Tema 6. AREAS de APLICACIÓN

6.1. Bases de Datos

6.2. Sistemas Expertos

6.3. Lenguaje Natural

TEMA 6. AREAS de APLICACIÓN

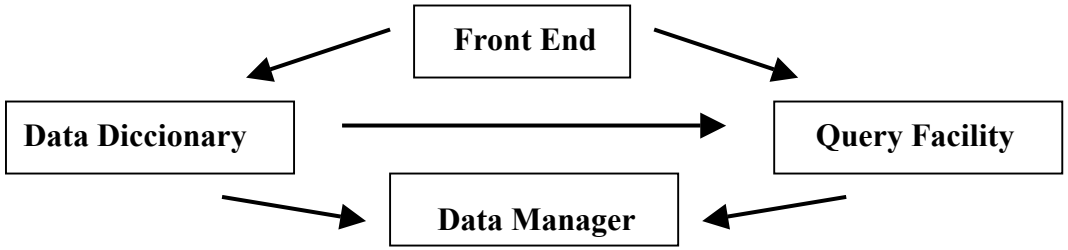
En este tema se presentan 3 áreas clásicas de aplicación de la Programación Lógica.

6.1 Bases de Datos

Los sistemas de gestión de bases de datos relacionales se caracterizan porque almacenan los datos en "tablas" o "relaciones" y por las operaciones de tipo algebraico que se realizan sobre dichas tablas. Esta información se puede representar de forma adecuada en un lenguaje lógico (Prolog). Se verá en concreto las similitudes entre las preguntas Prolog y las que se hacen en las B.D.R.

A) Componentes de un S.G.B.D.R.:

Los 4 componentes del sistema y sus relaciones vienen dados por la figura siguiente:



Front - End (interfaz con el usuario)

- Acepta la entrada del usuario comprobando si es un comando o pregunta válida.
- Pasa la entrada al diccionario, para definir nuevas tablas, o al módulo de preguntas, para acceder a la información en la B.D.
- Devuelve al usuario la respuesta a su pregunta o los errores que puedan aparecer al ejecutarla.

- De entre los posibles modelos de interfaz (sistemas gráficos, dirigidos por menus, etc), el más extendido es el SQL ("Structured Query Languaje") que define un conjunto de comandos (que representan las operaciones del álgebra relacional) con el que el usuario se comunica con la B.D.
- Se verá la relación entre una pregunta SQL y una pregunta Prolog.

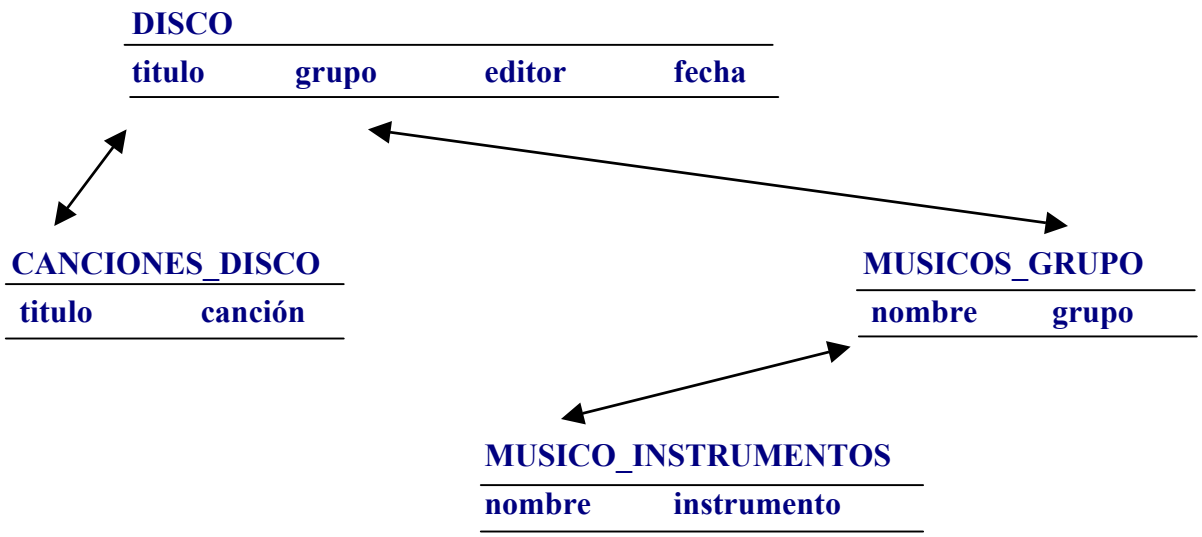
Data Diccionary (diccionario de datos)
- Define la terminología en la que el usuario puede formular las preguntas, manteniendo un catálogo de los nombres de las tablas y sus columnas. La información a añadir debe coincidir con alguna definición de tabla.
- Prolog no posee un diccionario de datos ya que el manejo de los datos es totalmente flexible: se pueden crear términos en cualquier momento sin restringirlos a una clase de información determinada. Sin embargo, el diccionario de datos se precisa para mantener el orden en la B.D.

Query facility (módulo de preguntas)
- Añade información a la B.D. o la extrae de ella.
- Una pregunta se forma con términos del diccionario.
➢ Este módulo traduce la representación de la pregunta (tipo diccionario) a una pregunta Prolog ejecutable.
- La respuesta será otra tabla (parte de ella, o combinación de varias) lógica (no necesariamente física).

