Redes Semánticas

- Las redes semánticas son estructuras utilizadas para la representación de conocimiento en Inteligencia Artificial.

- Son especialmente útiles para representar conocimiento de taxonomías.

- Una red semántica es un grafo dirigido en el cual los nodos corresponden a:
  - *Constantes de Relación* tales como clases (como en los lenguaje orientado a objetos) y propiedades.
  - *Instancias*: objetos de alguna clase.

- Un arco en una red semántica puede tener diversas etiquetas:

  - *Instancia de* o id, definidas entre objetos y clases. Si N_O es un nodo que representa a un objeto O (instancia) y N_C es un nodo para una clase C, entonces un arco de instancia une a N_O y N_C en el grafo para indicar que O pertenece a la clase C.
  - *Subclase de* o sd, definidas entre subclases y clases. Si N_{SC} es un nodo que representa a una clase SC y N_C es un nodo para una clase C, entonces un arco de subclase une a N_{SC} y N_C en el grafo para indicar que SC es subclase de C.
  - Otras etiquetas que representan atributos de clases. Por ejemplo, podemos considerar las etiquetas “tiene” o t y “come”o c.

  En la figura, se muestra un ejemplo de una red semántica la cual tenemos dos objetos `Piolin` y `Pedro` y diversas clases de objetos. De esta red, se puede inferir que `Piolin` es un canario y que, dado que también es un `Ave` que tiene `Alas`.

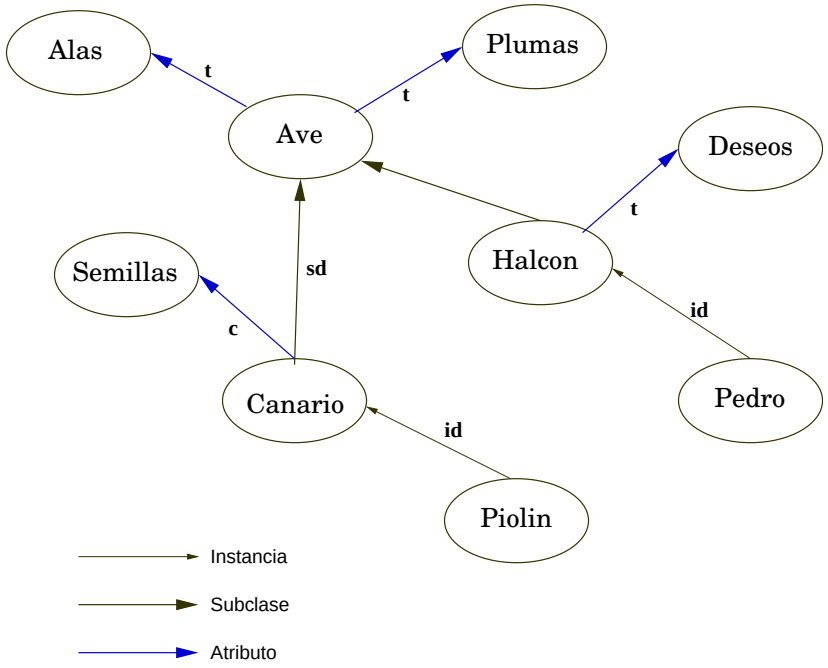


Figura 9: Red Semántica

- No hay una semántica precisa para las redes semánticas, y es el diseñador de una red semántica quien debe decidir cuáles son los mecanismos de herencia de propiedades y los mecanismos de resolución de conflictos.

