

TALLER**RAZONAMIENTO LÓGICO**

Profesor: Ingrid Durley Torres & Jaime Alberto Guzmán Luna

Contenido del taller:

1. Introducción a SWI-Prolog
2. Manejo Inicial
3. Ejemplo con razonamiento
4. Actividad propuesta

Actualmente existen muchos intérpretes de *Prolog*. Cada uno tiene diferencias en su funcionamiento. Durante este curso se empleará la versión *SWI-Prolog*. Este intérprete está disponible de manera gratuita y es de código abierto. Por este motivo, durante esta sesión de taller, se practicarán algunos de los conceptos requeridos para el buen manejo del intérprete.

El objetivo, comprender y utilizar las funciones básicas del interprete *SWI-Prolog* construyendo conocimiento, a partir de sus representaciones en lógica de predicados y generando inferencias a partir del razonamiento.

INTRODUCCION

Como usarlo? El intérprete *SWI-Prolog* nos permitirá, probar los programas de *Prolog* que se desarrollen durante el curso.

Entrar y salir de SWI-Prolog Lo primero que se debe saber del intérprete es como entrar a él:

<https://swish.swi-prolog.org/>

Esto ejecutará el intérprete y se mostrará lo siguiente en pantalla:

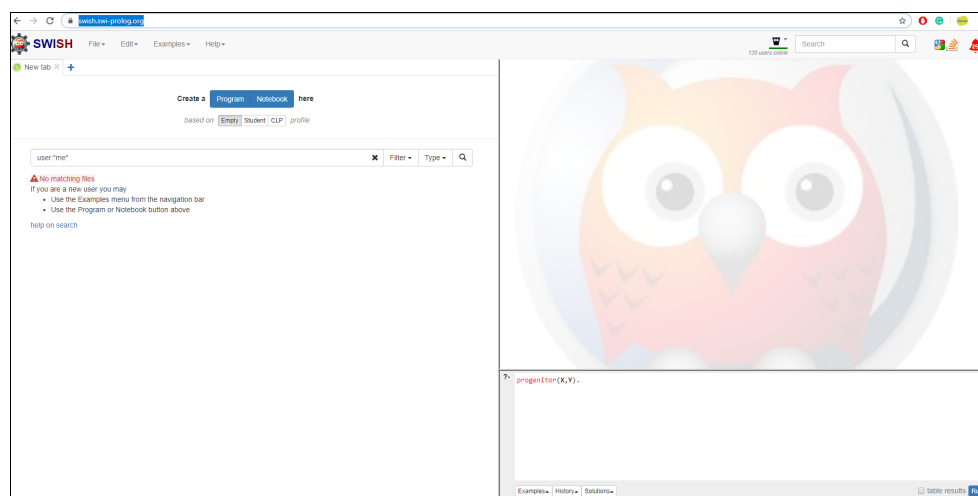
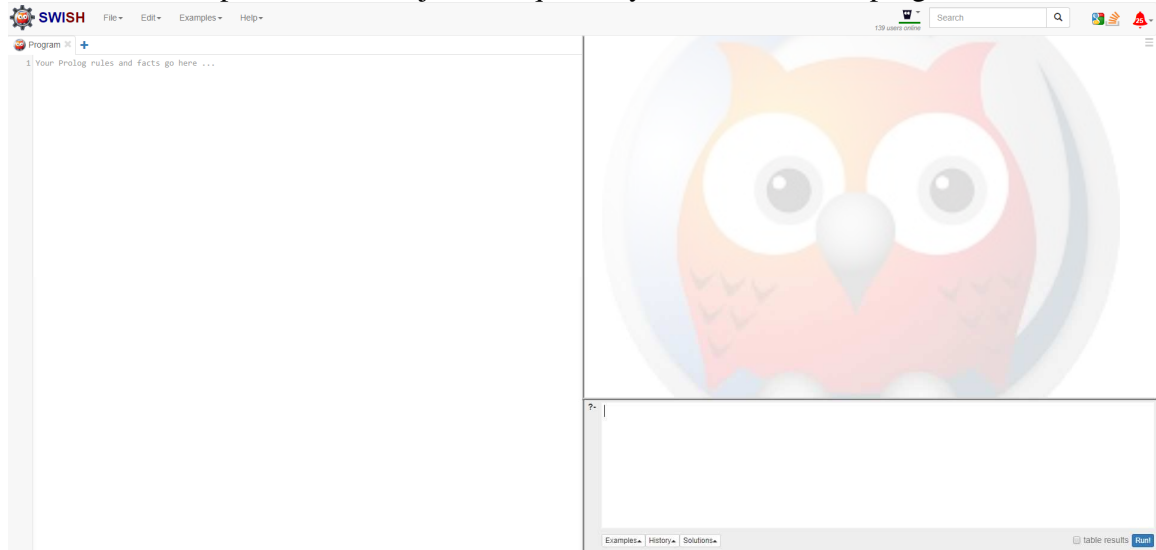