Data Manager (gestor de datos)
- Se ocupa de almacenar y acceder a los datos.
- Un S.G.B.D.R. puede aprovechar la ventaja que Prolog le brinda con su B.D. interna: las tablas se almacenan como estructuras Prolog y el acceso vendrá dado por los mecanismos predefinidos del lenguaje para el manejo de datos (sin necesidad de codificar primitivas especiales).

B) Diseñando una B.D. Ejemplo: Catálogo de discos

Escribiremos un programa para catalogar los discos de una colección. La información a almacenar se puede representar de la siguiente forma:



Un SGBDR debe exigir que cada tabla tenga una "clave primaria" (columna que hace que cada fila sea única).
Prolog no exige que cada término de su B.D. sea único, por lo que no existe en Prolog mecanismos predefinidos para mantener la integridad de la BD. Estas restricciones deben ser impuestas por la aplicación.

C) El Diccionario de Datos

Mantiene la información sobre las tablas. Cada entrada es una plantilla de la tabla que se define, dando su nombre y el de sus columnas.
- En SQL esta información se añade con el comando CREATE TABLE.
- En Prolog, las definiciones de tablas pueden venir dadas como una colección de estructuras dentro de hechos Prolog:

tabla(disco(titulo, grupo, editor, fecha)).
tabla(canciones_disco(título, canción)).
tabla(musicos_grupo(nombre, grupo)).
tabla(musico_instrumentos(nombre, instrumento)).

- En SQL el diccionario puede contener información sobre tipos de datos. Prolog no es tipado pero esta información se puede añadir de otras formas:
- - En la definición de las tablas: **tabla(musico_instrumentos(nombre:atomo, instrumento:atomo)).**
- - o mediante nuevos hechos añadidos: **tipo(nombre, atomo).**
tipo(instrumento, atomo).

En ambos casos se necesita escribir procedimientos que verifiquen los tipos de datos al añadir una nueva fila a una tabla.

D) La Base de Datos

Una vez conocido cómo estructurar la información, podemos almacenarla en las tablas.

Ejemplo:

disco('Into de music', 'Van Morrison', 'Phonogram', 1979).
disco('Devocionario', 'Golpes Bajos', 'Nuevos Medios', 1985).

musicos_grupo('Van Morrison', 'Van Morrison').
musicos_grupo('Toni Marcus', 'Van Morrison').
musicos_grupo('Mark Jordan', 'Van Morrison').
musicos_grupo('Herbie Armstrong', 'Van Morrison').
musicos_grupo('German Coppini', 'Golpes Bajos').
musicos_grupo('Pablo Novoa', 'Golpes Bajos').
musicos_grupo('Luis Garcia', 'Golpes Bajos').
musicos_grupo('Teo Cardalda', 'Golpes Bajos').

```
musico_instrumentos('Van Morrison', guitarra).
musico_instrumentos('Van Morrison', armonica).
musico_instrumentos('Toni Marcus', violin).
musico_instrumentos('Toni Marcus', viola).
musico_instrumentos('Mark Jordan', teclados).
musico_instrumentos('Herbie Armstrong', guitarra).

musico_instrumentos('German Coppini', voz).
musico_instrumentos('Pablo Novoa', guitarra).
musico_instrumentos('Luis Garcia', bajo).
musico_instrumentos('Teo Cardalda', piano).
```

```
canciones_disco('Into the music', 'Briht side of the road).
canciones_disco('Into the music', 'Full force gale').
canciones_disco('Into the music', 'Stepping out queen').
canciones_disco('Into the music', 'Troubadours').

canciones_disco('Devocionario', 'Prologo').
canciones_disco('Devocionario', 'Desconocido').
canciones_disco('Devocionario', 'La virgen loca').
canciones_disco('Devocionario', 'Travesuras de Till').
canciones_disco('Devocionario', 'Santos de devocionario').
```

- En SQL se añade la información con el comando INSERT:

INSERT
INTO nombre-tabla
VALUES (columna 1, columna 2, ...)

- En Prolog se haría mediante el siguiente predicado:

```
insert(Tabla, Argumentos, Termino)  :-  Termino =.. [Tabla | Argumentos],  assertz(Termino).
```

que, dados el nombre de una tabla y una lista de argumentos, devuelve una estructura Prolog con dicha información que es añadida a la B.D.

Ejemplo:

```
? insert(canciones_disco, ['Devocionario', 'Ayes'], T).
  T = canciones_disco('Devocionario', 'Ayes')
```

E) Transformando preguntas SQL en preguntas Prolog

La información se obtiene en un sistema SQL mediante SELECT:

```
SELECT    columna(s)
FROM      tabla(s)
WHERE     predicado
```

Esta forma general se puede usar para hacer 3 tipos básicos de preguntas:

a) Preguntas que devuelven todas las filas de todas las columnas de una o más tablas.

b) Preguntas que devuelven sólo algunas filas.

c) Preguntas que devuelven sólo algunas columnas.

Para construir preguntas Prolog a partir de la representación del diccionario se usarán los predicados evaluables: **functor, arg , (=..)**

Preguntas tipo **a)** Se desea seleccionar toda la información de la tabla "musicos_grupo".

