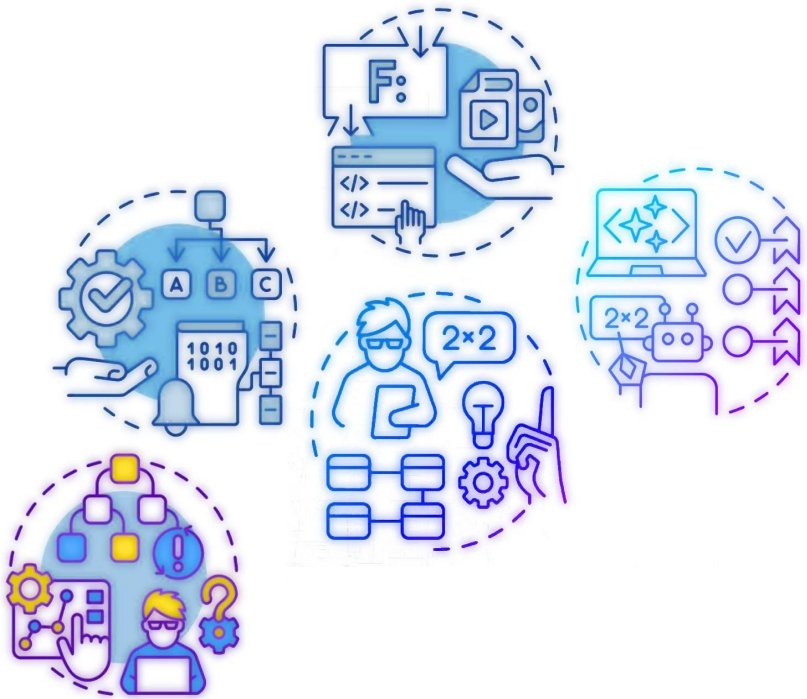


Paradigma Lógico

Esp. Ing. Viviana A. Santucci
AIA Federico Castoldi
Ing. J. Exequiel Benavidez
Ing. Jimena Bourlot

Tecnologías de la Programación

1 – ELEMENTOS DE UN PROGRAMA EN PROLOG

Un programa en Prolog está conformado por una serie de elementos:
Base de Conocimiento: representado por un conjunto de afirmaciones (hechos y reglas) representando los conocimientos que poseemos en un determinado dominio de campo de nuestra competencia.

Motor de Inferencia: es el que se encarga de la "ejecución del programa". Que en esencia es un comprobador de teoremas, el cual utiliza la Regla Inferencia.

Resolución (Intérprete de comandos): cuyo objetivo es permitir la posibilidad de responder consultas.

1 – ELEMENTOS DE UN PROGRAMA EN PROLOG

El hecho de programar en Prolog consiste en brindar a la computadora un universo Finito en forma de hechos y reglas, proporcionando los medios para realizar inferencias de un hecho a otro. A continuación si se hacen las preguntas adecuadas, Prolog buscará la respuesta en dicho universo y las presentará en pantalla.
Se puede resumir que la estructura de un programa en Prolog es:

LÓGICA + CONTROL = PROGRAMA

1 – ELEMENTOS DE UN PROGRAMA EN PROLOG

Los pasos a seguir para escribir un programa en Prolog:

a) Declarar **HECHOS** sobre los objetos y relaciones.

b) Definir **REGLAS** sobre los objetos y relaciones.

c) Hacer **PREGUNTAS** sobre los objetos y relaciones.

1 – ELEMENTOS DE UN PROGRAMA EN PROLOG

a) HECHOS: Expresan relaciones entre objetos.

Supongamos que queremos expresar el hecho de que "un coche tiene ruedas". Este hecho, consta de dos objetos, "coche" y "ruedas", y de una relación llamada "tiene".