Figura 1. Interfaz web prolog

Crear una nueva pestaña de trabajo a la izquierda y seleccionar crear programa.



Una vez que aparece esta interfaz, el sistema nos indica donde debemos registrar todo el conocimiento (*Your Prolog rules and facts go here*), mientras en la parte derecha inferior de la pantalla: *prompt de Prolog* (?-), podremos ingresar los comandos que vayamos a probar como consultas.

Una vez que se ha cargado un programa o se ha creado uno, podremos hacer preguntas sobre la base de conocimientos que se encuentra cargada en memoria. Es importante recordar que después de cada enunciado, se debe de colocar un punto (.) para indicar el final del mismo, o un punto y coma, si continua, según el caso.

Manejo inicial swi prolog

Conceptos de consola y editor.

Consola: Es donde se pueden hacer las preguntas sobre los hechos y las reglas.

Editor: es donde se pueden escribir los programas (hechos y reglas)

Crear un archivo para editar programas en Prolog

En la consola ingresar a "*File -> New..*", aquí dar el nombre del archivo con extensión *pl*.

Ejemplo: clase1.pl y ponerlo en la ubicación que queramos.

Consultar un archivo.

Consultar un archivo reciente, o copiar todos los hechos y reglas en el editor, para poder hacer preguntas sobre esa base de conocimiento, desde la consola.

RAZONAMIENTO LÓGICO: LÓGICA PROPOSICIONAL

La **conjunción**, “y”, la representaremos poniendo una coma entre los objetivos “,” y consiste en *objetivos* separados que *Prolog* debe satisfacer, uno después de otro:

X, Y

La **disyunción**, “o”, tendrá éxito si se cumple alguno de los objetivos que la componen. Se utiliza un punto y coma “;” colocado entre los objetivos:

$X ; Y$

La **negación** lógica no puede ser representada explícitamente en *Prolog*, sino que se representa implícitamente por la falta de aserción: “no”, tendrá éxito si el objetivo *X* fracasa. No es una verdadera negación, en el sentido de la Lógica, sino una negación “por fallo”. La representamos con el predicado predefinido **not** o con $\backslash+$

$\text{not}(X)$

$\backslash+ X$

La **implicación** o **condicional**, sirve para significar que un hecho depende de un grupo de otros hechos. En castellano solemos utilizar las palabras “si ... entonces ...”. En *Prolog* se usa el símbolo “:-” para representar lo que llamamos una **regla**:
cabeza_de_la_regla :- cuerpo_de_la_regla.

La cabeza describe el hecho que se intenta definir; el cuerpo describe los objetivos que deben satisfacerse para que la cabeza sea cierta. Un mismo nombre de variable representa el mismo objeto siempre que aparece en la regla. Así, cuando *X* se instancia a algún objeto, todas las *X* de dicha regla también se instancian a ese objeto (*ámbito de la variable*).