- En SQL:

```
SELECT    *
FROM      MUSICOS_GRUPO;
```

- En Prolog la pregunta equivalente es: **? musicos_grupo(X,Y).**

Para generar esta pregunta se puede definir el predicado **from,** para ser usado en modo (in,out,out), de la siguiente forma:

```
from(Nombre, Tabla, Pregunta) :-  tabla(Tabla),                  % generador (de valores para Tabla)
                                  functor(Tabla, Nombre, Arg), ! ,  % test (cuyo funtor sea Nombre)
                                  functor(Pregunta, Nombre, Arg).   % functor en modo de uso (out,in,in)
```

Ejemplo: **? from(musicos_grupo, T, P).**
T = musicos_grupo(nombre, grupo)
P = musicos_grupo(X,Y)

Preguntas tipo **b)** Se desea seleccionar sólo los músicos que tocan la guitarra.

- En SQL:

```
SELECT      *
FROM        MUSICO_INSTRUMENTOS
WHERE       INSTRUMENTO = guitarra;
```

- En Prolog, la pregunta equivalente es **? musico_instrumentos(X, guitarra).**

Para generar la pregunta se puede definir el predicado **where,** para ser usado en modo (in,in,in,out), de la siguiente forma:

1. Mediante **from** en modo (in,out,out) se obtiene la Tabla y la estructura genérica Pregunta a partir de un Nombre dado.
2. Usando **arg** en modo (out,in,in) se obtiene el lugar que ocupa el argumento Arg en la Tabla.
3. Usando finalmente **arg** en modo (in,out,in) se instancia el argumento de lugar N por Val en la estructura Pregunta.

```
where(Nombre, Arg, Val, Pregunta)  :-  from(Nombre, Tabla, Pregunta),
                                       arg(N, Tabla, Arg),
                                       arg(N, Pregunta, Val).
```

Ejemplo: **? where(musico_instrumentos, instrumento, guitarra, P).**
P = musico_instrumentos(X, guitarra).

Preguntas tipo **c)** Se desean todos los títulos de los discos y sus grupos correspondientes (sin editor ni fecha)

- En SQL:
```
SELECT   TITULO, GRUPO
FROM     DISCO;
```

- En Prolog:
```
? disco(Titulo, Grupo, _ , _)
```

Para generar esta pregunta se usa un método similar pero ahora se deben imprimir sólo las columnas de la pregunta. Definimos el predicado **select**, para ser usado en modo (in,in), de la siguiente forma:

```
select(Argvals, Nombre)  :-  from(Nombre, Tabla, Pregunta),
                             get_nums(Argvals, Tabla, Argnums),
                             get_instance(Pregunta, Argnums).
```

"get_nums(LA, T, LN)": "genera una lista LN con los lugares que ocupan los argumentos de LA en la tabla T".

```
get_nums([ ], _ , [ ]).
get_nums([C | R], Tabla, [N | AN]) :-  arg(N, Tabla, C), get_nums(R, Tabla, AN).
```

"get_instance (P, LN)" : "ejecuta la pregunta P y determina qué columnas escribir con la información de LN".

```
get_instance(P, LN)  :- call(P), get_columns(P, LN, Lista), output(Lista), fail.
get_instance( _ , _ ).

get_columns(P, [ ], [ ]).
get_columns(P, [C | R], [Val | RL]  :- arg(C, P, Val), get_columns(P, R, RL).
output([ ])  :- nl.
output([C | R]) :- write(C), tab(4), output(R).
```

Ejemplo: **? select([titulo, grupo], disco).**
Into de music Van Morrison
Devocionario Golpes Bajos

Siguiendo estas técnicas, se pueden escribir programas para traducir otras preguntas SQL a preguntas Prolog del tipo siguiente:

```
SQL:    SELECT    *
        FROM      DISCO
        WHERE     FECHA > 1978;

Prolog: ?  where(disco, (fecha, >, 1978), X).

SQL:    SELECT    *
        FROM      DISCO
        WHERE     FECHA BETWEEN 1977 AND 1980;

Prolog: ?  where(disco, (fecha, between, (1977, 1981)), X).

SQL:    SELECT    *
        FROM      MUSICOS_GRUPO
        WHERE     NOMBRE IN ['German Coppini','Pablo Novoa'];

Prolog: ? where(musicos_grupo, (nombre, in, ['German Coppini', 'Pablo Novoa']), X).
```

SQL:
```
SELECT *
FROM MUSICO
WHERE NOMBRE = 'German Coppini'
OR INSTRUMENTO = guitarra
```

Prolog:
```
? where(musico_instrumentos, (nombre, == 'German Coppini'), or, (instrumento, ==, guitarra), X).
```

Para todas estas traducciones se debe modificar el programa Prolog del predicado "where".

Otras consultas usuales que se realizan en SQL, y pueden ser traducidas a preguntas Prolog, se refieren a:
- Operaciones aritméticas sobre las tablas.
- Ordenaciones de los resultados de la tabla.
- Acceso a más de una tabla a la vez.
- Modificación de las definiciones de las tablas en el diccionario.