La forma de representarlo en PROLOG es: **tiene (coche,ruedas).**

Los hechos son cláusulas definidas

1 – ELEMENTOS DE UN PROGRAMA EN PROLOG

Llamaremos cláusulas definidas, a las fórmulas con el siguiente formato (sintaxis):

$$A0 \ A1 \wedge \ldots \wedge An$$

O equivalentemente

$$A0 \vee \neg A1 \vee \ldots \vee \neg An$$

donde A0, ..., An son fórmulas atómicas, y todas las variables que ocurran en una fórmula son (implícitamente) cuantificadas en forma universal. Los hechos son cláusulas definidas en donde n = 0.

A0 es llamado cabecera de la cláusula y (A1 $\wedge$ ... $\wedge$ An) cuerpo de la misma.

1 – ELEMENTOS DE UN PROGRAMA EN PROLOG

Ejemplo

- Tomás es hijo de Juan
- Ana es hija de Tomás
- Juan es hijo de Marcos
- Alicia es hija de Juan

El nieto de una persona es el hijo de un hijo de dicha persona

1 – ELEMENTOS DE UN PROGRAMA EN PROLOG

Ejemplo:
Para la última sentencia:
Para todo X e Y, nieto(X, Y) si existe un Z tal que hijo(X, Z) e hijo(Z, Y)

Formalizando y transformando:

- $\forall X\ \forall Y$ (nieto(X, Y) $\exists Z$ (hijo(X, Z) $\wedge$ hijo(Z, Y)))
- $\forall X\ \forall Y$ (nieto(X, Y) $\vee \neg \exists Z$ (hijo(X, Z) $\wedge$ hijo(Z, Y)))
- $\forall X\ \forall Y$ (nieto(X, Y) $\vee\ \forall Z \neg$(hijo(X, Z) $\wedge$ hijo(Z, Y)))
- $\forall X\ \forall Y\ \forall Z$ (nieto(X, Y) $\vee \neg$(hijo(X, Z) $\wedge$ hijo(Z, Y)))
- $\forall X\ \forall Y\ \forall Z$ (nieto(X, Y) (hijo(X, Z) $\wedge$ hijo(Z, Y)))

1 – ELEMENTOS DE UN PROGRAMA EN PROLOG

Ejemplo:

Formalizando el ejemplo tenemos los siguiente hechos:

```
hijo(tomás, juan)
hijo(ana, tomás)
hijo(juan, marcos)
hijo(alicia, juan)
```

1 – ELEMENTOS DE UN PROGRAMA EN PROLOG

b) Reglas: Las reglas se utilizan en Prolog para significar que un hecho depende de uno o más hechos. Es la representación de las implicaciones lógicas del tipo p ---> q (p implica q).

Ejemplo: sabe(persona):-estudia(persona).

1 – ELEMENTOS DE UN PROGRAMA EN PROLOG

Algunas características son:

- Una regla consiste en una cabeza y un cuerpo, unidos por el signo " :- ".
- La cabeza está formada por un único hecho.
- El cuerpo puede ser uno o más hechos (conjunción de hechos), separados por una coma (","), que actúa como el "y" lógico.
- Las reglas finalizan con un punto (".").

