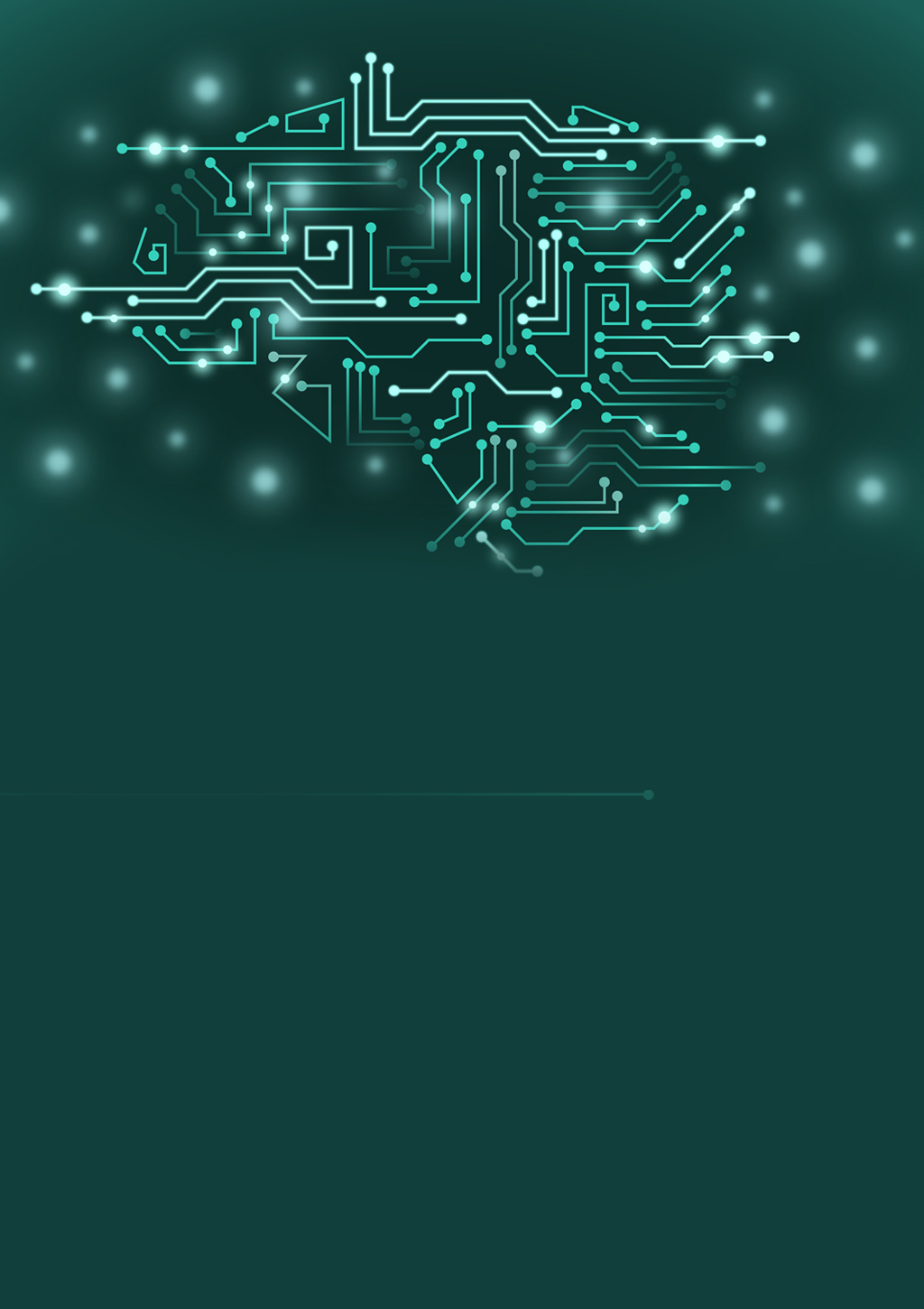
****

**PROLOG un lenguaje declarativo**

**Alois Carrera**

**Angel Cantoral**

**Ismael Guerrero**

**Prólogo**

En el presente escrito abarcaremos temas relacionados con lenguajes de programación, para ser directos, con un lenguaje en específico: PROLOG. Pero antes de adentrarnos en algo tan complejo para principiantes en este mundo de la programación, primero aclararemos términos importantes.

Un Lenguaje de Programación es un sistema estructurado de comunicación, prácticamente como lo es un ser humano, que permite la comunicación por medios como signos, palabras, sonidos o simples gestos. Ahora, haciendo referencia al lenguaje de programación, este está organizado de una manera específica para que aparatos puedan entenderse entre sí y a que a la vez se interpreten instrucciones que se deberán ejecutar. El término programación se define como un conjunto de instrucciones consecutivas, que de forma ordenada llevan a la ejecución una tarea específica. Las instrucciones reciben un nombre, que es código fuente, y es único para todos los lenguajes, ya que está diseñado para cumplir una función específica. Pueden utilizar diferentes bases para el control de comportamientos de dispositivos, también pueden ser usados para crear programas informáticos, o en el caso de PROLOG, resolver problemas que implican objetos y relaciones entre objetos.

El escrito está orientado a todo tipo de personas, tanto estudiantes como personas independientes que han decidido aprender o recordar algo que ya sabían sobre el lenguaje, pero que han olvidado por alguna razón. En principio este escrito fue hecho como un proyecto de la materia Herramientas Aplicadas a la Inteligencia Artificial, que es una asignatura dentro del plan de estudio de la carrera Lic. En ingeniería en Sistemas Computacionales, 3er año II semestre, de la Universidad Tecnológica de Panamá y está facilitada por el Prof. Dr. Cristian Pinzón.

Pero como todo proyecto, este busca expandirse para servir como base de conocimiento a las personas que deseen aprender un poco sobre este lenguaje de programación tan poderoso.

Índice

[1. Introducción 3](#_Toc27561693)

[1.1 ¿Qué es PROLOG? 3](#_Toc27561694)

[1.2 ¿Por qué utilizar PROLOG? 5](#_Toc27561695)

[1.3 Historia 7](#_Toc27561696)

[1.4 PROLOG y la programación lógica 8](#_Toc27561697)

[1.5 Resumen 9](#_Toc27561698)

[Glosario 10](#_Toc27561699)

[Cuestionario 12](#_Toc27561700)

[2. Fundamentos de PROLOG 13](#_Toc27561701)

[2.1 Los hechos de PROLOG 13](#_Toc27561702)

[2.2 Las Consultas de PROLOG 14](#_Toc27561703)

[2.3 Las reglas de PROLOG 17](#_Toc27561704)

[2.4 La sintaxis PROLOG 19](#_Toc27561705)

[2.5 Significado declarativo y procedural de los programas 22](#_Toc27561706)

[2.6 Programación lógica y Sistemas Expertos 23](#_Toc27561707)

[2.7 Representación del conocimiento 24](#_Toc27561708)

[2.8 Mecanismos de razonamiento 26](#_Toc27561709)

[2.9 Generación de explicación 32](#_Toc27561710)

[2.9.1 Introducción de incertidumbre 33](#_Toc27561711)

[3. Interfaz de Usuario en PROLOG 36](#_Toc27561712)

[3.1 Ejemplos resueltos de PROLOG 44](#_Toc27561713)

[Referencias 51](#_Toc27561714)

# 1. Introducción

Contenido

1.1. ¿Qué es PROLOG?

1.2. ¿Por qué utilizar PROLOG?

1.3. Historia

1.4. PROLOG y la programación lógica

1.5. Resumen

Glosario

Cuestionario

# 1.1 ¿Qué es PROLOG?

PROLOG es un lenguaje de programación declarativo. Este tipo de lenguajes declarativos se diferencian de los lenguajes imperativos o procedurales, que están basados en formalismos abstractos (PROLOG está basado en la lógica de predicados de primer orden y LISP, que es otro lenguaje de programación declarativa), y por tanto su semántica no depende de la máquina en la que se está ejecutando.

PROLOG es un lenguaje de programación diseñado para representar y utilizar el conocimiento que se tiene sobre un determinado dominio. Los programas en PROLOG responden preguntas sobre el tema del cual tienen conocimiento. Forma parte de lo que se conoce como programación declarativa. En lenguajes tradicionales se indica cómo resolver un problema, en la programación lógica se establecen hechos y reglas que dan indicaciones para saber qué hacer para resolver el problema. La popularidad de este lenguaje se debe a su capacidad de deducción, además de que es un lenguaje fácil de usar por su semántica y sintaxis.

Las sentencias en estos lenguajes se entienden sin necesidad de hacer referencia al nivel máquina para explicar los efectos colaterales. Por tanto, un programa escrito en un lenguaje declarativo puede usarse como una especificación o una descripción formal de un problema. Otra ventaja de los programas escritos en lenguajes declarativos es que se pueden desarrollar y comprobar poco a poco, y pueden ser sintetizados o transformados sistemáticamente.

Este lenguaje es muy útil para resolver problemas que implican objetos y relaciones entre objetos, es por eso que está basado en los siguientes mecanismos básicos:

* Unificación.
* Estructuras de datos basados en árboles.
* Backtracking automático.

La sintaxis del lenguaje consiste en los siguiente:

* Declarar Hechos sobre objetos y sus relaciones.
* Hacer preguntas sobre objetos y sus relaciones.
* Definir reglas sobre objetos y sus relaciones.

Elementos de PROLOG:

* Hechos
* Variables
* Consultas
* Conjunciones
* Reglas

Paradigmas de programación:

* Imperativa
* Orientada a Objetos
* Funcional
* Declarativa

# 1.2 ¿Por qué utilizar PROLOG?

PROLOG, en comparación con otros lenguajes de programación que presten las características necesarias para el desarrollo de, por ejemplo, sistemas expertos, está muy igualado con los más conocidos, y si no, un poco más por encima.

Algunas de las características propias de este lenguaje son:

* Recursión
* Inteligencia Artificial (Inferencia y representación del conocimiento)
* Procesamiento de lenguaje (Autómatas)
* Basado en lógica y programación declarativa.
* No se especifica cómo debe hacerse, sino qué debe lograrse.
* Una característica importante en PROLOG y que lo diferencia de otros lenguajes de programación, es que una variable sólo puede tener un valor mientras se cumple el objetivo.
* El programador se concentra más en el conocimiento que en los algoritmos.
  + ¿Qué es conocido? (hechos, reglas)
  + ¿Qué preguntar? (Cómo resolverlo)

En este lenguaje se llega a una solución infiriéndola desde algo ya conocido.

**Ventajas y desventajas de PROLOG.**

**Ventajas:**

* Habilidad para calcular de forma procedural.
* Facilidad para programar ya que se pueden escribir programas rápidamente con pocos errores originando programas claramente legibles, aún si no se conoce bien el lenguaje.
* No hay que pensar demasiado en la solución del problema, ya que PROLOG infiere sus respuestas basándose en las reglas declaradas dentro del programa.
* PROLOG utiliza un mecanismo de búsqueda independiente de la base de hechos. Es una buena estrategia puesto que garantiza el proceso de todas las posibilidades.

**Desventajas:**

* La resolución automática podría dar una respuesta incorrecta a una consulta. Son poco eficientes.
* PROLOG algunas veces es incapaz de reconocer que un problema es inaplicable o suficiente. Si el programa no contiene suficiente información para contestar una consulta, es incapaz de reconocerlo y responde no.