Representación de Redes Semánticas en Prolog

Para modelar redes semánticas en PROLOG, necesitamos codificar el grafo que la representa. Para esto, utilizaremos los predicados:

`instancia_de(Objeto,Clase)`: es utilizado para decir que un `Objeto` pertenece a una `Clase` determinada. Para representar la red de la figura 9, tendremos las cláusulas unitarias:

```
instancia_de(piolin,canario).
instancia_de(pedro,halcon).
```

`subclase_de(Clase1,Clase2)`: indica que la `Clase1` está contenida en o forma parte de la `Clase2`. Para la red de la figura, tendremos:

```
subclase_de(canario,ave).
subclase_de(halcon,ave).
```

`tiene_propiedad(Clase1,Propiedad,Clase2)`: Esto involucra una relación entre dos clases e indica que `Clase1`) está relacionada con `Clase2` mediante la propiedad `Propiedad`. Para la red de la figura:

```
tiene_propiedad(canario,come,semillas).
tiene_propiedad(ave,tiene,alas).
tiene_propiedad(ave,tiene,plumas).
tiene_propiedad(halcon,tiene,deseos).
```

Para que nuestra red semántica sea más interesante, consideremos el siguiente conjunto de cláusulas:

```
instancia_de(tweety,canario).
instancia_de(pedro,halcon).
instancia_de(juan,halcon).
instancia_de(juancho,cocodrilo).
instancia_de(marcela,pinguino).

subclase_de(ave,oviparo).
subclase_de(reptil,oviparo).
subclase_de(canario,ave).
subclase_de(halcon,ave).
subclase_de(ave,animal).
subclase_de(reptil,animal).
subclase_de(cocodrilo,reptil).
subclase_de(pinguino,ave).
```

```
tiene_propiedad(canario,come,semillas).
tiene_propiedad(ave,tiene,alas).
tiene_propiedad(ave,tiene,plumas).
tiene_propiedad(halcon,tiene,deseos).
tiene_propiedad(ave,puede,volar).
tiene_propiedad(reptil,tiene,escamas).
```

Dada una red semántica, uno desea ser capaz de realizar consultas relativas a la red. Por ejemplo, ¿es `hector` un ave? O bien, en forma más general, querremos responder la pregunta ¿a qué clase pertenece un objeto? Para esto, podemos escribir:

```
es(Clase,Obj):- instancia_de(Obj,Clase).
es(Clase,Obj):- instancia_de(Obj,Clasep),
                subc(Clasep,Clase).
subc(C1,C2):- subclase_de(C1,C2).
subc(C1,C2):- subclase_de(C1,C3),
              subc(C3,C2).
```

El predicado `es(Clase,X)` se satisface cuando `X` pertenece a la clase `Clase`. Para responder esta pregunta estamos suponiendo que un objeto es una instancia de una clase `C` si es una instancia de `C` o de cualquier subclase de `C`. la

relación `subc` es la clausura transitiva de `subclase_de`. De este modo, con las reglas anteriores y la red del ejemplo, observamos lo siguiente:

```
?- es(X,juan).
X = halcon ;
X = ave ;
X = oviparo ;
X = animal ;
No
?-  es(animal,X).
X = piolin ;
X = pedro ;
X = juan ;
X = juancho ;
X = marcela ;
No
```

Supongamos que queremos preguntar qué propiedades tienen objetos cualesquiera:

```
propiedad(Obj,Prop):- es(Clase,Obj),
                      tiene_propiedad(Clase,Fun,Arg),
                      Prop =.. [Fun,Arg].
```

Así, observamos lo siguiente:

```
?- propiedad(piolin,X).
X = come(semillas) ;
X = tiene(alas) ;
X = tiene(plumas) ;
X = puede(volar) ;
No
```

Ahora, supongamos que queremos establecer que los pingüinos no pueden volar. Suponga entonces que agregamos:

```
tiene_propiedad(pinguino,no_puede,volar).
```

Con lo cual observamos:

```
?- propiedad(marcela,X).
X = no_puede(volar) ;
X = tiene(alas) ;
X = tiene(plumas) ;
X = puede(volar) ;
No
```

Obviamente, obtenemos que Marcela puede y no puede volar. Esto ocu-

rre pues no hemos incorporado la información de incompatibilidad entre `puede` y `no_puede`. Para realizar esto, podemos inventar un nuevo predicado `incompatible(Prop1,Prop2)`. En el ejemplo, escribiremos:

```
incompatible(puede(X),no_puede(X)).
```

De modo que volar y no volar son propiedades incompatibles. Ahora, ¿cómo manejamos esta información de incompatibilidad al responder consultas?