1 – ELEMENTOS DE UN PROGRAMA EN PROLOG

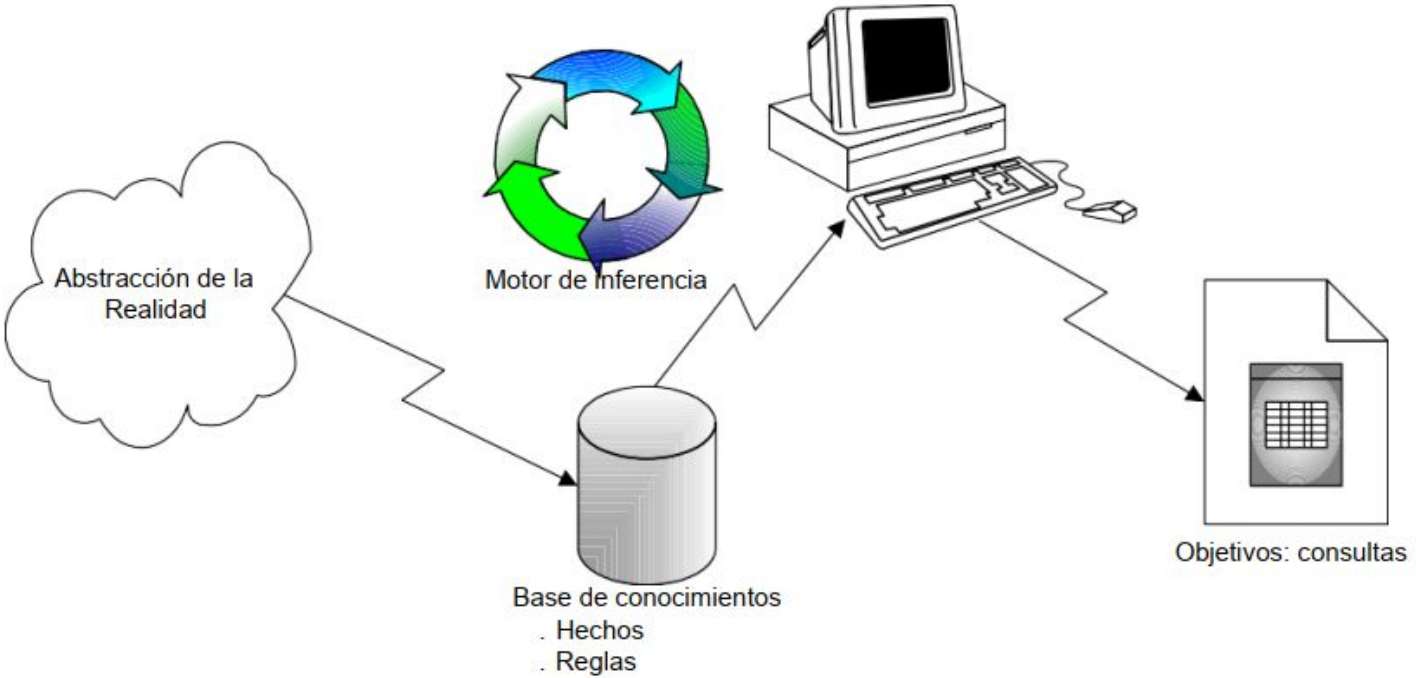
c) Preguntas o Consultas: son las herramientas que tenemos para recuperar información desde Prolog. Al hacer una pregunta a un programa lógico queremos determinar si esa pregunta es consecuencia lógica del programa.
Prolog considera que todo lo que hay en la Base de conocimiento es verdad, y lo que no, es falso. De manera que si Prolog responde “yes” es que ha podido demostrarlo, y sino,es que no lo ha podido demostrar (no debe demostrarse como“falso” sino con lo que Prolog conoce no puede demostrarse su veracidad).

1 – ELEMENTOS DE UN PROGRAMA EN PROLOG

Cuando se hace una pregunta a Prolog, éste efectuará una búsqueda por toda la Base de Conocimiento intentando encontrar hechos que coincidan con la pregunta.

En la siguiente figura se ilustra los elementos que intervienen en un programa desarrollado bajo el enfoque del paradigma lógico.

1 – ELEMENTOS DE UN PROGRAMA EN PROLOG



1 – ELEMENTOS DE UN PROGRAMA EN PROLOG

Un programa en Prolog es conformado por un conjunto de hechos y reglas que representan el problema que se pretende resolver.