**Algunos juegos y programas populares que han programado en PROLOG:**

* Problema de las 8 reinas: El juego de las 8 reinas consiste en poner sobre un tablero de ajedrez ocho reinas sin que estas se amenacen entre ellas. Para resolver este problema se emplea el esquema backtracking.
* Analogía: para resolver cuestiones de analogía geométrica en test de inteligencia.
* Sistemas expertos: pueden percibir el ambiente mediante sensores y actúan sobre ese ambiente por medio de efectores.
* Compiladores: la construcción de compiladores e intérpretes son campos de desarrollo muy adecuados para PROLOG.
* Torres de hanoi.

# 1.3 Historia

PROLOG, cuyo nombre proviene del francés “PROgrammation en LOGique”, es un lenguaje de programación declarativa muy utilizado en Inteligencia Artificial, principalmente en Europa.

El lenguaje fue creado a principios de los años 70 en la Universidad de Aix-Marseille, por los profesores Alain Colmerauer y Philippe Roussel, como resultado de un proyecto de procesamiento de lenguajes naturales. Alain Colmerauer y Robert Pasero trabajaban en la parte del procesamiento del lenguaje natural y Jean Trudel y Philippe Roussel en la parte de deducción e inferencia del sistema. Interesado por el método de resolución SL, Trudel persuadió a Robert Kowalski para que se uniera al proyecto, dando lugar a una versión preliminar del lenguaje PROLOG a finales de 1971, cuya versión definitiva apareció en el año 1972.

La primera versión de PROLOG fue programada en ALGOL W e, inicialmente, se trataba de un lenguaje totalmente interpretado. En 1983, David H.D. Warren desarrolló un compilador capaz de traducir PROLOG en un conjunto de instrucciones de una máquina abstracta, denominada Warren Abstract Machine. Se popularizó en la década de los 80’ por la aparición de herramientas para microordenadores y su adopción para el desarrollo del proyecto de la quinta generación de computadores, para el que se desarrolló la implementación paralelizada del lenguaje llamada KL1.

Las primeras versiones del lenguaje diferían, en sus diferentes implementaciones, en muchos aspectos de sus sintaxis, empleándose mayormente como forma normalizada el dialecto propuesto por la universidad de Edimburgo, hasta que en 1995 se estableció un estándar ISO, llamado ISO-PROLOG.

# 1.4 PROLOG y la programación lógica

A continuación, se verá una breve perspectiva del lenguaje en cuestión.

PROLOG se compone de una serie de hechos, relaciones concretas entre objetos de datos y un conjunto de reglas, es decir, un patrón de relaciones entre los objetos de la base de datos. Estos hechos y reglas se introducen en la base de datos a través de una operación de consulta.

Un programa se ejecuta cuando el usuario introduce a una pregunta un conjunto de términos que deben ser todos ciertos. Los hechos y las reglas de la base de datos se usan para determinar cuáles sustituciones de variables de la pregunta son congruentes con la información de la base de datos.

Como intérprete, PROLOG solicita entradas al usuario. El usuario digita una pregunta o un nombre de función. La respuesta correcta será ‘yes’ y la falsa será ‘no’, se imprimen estas respuestas, así como una asignación a las variables de la pregunta que hacen cierta la pregunta, es decir, que unifican la pregunta. La ejecución de PROLOG, aunque se basa en la especificación de predicados, opera en forma muy parecida a un lenguaje aplicativo como LISP o ML. El desarrollo de las reglas en PROLOG requiere el mismo pensamiento recursivo que se necesita para desarrollar programas en otros lenguajes aplicativos. PROLOG tiene una sintaxis y semántica simples. Puesto que busca relaciones entre una serie de objetos, la variable y la lista son las estructuras de datos básicas que se usan. Una regla se comporta de forma muy parecida a un procedimiento, excepto que el concepto de unificación es más complejo que el proceso relativamente sencillo de sustitución de parámetros por expresiones.

# 1.5 Resumen

En este capítulo de PROLOG se establecen las definiciones sobre los lenguajes de programación y de la herramienta PROLOG así. Se ve de manera breve la historia de esta herramienta para saber cómo ha llegado a ser lo que es hoy en día. Se realizan observaciones importantes del por qué utilizar PROLOG, dándole a los lectores las características de la herramienta, así como también las ventajas y las desventajas, sin necesidad de sugestionar la decisión final del lector, simplemente colocando estos apartados para que la persona saque sus propias conclusiones sobre la herramienta y decida por sí misma si le conviene utilizarla o no. Se habló de PROLOG en la programación lógica y sus herramientas para la creación de sistemas expertos, que no son más que sistemas que nos permiten partir de hechos y aplicar reglas para obtener nuevos hechos, llamados hechos inferidos. Reemplazan o complementan el conocimiento el conocimiento del experto, según el caso. Están compuestos por una base de reglas que contiene todas las reglas conocidas por el experto en cuestión y que permite llegar a conclusiones a través de estas. También se componen de una base de hechos que representan todo lo que se conoce por parte del usuario, así como también los hechos inferidos hasta el momento. Una interfaz de usuario permitirá comunicarse de forma clara con las distintas personas que utilicen el sistema. El núcleo del sistema experto es su motor de inferencia. Es el que va a escoger y aplicar las reglas, y ejecutar las interacciones con el usuario. Puede ser de razonamiento deductivo si parte de los hechos para obtener otros nuevos o de razonamiento inductivo si parte de un objetivo y busca cómo alcanzarlo.

# Glosario

1. **ALGOL W**

ALGOL W es un Lenguaje de programación. Se basa en una propuesta para ALGOL X como sucesor de ALGOL 60. Representaba una modificación relativamente conservadora de ALGOL 60, agregando cadenas, cadenas de bits, números complejos y referencias a tipos de datos de registro. Introdujo la declaración While, remplazaron lo switch’s con la declaración case. Fortalecieron el lenguaje de forma general.

1. **Backtracking**

Es un algoritmo general para encontrar todas o algunas soluciones a problemas computacionales, en particular problemas de satisfacción de restricciones, que incrementa gradualmente a los candidatos soluciones y abandona a otro candidato tan pronto determine que no funciona como solución válida.

1. **Hechos**

Un hecho, en PROLOG, es una relación entre objetos. Ejemplo: La capital de Francia es París. La escritura en PROLOG sería:

capital(francia,parís).

1. **Intérprete**

Es un programa informático capaz de analizar y ejecutar otros programas.

1. **LISP**

Es una familia de lenguajes de programación de computadora de tipo multiparadigma con larga historia y una inconfundible y útil sintaxis basada en la notación polaca.

1. **Procedural**

También conocida como la generación por procedimientos, algo es considerado como procedural o procedimental cuando ha sido creado mediante un algoritmo.

1. **Programación** **declarativa**

Es un paradigma de programación que está basado en el desarrollo de programas especificando o declarando un conjunto de condiciones, proposiciones, afirmaciones, restricciones, ecuaciones o transformaciones que describen el problema y detallan su solución.

1. **Recursión**

es la forma en la cual se especifica un proceso basado en su propia definición.

1. **Reglas**

Una Regla consta de dos partes, una cabeza y un cuerpo. La cabeza y el cuerpo están unidos mediante el símbolo :-

*cabeza :- cuerpo.*

El cuerpo puede estar formado por varios hechos.

*cabeza :- hecho1, hecho2, …, hecho.*

# Cuestionario

* ¿Qué es PROLOG?
* ¿Qué es un sistema experto?
* ¿Cuáles son las ventajas y desventajas de PROLOG?
* ¿En qué lenguaje de programación fue elaborada la primera versión de PROLOG?
* ¿Cuáles son las características propias del lenguaje PROLOG?
* ¿Cuáles son los mecanismos en los que está basado PROLOG?
* ¿En qué lógica está basado PROLOG?
* ¿Qué es un lenguaje de programación?
* ¿En qué consiste la sintaxis de PROLOG?
* ¿En qué año apareció la versión final de PROLOG?
* ¿Cuáles son los elementos de PROLOG?
* ¿Cuáles son los paradigmas de programación de PROLOG?

# 2. Fundamentos de PROLOG

# 2.1 Los hechos de PROLOG

Para explicar los fundamentos de PROLOG vamos a utilizar el típico ejemplo de las relaciones familiares.

Para decir que Laura es uno de los dos progenitores de Damián, podríamos declarar el siguiente hecho PROLOG:

*progenitor(Laura, Damián)*

“progenitor” es el nombre de la relación o nombre de predicado y “Laura” y “Damián” son los argumentos. Los hechos acaban siempre con punto. Nosotros interpretaremos que Laura, primer argumento de la relación, es la madre de Damián, segundo argumento de la relación. Sin embargo, este orden es arbitrario y cada programador puede darle su propio significado. Los nombres de las relaciones y los argumentos que se refieren a objetos o personas concretas se escribirán con minúscula. Otros ejemplos de hechos pueden ser los siguientes:

*le\_gusta\_a(Juan, María)*

*valioso(oro)*

*tiene(Juan, libro)*

*da(juan, libro, María)*

Los nombres también son arbitrarios y el programador decidirá la interpretación que haga de ellos. La relación *le\_gusta\_a(Juan, María)* es equivalente a la relación a(b, c), aunque para que la interpretación sea más sencilla, se recomienda que los nombres se elijan de forma que ayuden a su interpretación. Los hechos no tienen que reflejar el mundo real necesariamente, pero será única y exclusivamente lo que PROLOG tomará como verdadero. Un conjunto de hechos (también llamados cláusulas), junto con un conjunto de reglas, forman lo que se llama una base de datos PROLOG.

# 2.2 Las Consultas de PROLOG

Sobre un conjunto de hechos se pueden realizar una serie de preguntas. Por ejemplo:

*?- le\_gusta\_a(Juan, María)*

PROLOG busca automáticamente en la base de datos si existe un hecho que se puede unificar con el hecho que aparece en la pregunta. PROLOG contestará “SI” si encuentra ese hecho y “NO” si no lo encuentra. La contestación “NO” no implica que el hecho sea falso (aunque sí lo sea para nuestra base de datos), sino que no se puede probar (en general) que sea verdadero con el conocimiento almacenado en la base de datos.

Para realizar preguntas más interesantes, como, por ejemplo, qué le gusta a María o cuáles son los padres de Damián, se usarán las variables. En PROLOG las variables empiezan por mayúscula. Por ejemplo:

*?-le\_gusta\_a(María, X)*

*?-progenitor(Y, Damián)*

Para obtener la o las respuestas, PROLOG recorre la base de datos hasta encontrar el primer hecho que coincide con el nombre de la relación y su aridad y con los argumentos que no son variables. Marca esa posición para poder recordar dónde se quedó en el recorrido por la base de datos. La o las variables se instancian al valor que le corresponde según el lugar que ocupan en la relación, y ese valor es la respuesta que proporciona PROLOG. Si pulsamos RETURN no obtendremos más que la primera respuesta. Si se quieren obtener todas las respuestas (para el caso de que exista más de una) se teclea “;”. Cuando pulsamos “;”, PROLOG sigue automáticamente la búsqueda desde la marca de posición en la que se había quedado en la base de datos. Se dice entonces que PROLOG intenta resatisfacer la pregunta. Se desinstancian las variables que se habían instanciado, y sigue buscando otro hecho que coincida sintácticamente con la pregunta. A este mecanismo se le llama backtracking, y PROLOG lo hace automáticamente. Para resolver preguntas más complejas, como, por ejemplo, ¿se gustan Juan y María? o ¿tienen Ana y Damián un progenitor común, es decir, son hermanos? o ¿quién es el nieto(s) de Tomás?, se utilizan conjunciones de objetivos, es decir, preguntas separadas por comas, que en PROLOG corresponden a la “Y” lógica.