Un primer intento se observa en la siguiente redefinición del predicado `propiedad`.

```
es(Clase,Obj,0):- instancia_de(Obj,Clase).
es(Clase,Obj,Prio):- instancia_de(Obj,Clasep),
                     subcn(Clasep,Clase,Prio).

subcn(C1,C2,1):- subclase_de(C1,C2).
subcn(C1,C2,N):- subclase_de(C1,C3),
                 subcn(C3,C2,M), N is M+1.

propiedad(Obj,Prop,Prio):-
        es(Clase,Obj,Prio),
        tiene_propiedad(Clase,Fun,Arg),
```

```
        Prop =.. [Fun,Arg].

propiedad(Obj,Prop):- propiedad(Obj,Prop,Prio),
                      \+ incomp(Obj,Prop,Prio).

incomp(Obj,Prop,Prio):- incompatible(Prop,Propp),
                        propiedad(Obj,Propp,Priop),
                        Priop =< Prio.
```

Obsérvese que se redefinen algunos conceptos incorporando una medida de distancia. De modo que las incompatibilidad se resuelven apelando a la *especificidad*. Esto es, si un objeto hereda dos propiedades incompatibles, entonces se acepta aquella que es más específica. Así, en el caso de `marcela`, observaremos:

```
| ?- propiedad(marcela,X).
X = no_puede(volar) ;
X = tiene(alas) ;
X = tiene(plumas) ;
No
```

Nótese que el manejo de incompatibilidades no funciona cuando se heredan

propiedades de clases a igual distancia. ¿Qué ocurre en nuestra implementación?

Extensiones de Redes Semánticas

- Las redes semánticas son fáciles de extender para modelar casi cualquier propiedad en representación de conocimiento taxonómico.

- Sin embargo, como la semántica es poco claro, sería complejo pensar en intercambiar conocimiento entre distintas aplicaciones que funcionan con este sistema.

- Pensando en el intercambio de conocimiento (especialmente orientado al Web Semántico) hoy en día se están desarrollando lenguajes que se fundan en las redes semánticas.

- Tal es el ejemplo de DAML(DARPA Agent Markup Language) que se utiliza para representar información semántica que será utilizada en el Web Semántico.

- DAMLestá basado en lógica descriptiva (*Description Logics*) y está construido sobre RDF (Resource Description Language).

- La descripción de distintas “areas de conocimiento” usando este lenguaje se conocen como *ontologías*.

- DAML se escribe usando XML. El siguiente es un ejemplo de descipción de una clase:

```
<!-- Definicion de clases subclases -->

  <daml:Class rdf:ID="Product">
    <rdfs:label>Product</rdfs:label>
    <rdfs:comment>An item sold by Super Sports Inc.</rdfs:comment>
  </daml:Class>

  <daml:Class rdf:ID="BackPack">
    <rdfs:label>Back Pack</rdfs:label>
    <rdfs:subClassOf rdf:resource="#Product"/>
  </daml:Class>


<!-- Definición de Propiedades -->

<daml:ObjectProperty rdf:ID="usedFor">
```

```
    <rdfs:label>usedFor</rdfs:label>
    <rdfs:comment>The activity for which a product is used</rdfs:comment>
    <daml:domain rdf:resource="#Product"/>
    <daml:range rdf:resource="#Activity"/>
  </daml:ObjectProperty>

<!-- Definición de una instancia -->

 <ss:BackPack rdf:ID="ReadyRuck">
    <rdfs:label>Ready Ruck back pack</rdfs:label>
    <rdfs:comment>The ideal pack for your most rugged adventures</rdfs:comment>
    <ss:productNumber>23456</ss:productNumber>
    <ss:packCapacity>45</ss:packCapacity>
    <ss:usedFor rdf:resource="#Hiking"/>
  </ss:BackPack>
```

- Actualmente, hay proyectos en los que se pretende modelar todo el conocimiento.
- El proyecto OpenCyc, contiene una ontología con unos 6.000 conceptos y más de 60.000 afirmaciones sobre ellos.

Procesamiento de Lenguaje Natural en Prolog

¿Por qué?