Ante una determinada pregunta sobre el problema, el Prolog utilizará estos hechos y reglas para intentar demostrar la veracidad o falsedad de la pregunta que se le ha planteado

1 – ELEMENTOS DE UN PROGRAMA EN PROLOG

La forma en la que el usuario indagará al programa es por medio de cláusulas, las cuales en principio van a tener el siguiente formato:

$\forall(\neg(A1 \wedge ... \wedge Am))$

Estas fórmulas son denominadas objetivos definidos, y usualmente se escriben como:

A1, ..., Am

En donde las Ai son fórmulas atómicas denominadas subobjetivos

1 – ELEMENTOS DE UN PROGRAMA EN PROLOG

Ejemplos de consultas al programa y las correspondientes definición de Objetivos

Es Ana hija de Tomás: hijo(ana, tomás)

Como no hay variables se obtiene simplemente “Si”
Quien es nieto de Ana: nieto(X, ana)
Como el programa no contiene información acerca de nietos de Ana, la respuesta es “Nadie”

2 – PRINCIPIO DE RESOLUCIÓN O REGLA DE INFERENCIA

Es un algoritmo que, a partir de la negación de la pregunta y los hechos y reglas del programa, intenta llegar al absurdo para demostrar que la pregunta es cierta.

Este principio de resolución fue propuesto por Robinson, que propone una regla de inferencia a la que llama resolución, mediante la cual la demostración de un teorema puede ser llevada a cabo de manera automática.

2 – PRINCIPIO DE RESOLUCIÓN O REGLA DE INFERENCIA

En definitiva la resolución es una regla de inferencia que permite a la computadora decir qué proposiciones siguen lógicamente a otras proposiciones. Prolog utiliza este principio de resolución y trabaja con cláusulas de Horn. Utiliza la unificación para intentar identificar las partes derecha e izquierda de las cláusulas de una forma lógica, investigando los valores de la variable que permitirán una identificación correcta.

A partir de la Regla de Resolución puede concebirse un sistema de demostración automática que evalúe los programas en lógica,mediante la aplicación reiterada de la Regla de Resolución.

La implementación de la Regla de Inferencia en Prolog se basa en los conceptos de Unificación y Backtracking.

2 – PRINCIPIO DE RESOLUCIÓN O REGLA DE INFERENCIA

Cláusulas de Horn:

Son un tipo específico de cláusulas lógicas que juegan un papel fundamental en la programación lógica y en sistemas de inferencia.

Son disyunciones de literales donde a lo sumo uno de los literales es positivo.

2 – PRINCIPIO DE RESOLUCIÓN O REGLA DE INFERENCIA

Una cláusula de Horn puede tener una de las siguientes formas:

- Un literal positivo: Por ejemplo, P. Esto se interpreta como "P es verdadero".
- Literales negativos y a lo sumo un literal positivo: Por ejemplo, Q, R => P (que es equivalente a ¬Q ∨ ¬R ∨ P). Esto se lee como "si Q y R son verdaderos, entonces P es verdadero". A la parte Q, R se la llama el cuerpo y a P la cabeza de la cláusula.
- Solo literales negativos: Por ejemplo, Q, R => false (que es equivalente a ¬Q ∨ ¬R). Esto se interpreta como "es imposible que Q y R sean verdaderos simultáneamente", o "no Q o no R".

3 – LA UNIFICACIÓN

Mecanismo mediante el cual las variables lógicas toman valor en Prolog.
Cuando una variable no tiene valor se dice que está libre.Pero una vez que se le asigna valor, éste ya no cambia, por eso se dice que la variable está ligada.

Ejemplo:
?-docente(X).
docente(‘alejandra’) el valor Alejandra se liga a la variable X
nota(X).
nota(7) el valor 7 se liga a la variable X

3 – LA UNIFICACIÓN

Se dice que dos términos unifican cuando existe una posible ligadura (asignación de valor) de las variables,tal que ambos términos son idénticos sustituyendo las variables por dichos valores.

Por ejemplo: a(X,3) y a(4,Z) unifican dando valores a las variables: X vale 4, Z vale 3. Obsérvese que las variables de ambos términos entran en juego.

3 – LA UNIFICACIÓN

La unificación no debe confundirse con la asignación de los lenguajes imperativos puesto que representa la igualdad lógica.

Muchas veces unificamos variables con términos directamente y de manera explícita.