*?-le\_gusta\_a(Juan, María), le\_gusta\_a(María, Juan).*

*?-progenitor(X, Ana), progenitor(X, Damián).*

*?-progenitor(Tomas, X), progenitor(X, Y).*

En una conjunción de objetivos correspondientes a la misma pregunta, la misma variable se refiere al mismo objeto (en el segundo ejemplo, X se refiere a la misma persona). En preguntas u objetivos distintos, el mismo nombre de variable se refiere a distintos objetos (la X del segundo y el tercer ejemplo se refieren a personas distintas). Para buscar las soluciones a una conjunción de objetivos, PROLOG establece una marca de posición para cada objetivo, y recorre toda la base de datos en la búsqueda de cada objetivo.

Ejemplo:

Suponemos definida la siguiente base de datos de relaciones familiares:

*progenitor(Clara, José)*

*progenitor(Tomas, José)*

*progenitor(Tomas, Isabel.*

*progenitor(José, Ana)*

*progenitor(José, Patricia)*

*progenitor(Patricia, Jaime)*

Para demostrar si Clara es bisabuela de Jaime, utilizaríamos la siguiente conjunción de objetivos:

*?-progenitor(Clara, X), progenitor(X,Y), progenitor(Y, Jaime).*

Para explicar el funcionamiento del backtracking que permite obtener todas las respuestas de la anterior conjunción de objetivos, vamos a utilizar el esquema de deducción natural.

*?-progenitor(Clara, X), progenitor(X,Y), progenitor(Y, Jaime).*

*½X=jose*

*?-progenitor(Clara, José), progenitor(José, Y), progenitor(Y, Jaime).*

*½Y=Ana ½Y=Patricia*

*progenitor(José, Ana), progenitor(Ana, Jaime)*

*progenitor(José, patricia), progenitor(Patricia, Jaime)*

Explicación: Se intenta satisfacer X del primer objetivo de la conjunción de objetivos, progenitor(Clara, X). El objetivo coincide con el primer hecho de la base de datos, donde se quedará su marca de posición asociada, y la variable X se instancia a “José”. Dentro de una misma pregunta, las variables con el mismo nombre se refieren al mismo objeto, por tanto, se sigue comprobando si se puede demostrar que “progenitor(José, Y), progenitor(Y, Jaime)”. Para el objetivo progenitor(José, Y) se empieza a buscar en la base de datos desde el principio, hasta encontrar un hecho que unifique. El primer hecho que coincide es progenitor(José, Ana), por tanto Y se instancia a “Ana“ y en este hecho se queda la marca de posición para este objetivo. Ahora se comprueba si se puede demostrar que es verdad progenitor(Ana, Jaime). Como este hecho no está en la base de datos, esta no es una solución, y se señala en el árbol de derivación natural. La última variable que se había instanciado se des instancia, y se sigue buscando en la base de datos, desde la marca de posición asociada, otro hecho que coincida con el objetivo actual. En este caso, progenitor(José, Patricia). Y se instancia a “patricia”. Ahora se demostrará si se cumple progenitor(Patricia, Jaime). Como este sí que es un hecho de la base de datos, la instanciación X=José e Y=patricia es una solución a la conjunción de objetivos del ejemplo, y se señala en el árbol con el símbolo o. Y no hay más soluciones, porque no hay otro hecho de la base de datos que coincida con progenitor(Clara, X).

# 2.3 Las reglas de PROLOG

Existe en PROLOG la posibilidad de definir la relación “abuelo(X,Y)” o la relación “tio(X,Y)” como reglas, además de poderlo hacer como hechos o como conjunción de objetivos.

Ejemplo:

*abuelo(X, Y):- progenitor(X, Z), progenitor(Z, Y)*

*tio(X, Y):- progenitor(Z, Y), progenitor(V, Z), progenitor(V, X)*

A la primera parte de la regla se le llama cabeza o conclusión, el símbolo ":-" es el condicional (SI), y a la parte de la regla que está después de “:-“ es el cuerpo o parte condicional. El cuerpo puede ser una conjunción de objetivos separados por comas. Para demostrar que la cabeza de la regla es cierta, se tendrá que demostrar que es cierto el cuerpo de la regla. Por lo visto hasta ahora, las cláusulas PROLOG son de tres tipos: hechos, reglas y preguntas. Las cláusulas PROLOG consisten en una cabeza y un cuerpo. Los hechos son cláusulas que tienen cabeza, pero no tienen cuerpo. Las preguntas sólo tienen cuerpo. Las reglas tienen siempre cabeza y cuerpo. Los hechos son siempre ciertos. Las reglas declaran cosas que son ciertas dependiendo de una condición. El programa PROLOG (o base de datos PROLOG) está formado por hechos y reglas y para PROLOG no hay ninguna distinción entre ambas. Las preguntas se le hacen al programa para determinar qué cosas son ciertas.

Con la definición del tipo de reglas anterior se pueden resolver problemas interesantes, sin embargo, la gran potencia del PROLOG está en la definición de reglas recursivas. Como hemos visto en ejemplos anteriores se puede definir la relación progenitor, y las reglas abuelo, bisabuelo, tatarabuelo, etc. En general, puede ser interesante definir la relación predecesor(X,Y). Un predecesor de X podrá ser el progenitor de X. También será predecesor si es abuelo/a, si es tatarabuelo/a, etc., es decir, necesitaríamos un conjunto de reglas como:

*predecesor(X, Y):-progenitor(X, Y).*

*predecesor(X, Y):-progenitor(X,Z), progenitor(Z, Y).*

*predecesor(X, Y):-progenitor(X, Z), progenitor(Z, V),*

*progenitor(V, Y).*

La definición de varias reglas con el mismo nombre de relación equivale en PROLOG a la “O” lógica o disyunción. Pero la definición de este conjunto de reglas es infinita, nunca terminaríamos de escribirlo. La siguiente regla recursiva nos permitirá definirlo.

Ejemplo:

*predecesor(X, Y):-progenitor(X, Y)*

*predecesor(X, Y):-progenitor(X, Z), predecesor(Z, Y)*

Ilustración 1. Diagrama de predecesor

La primera definición de predecesor corresponde al caso más sencillo en el que el predecesor corresponde al progenitor. Es lo que llamaremos la regla de salida de la recursividad. Si al intentar comprobar si una persona es predecesora de otra, se halla en la base de datos que la primera persona es progenitora de la segunda, se habrá demostrado y PROLOG responderá que sí. En caso contrario, se intentará demostrar que es predecesor utilizando la segunda regla, que contiene la llamada recursiva.

# 2.4 La sintaxis PROLOG

Los objetos o términos PROLOG pueden ser objetos simples o estructuras. Los objetos simples pueden ser constantes o variables. Las constantes serán átomos o números. Los átomos empiezan con letra minúscula (nunca con números), pueden contener caracteres especiales y pueden ser nombres entre comillas simples. Los números serán enteros o reales, sin una definición explícita de tipos. PROLOG se utiliza para una programación simbólica, no numérica, por eso los enteros se utilizarán por ejemplo para contar el número de elementos de una lista, pero los reales son poco utilizados. Las variables empiezan con mayúscula o con subrayado. Las variables anónimas son aquellas cuyo nombre es sólo el carácter subrayado (\_). Se usan cuando no es importante el nombre de la variable o cuando la variable no puede unificar con otra, dentro de la misma cláusula.

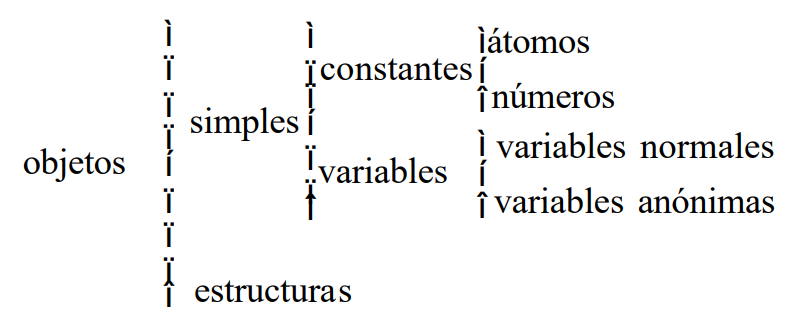


Ilustración 2. Composición de Prolog

Por ejemplo:

*tiene\_un\_hijo(X):-progenitor(X, Y)*

“Y” no unifica con otra variable en la definición de la relación “tiene\_un\_hijo”, por lo que es aconsejable sustituirla por una variable anónima (en algunos entornos de programación PROLOG se advierte sobre la conveniencia de ello).

*tiene\_un\_hijo(X):-progenitor(X, \_)*

Es importante señalar que el alcance de una variable es la cláusula donde aparece, y el alcance de una constante es todo el programa PROLOG.La sintaxis de las estructuras es la misma que la de los hechos. Los funtores de las estructuras son los nombres de los predicados de hechos y reglas. Los argumentos de los hechos y las reglas son los componentes de las estructuras.

Ejemplo:

*tiene(Juan, libro).*

*tiene(juan, libro(don\_quijote, autor(Miguel, Cervantes), 3256)).*

Las estructuras se pueden representar mediante árboles. Por ejemplo, un punto en 2- dimensiones se puede representar con la estructura punto(X,Y) y un punto en 3- dimensiones con la estructura punto(X,Y,Z). Ambas estructuras son distintas porque tienen distinta aridad o número de argumentos. La estructura en árbol correspondiente sería:

Punto

Punto

X

Y

X

Y

Z

Ilustración 3. Dimensiones

Un segmento se puede representar con 2 puntos, segmento(punto(X1,Y1), punto(X2,Y2)); y un triángulo con 3 puntos, triangulo(punto(X1,Y1), punto(X2,Y2), punto(X3,Y3)). La operación más importante sobre términos es la unificación. Dos términos pueden unificarse si son idénticos o las variables de ambos términos pueden instanciarse a objetos tales que después de la sustitución de las variables por esos objetos, los términos sean idénticos.

Ejemplo:

*fecha(D,M,1998) = fecha(D1,mayo,A)*

*piloto(A,Londres) = piloto(londres,paris)*

*punto(X,Y,Z) = punto(X1,Y1,Z1)*

*f(X,a(b,c)) = f(Z,a(Z,c))*

En el caso de que “?-X=Y” sea una pregunta PROLOG, puede ocurrir varias cosas:

1. Que una de las dos variables no esté instanciada pero la otra sí, entonces el objetivo se satisface y la variable que no estaba instanciada queda instanciada al valor de la variable que estaba instanciada;
2. Que ambas variables estén sin instanciar, entonces el objetivo se satisface y además ambas variables quedan compartidas, es decir, que cuando una de las dos variables se instancie, la otra quedará instanciada a lo mismo;
3. Sí X e Y son constantes ambas tienen que ser el mismo objeto, y si son estructuras, tienen que tener el mismo funtor, el mismo número de argumentos y que coincidan todos los argumentos.

Usando solamente la unificación se pueden resolver problemas interesantes.

Ejemplo:

*vertical(segmento(punto(X,Y1),punto(X,Y2))*

*horizontal(segmento(punto(X1,Y),punto(X2,Y))*

Un segmento vertical en 2-D está formado por dos puntos, en los cuales la coordenada X coincide. Y un segmento horizontal en 2-D está formado por dos puntos, en los cuales la coordenada Y coincide.