- El Procesamiento de Lenguaje es esencial para la comunicación entre humanos y máquinas a través de lenguajes naturales.
- El desarrollo de esta área permite la posibilidad de construir traductores, interfaces a sistemas de información, bases de datos, computación ubícua, etc.
- Es posible argumentar que estas interfaces son más naturales para los seres humanos.
- Dependiendo de la entrada, voz o texto, se tienen niveles de complejidad distintos.
- Esta área ha sido tradicionalmente difícil... Las aplicaciones actuales de len-

guaje natural tienen grandes restricciones. Generalmente se limitan a dominios bastante acotados.

- El gran problema es la ambigüedad...

- Nos enfocaremos al procesamiento de lenguaje natural escrito como interfaces a bases de conocimiento.

Elementos del lenguaje natural

Varios aspectos esenciales que es necesario manejar al momento de construir un sistema de lenguaje natural.

Esto da origen a varios niveles de entendimiento.

Fonética La fonética define cómo se forman las palabras a partir de sonidos.

Morfología Se preocupa de la composición de las palabras. A partir de la morfología es posible obtener información sobre la función de las palabras. Ejemplo: *in-determinada-mente*.

Sintaxis La sintaxis describe la forma del lenguaje. Está especificada usualmente por una gramática. Los lenguajes naturales son complicados de definir con herramientas formales, como gramáticas libres de contexto.

Semántica La semántica define el significado de oraciones, frases y expresiones del lenguaje.

Pragmática La pragmática explica cómo una frase se relaciona con el contexto en el cual se está hablando.

Para entender completamente el significado de los tres últimos aspectos, consideremos las siguientes oraciones que se podrían haber dicho en la introducción de este curso:

Éste es un curso de Inteligencia Artificial
Las ranas hacen ruido mientras duermen
Las furiosas iguanas verdes y sin color, duermen placenteramente.
Cuatro elefantes en desde siempre ocho.

Dentro de esta clase, la primera frase tiene sentido en los tres aspectos. La segunda sólo en los dos primeros, la tercera, sólo en la sintaxis. La última frase no tiene sentido ni sintáctico ni semántico ni pragmático.

Análisis Sintáctico de Lenguajes Naturales

- En esta sección veremos algunos elementos computacionales para el análisis frases del lenguaje natural.

- A nivel sintáctico, los lenguajes están definidos por las oraciones legales que pertenecen a ellos. Las oraciones son secuencias de símbolos.

- Las oraciones legales de un lenguaje natural son analizadas a través de la gramática del lenguaje.

Gramática Castellana

- El la gramática castellana, por ejemplo, se distingue los siguientes elementos esenciales para expresar ideas o afirmaciones:

  Frases: expresan una idea, sin juicio ni aserción asociada. Ejemplo "*de vez en cuando*".
  Oraciones: Es una *unidad intencional* de la gramática. Contiene afirmaciones o juicios. Una oración siempre contiene al menos un verbo.
  Bimembres: Similares a las oraciones, pero no tienen verbo. Ejemplo: "*Buenos días a todos*".
  Período: Serie de oraciones, frecuentemente separadas por un **;**.

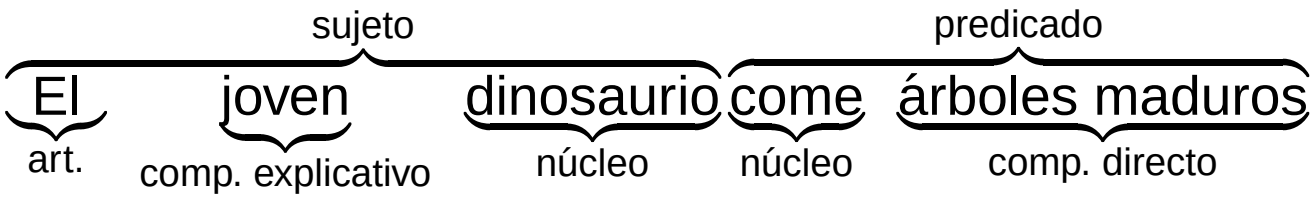
- Típicamente, una oración está compuesta por un **sujeto** y un **predicado**.