3 – LA UNIFICACIÓN

Para saber si dos términos unifican podemos aplicar las siguientes normas:

- Una variable siempre unifica con un término, quedando ésta ligada a dicho término.
  Ejemplo:

```
alumno_regular(X):-alumno(X),
        aprobo_primer_parcial(X),
        aprobó_segundo_parcial(X).
?-alumno_regular (juan).
```

3 – LA UNIFICACIÓN

- Dos variables siempre unifican entre sí, además, cuando una de ellas se liga a un término, todas las que unifican se ligan a dicho término.
- Para que dos términos unifiquen, deben tener el mismo functor y la misma aridad. Después se comprueba que los argumentos unifican uno a uno manteniendo las ligaduras que se produzcan en cada uno.

Ejemplo:

enseña(alejandra,juan).

?-enseña(Quien,X).

Donde: Quien se unifica con el valor alejandra, y X con juan.

3 – LA UNIFICACIÓN

Si algún término no unifica, ninguna variable queda ligada.
Ejemplo: Definición de hechos y reglas:

```
alumno (jose).
alumno (ana).
alumno (juan).
aprobo_primer_parcial(jose).
aprobo_primer_parcial(juan).
aprobo_primer_parcial(ana).
aprobo_segundo_parcial(jose).
aprobo_segundo_parcial(ana).

alumno_regular(X):-alumno(X),aprobo_primer_parcial(X), aprobo_segundo_parcial(X)
```

3 – LA UNIFICACIÓN

Consultas:

?- alumno_regular(jose).
true

?- alumno_regular(juan).
false

?- alumno_regular(X).
X = jose
X = ana
2 soluciones.

4 – BÚSQUEDA DE SOLUCIONES

Una llamada concreta a un predicado, con unos argumentos concretos, se denomina objetivo (en inglés, goal). Todos los objetivos tienen un resultado de éxito o fallo tras su ejecución indicando si el predicado es cierto para los argumentos dados, o por el contrario, es falso.

Cuando un objetivo tiene éxito las variables libres que aparecen en los argumentos pueden quedar ligadas. Estos son los valores que hacen cierto el predicado. Si el predicado falla, no ocurren ligaduras en las variables.

4 – BÚSQUEDA DE SOLUCIONES

Ejemplo: Se definen los siguientes hechos:

```
docente(ana).
docente(juan).
docente(pedro).
```

El caso más básico es aquél que no contiene variables:

```
?- docente(ana).
Verdadero
?- docente(maria).
Falso
```

4 – BÚSQUEDA DE SOLUCIONES

Si utilizamos una variable libre:

?- docente (X).
es posible que existan varios valores para dicha variable que hacen cierto el objetivo.
X =ana
X = juan
X = pedro

En este caso obtenemos todas las combinaciones de ligaduras para las variables que hacen cierto el objetivo.

4 – BÚSQUEDA DE SOLUCIONES

Las secuencias de objetivos o consultas tienen las siguientes características:

- Los objetivos se ejecutan secuencialmente por orden de escritura (es decir, de izquierda a derecha).
- Si un objetivo falla, los siguientes objetivos ya no se ejecutan.Además la conjunción,en total, falla.
- Si un objetivo tiene éxito,algunas o todas sus variables quedan ligadas,y por lo tanto,dejan de ser variables libres para el resto de objetivos en la secuencia.
- Si todos los objetivos tienen éxito, la conjunción tiene éxito y mantiene las ligaduras de los objetivos que la componen.

4 – BÚSQUEDA DE SOLUCIONES

- Partiendo de un objetivo a probar se busca las aserciones que pueden probar el objetivo. Este proceso de búsqueda de soluciones,se basa en dos conceptos: la **Unificación** y el **Backtracking**.

- En el proceso de Unificación cada objetivo determina un subconjunto de cláusulas susceptibles de ser ejecutadas (puntos de elección). Prolog selecciona el primer punto de elección y sigue ejecutando el programa hasta determinar si el objetivo es verdadero o falso.