6.2 Sistemas Expertos

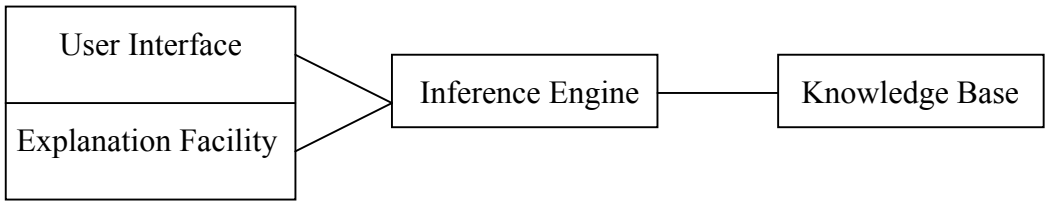
Los sistemas expertos no se comportan como otras clases de programas: en lugar de realizar una serie de tareas, deben tener un cuerpo de conocimiento y deben ser capaces de manipular dicho conocimiento para obtener conclusiones (mediante algún método de resolución de problemas). Prolog es un lenguaje ideal para construir sistemas expertos al ser descriptivo.

Algunas de las características más sobresalientes de un S.E. son:
- Tiene la capacidad de adquirir conocimiento con el tiempo, sin límite.
- Puede predecir cosas que serán ciertas basándose en el conocimiento que posee.
- Puede agrupar conocimiento de diferentes formas.
- Puede dar conclusiones con un cierto grado de certeza.
- Suele usar "heurísticas" para obtener rápido una solución (y mantener el tamaño del sistema dentro de límites razonables).
- Puede explicar cómo se ha llegado a una conclusión.

Todas estas características pueden realizarse en Prolog con su base de datos predefinida y su método para resolver problemas.

Componentes de un Sistema Experto

Los Sistemas Expertos se componen de lo siguiente:



A) Knowledge Base (La Base de Conocimiento)

Este módulo contiene el conocimiento. Hay distintos métodos para describirlo, todos ellos se pueden representar fácilmente en Prolog.

a) Redes semánticas: describen relaciones entre objetos. La relación más simple es "isa". Por ejemplo, "un perro es un animal" o "un árbol es una planta". Se traducen a hechos Prolog:

isa(perro, animal). **isa(arbol, planta).**

b) Reglas de Producción: describen el conocimiento como reglas "if-then". Por ejemplo, "Si es sábado podemos dormir hasta tarde". Se traducen a reglas Prolog:

dormir(tarde) :- hoy(sabado).

c) "Frames": son útiles cuando se necesita coleccionar diferentes partes de información antes de encontrar la solución. Por ejemplo, para elegir un restaurante se necesita decidir el tipo de comida, gasto a realizar, etc.
Como Prolog permite construir estructuras de datos arbitrarias, esta información se traduce fácilmente:

restaurante([nombre : arzak, tipo : selecto, precio : caro, lugar : san_sebastian]).

d) Taxonomías: describen el conocimiento como lo hacen los "frames" y las redes semánticas conjuntamente.
Por ejemplo, se puede tener un "frame" que describe las características de los gatos en general y otro para las características de un gato particular; y definir luego la relación entre ambos frames:

gato ([familia : felinos, genero : domestico]).
beltza ([color : negro]).
isa (beltza, gato).

La relación "isa" describe una herencia: todas las características de "gato" son también caracterísitcas de "beltza".

B) Inference Engine (Motor de Inferencia)

La información en la Base de Conocimiento es estática hasta que algo exterior hace que comience a funcionar. Este trabajo lo realiza el "motor de inferencia". Las dos formas en que puede trabajar un motor de inferencia son:

- *"encadenamiento hacia atrás"*: comenzar con la conclusión hasta encontrar evidencias que soporten dicha conclusión.
- *"encadenamiento hacia adelante"*: comenzar con la evidencia disponible y llegar a las conclusiones que dicha evidencia soporta.

El primero de estos métodos es la estrategia de solución propia de Prolog. Otras técnicas de inferencia también pueden describirse en Prolog.

C) Explanation Facility (Módulo de explicaciones)

Los Sistemas Expertos añaden información al preguntar al usuario, esta parte del sistema se llama el módulo consulta.

Además, los usuarios pueden preguntarle al S.E. cómo ha obtenido una conclusión particular o por qué el sistema le ha hecho una pregunta dada. Esta parte es el módulo de explicaciones.

Ejemplo de un Sistema Experto en Prolog

Para ilustar los conceptos anteriores se mostrará un sistema experto de juguete en Prolog (siguiendo las ideas de los metaintérpretes Prolog) donde el conocimiento se representa por Reglas de Producción y el motor de inferencia es el propio de Prolog (encadenamiento hacia atrás). Habrá que añadir las características propias del diálogo con el usuario y la generación de explicaciones.

Ejemplo: El aprendiz de hornero

Las siguientes reglas de producción nos indicarán en qué nivel N del horno se debe colocar un plato P dependiendo de su tipo y tamaño. Constituyen la Base de Conocimiento.

```
% coloca (P, N):     El plato P debe colocarse en el nivel N del horno.
coloca(P, alto)  :-  pasteleria(P), tamaño(P, pequeño).
```

```
coloca(P, medio)  :-  pasteleria(P), tamaño(P, grande).
coloca(P, medio)  :-  plato_fuerte(P).
coloca(P, bajo)   :-  coccion_lenta(P).
pasteleria(P)     :-  tipo(P, pastel).
pasteleria(P)     :-  tipo(P, pan).
plato_fuerte(P)   :-  tipo(P, carne).
coccion_lenta(P)  :-  tipo(P, postre).
```

Consultando al usuario:

El S.E. consultará al usuario sobre el tipo y tamaño de su plato. Las respuestas se guardarán como información en la B.C., de manera que el sistema las conozca para el futuro.

```
% Cáscara interactiva que interroga al usuario
resuelve(true).
resuelve((A,B))    :-  resuelve (A), resuelve(B).
resuelve(A)        :-  clause(A,B), resuelve(B).
resuelve(A)        :-  preguntable(A),  not(conocido(A)), pregunta(A, Resp), responde(Resp, A).
pregunta(A, Resp)  :-  visualiza_pregunta(A), read(Resp).
responde(si, A)    :-  assert(A).
responde(no, A)    :-  assert(falso(A)), fail.
conocido(A)  :-  A
conocido(A)  :-  falso(A).
visualiza_pregunta(A)  :-  write(A), write(' ¿(si/no)? ').
preguntable(tamaño(P,T)).
preguntable(tipo(P,T)).
```

Ejemplo de sesión:

```
? resuelve(coloca(bistec, N)).
tipo(bistec, pastel) ¿(si/no)? no.        /* añade a la B.C.: falso(tipo(bistec, pastel)). */
tipo(bistec, pan) ¿(si/no)? no.           /* añade a la B.C.: falso(tipo(bistec, pan)). */
tipo(bistec, carne) ¿(si/no)? si.         /* añade a la B.C.: tipo(bistec, carne). */
N = medio

? resuelve(coloca(croissant, N)).
tipo(croissant, pastel) ¿(si/no)? si.     /* añade a la B.C.: tipo(croissant, pastel). */
tamaño(croissant, pequeño) ¿(si/no)? si.  /* añade a la B.C.: tamaño(croissant, pequeño).*/
N = alto

? resuelve(coloca(flan, N)).
tipo(flan, pastel) ¿(si/no)? no.          /* añade a la B.C.: falso(tipo(flan, pastel)). */
tipo(flan, pan) ¿(si/no)? no.             /* añade a la B.C.: falso(tipo(flan, pan)). */
tipo(flan, carne) ¿(si/no)? no.           /* añade a la B.C.: falso(tipo(flan, carne)). */
tipo(flan, postre) ¿(si/no)? si.          /* añade a la B.C.: tipo(flan, postre). */
N = bajo

? resuelve(coloca(croissant, N)).
N = alto
```

¡Ha aprendido!

Generando explicaciones:

Se puede permitir al usuario preguntas de dos tipos:

- **¿para?** ¿para qué quieres saberlo? (Posible respuesta a una pregunta del sistema).
- **¿cómo?** ¿cómo lo has averiguado? (Posible pregunta tras una respuesta del sistema).

En el caso de las preguntas **¿para?** el sistema debe llevar cuenta de las reglas que intenta aplicar al deducir una respuesta. Añadiremos al programa anterior un argumento extra que contenga las reglas aplicadas hasta el momento.

```
% Cáscara interactiva que interroga al usuario y responde a preguntas ¿para?
resuelve(A) :- resuelve(A, [ ]).
resuelve(true, _ ) :- !.
resuelve( (A,B), Reglas ) :- !, resuelve(A, Reglas), resuelve(B, Reglas).
resuelve(A, _ ) :- falso(A), !, fail.                    /* Negación del usuario */
resuelve(A, _ ) :- cierto(A), !.                         /* Afirmación del usuario */
resuelve(A, Reglas) :- clause(A,B), resuelve(B, [regla(A,B) | Reglas]).
resuelve(A, Reglas) :- preguntable(A), pregunta(A, Resp), responde(Resp, A, Reglas).

/* Ampliamos "responde" con un argumento extra y permitimos la pregunta ¿para? */
pregunta(A, Resp) :- visualiza_pregunta(A), read(Resp).
visualiza_pregunta(A) :- write(A), write(' ¿(si/no/para)? ').
responde(si, A, _ ) :- !, assert(cierto(A) ).
responde(no, A, _ ) :- !, assert(falso(A)), fail.
responde(para, A, [Regla | Reglas]) :- !, visualiza_regla(Regla), pregunta(A, Resp), responde(Resp, A, Reglas).
responde(para, A, [ ]) :- write(' No puedo dar mas explicaciones'), nl, pregunta(A, Resp), responde(Resp, A, [ ]).
visualiza_regla(regla(A,B) ) :- write(' SI '), escribe_cuerpo(B), write(' ENTONCES '), write(A), nl.
escribe_cuerpo((A,B)) :- !, write(A), write('&'), escribe_cuerpo(B).
escribe_cuerpo(A) :- write(A), nl.
```

Ejemplo de sesión con preguntas ¿para?:

```
? resuelve(coloca(flan, N)).
   tipo(flan, pastel)  ¿(si/no/para)?      para.
    SI tipo(flan, pastel)
    ENTONCES pasteleria(flan)
   tipo(flan, pastel)  ¿(si/no/para)?      para.
    SI pasteleria(flan)  &  tamaño(flan, pequeño)
    ENTONCES coloca(flan, alto)
   tipo(flan, pastel)  ¿(si/no/para)?      para.
    No puedo dar mas explicaciones.
   tipo(flan, pastel)  ¿(si/no/para)?      no.
   tipo(flan, pan)  ¿(si/no/para)?         no.
   tipo(flan, carne)  ¿(si/no/para)?       no.
   tipo (flan, postre)  ¿(si/no/para)?     para.
    SI  tipo(flan, postre)
    ENTONCES  coccion_lenta(flan).
   tipo(flan, postre)  ¿(si/no/para)?      si.
   N = bajo.
```