Se llaman **cláusulas** de un predicado, tanto a los hechos como a las reglas. Una colección de cláusulas forma una **Base de Conocimientos**.

EJEMPLO 1.

Diseñar un SISTEMA DE REGLAS DE PRODUCCIÓN, que permita decidir automáticamente si a un cliente de una sucursal bancaria se le va a conceder un préstamo o no.

1. La cantidad del préstamo no puede superar, los 10 millones de pesos y la utilización del préstamo no es relevante, para su concesión.
 - a. La decisión de la concesión depende de la edad, el sueldo (si es estable o no y la cantidad) y si tiene avales o no.
 - i. Si el sueldo es estable (trabajador fijo o funcionario,) no hace falta que tenga avaladores, si gana más de 1500000 pesos mensuales.
 - ii. Si el sueldo es estable pero inferior o igual a 1500000 pesos necesita, al menos, un avalador.

El sistema debe permitir demostrar que la respuesta a la petición de un préstamo de 5 millones de pesos por una persona de 31 años, funcionaria, que gana 1550000 pesos mensuales y no tiene avalador, es afirmativa.

Actividad 1: Construya la correspondiente base de hechos sobre la que se le permitirá razonar (basta copiar y pegar).

% Hechos %

edad(juan, 40).

edad(sara, 36).

edad(cristobal, 55).

edad(tatiana, 30).

edad(marcela, 28).

edad(mauricio, 45).

edad(diana, 26).

funcionario(juan).

funcionario(mauricio).

trabajador_fijo(sara).

trabajador_fijo(tatiana).

trabajador_fijo(marcela).

trabajador_turno(diana).

trabajador_turno(cristobal).

sueldo(juan, 2000000).

sueldo(sara, 960000).

sueldo(cristobal, 1580000).

sueldo(tatiana, 100000).

sueldo(marcela, 500000).

sueldo(mauricio, 1600000).

prestamo(juan, 10000000).

prestamo(sara, 16000000).

prestamo(cristobal, 5000000).

prestamo(tatiana, 10000000).

prestamo(marcela, 5000000).

prestamo(mauricio, 5000000).

prestamo(diana, 800000).

% REGLAS %

cliente(X):- funcionario(X);trabajador_fijo(X); trabajador_turno(X).

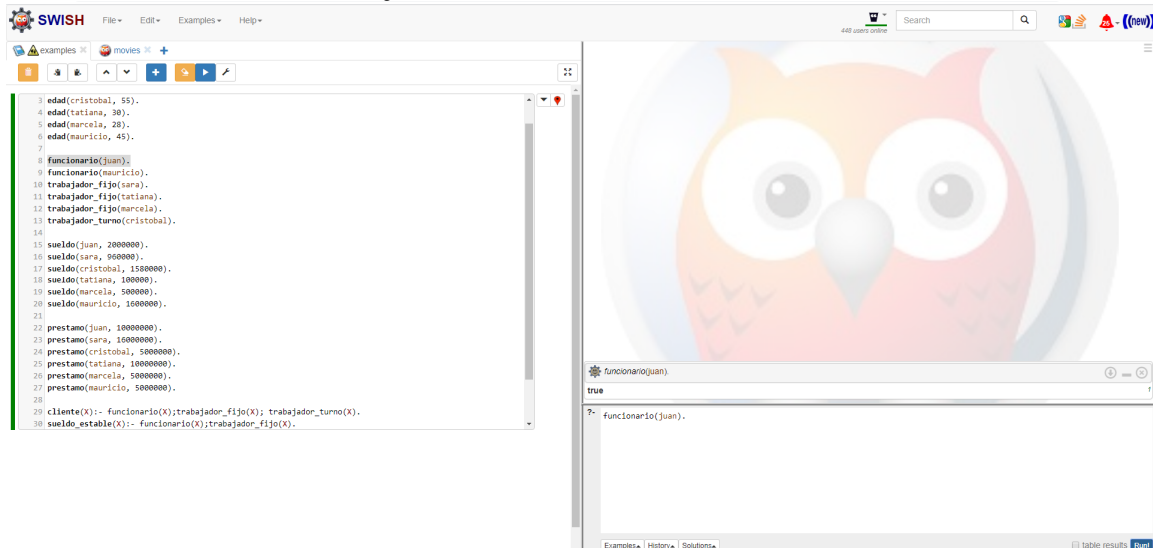
sueldo_estable(X):- funcionario(X);trabajador_fijo(X).