# 2.5 Significado declarativo y procedural de los programas

En un lenguaje declarativo puro, sería de esperar que el orden en el que aparecen los hechos y las reglas en la base fuera independiente de los datos, sin embargo, en PROLOG no es así. El significado declarativo tiene que ver sólo con las relaciones definidas por el programa. De esta manera, el significado declarativo determina cuál será la salida del programa. Por otro lado, el significado procedural determina cómo se ha obtenido esta salida; es decir, como evalúa las relaciones PROLOG. Si tenemos un conjunto de hechos y reglas con el mismo nombre de relación y la misma aridad, puede ser conveniente que los hechos estén situados en la base de datos antes que las reglas, (sobre todo, si los hechos son excepciones de las reglas). Además, también suele ser aconsejable poner la regla para salirse de la recursividad antes que la regla recursiva. La habilidad de PROLOG para calcular de forma procedural es una de las ventajas específicas que tiene el lenguaje. Como consecuencia esto anima al programador a considerar el significado declarativo de los programas de forma relativamente independiente de su significado procedural. Es decir, las ventajas de la forma declarativa de este lenguaje son claras (es más fácil pensar las soluciones y muchos detalles procedurales son resueltos automáticamente por el propio lenguaje) y podemos aprovecharlas. Los aspectos declarativos de los programas son, habitualmente, más fáciles de entender que los procedurales. Esta es la principal razón por la que el programador debe concentrarse en el significado declarativo y evitar distraerse por los detalles de cómo se ejecutan los programas.

El acercamiento a la programación declarativa no es suficiente. Una vez se tenga el concepto claro, y cuando se trabaje con programas grandes, los aspectos procedurales no se pueden ignorar por completo por razones de eficiencia. No obstante, el estilo declarativo a la hora de pensar en PROLOG debe ser estimulado y los aspectos procedurales ignorados para favorecer el uso de las restricciones.

# 2.6 Programación lógica y Sistemas Expertos

**¿Qué es un Sistema Experto?**

Un sistema experto es un programa que se comporta como un experto para algún dominio de aplicación, normalmente reducido. Debe ser capaz de explicar las decisiones que ha ido tomando y el razonamiento subyacente. Algunas veces es interesante que los sistemas expertos manejen incertidumbre y conocimiento incompleto. Para el desarrollo de un sistema experto se distinguen tres módulos:

1. La base de conocimiento
2. El motor de inferencia
3. La interfaz con el usuario

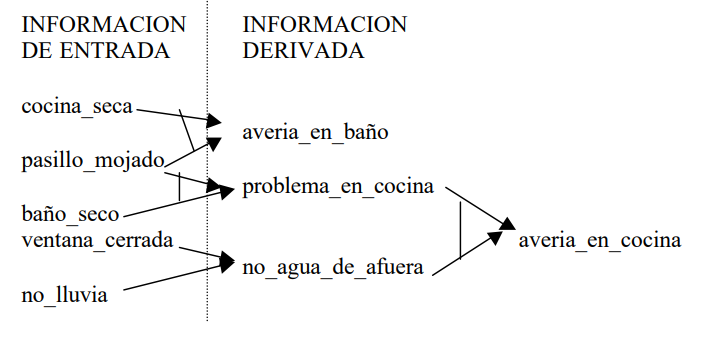
La base de conocimiento contiene el conocimiento específico del dominio de la aplicación, es decir, un conjunto de hechos, un conjunto de reglas que definen relaciones en el dominio, y métodos, heurísticos e ideas para resolver problemas en el dominio. El motor de inferencia contiene los programas necesarios para manejar el conocimiento de la base. La interfaz de usuario permitirá una comunicación fácil y agradable entre el usuario y el sistema, proporcionando detalles del proceso de resolución del problema. El motor de inferencia y la interfaz de usuario se pueden ver como un módulo, que le llamaremos shell. Teóricamente, el shell es independiente de la base de conocimiento. Sin embargo, para sistemas expertos complejos, un mismo shell no funciona todo lo bien que sería deseable para distintas bases de conocimiento, a no ser que los dominios de las aplicaciones sean muy similares. Aunque se necesiten algunas modificaciones en el shell cuando se cambia de dominio, los principios del shell permanecen invariables para distintas bases de conocimiento.

# 2.7 Representación del conocimiento

Las reglas “si-entonces”, también llamadas reglas de producción constituyen el formalismo más popular para representar conocimiento. En los sistemas expertos tienen las siguientes características deseables:

* Modularidad: cada regla define una parte de conocimiento pequeña y relativamente independiente.
* Incremental: se pueden añadir nuevas reglas a la base de conocimiento, que son relativamente independientes de las otras reglas.
* Modificable: se pueden cambiar reglas por otras nuevas.
* Transparente: es capaz de explicar las decisiones tomadas y las soluciones planteadas.

Para desarrollar un sistema experto que resuelva un problema real en un dominio concreto, el primer paso es consultar con humanos expertos en el dominio y después aprender de ese dominio uno mismo. Este paso supone un gran esfuerzo. Para los ejemplos de este capítulo, se va a considerar una base de conocimiento juguete que trata de diagnosticar averías de agua en un piso. La avería puede darse en el baño o en la cocina. En ambos casos, la avería provoca que haya agua en el suelo del pasillo. Solamente puede haber avería en el baño o en la cocina. La base de conocimiento se muestra como una red de inferencia. Las líneas que unen flechas significan “Y” lógica. Si las flechas no están unidas significa “O” lógica.



# 2.8 Mecanismos de razonamiento

Cuando el conocimiento se ha representado, se necesita un mecanismo de razonamiento para obtener conclusiones a partir de la base de conocimiento. Para las reglas “si-entonces” existen básicamente dos formas de razonamiento:

**El mecanismo de razonamiento encadenado hacia detrás.**

El razonamiento encadenado hacia detrás parte de una hipótesis e intenta demostrarla, buscando hechos que evidencien la hipótesis. Este mecanismo de razonamiento se implementa en PROLOG de manera trivial, como muestra el siguiente ejemplo.

Ejemplo: Mecanismo de razonamiento encadenado hacia detrás en PROLOG.

*hecho(averia\_en\_baño):-*

*hecho(pasillo\_mojado),*

*hecho(cocina\_seca).*

*hecho(problema\_en\_cocina):-*

*hecho(pasillo\_mojado),*

*hecho(baño\_seco).*

*hecho(no\_agua\_de\_afuera):-*

*(*

*hecho(ventana\_cerrada)*

*;*

*hecho(no\_lluvia)*

*).*

*hecho(averia\_en\_cocina):-*

*hecho(problema\_en\_cocina),*

*hecho(no\_agua\_de\_afuera).*

Los hechos reales en cada ocasión, es decir, las evidencias, se implementarán como hechos de la base de datos PROLOG. Por ejemplo:

*hecho(pasillo\_mojado).* %si no pongo hecho da problemas de

*hecho(baño\_seco).* % predicado no definido para aquellos

*hecho(ventana\_cerrada).* %predicados que no son hechos

Con esto se podría demostrar si es cierta la hipótesis: *?-hecho(averia\_en\_cocina).*

Los usuarios de un sistema experto, sin embargo, no tienen por qué conocer PROLOG. Además, no hay distinción entre la base de conocimiento (que para nosotros es el conjunto de hechos) del resto del programa (las reglas), ya que para PROLOG es lo mismo. Sin embargo, sería deseable esa distinción. El uso de la notación operador en PROLOG puede hacer que la sintaxis del sistema experto sea más amigable para un usuario que no conoce PROLOG.

Ejemplo Razonamiento encadenado hacia detrás utilizando la notación operador.

*:-op(800,fx,si).*

*:-op(700,xfx,entonces).*

*:-op(300,xfy,o).*

*:-op(200,xfy,y).*

*si*

*pasillo\_mojado y cocina\_seca*

*entonces*

*averia\_en\_baño.*

*si*

*pasillo\_mojado y baño\_seco*

*entonces*

*problema\_en\_cocina.*

*si*

*ventana\_cerrada o no\_lluvia*

*entonces*

*no\_agua\_de\_fuera.*

*si*

*problema\_en\_cocina y no\_agua\_de\_fuera*

*entonces*

*averia\_en\_cocina.*

Las evidencias se podrían representar con los siguientes hechos:

*hecho(pasillo\_mojado).*

*hecho(baño\_seco).*

*hecho(ventana\_cerrada).*

Los siguientes predicados interpretarán las reglas anteriores:

*es\_verdad(P):- hecho(P).*

*es\_verdad(P):-*

*si Condicion entonces P,*

*es\_verdad(Condicion).*

*es\_verdad(P1 y P2):-*

*es\_verdad(P1),*

*es\_verdad(P2).*

*es\_verdad(P1 o P2):-*

*(*

*es\_verdad(P1)*

*;*

*es\_verdad(P2)*

*).*

Y se podría comprobar si es cierta la hipótesis:

*?-es\_verdad(averia\_en\_cocina).*

**El mecanismo de razonamiento encadenado hacia delante.**

El mecanismo de razonamiento encadenado hacia delante parte de las evidencias y busca las conclusiones de esas evidencias, añadiéndolas a su vez a la base de conocimiento. Suponiendo que las reglas están almacenadas con la fórmula “si-entonces” del ejemplo anterior, los siguientes predicados constituyen la forma de razonamiento encadenado hacia delante.

Ejemplo Razonamiento encadenado hacia delante en PROLOG.

*hacia\_delante:-*

*findall(P,hecho(P),L), % recoge todas las evidencias en una lista L*

*hacia\_delante(L).*

*hacia\_delante(L):-*

*(*

*nuevo\_hecho\_derivado(L,P),*

*!,*

*write(‘Derivado: ‘), write(P), nl,*

*append([P],L,L1), %append/3 es el predicado predefinido*

*hacia\_delante(L1) %correspondiente a concatenar*

*;*

*write(‘No hay mas hechos’)*

*).*

*nuevo\_hecho\_derivado(L,Conclusion):-*

*si Condicion entonces Conclusion,*

*not miembro(Conclusion, L),*

*hecho\_compuesto(L,Condicion).*

*hecho\_compuesto(L,Condicion):-*

*miembro(Condicion,L). % Es un hecho simple, está incluido en la lista*

*hecho\_compuesto(L,Cond1 y Cond2):-*

*hecho\_compuesto(L,Cond1),*

*hecho\_compuesto(L,Cond2).*

*hecho\_compuesto(L,Cond1 o Cond2):-*

*(*

*hecho\_compuesto(L,Cond1)*

*;*

*hecho\_compuesto(L,Cond2)*

*).*

**¿Cuándo utilizar cada mecanismo de razonamiento?**

Si solamente se tiene una hipótesis a demostrar, el mecanismo de razonamiento a utilizar es el encadenamiento hacia detrás. Si hay muchas hipótesis y no hay ninguna razón para empezar con alguna de ellas en particular, es mejor utilizar el encadenamiento hacia delante. Si hay muchas evidencias y pocas hipótesis o conclusiones será más conveniente utilizar el encadenamiento hacia detrás; si hay pocas evidencias y muchas hipótesis será más útil el encadenamiento hacia delante. En algunas tareas es interesante una combinación de ambos mecanismos de razonamiento, hacia delante y hacia detrás. Por ejemplo, los diagnósticos médicos. Las evidencias iniciales de la enfermedad de un paciente pueden proporcionar una hipótesis inicial, utilizando un razonamiento hacia delante. La hipótesis inicial será confirmada o rechazada, utilizando evidencias adicionales y el razonamiento hacia detrás.