En el caso de las preguntas **¿cómo?** se incluirá en el sistema un meta-intérprete que resuelve el objetivo construyendo a la vez una prueba, y un procedimiento que interpreta la prueba para generar una explicación. El predicado "como" no hace preguntas al usuario, por lo que se debe resolver antes el objetivo y después preguntar cómo se ha resuelto.

```
% Cáscara para preguntas ¿cómo?
:- op(10, xfy, '<--').
como(Objetivo)  :-  prueba(Objetivo, Prueba), interpreta(Prueba).

prueba(true, true)  :-  !.
prueba((A,B), (PA,PB))  :- !, prueba(A, PA), prueba(B, PB).
prueba(A, _ )  :-  falso(A), !, fail.
prueba(A, (A <-- usuario))  :-  cierto(A), !.
prueba(A, (A <-- Prueba))  :-  clause(A, B), prueba(B, Prueba).

interpreta( (P1,P2) )  :-  !, interpreta(P1), interpreta(P2).
interpreta(Prueba)  :-  usuario(Prueba, Hecho), !, write(Hecho), write(' es una afirmación del usuario '), nl.
interpreta(Prueba)  :- hecho(Prueba, Hecho), !, write(Hecho), write(' es un hecho de la base de conocimiento \n').
interpreta(Prueba)  :- regla(Prueba, Cabeza, Cuerpo, Prueba_cuerpo), write(Cabeza),  write(' se demuestra usando la regla: '),
                       nl, visualiza_regla(regla(Cabeza, Cuerpo)),  interpreta(Prueba_cuerpo).

usuario((Hecho <-- usuario), Hecho).
hecho((Hecho <-- true), Hecho).
regla((Cabeza<-- Prueba_cuerpo), Cabeza, Cuerpo, Prueba_cuerpo) :- probado_por(Prueba_cuerpo, Cuerpo).
probado_por((PA, PB), (A, B)) :- !, probado_por(PA, A), probado_por(PB, B).
probado_por((A <-- _ ), A).
```

Ejemplo de sesión con preguntas ¿cómo?:

?- resuelve(coloca(pizza,K)).

tipo(pizza, pastel) ¿(si/no/para)? no.

tipo(pizza, pan) ¿(si/no/para)? si.

tamaño(pizza, pequeño) ¿(si/no/para)? no.

tamaño(pizza, grande) ¿(si/no/para)? si.

K = medio

?- como(coloca(pizza,medio)).

coloca(pizza, medio) se demuestra usando la regla:

SI pasteleria(pizza) & tamaño(pizza, grande)

ENTONCES coloca(pizza, medio)

pasteleria(pizza) se demuestra usando la regla:

SI tipo(pizza, pan)

ENTONCES pasteleria(pizza)

tipo(pizza, pan) es una afirmación del usuario

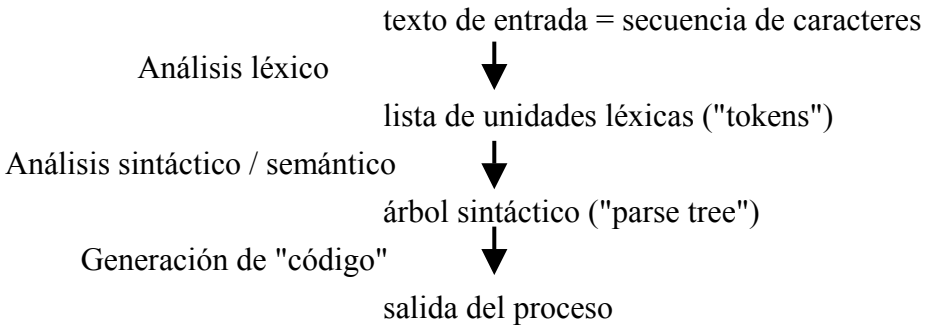
tamaño(pizza, grande) es una afirmación del usuario

6.3 Lenguaje Natural

El procesamiento del lenguaje tiene una amplia gama de aplicaciones:

- intérpretes y compiladores
- interfaz con bases de datos y/o con sistemas expertos
- traducción automática del lenguaje natural

Las principales fases del procesamiento son:



Nos centraremos aquí en el análisis sintáctico que a partir de una lista de unidades, construye la estructura sintáctica de la "frase" (en forma de árbol), según sea la gramática del lenguaje.

Gramáticas independientes del contexto (G.I.C.)

- Las frases del lenguaje son secuencias finitas de unidades léxicas ("tokens" o palabras) que vendrán representadas como listas de átomos Prolog.
- Una gramática es un sistema de reglas que define las frases del lenguaje.

- Las reglas de una G.I.C. son de la forma: **<cabeza> --> <cuerpo>**
  donde <cabeza> es un símbolo no-terminal y <cuerpo> es una serie de símbolos terminales y/o no-terminales separados por comas.
- Los símbolos no-terminales (escritos como átomos Prolog) representan categorías sintácticas.
- Los símbolos terminales (escritos como listas de átomos Prolog) representan (partes de) frases concretas.
- El significado de una regla es que una frase correspondiente a la categoría sintáctica de la cabeza puede tener la forma indicada por el cuerpo.

Ejemplo: Gramática independiente del contexto para un fragmento del castellano.