no_avaluador(X):- sueldo_estable(X), sueldo(X,Z), Z>= 1500000.

un_avaluador(X):- prestamo(X, Y), Y =< 10000000, sueldo_estable(X), sueldo(X,Z), Z< 1500000.

Actividad 2: Elabore consultas sobre los hechos y sobre las reglas que ya tiene incorporadas.
Por ejemplo:

funcionario(juan).



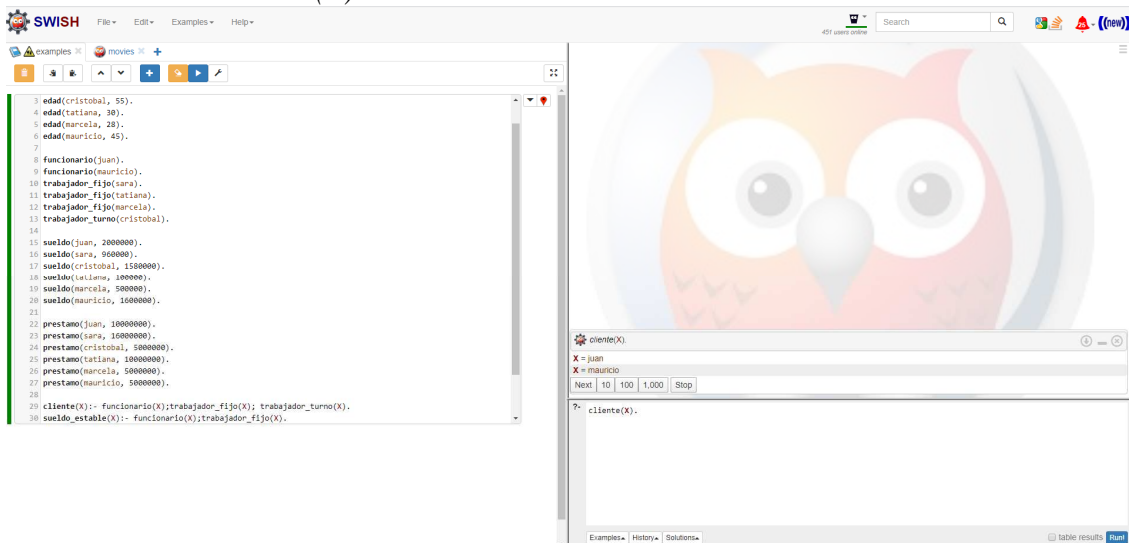
The screenshot shows the SWISH Prolog environment. The left pane contains a list of facts and rules. The right pane shows the query `funcionario(juan).` and the result `true`.

```

1 edad(cristobal, 55).
2 edad(tatiana, 30).
3 edad(marcela, 28).
4 edad(mauricio, 45).
5
6 funcionario(juan).
7 funcionario(mauricio).
8 trabajador_fijo(sara).
9 trabajador_fijo(tatiana).
10 trabajador_fijo(marcela).
11 trabajador_turno(cristobal).
12
13 sueldo(juan, 2000000).
14 sueldo(sara, 900000).
15 sueldo(cristobal, 1500000).
16 sueldo(tatiana, 1000000).
17 sueldo(marcela, 500000).
18 sueldo(mauricio, 1600000).
19
20 prestamo(juan, 10000000).
21 prestamo(sara, 10000000).
22 prestamo(cristobal, 5000000).
23 prestamo(tatiana, 10000000).
24 prestamo(marcela, 5000000).
25 prestamo(mauricio, 5000000).
26
27 cliente(X):- funcionario(X);trabajador_fijo(X); trabajador_turno(X).
28 sueldo_estable(X):- funcionario(X);trabajador_fijo(X).
  