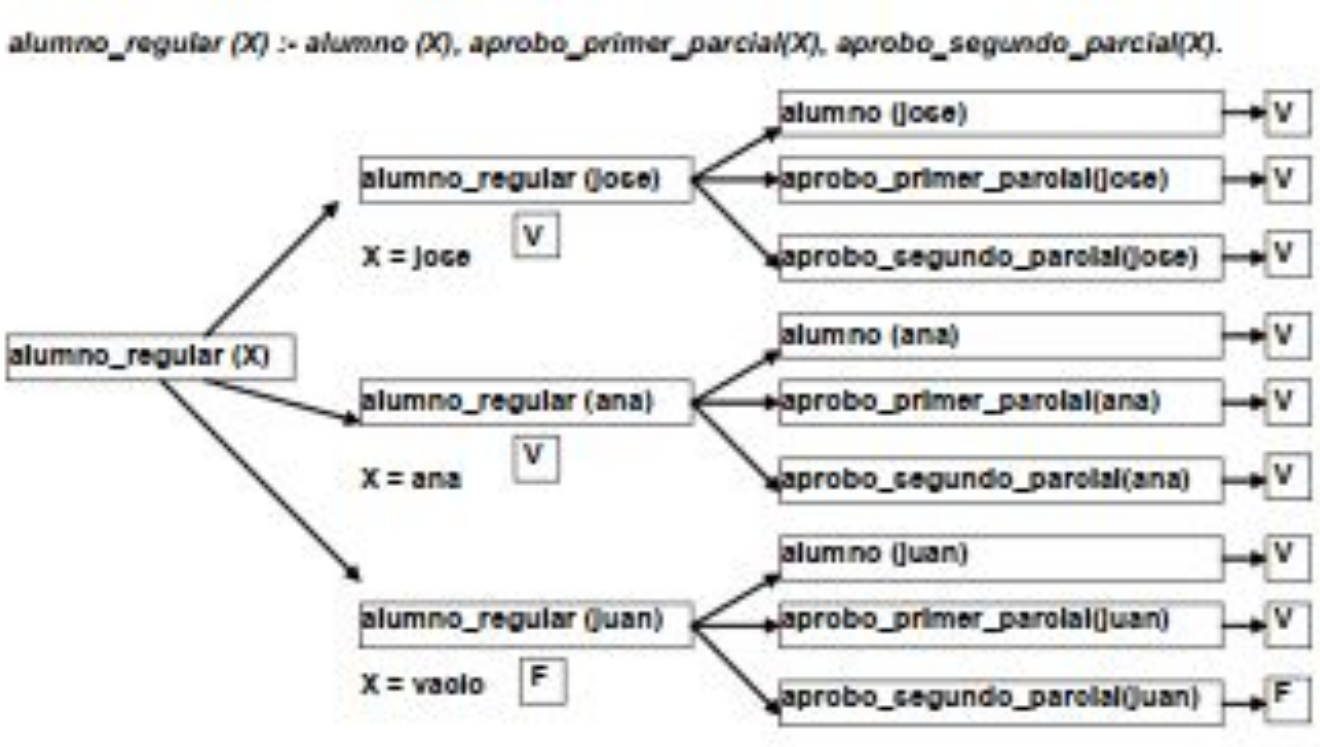
5 – BACKTRACKING

Es una técnica general que consiste en recorrer sistemáticamente todos los caminos posibles. Cuando un camino no conduce a la solución, se retrocede al paso anterior para buscar un nuevo camino.

El algoritmo de backtracking es utilizado para la resolución de diversos problemáticas, Prolog lo incorpora para su mecanismo de búsqueda de soluciones. En el caso de existir una solución seguro la encuentra, el problema es el tiempo de procesamiento.