  El sujeto es de quien se habla en la oración.

  El predicado es lo que se dice del sujeto.

- Tanto el sujeto como el predicado pueden tener *complementos*. Estos complementos se clasifican en tipos.

Las siguientes frases muestran ejemplos de frases legales del castellano:



- Las oraciones del castellano no tienen todas esta estructura. Por ejemplo, las siguientes oraciones también son legales:

  Nos encontramos al lado del food-garden.

  Llovía a cántaros.

  Felipe y Andrés han salido.

Gramáticas Libres de Contexto y LNs

- Para poder analizar en lenguaje natural de manera computacional, necesitamos elementos para analizar computacionalmente las oraciones del lenguaje.

- Para realizar análisis sintáctico de lenguajes formales se utilizan con frecuencia las *gramáticas libres de contexto*.

  En el procesamiento de lenguaje natural, también se utilizan estos formalismos.

- Una gramática libre de contexto G es una tupla $G = (V, T, P, S)$, donde

  - V es un conjunto de variables,
  - T un conjunto de terminales,
  - P son las producciones y
  - S es el símbolo inicial de la gramática.

- Si $T = \{a, b, c\}$ y P está compuesto por las siguientes producciones:

$$S \to c$$
$$S \to aSb$$

  Entonces decimos que la gramática *genera* la palabra $aaacbbb$ porque, desde el símbolo inicial S es posible producir dicha palabra:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaacbbb$$

- El lenguaje formal generado por esta gramática está descrito por $L = \{a^i c b^i | i \geq 0\}$.

GLCs y castellano

- Por ejemplo, para el *castellano* podríamos definir la siguiente producción para oraciones:

$$[\texttt{oracion}] \longrightarrow [\texttt{sujeto}]\,[\texttt{predicado}]$$

- Aquí la variable oración describe todas las oraciones que son válidas en el lenguaje castellano y que están compuestas por un sujeto seguido de un predicado.

- Antes de continuar, veremos cómo es posible manejar directamente las GLCs en Prolog.

Manejos de GLC's en Prolog

- En PROLOG es posible manejar GLC directamente. El siguiente es el código PROLOG para nuestros ejemplos:

```
s --> [c].
s --> [a],s,[b].

oracion --> sujeto,predicado.
```

  donde los terminales están entre paréntesis cuadrados y las variables se escriben directamente.

- A partir de esta especificación, PROLOG genera las siguientes reglas:

```
s([c|A], A).
s(A, B) :-
        A = [a|C],
        s(C, D),
        D = [b|B].
```

```
oracion(A, B) :-
        sujeto(A, C),
        predicado(C, B).
```

- El predicado `s(L,R)` se satisface si `L` es una lista de átomos generada por la variable `s` de la gramática más un *resto* `R`. Así:

```
?- s([a,a,a,c,b,b,b,j],L).
L = [j] ;
No
```

- Es posible generar todas las palabras generadas por la gramática con la siguiente consulta:

```
?- s(L,[]).
L = [c] ;
L = [a, c, b] ;
L = [a, a, c, b, b] ;
L = [a, a, a, c, b, b, b] ;
L = [a, a, a, a, c, b, b, b, b]
Yes
```

- Es necesario tener cuidado con ciertas gramáticas, que por su orden de escritura, pueden conducir comportamientos indeseables. Éste es el caso de:

```
s --> s,[c].
s --> [c].
```

que no responde frente a la consulta `s(L,[])`.

Procesamiento de Lenguaje Natural en Prolog

- Supongamos que proponemos la siguiente GLC para un castellano simplificado:

```
oracion --> sujeto,predicado.

sujeto --> pronombre_personal.
sujeto --> articulo,sustantivo.
sujeto --> articulo,sustantivo,adjetivo.
sujeto --> sujeto,conjuncion,sujeto.
```

- Una `oracion` es un `sujeto` seguido de un `predicado`. Nótese que tenemos dos definiciones para un `sujeto`.