```

Query: `funcionario(juan).`
Result: `true`

cliente(X).



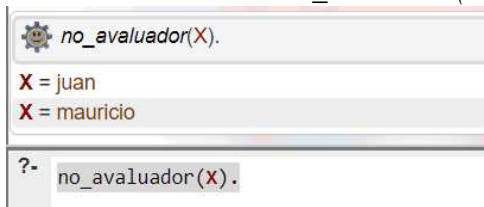
The screenshot shows the SWISH Prolog environment. The left pane contains a list of facts and rules. The right pane shows the query `cliente(X).` and the result `X = mauricio`.

```

3 edad(cristobal, 55).
4 edad(tatiana, 30).
5 edad(marcela, 28).
6 edad(mauricio, 45).
7
8 funcionario(juan).
9 funcionario(mauricio).
10 trabajador_fijo(sara).
11 trabajador_fijo(tatiana).
12 trabajador_fijo(marcela).
13 trabajador_turno(cristobal).
14
15 sueldo(juan, 2000000).
16 sueldo(sara, 900000).
17 sueldo(cristobal, 1500000).
18 sueldo(tatiana, 1000000).
19 sueldo(marcela, 500000).
20 sueldo(mauricio, 1600000).
21
22 prestamo(juan, 10000000).
23 prestamo(sara, 10000000).
24 prestamo(cristobal, 5000000).
25 prestamo(tatiana, 10000000).
26 prestamo(marcela, 5000000).
27 prestamo(mauricio, 5000000).
28
29 cliente(X):- funcionario(X);trabajador_fijo(X); trabajador_turno(X).
30 sueldo_estable(X):- funcionario(X);trabajador_fijo(X).
  
```

Query: `cliente(X).`
Result: `X = mauricio`

no_avaluador(X).



The screenshot shows the SWISH Prolog environment. The left pane contains a list of facts and rules. The right pane shows the query `no_avaluador(X).` and the result `X = mauricio`.

```

1 no_avaluador(X).
2
3 X = mauricio
4 X = mauricio
  
```

Query: `no_avaluador(X).`
Result: `X = mauricio`

- Indique que reglas se dispararon y en que orden

Actividad 3.

Incorpore una regla que indique

- iii. *Si el sueldo es estable pero inferior a 900000 pesos necesita, al menos, dos avaladores.*

Pruebe su consulta.

Actividad 4.

Incorpore el conocimiento necesario, que permita validar la siguiente inferencia:

Si X no es cliente, entonces X no accede al préstamo

Camilo no es cliente, entonces camilo, no accede al préstamo

Pruebe nuevamente la consulta

ESTUDIO DEL FORALL.

El forall es un predicado, o sea, no conviene pensar qué “hace”, sino qué relaciona, y/o cuándo se verifica.

- Definición de forall: El forall recibe dos parámetros, los dos deben ser consultas. Las consultas se hacen sobre predicados, o sea que forall es un predicado de orden superior, porque puede manejar predicados dentro de sus parámetros.
- ¿Cuándo se verifica?: Cuando a todos los que les pasa lo primero, les pasa lo segundo. De una forma más “técnica”, cuando todas las respuestas de la primer consulta son respuestas de la segunda.
- Cuando es posible aplicarlo? en las situaciones donde decimos “*a todos los que ...blah... les pasa ...bleh...*” es probable que el forall vaya bien.

EJEMPLO 2:

Actividad 5: Construya la correspondiente base de hechos sobre la que se le permitirá razonar (basta copiar y pegar).