# 2.9 Generación de explicación

El razonamiento encadenado hacia delante va escribiendo la explicación de cómo se ha ido generando la respuesta. Para el razonamiento encadenado hacia detrás, se generará un árbol de prueba de cómo la conclusión final deriva de los hechos y las reglas de la base de conocimiento. Se define “<=” como un operador infijo y se modifica el predicado “es\_verdad” del ejemplo anterior, devolviendo como segundo argumento la prueba de su veracidad, como se muestra en el siguiente ejemplo.

Ejemplo Razonamiento encadenado hacia detrás generando explicación

*:-op(800,xfx,<=).*

*es\_verdad(P,P):-hecho(P).%La demostración de que el hecho(P) es verdad es P*

*es\_verdad(P,P<=CondPrueba):-*

*si Condicion entonces P,*

*es\_verdad(Condicion,CondPrueba).*

*es\_verdad(P1 y P2,Prueba1 y Prueba2):-*

*es\_verdad(P1,Prueba1),*

*es\_verdad(P2,Prueba2).*

*es\_verdad(P1 o P2, Prueba):-*

*(*

*es\_verdad(P1,Prueba)*

*;*

*es\_verdad(P2,Prueba)*

*).*

# 2.9.1 Introducción de incertidumbre

Hasta aquí se ha asumido para la representación del conocimiento, que los dominios de los problemas son categóricos, es decir, que las respuestas a todas las preguntas son verdadero o falso. Sin embargo, algunos dominios de expertos son más inciertos y la respuesta puede tener más matices (por ejemplo, la respuesta puede ser verdadera, altamente probable, probable, improbable o imposible). Una forma de expresar esta incertidumbre es mediante la teoría de la certeza, que permite asignar un número real a cada hecho o regla entre 0 y 1, que determine el grado de certeza o la medida de creencia de dicho hecho o regla. Esta teoría fue desarrollada en los años 70 durante la construcción de MYCIN, un sistema experto para diagnóstico médico en enfermedades de infecciones de sangre. En el siguiente ejemplo se muestra cómo la teoría de la certeza es aplicada para un uso efectivo.

Ejemplo Razonamiento con encadenamiento hacia detrás y grado de certeza.

Las evidencias se escribirán utilizando otro argumento que indicará el grado de certeza. Por ejemplo:

*hecho(pasillo\_mojado,1). % El pasillo está mojado*

*hecho(baño\_seco,1). % El baño está seco*

*hecho(cocina\_seca,1). % La cocina no está seca*

*hecho(no\_lluvia,0.8). % Probablemente no ha llovido*

*hecho(ventana\_cerrada,0). % La ventana no está cerrada*

Las reglas se escribirán con su certeza asociada de la siguiente forma:

sí Condición entonces Conclusión: Certeza

Por ejemplo:

*si*

*pasillo\_mojado y baño\_seco*

*entonces*

*problema\_en\_cocina : 0.9.*

Los predicados PROLOG que interpreten estos hechos y reglas se definen como sigue.

*:-op(800,xfx,:).*

*certeza(P,Cert):-hecho(P,Cert). % La certeza de los hechos va incluida en ellos*

*certeza(Cond1 y Cond2,Cert):-%La certeza de una condición compuesta por Y*

*certeza(Cond1,Cert1), % es el mínimo de las condiciones simples*

*certeza(Cond2,Cert2),*

*minimo(Cert1,Cert2,Cert).*

*certeza(Cond1 o Cond2,Cert):-%La certeza de una condición compuesta por O*

*certeza(Cond1,Cert1), % es el máximo de las condiciones simples*

*certeza(Cond2,Cert2),*

*maximo(Cert1,Cert2,Cert).*

*certeza(P,Cert):-*

*sí Condición entonces P: C1, % La certeza de una conclusión es la*

*certeza(Condicion,C2), % certeza de la regla multiplicada por la*

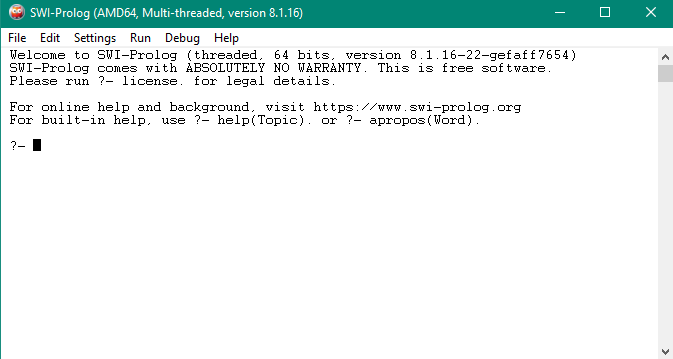
*Cert is C1\*C2. % certeza de la condición*

Existen más aproximaciones al tratamiento del conocimiento inexacto, entre las que destacan las siguientes:

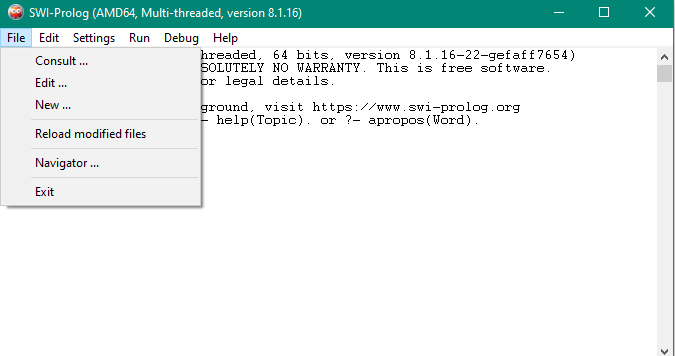
* **Aproximación bayesiana o probabilística.** Esta metodología se beneficia de que está fundada sobre unas bases estadísticas bien conocidas. Sin embargo, sólo podrá ser aplicada si se dispone de información sobre hechos pasados de los eventos y modelos probabilísticos disponibles para que sean fiables las conclusiones proporcionadas; la suma de las probabilidades a favor y en contra debe ser igual a 1, y se debe asumir la independencia de los datos.
* **Lógica difusa**. Esta metodología se define como una rama de la lógica clásica, que utiliza grados de pertenencia a conjuntos, en vez de la categórica pertenencia verdadero/falso.

# 3. Interfaz de Usuario en PROLOG

Esta sección del presente documento servirá para guiar a las personas nuevas en esta herramienta. Contendrá capturas y algunas reseñas explicativas para que los lectores conozcan las distintas ventanas que contiene el SWI-PROLOG.

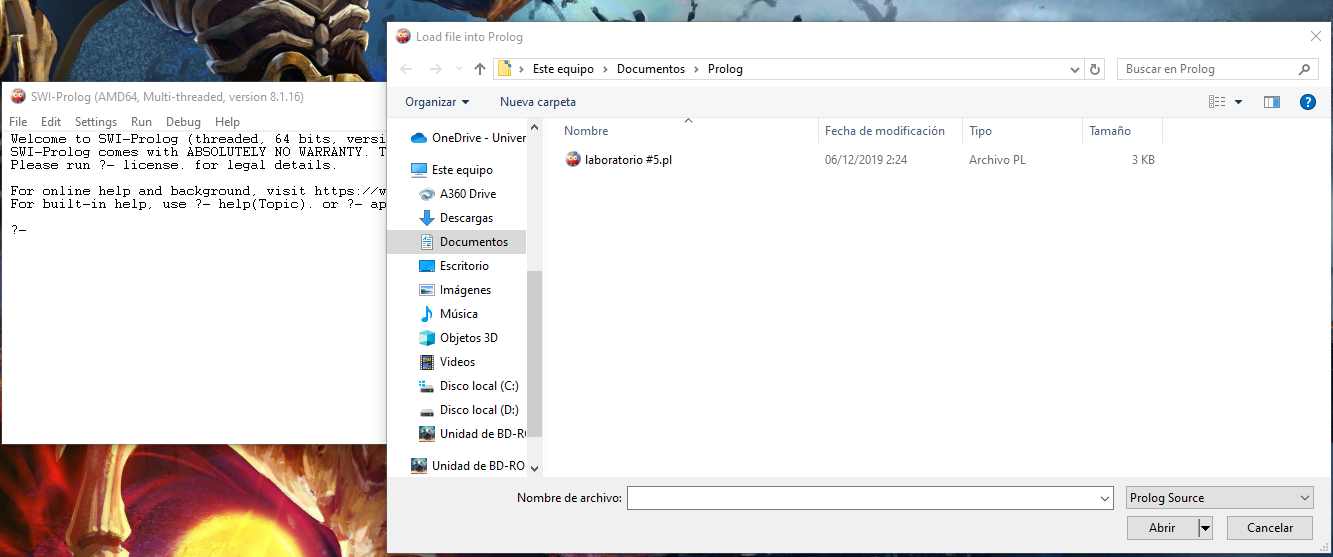


Esta será la ventana principal que se abrirá nada más ejecutarse el SWI-PROLOG. Mostrará un mensaje de bienvenida donde presentará la versión del programa, algunas condiciones y datos de soporte técnico. El signo de interrogación con el guion representa la clave para realizar una consulta.

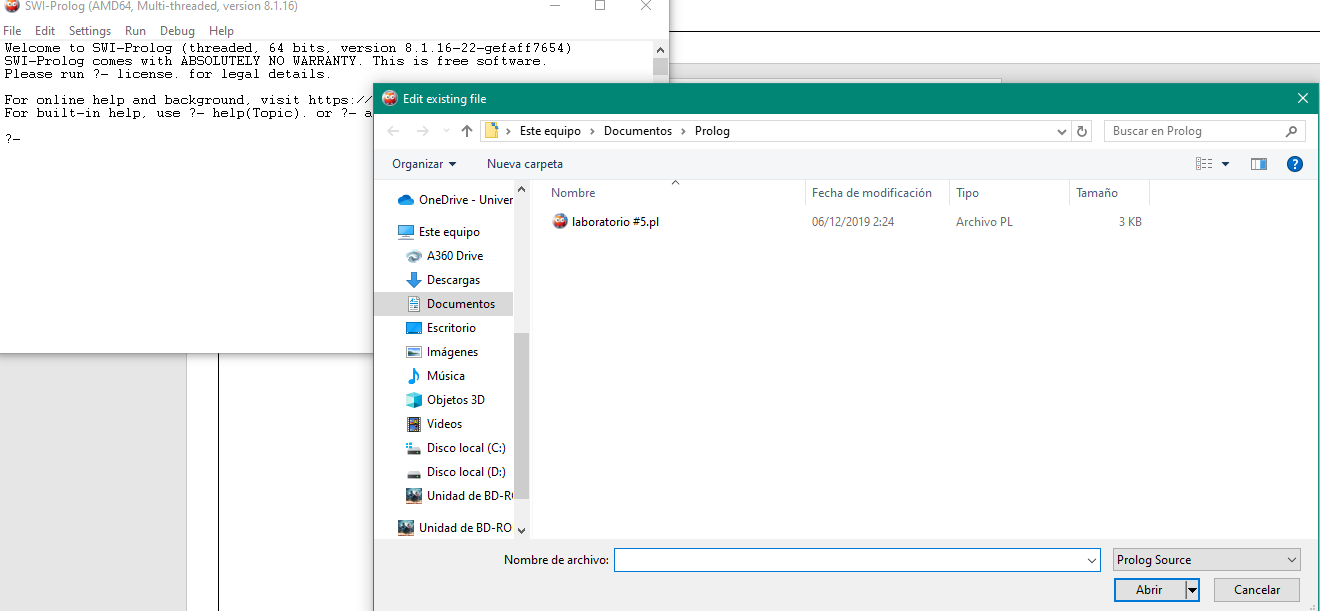


En la pestaña “File” se desplegará un menú dónde se encontrarán opciones como:

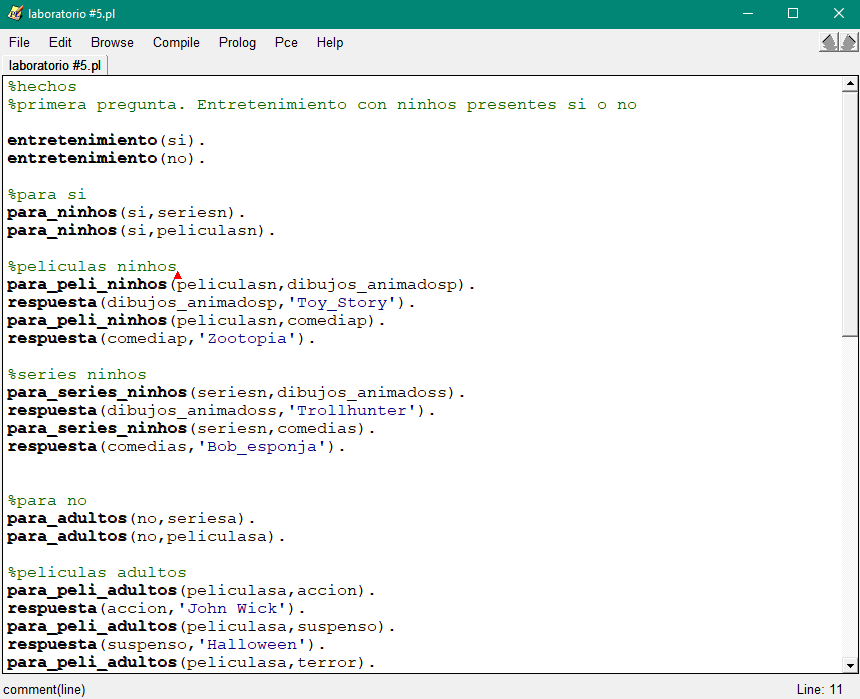
* Consult
* Edit
* New
* Reload modified files
* Navigator
* Exit



Dentro de la opción consult, se abrirá la carpeta por defecto de PROLOG donde se guardan todos los archivos “.pl” creados en la herramienta. En esta sección se podrán cargar los archivos para empezar a interactuar (realizar consultas) con ellos en el programa.

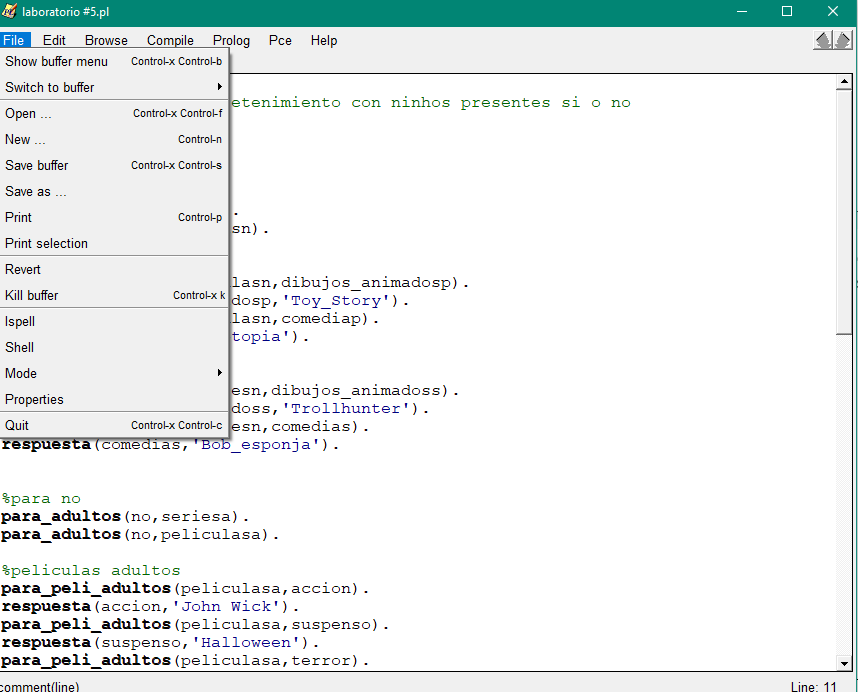


En la sección edit se abrirá una carpeta similar a la de consult, sólo que aquí en vez de consultar los datos guardados en la base de datos, enviará directo a una sección donde se podrá editar el código fuente del archivo. Imagen a continuación:

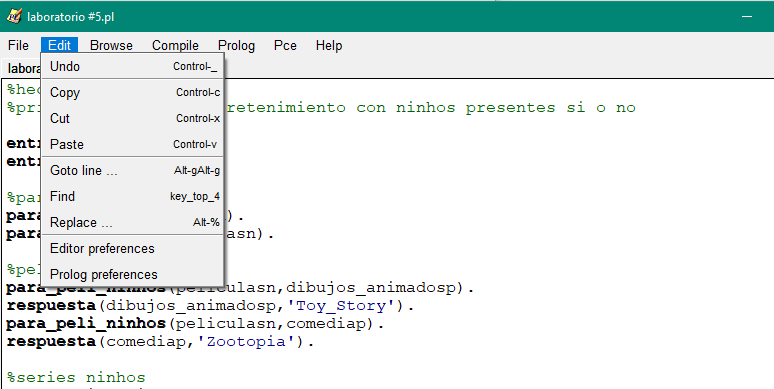


Como se puede observar, se muestra el código del archivo y se puede editar a voluntad del usuario los hechos y reglas.

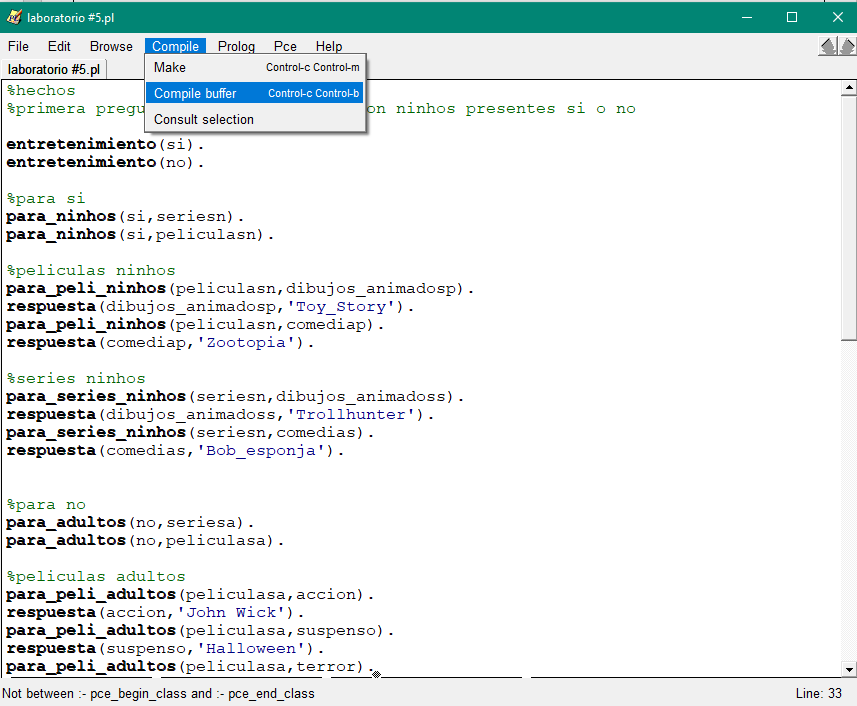
Lo más importante de esta sección:



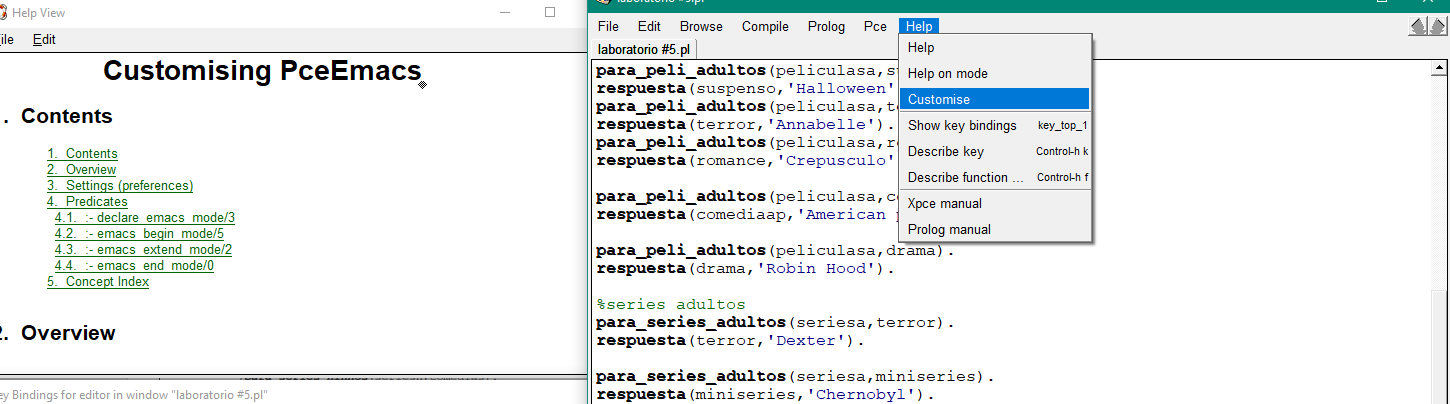
Entre lo importante a destacar, tenemos la pestaña file, donde encontraremos las opciones de abrir directamente un archivo, tal como lo tiene la ventana principal de la herramienta. También podemos ver opciones de guardado, de modo y de propiedades. La opción Kill buffer corta de manera automática la compilación del programa.

