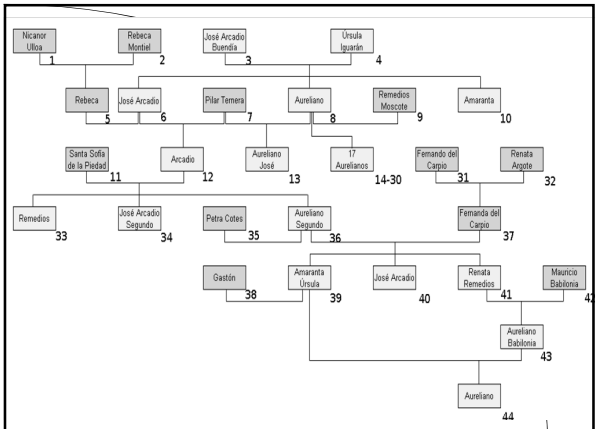
- Casos bases:
  - Un padre y una madre son ancestros de sus hijos
- Recursión:
  - Alguien es mi ancestro si es padre o madre de un ancestro mío

Reglas para ancestro (padre)

- `ancestro(A,D) :-`
  `padre(A,D).`
- `ancestro(A,D):-`
  `padre(A,SomeOne),`
  `ancestro(SomeOne,D).`

Un pequeño problema

- `ancestro('Jose Arcadio','Aureliano').`
- Ciclos
  - Prolog usa Depth First Search
  - Si encuentra un ciclo no puede salir de él.



Operadores

- Prolog nos permite definir operadores propios
- Op(Prioridad,Formato,Nombre).
  - op(700,xfx,>).
  - op(500,yfx,+).
  - op(400,yfx,*).
- Mayor prioridad menor preferencia

Listas

- Un tipo usado frecuentemente en Prolog son las listas
  - [‘JosE Arcadio’,’Aureliano’,’Renata Remedios’]
- La lista se divide en dos partes:
  - [Primer elemento| Resto]
  - [H|T]

Redefiniendo el árbol genealógico de los Buendía

- Dos nuevos predicados y dos operadores
  - Operadores para género
    - Masculino mas
    - Femenino fem
  - Predicado info/2
    - Primer argumento es un identificador único
    - Segundo argumento es un nombre-genero
  - Predicado hijos
    - Primer argumento es un identificador
    - Segundo argumento es una lista de identificadores

El archivo de info.pl

- Contiene los hechos
  - `info(1, mas 'Nicanor Ulloa').`
  - `info(2, fem 'Rebeca Montiel').`
  - `info(3, mas 'José Arcadio Buendía').`
- Pero además se ahorra el caso los 17 Aurelianos
  - ```
    info(ID, mas 'Aureliano'):-
        ID > 13,
        ID < 31.
    ```


Argumentos obligatorios

- Problema si preguntamos
  - info(X, mas 'Aureliano').
  - X es un argumento necesario
- En la documentación de sicstus se marcan los argumentos con
  - ? Argumento puede o no ser una variable
  - + Argumento no puede ser variable
  - - Argumento necesita ser variable

Archivos hijos.pl

- ```
  hijos(1,[5]).
  hijos(2,[5]).
  hijos(3,[6,8,10]).
  hijos(4,[6,8,10]).
  …
  ```


Hijo de alguien

- ```
  hijo(H,P):-
      info(H, mas Nombre),
      hijos(P,Hs),
      en_hijos(H,Hs).
  ```
- Tres casos casos:
  - H es el primer elemento de la lista
  - H está en el resto de la lista
  - H no está en la lista

Hijo de alguien (cont.)

- `en_hijos(H,[]):- fail.`
- `en_hijos(H,[H|R]).`
- ```
  en_hijos(H,[_|R]):-
      en_hijos(H,R).
  ```


Pattern matching

- Utilizamos unificación para hacer pattern matching
  - Listas hacen matching con [Head|Tail]
  - [] representa la lista vacía
  - _ representa cualquier valor

Fallo

- Se utiliza el átomo 'fail' para hacer un predicado falso.

Abrir familia.pl

- Intentar los predicados padres, abuelos, ancestros.
- Intentar el predicado hermano.
- Intentar el predicado hermanos.

Primogénito

```
primer_hijo(P,H):-
    info(H, mas N),
    hijos(P,Hs),
    en_hijos(H,Hs),!.
```

- El operador "!" para la búsqueda, se le denomina 'cut'.

Accumuladores

- En hermanos utilizamos un acumulador
- Es decir, un resultado parcial del cómputo
- add_hermano/4 usa esta técnica
  - De quien buscamos el hermanos
  - Lista de posibles hermanos
  - Acumulador
  - Resultado

Add_hermano

- Busca construir una lista de hermano únicos
  - Comenzamos con una lista vacía de hermanos
  - Se recorre la lista de posibles hermanos
  - Uno no puede ser su propio hermano
  - Sí hermano ya en la lista no agregarlo
  - Se pasa la lista parcial de hermanos único (Acc) al siguiente predicado
  - Un argumento es el resultado final, que se define hasta que ya no haya más posibles hermanos

Accumuladores

```
- add_hermanos(E,[],Acc,Acc).
- add_hermanos(E,[H|Rest],Acc,Res):-
    E \== H,
    \+ en_hermanos(H,Acc),
    add_hermanos(E,Rest,[H|Acc],Res),!.
- add_hermanos(E,[H|Rest],Acc,Res):-
    add_hermanos(E,Rest,Acc,Res),!.
```

Conceptos claves

- Hechos son una base de conocimiento/base de datos
- Reglas son programas
- "Queries" ejecución de programas
- Prolog ejecuta una búsqueda de hechos para tratar de hacer verdaderas las reglas

Conceptos claves

- Las premisas de las reglas son conjunciones de situaciones necesarias para hacer al predicado verdadero.
- Las reglas alternativas son disyunciones de condiciones alternativas para hacer dicho predicado verdadero.

Conceptos claves

- Unificación rige el pattern matching
- Es posible definir reglas recursivas y utilizar patrones de programación como el de acumuladores.
- Necesario manejar:
  - Listas
  - Fail
  - cut.