% Hechos %

dulce(chocolate).
dulce(caramelo).
dulce(durazno).
amargo(radicheta).
amargo(cebada).

leGusta(roque,chocolate).
leGusta(roque,radicheta).
leGusta(pepe,caramelo).

% ... y muchos hechos más que describen los gustos de un grupo de personas

colorDePelo(roque,colorado).
colorDePelo(pepe,castanio).

```
% ... etc. con los colores de pelo
```

```
vive(roque,buenosAires).
```

```
vive(pepe,mendoza).
```

```
vive(lucas,salliquelo).
```

```
% ... y donde vive cada persona de la que queremos hablar
```

```
ciudadGrande(buenosAires).
```

```
ciudadGrande(mendoza).
```

```
% ... y así todas las ciudades grandes
```

Actividad 6.

Vamos a definir **esTierno**, donde decimos que una persona es tierna si todas las cosas que le gustan son dulces.

Estamos en un caso candidato a *forall*: “a todas las cosas que le gustan les tiene que pasar ser dulces”. Basado en lo anterior vamos a crear la base de reglas sencilla en Prolog así:

% REGLAS %

```
esTierno(Persona):- forall(leGusta(Persona, Alimento), dulce(Alimento)).
```

Significado:

... una Persona es tierna si ... todas las cosas que le gustan son dulces, exactamente lo que dijimos en español.

Actividad 7.

Realicemos la siguiente consulta:

```
?- esTierno(roque).
```

Que respuesta da? Porque cree que se da esta respuesta?

Actividad 8.

Realicemos la siguiente otra consulta:

```
?- esTierno(pepe).
```

Que respuesta da? Porque cree que se da esta respuesta?

Actividad 9.

¿Qué pasa si no hay soluciones para el primer parámetro de forall?

- ¿Qué pasa si consultamos si lucas es tierno y en nuestra base de conocimientos no hay nada que le guste?
 - `?- esTierno(lucas).`

Que respuesta da? Porque cree que se da esta respuesta?

FORALL E INVERSIBILIDAD

Actividad 10.

Qué pasa con las variables y la inversibilidad. ¿Será inversible el predicado esTierno? Hagamos la consulta con una variable en el argumento.

- `?- esTierno(X).`

Que respuesta da? Porque cree que se da esta respuesta?

Explicación.

En este caso la variable Persona llega sin ligar al forall. Entonces la primer consulta es:

- `leGusta(Persona, Alimento).`

Para cada una de las respuestas a esta consulta, se tiene que verificar

- `esDulce(Alimento)`

donde Alimento es lo que ligó la primer consulta.

Detalles:

- ¿Cuáles son las respuestas a la primer consulta? Todos los pares (persona,alimento) relacionados por **leGusta**.
- Entonces, el forall sólo se va a verificar si cualquier cosa que le guste a alguien, no importa a quién, es dulce.
- Con la base de conocimiento actual esto no se da y por lo tanto la respuesta es **false** y no nos devuelve la lista de personas que cumplen con ser tiernos como lo hemos hecho al principio del taller.

Actividad 11.

Analizando la actividad anterior, la regla propuesta no es lo que se quiere, ya que se debe tener la capacidad de poder consultar por las variables y retornar las personas que cumplen con la condición. Para lograr lo que queremos, tenemos que lograr que la variable **Persona** llegue ligada al **forall** mediante la siguiente regla:

```
esTierno(Persona):-
    persona(Persona),
    forall(leGusta(Persona,
    Alimento),esDulce(Alimento)).
```

Igualmente incluir los siguientes hechos:

```
persona(roque).
persona(pepe).
persona(lucas).
```


Hagamos la consulta nuevamente con una variable en el argumento.

- `?- esTierno(X).`

¿Que respuesta da? ¿Porque cree que se da esta respuesta?

Por último, repetir las consultas con las instancias de roque, pepe y lucas para verificar su comportamiento.