Reglas de la gramática:

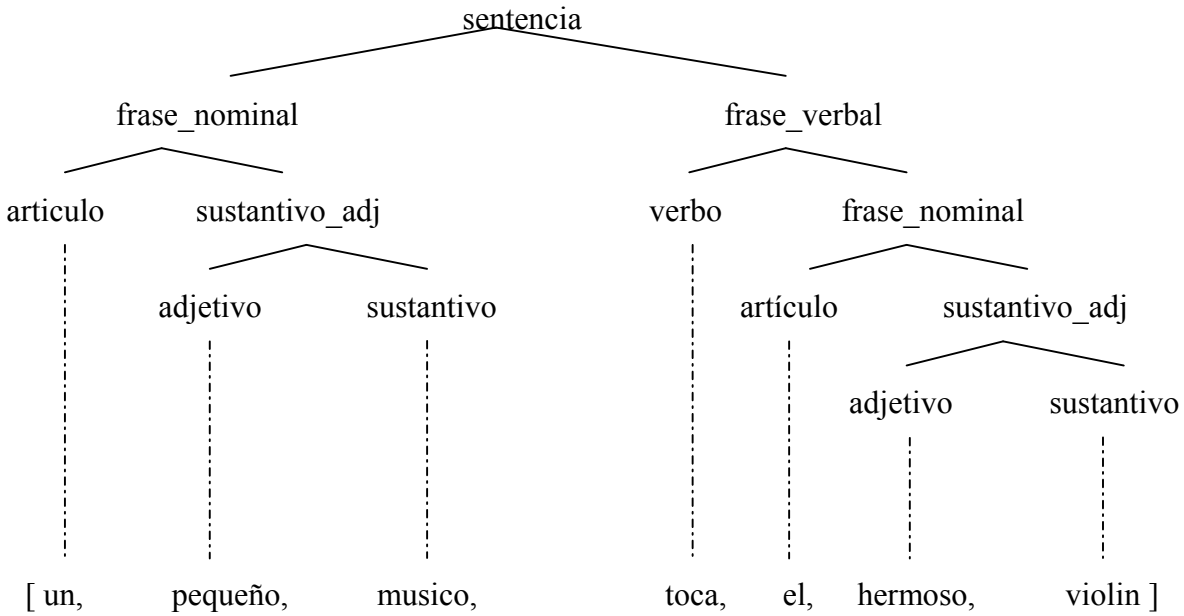
```
sentencia   -->   frase_nominal, frase_verbal.
frase_nominal --> articulo, sustantivo_adj.
frase_nominal --> sustantivo_adj.
sustantivo_adj --> sustantivo.
sustantivo_adj --> adjetivo, sustantivo.
frase_verbal --> verbo.
frase_verbal --> verbo, frase_nominal.
```

Reglas del vocabulario:

```
articulo --> [el].          adjetivo --> [pequeño].     sustantivo --> [musico].      verbo --> [toca].
articulo --> [la].          adjetivo --> [pequeña].     sustantivo --> [bailarina].   verbo --> [baila].
articulo --> [un].          adjetivo --> [hermoso].     sustantivo --> [violin].
articulo --> [una].         adjetivo --> [hermosa].     sustantivo --> [danza].
```

Una frase pertenece a la categoría sintáctica del símbolo no-terminal S si puede generarse a partir de S por aplicación sucesiva de las reglas. Si la generación es posible, la frase tendrá un árbol de análisis.

Ejemplo:



Otras sentencias de este lenguaje son:

[la, hermosa, bailarina, baila, una, danza]
[el, violin, toca, un musico]
[una, hermoso, musico, baila, un, danza]

Como se ve hay problemas de concordancia de géneros y de significado.

Representación en Prolog de una G.I.C.

Para representar en Prolog las reglas de una G.I.C. se puede asociar a cada símbolo no-terminal un predicado Prolog que reconozca las frases de la correspondiente categoría sintáctica.
Se convierten entonces las reglas de la G.I.C. en cláusulas Prolog.

Ejemplo:

"sentencia(X)" : la frase X es una sentencia
"articulo([el])" : "el" es un artículo.

La forma más directa de hacerlo sería:

```
sentencia(X) :- concatenar(Y,Z,X), frase_nominal(Y), frase_verbal(Z).
articulo([el]).
```

Esta forma es demasiado ineficiente, hay muchos intentos fallidos de descomponer X en la forma adecuada. Para mejorarlo, se puede hacer uso de la idea de *listas-diferencia*:

"sentencia(X,R)" : X-R es una sentencia.
"articulo(X,R)" : X-R es un artículo.

De esta manera las reglas anteriores de la gramatica se pueden escribir en Prolog:

```
sentencia(S,R)       :- frase_nominal(S,Z), frase_verbal(Z,R).
frase_nominal(S,R)   :- articulo(S,U), sustantivo_adj(U,R).
frase_nominal(S,R)   :- sustantivo_adj(S,R).
sustantivo_adj(S,R)  :- sustantivo(S,R).
sustantivo_adj(S,R)  :- adjetivo(S,U), sustantivo(U,R).
frase_verbal(S,R)    :- verbo(S,R).
frase_verbal(S,R)    :- verbo(S,U), frase_nominal(U,R).
```

Las reglas del vocabulario quedarían traducidas a hechos Prolog:

```
articulo([el | R], R).
articulo([la | R], R).
-  -  -  -  -  -  -
verbo([baila | R], R).
```

Ahora se pueden hacer preguntas para analizar o generar sentencias.