Las etapas por las que pasa el algoritmo se pueden expresar mediante un árbol de expansión (implícito en el algoritmo)

5 – BACKTRACKING



5 – BACKTRACKING

Las características generales:

- Cada solución es el resultado de una secuencia de decisiones.
- Las decisiones pueden deshacerse ya sea porque no lleven a una solución o porque se quieran explorar todas las soluciones (para obtener la solución óptima)
- Existe una función objetivo que debe ser satisfecha u optimizada por cada selección
- Las etapas por las que pasa el algoritmo se pueden representar mediante un árbol de
- expansión.
- El árbol de expansión no se construye realmente, sino que está implícito en la ejecución del algoritmo.
- Cada nivel del árbol representa una etapa de la secuencia de decisiones.

5 – BACKTRACKING

- Funcionamiento :
- Cuando se va ejecutar un objetivo, Prolog sabe de antemano cuántas soluciones alternativas puede tener.
- Cada una de las alternativas se denomina punto de elección.Dichos puntos de elección se anotan internamente y de forma ordenada. Para ser exactos, se Introducen en una pila.
- Se escoge el primer punto de elección e ejecuta el objetivo eliminando punto de elección en el proceso.
- Si el objetivo tiene éxito se continúa con el siguiente objetivo aplicando estas mismas normas.
- Si el objetivo falla, Prolog da marcha atrás recorriendo los objetivos que anteriormente sí tuvieron éxito (en orden inverso) y deshaciendo las ligaduras de sus variables. Es decir, comienza el backtracking.

5 – BACKTRACKING

- Cuando uno de esos objetivos tiene un punto de elección anotado, se detiene el backtracking y se ejecuta de nuevo dicho objetivo usando la solución alternativa. Las variables se ligan a la nueva solución y la ejecución continúa de nuevo hacia adelante. El punto de elección se elimina en el proceso.
- El proceso se repite mientras haya objetivos y puntos de elección anotados. De hecho, se puede decir que un programa Prolog ha terminado su ejecución cuando no le quedan puntos de elección anotados ni objetivos por ejecutar en la secuencia.

El Backtracking se puede controlar mediante el uso de dos predicados:

El corte (!)

El fail

5 – BACKTRACKING

Ejemplo:

```
% Hechos
es_estudiante(maria).
es_estudiante(pedro).
es_estudiante(ana).

es_deportista(pedro).
es_deportista(juan).
es_deportista(maria).
```

```
% Regla
persona_activa(X) :-
    es_estudiante(X),    % Condición 1: X es estudiante
    es_deportista(X).    % Condición 2: X es deportista
```

```
?- persona_activa(Alguien).
```

6 – PREDICADO DE CORTE

Es un predicado predefinido que no recibe argumentos. Se representa mediante un signo de admiración (!). Es un predicado que siempre se cumple, que genera un resultado verdadero en la primera ejecución, y falla en el proceso de backtracking, impidiendo dicho retroceso.

El corte tiene la propiedad de eliminar los puntos de elección del predicado que lo contiene. Es decir, cuando se ejecuta el corte, el resultado del objetivo (no sólo la cláusula en cuestión) queda comprometido al éxito o fallo de objetivos que aparecen a continuación. Es como si a Prolog "se le olvidase" que dicho objetivo puede tener varias soluciones.

6 – PREDICADO DE CORTE

Otra forma de ver el efecto del corte es pensar que solamente tiene la propiedad de detener el backtracking cuando éste se produce. Es decir, en la ejecución normal el corte no hace nada. Pero cuando el programa entra en backtracking y los objetivos se recorren marcha atrás,al llegar corte el backtracking se detiene repentinamente forzando el fallo del objetivo.