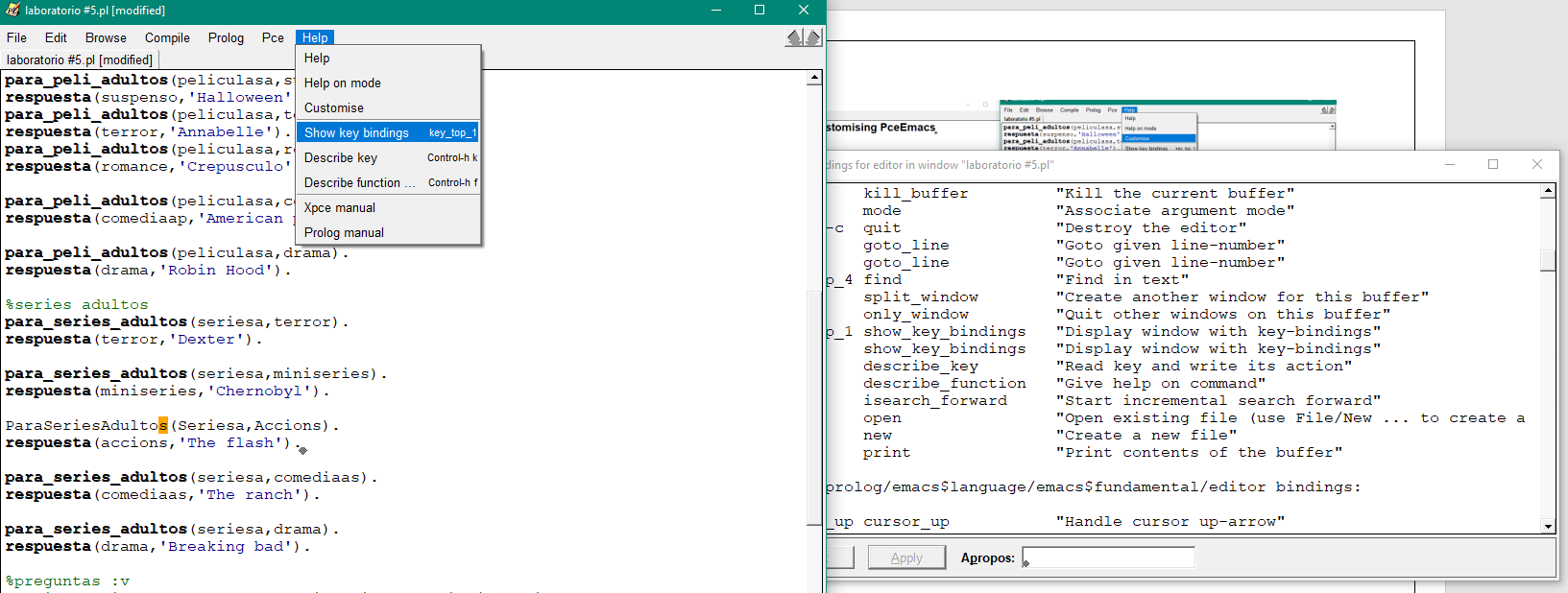


En la pestaña de edit, tendremos la opción de deshacer, copiar, cortar y pegar. Goto line que sirve para viajar a una línea de forma directa.



La pestaña de compile tiene opciones como compile buffer, que viene siendo la más importante de esta pestaña, ya que sirve para compilar el código que hemos trabajado en el editor.

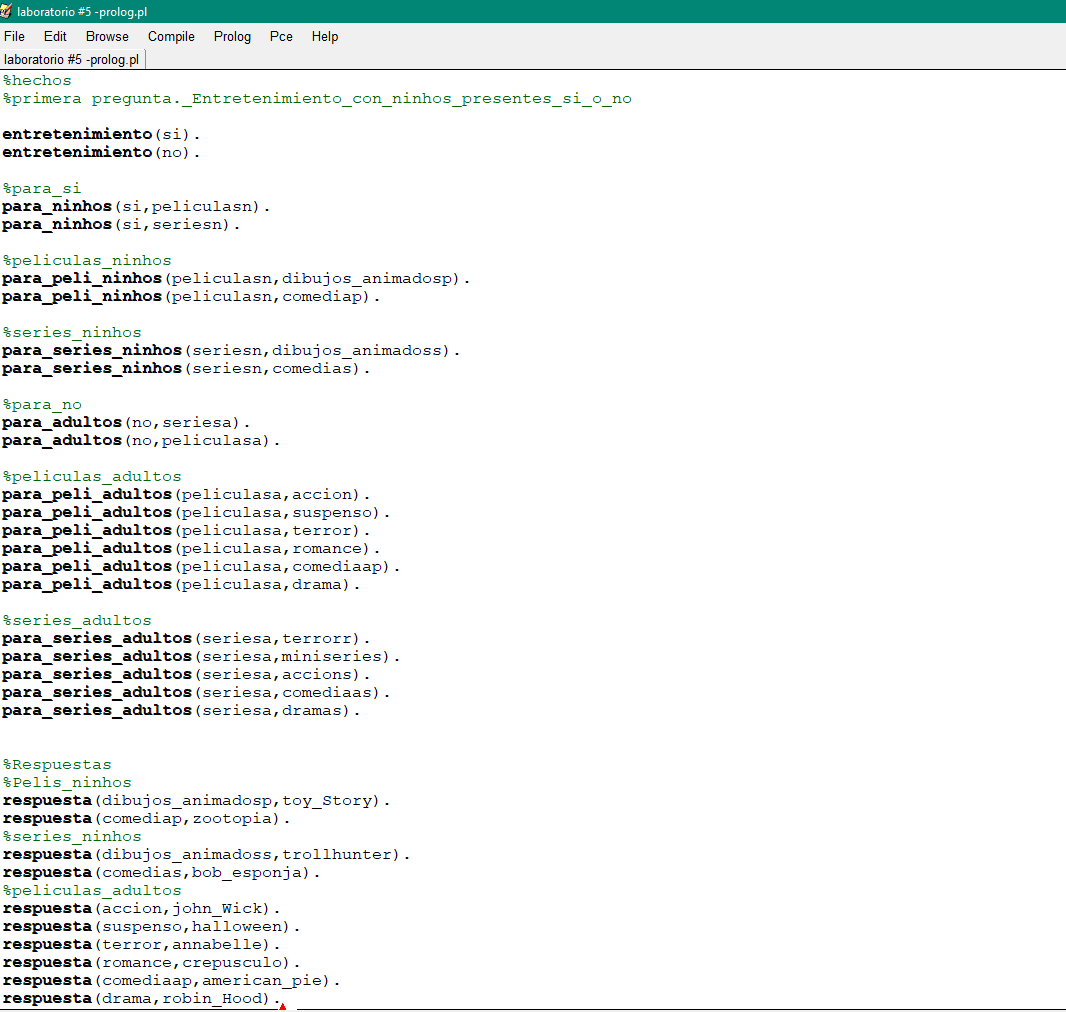


****En la sección de ayuda, nos encontramos con apartados como customize, que nos abrirá una ventana de vista de ayuda, que contiene información valiosa de datos poco conocidos sobre la aplicación.

En la misma pestaña de ayuda, habrá otro apartado importante de ver que será el show key bindings, que muestra una serie de comandos que servirán como atajos a la hora de utilizar el programa, para realizar funciones como abrir un archivo nuevo, crear un archivo nuevo, imprimir y deshacer, entre otras.

# 3.1 Ejemplos resueltos de PROLOG

**Ejemplo de un sistema experto para la elección de series y películas:**



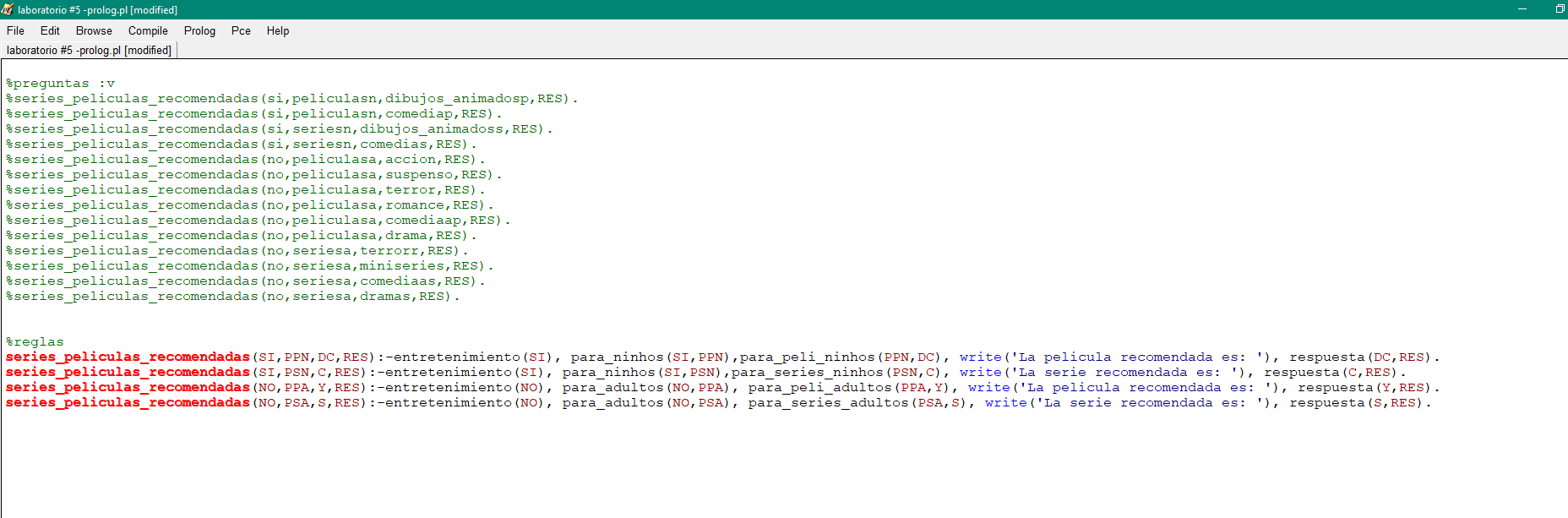
Se presenta el código del sistema experto, en esta parte tenemos solamente los hechos que utilizarán. Como ya sabemos, se tienen que crear los hechos con variables entre paréntesis, indicando que esas son las opciones que tiene que guardar la base de datos. En este caso se quiere llegar a que el sistema capte que la variable entretenimiento tendrá 2 ramas, si o no. Dependiendo de lo que el usuario decida, se va expandiendo el árbol, es por esto que los hechos están planteados de esa manera.

Para\_ninhos (si,peliculasn)

Para\_ninhos (si,seriesn)

Por ejemplo, indica que cuando el usuario seleccione el tipo de entretenimiento y escoja sí, entrará en el apartado de niños, donde las siguientes respuestas serán: Series o películas. Y de esa forma irían planteados los hechos hasta el final, que hay un leve cambio.

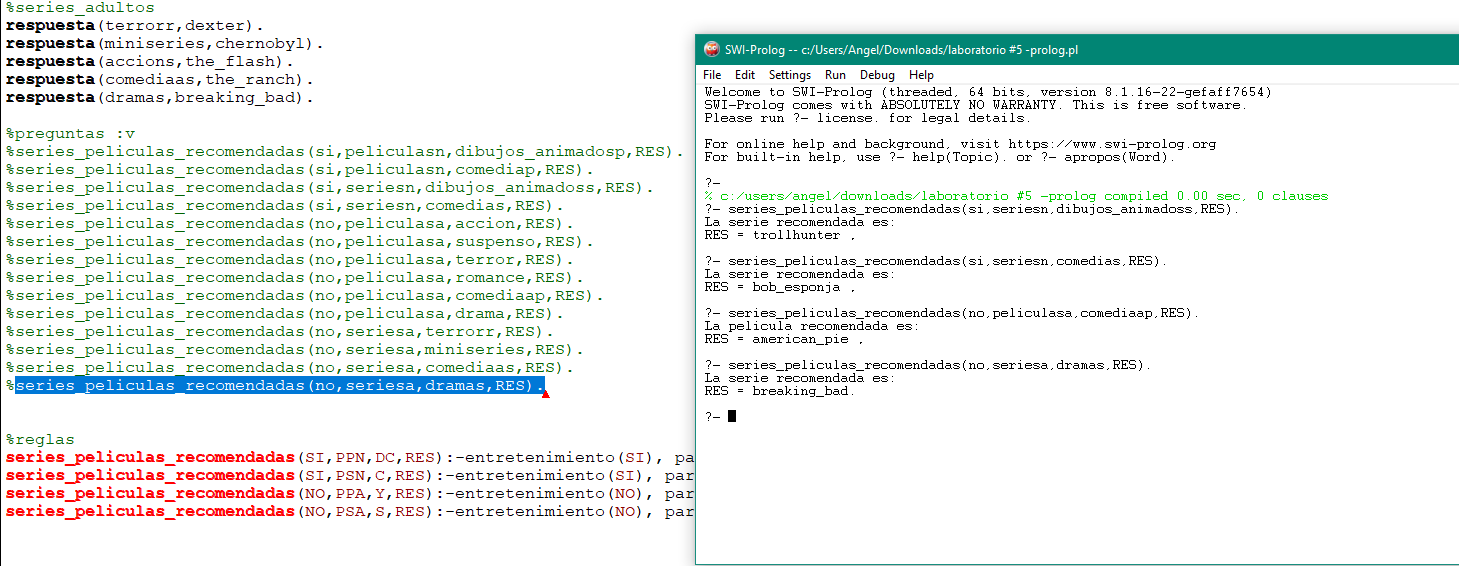
Como se puede observar en el comentario %Respuestas, nos indica que es una variable que va a captar todos los movimientos que haga el usuario del sistema experto. Para simplificarlo, todas las variables que comienzan con para\_... indican que es el penúltimo nivel del árbol, y todo lo que esté a la derecha de la “,” indicará la última decisión que el usuario tomará, es por eso que la variable respuesta tiene como parámetro, en todos los casos, la última decisión que tomará el usuario, teniendo entonces en el lado derecho de la “,” la respuesta que el sistema imprimirá en pantalla.