Ejemplos de preguntas:

? sentencia([un, pequeño, musico, toca, el, hermoso, violin], [ ]).
si
? sentencia(S, [ ]).
(genera todas las sentencias del lenguaje)
? frase_nominal([un, pequeño, musico, toca, el, violin], R).
R = [toca, el violin]

Nota: Algunos sistemas Prolog permiten escribir las reglas de la gramática directamente (como términos construidos con el operador -->) y las traducen automáticamente a cláusulas de la forma "sentencia(S,R)" como las vistas.

Ejemplos de reglas traducidas:

1)	p --> q, [a,b], r.	se traduce a	p(X,Y) :- q(X, [a,b | Z]), r(Z,Y).
2)	q --> r, p, [a,b], q, [a,b], p.	se traduce a	q(X,Y) :- r(X,Z), p(Z, [a,b | U]), q(U, [a,b | T]), p(T, Y).
3)	r --> p, [b], [b, a], q.	se traduce a	r(X,Y) :- p(X, [b,b,a | Z]), q(Z,Y).
4)	p --> [b,c].	se traduce a	p([b, c | R], R).

Argumentos adicionales

Se pueden añadir argumentos adicionales a los símbolos no-terminales de las reglas, que se conservan al traducir las reglas a cláusulas. De esta forma, el programa Prolog que resulta es más potente que una G.I.C.

Ejemplo: **Concordancia de género y número en el lenguaje anterior**

```
sentencia(G,N)  -->  frase_nominal(G,N), frase_verbal(N).
frase_nominal(G,N) -->   articulo(G,N), sustantivo_adj(G,N).
frase_nominal(G,N) -->   sustantivo_adj(G,N).
sustantivo_adj(G,N) -->   sustantivo(G,N).
sustantivo_adj(G,N) -->   adjetivo(G,N), sustantivo(G,N).
frase_verbal(N)  -->   verbo(N).
frase_verbal(N)  -->   verbo(N), frase_nominal(G1, N1).

articulo(m,s)  -->  [el].      adjetivo(m,s)  -->  [pequeño].    sustantivo(m,s) -->  [musico].      verbo(s) -->  [toca].
articulo(f,s)  -->  [la].      adjetivo(f,s)  -->  [pequeña].    sustantivo(f,s) -->  [bailarina].   verbo(s) -->  [baila].
articulo(m,s)  -->  [un].      adjetivo(m,s)  -->  [hermoso].    sustantivo(m,s) -->  [violin]
articulo(f,s)  -->  [una].     adjetivo(f,s)  -->  [hermosa].    sustantivo(f,s) -->  [danza].
```

Ahora no se generaría una sentencia del tipo **[una, hermoso, musico, baila, un, danza]** porque no concuerda en el género.

Ejemplo de una regla de la nueva gramática traducida a Prolog:

```
frase_verbal(N,S,R)  :-   verbo(N,S,U), frase_nominal(G1, N1, U, R).
```

Arbol sintáctico

Con la idea de la inclusión de argumentos adicionales, podemos lograr que durante el análisis de una frase se construya su árbol sintáctico. Para nuestro lenguaje, añadimos un argumento extra (el primero) que constituye el árbol sintáctico de la categoría correspondiente.

Ejemplo:

```
sentencia(sent(FN, FV), G, N)          -->  frase_nominal(FN, G, N), frase_verbal(FV, N).
frase_nominal(fr_nom(A, S), G, N)      -->  articulo(A, G, N), sustantivo_adj(S, G, N).
frase_nominal(fr_nom(S), G, N)         -->  sustantivo_adj(S, G, N).
sustantivo_adj(sust_adj(S), G, N)      -->  sustantivo(S, G, N).
sustantivo_adj(sust_adj(A, S), G, N)   -->  adjetivo(A, G, N), sustantivo(S, G, N).
frase_verbal(fr_verbal(V), N)          -->  verbo(V, N).
frase_verbal(fr_verbal(V, FN), N)      -->  verbo(V, N), frase_nominal(FN, G1, N1).

articulo(art(el), m, s)  -->  [el].
- - - - - - - - - - - - - - - - - - - -
adjetivo(adj(pequeño), m, s)  -->  [pequeño].
- - - - - - - - - - - - - - - - - - - -
sustantivo(sust(musico), m, s)  -->  [musico].
- - - - - - - - - - - - - - - - - - - -
verbo(verbo(toca), s)  -->  [toca].
- - - - - - - - - - - - - - - - - - - -
```

Análisis de una frase usando las nuevas reglas:

? frase_nominal(FN, G, N, [la, hermosa, bailarina, baila, una, danza], R).

FN = fr_nom(art(la), sust_adj(adj(hermosa), sust(bailarina)))
G = f
N = s
R = [baila, una, danza]

Gramática de Cláusulas Definidas (G.C.D.)

La gramática usada se llama G.C.D. Como se ha visto en parte, consiste en una G.I.C. extendida de dos formas:

- Se puede pasar información entre las definiciones por medio de argumentos extras, para expresar condiciones dependientes del contexto.
- Una G.C.D. puede contener fines ejecutables que no son parte de la gramática misma, sino que se ejecutan (al analizarse una sentencia del lenguaje) para producir efectos laterales. Estos fines aparecen entre llaves.

Ejemplo:

frase_nominal(fr_nom(S), G, N) --> sustantivo_adj(S, G, N), { write('no hay articulo') }.

? frase_nominal(FN, _, _, [hermosa, bailarina], [ ]).
no hay articulo
FN = fr_nom(sust_adj(adj(hermosa), sust(bailarina)))