El corte se utiliza muy frecuentemente, cuanto más diestro es el programador más lo suele usar. Los motivos por los que se usa el corte son,por orden de importancia,son los siguientes:

6 – PREDICADO DE CORTE

- Para optimizar la ejecución. El corte sirve para evitar que por culpa del backtracking se exploren puntos de elección que, con toda seguridad, no llevan a otra solución (fallan). Para los entendidos, esto es podar el árbol de búsqueda de posibles soluciones.
- Para facilitar la legibilidad y comprensión del algoritmo que está siendo programado. A veces se sitúan cortes en puntos donde, con toda seguridad, no van a existir puntos de elección para eliminar,pero ayuda a entender que la ejecución sólo depende de la cláusula en cuestión.
- Para implementar algoritmos diferentes según la combinación de argumentos de entrada. Algo similar al comportamiento de las sentencias case en los lenguajes imperativos.
- Para conseguir que un predicado solamente tenga una solución. Esto nos puede interesar en algún momento. Una vez que el programa encuentra una solución ejecutamos un corte. Así evitamos que Prolog busque otras soluciones aunque sabemos que éstas existen.

7 – PREDICADO DE FALLO

Es un predicado predefinido, sin argumentos que siempre falla, por lo tanto, implica la realización del proceso de retroceso (backtracking) para que se generen nuevas soluciones.

Cuando la máquina Prolog encuentra una solución se detiene y devuelve el resultado de la ejecución. Con fail podemos forzar a que no se detenga y siga construyendo el árbol de búsqueda hasta que no queden más soluciones que mostrar.

8 – RECURSIVIDAD

Se basa en definir relaciones en términos de ellas mismas.
Si del lado derecho de una cláusula aparece en algún punto el mismo predicado que figura del lado izquierdo, se dice que la cláusula tiene llamado recursivo, es decir “se llama a sí misma” para Verificar que se cumple esa misma propiedad como parte de la condición que define a la regla,y sobre algún posible valor de sus variables.

En una definición recursiva, es necesario considerar dos casos:

- Caso Básico: Momento en que se detiene el proceso o cómputo.
- Caso Recursivo: Suponiendo que ya se ha solucionado un caso más simple, se descompone el caso actual hasta llegar al caso más simple.

8 – RECURSIVIDAD

Por ejemplo, las siguientes reglas y hechos, expresan la idea de que se puede hablar de alguien si conocemos a ese alguien o si conocemos a alguien que nos habla de él.

regla1: habla_de(Uno,Otro) :- conoce(Uno,Otro),!.
regla2: habla_de(Uno,Otro) :- conoce(Uno,AlguienMas) , habla_de(AlguienMas,Otro).

hecho1: conoce(juan,maria).
hecho2: conoce(maria,jose).
hecho3: conoce(maria,ana).
hecho4: conoce(pedro,juan).

Y supongamos que queremos consultar lo siguiente: **?- habla_de(X,Y).**

8 – RECURSIVIDAD

¿Cuáles serán las respuestas que obtendremos?
X=juan, Y=maria;
X=maria, Y=jose;
X=maria, Y=ana;
X=pedro, Y=juan;

Estas primeras respuestas tienen que ver con el orden que siguen los hechos en nuestro programa lógico. Pero cuando a la última respuesta le agregamos “;”, gracias al backtracking intentará buscar más por alguna rama aún no explorada.

Nótese que las respuestas dadas hasta ahora, se deducen de aplicar la regla1 y luego la regla3; para la segunda respuesta, se aplicaron las reglas 1 y 4; para la tercera respuesta se aplicaron las reglas 1 y 5; y para la última respuesta se aplicaron las reglas 1 y 6.

8 – RECURSIVIDAD

Hasta acá no se usó nunca la regla2.Recién para dar la siguiente respuesta,aplica la regla 2 (dado que ya no tiene más combinaciones posibles de regla1 con algún hecho) y combinándola con, por Ej., la regla3, luego la regla1 y luego la regla4 obtenemos:

X=juan, Y=jose;

El resto de las respuestas serían:
X=juan,Y=ana;
X=pedro,Y=maria;

Pero aún podemos utilizar Prolog en su máxima potencia si trabajamos con estructuras infinitas o potencialmente infinitas, como podrían ser las listas. El poder de los programas lógicos está en su natural habilidad de manipular tipos de datos recursivos.