Veamos a continuación las reglas para que estos hechos empiecen a funcionar:

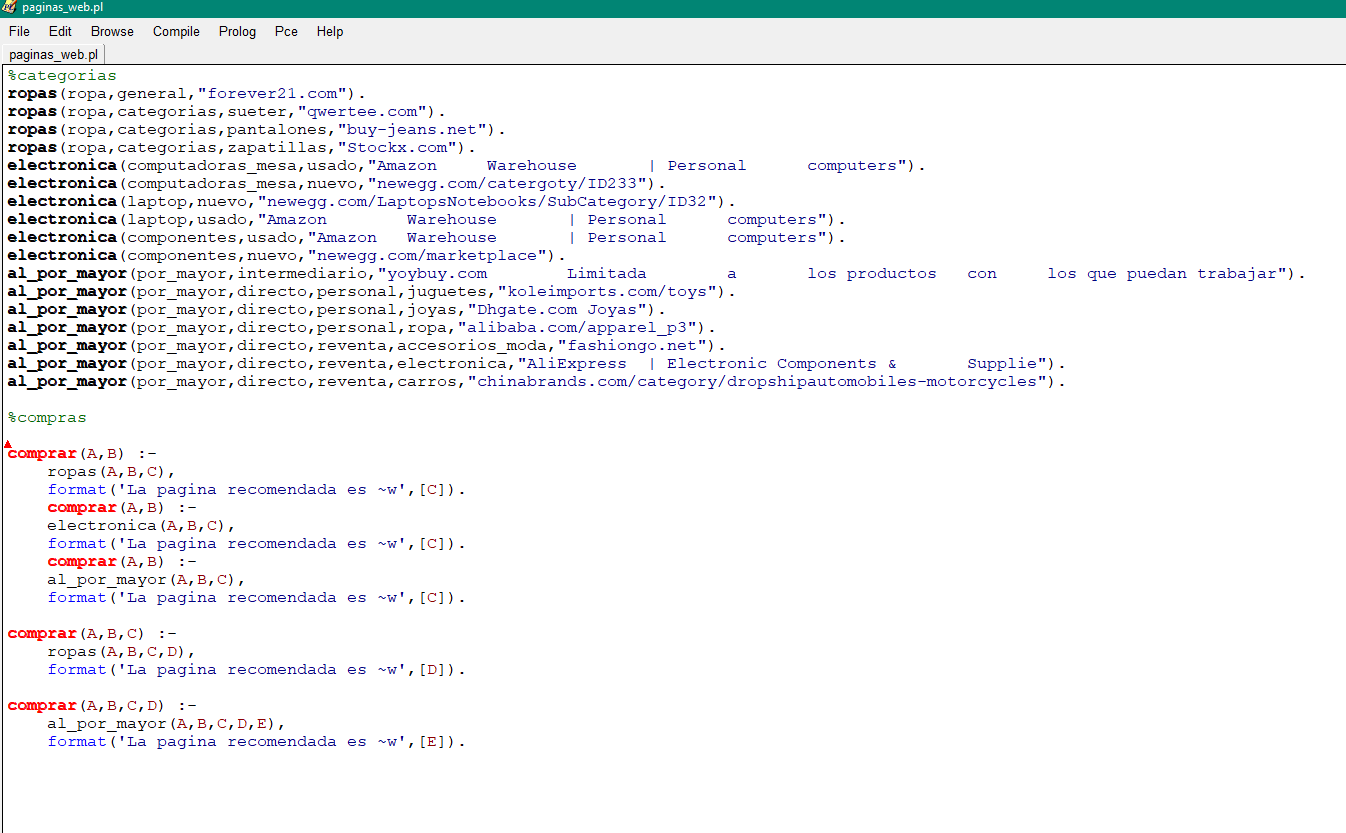
Todo lo escrito con el símbolo de porcentaje %, indicará un comentario, estos comentarios están puestos ahí para mayor facilidad a la hora de consultar datos al sistema experto, por simple comodidad del desarrollador del código.

Las reglas están planteadas de la siguiente manera: se crea el nombre de la primera regla y se le asignarán parámetros (variables cualesquiera), tantas como se vayan a utilizar, es decir, tantas como sean necesarios con respecto a los caminos que hay que tomar para llegar a la respuesta. Esto se continúa con un :- que jugará el papel principal en la cláusula que estamos declarando. Se colocan los hechos necesarios para llegar a la respuesta, estos seguidos por la variable a utilizar (deben ser las mismas puestas al principio de la regla, sólo que irán de forma secuencial).

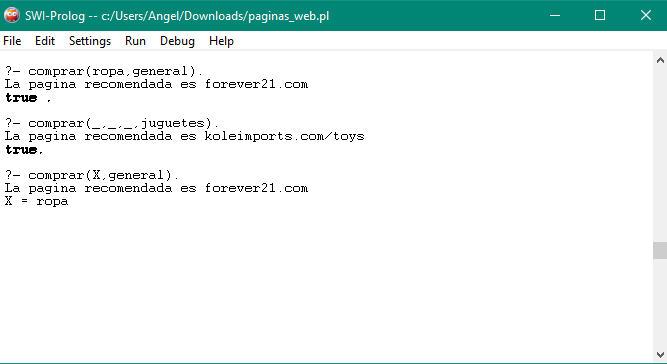
Al terminar la secuencia, se escribirá el comando write para realizar una impresión y se colocará el último hecho, que en nuestro caso será respuesta, con las dos últimas variables de la regla. Este proceso es repetitivo con las demás reglas, como en este ejemplo se trataba de un árbol con 4 niveles, serán necesarias 4 reglas para poder culminar el sistema experto.

En la sección de consulta se utilizarán consultas preparadas, como dijimos anteriormente, el desarrollador del sistema en este caso las preparó para facilitar el uso del sistema, de modo que lo único que hay que hacer es reciclarlas y ejecutarlas en la consola de consultas, se puede ver que para cada consulta el sistema entrega un resultado distinto, todo está organizado de esa forma con ayuda de los hechos que declaramos y las reglas que utilizamos para que estos funcionaran.

**Ejemplo de un sistema experto para recomendar páginas web de compras por internet:**



En este ejemplo de problema resuelto, nos encontramos con una sintaxis parecida a la del primer ejemplo, pero sólo en lo general, como, por ejemplo, los hechos se tienen que declarar con un orden secuencial que evite inconsistencias que provoquen errores.



Las consultas se realizan a través de la sintaxis declarada en las reglas y los hechos del sistema experto.

**Conclusiones**

**PROLOG** resulta ser una herramienta poderosa que tiene una gran variedad de utilidades en el ámbito de programación. A diferencia de algunos otros lenguajes de programación clásica, que dividen el problema en áreas y procedimientos, y que lo detallan de forma explícita paso a paso con una serie de instrucciones a seguir en orden lógico, **PROLOG** permite describir estructuras de datos, los objetivos y las relaciones por el simple hecho de ser un lenguaje declarativo. Cabe destacar este lenguaje tiene una estructura de programación enfocada a dar soluciones a los problemas de las áreas de las aplicaciones de la IA, mediante estructuras de programación orientadas a la resolución de problemas de una forma no tan clásica. Con lo aprendido en PROLOG, podemos decir que, como programación basada en hechos, predicados y reglas, se ve su potencial y el alcance que llega a tener hablando en el sentido de desarrollo de soluciones de problemas, que en otros lenguajes sería muy tedioso y poco efectivo.

No todas las personas del ámbito de la informática, ya sea programadores o investigadores conocen este lenguaje, aceptamos que ni siquiera nosotros conocíamos de su existencia antes de empezar a estudiarlo en el curso, pero la verdad es que nos perdimos durante mucho tiempo de una herramienta muy poderosa que da soluciones a problemas de forma directa, no tan extensos y eficaces. Uno de los motivos de este trabajo fue que las personas conocieran el lenguaje PROLOG, y mejor aún, que supieran qué es, qué contiene, por qué utilizarlo y cómo utilizarlo.

# Referencias

Andres, J., Jonathan, V., & Ortiz, A. (n.d.). *Prolog*. Retrieved from http://ferestrepoca.github.io/paradigmas-de-programacion/proglogica/tutoriales/prolog-gh-pages/Prolog.pdf

*El Lenguaje de Programación PROLOG*. (n.d.). Retrieved from http://mural.uv.es/mijuanlo/PracticasPROLOG.pdf

PROLOG.pdf. (n.d.). Retrieved December 17, 2019, from https://www.dropbox.com/preview/0750\_HERRAMIENTAS APLICADAS I.A/Laboratorios/PROLOG/PROLOG.pdf?role=personal

Tema 1. Introduccion PROLOG.pdf. (n.d.). Retrieved December 17, 2019, from https://www.dropbox.com/preview/0750\_HERRAMIENTAS APLICADAS I.A/Laboratorios/PROLOG/Tema 1. Introduccion PROLOG.pdf?role=personal

Introducción a Prolog.pdf. (n.d.). Retrieved December 17, 2019, from https://www.dropbox.com/home/0750\_HERRAMIENTAS APLICADAS I.A/Laboratorios/PROLOG?preview=Introducción+a+Prolog.pdf

*Guía de Uso Básico de Prolog*. (n.d.). Retrieved from http://www.lcc.uma.es/~pacog/apuntes/pd/guia.pdf

ProLog - EcuRed. (n.d.). Retrieved December 17, 2019, from https://www.ecured.cu/ProLog#Caracter.C3.ADsticas\_del\_lenguaje\_ProLog

PROLOG: PROLOG. (n.d.). Retrieved December 17, 2019, from http://blogramacion5.blogspot.com/2015/09/prolog.html

Lógica, P., De, M., & Rodríguez Díaz, A. (n.d.). *UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA*. Retrieved from http://blogramacion5.blogspot.com/2015/09/prolog.html

Alonso Jiménez, J. A. (n.d.). *Introducción a la programación lógica con Prolog*. Retrieved from https://www.cs.us.es/~jalonso/publicaciones/2006-int\_prolog.pdf

fjgaspar. (n.d.). *Cap1. PROLOG*. Retrieved from http://www.unizar.es/pde/fjgaspar/cap1.pdf