- A continuación definimos el resto de las producciones, las que incluyen los *terminales* de la gramática.

```
articulo --> [el];[la];[los];[las].
sustantivo -->
```

```
            [casa];[arbol];[gato];[gata];
            [perro];[avion];[mesa].

adjetivo --> [bello];[grande];[chico].

pronombre_personal --> [cecilia];[soledad];[hernan].

conjuncion --> [y];[o].

predicado --> verbo,complemento.

complemento --> complemento_directo.
complemento -->
    complemento_directo,
    complemento_indirecto.

complemento_directo --> sujeto.

verbo --> [tiene];[tienen];[juega];[canta];[salta].
```

- Con esta definición, podemos hacer la siguiente consulta:

```
?- oracion([cecilia,y,hernan,tienen,el,avion,grande],[]).
Yes
```

- Además:

```
?- oracion(L,[]).
L = [cecilia, tiene, cecilia] ;
L = [cecilia, tiene, soledad] ;
L = [cecilia, tiene, hernan] ;
L = [cecilia, tiene, el, casa]
Yes
```

- Evidentemente los resultados no son completamente satisfactorios. Debido a problemas en el número y género.

- Aparte de los problemas de número y género, ¿se pueden producir otros?

- La respuesta es **sí**. Dado que la definición de `sujeto` es recursiva por la izquierda, PROLOG falla frente a algunas consultas, por ejemplo:

```
?- oracion([cecilia,y,hernan],[]).
ERROR: Out of local stack
```

- Para solucionar este problema, se puede reescribir la gramática de manera que deje de ser recursiva por la izquierda.

Agregando Número y Género

- Es posible agregar argumentos a las variables de las gramáticas. Así, la siguiente gramática soluciona nuestro problema.

```
oracion --> sujeto(G,N),predicado(G,N).

sujeto(G,s) --> pronombre_personal(G).
sujeto(G,N) --> articulo(G,N),sustantivo(G,N).

sujeto(m,p) -->
        pronombre_personal(G),
        conjuncion,
        sujeto(_,_).

sujeto(m,p) -->
        articulo(G,N),sustantivo(G,N),
        conjuncion,
        sujeto(_,_).


articulo(f,s) --> [la];[una].
articulo(f,p) --> [las];[unas].
```

```
articulo(m,s) --> [el];[un].
articulo(m,p) --> [los];[unos].


sustantivo(m,s) --> [gato];[arbol].
sustantivo(m,f) --> [gatos];[arboles].
sustantivo(f,s) --> [gata];[mesa].
sustantivo(f,p) --> [luces];[gatas].

pronombre_personal(f) --> [cecilia];[soledad].
pronombre_personal(m) --> [hernan].

conjuncion --> [y];[o].

predicado(G,N) --> verbo(G,N),complemento.

complemento --> complemento_directo.
complemento --> complemento_directo,
    complemento_indirecto.

complemento_indirecto --> preposicion,sujeto(_,_).

preposicion --> [a];[sobre];[bajo];[de];[desde];[hacia].
```

```
complemento_directo --> sujeto(_,_).

verbo(_,s) --> [tiene];[come];[duerme];[salta];[es].
verbo(_,p) --> [tienen];[comen];[duermen];[saltan];[son].
```

Así,

```
?- oracion([soledad,y,hernan,tienen,una,gata,bajo,la,mesa],[]).
Yes

?- oracion([soledad,y,hernan,tiene,una,gata,bajo,la,mesa],[]).
No
```

Introducción al Análisis Semántico

- Hasta el momento sólo hemos dado una idea de cómo reconocer sintácticamente las frases, pero no nos hemos preocupado de su significado, ni menos cómo utilizar este significado.

- Es posible hacer que PROLOG construya una representación de las oraciones. Esto se hace agregando un argumento extra a las variables de la gramática.

- En el siguiente ejemplo, nos aprovecharemos del hecho que es posible insertar código PROLOG directamente en la especificación de una gramática libre de contexto.

  Así:

```
sustantivo --> [X],{member(X,[gato,casa,auto])}.
```

  Es equivalente a escribir:

```
sustantivo --> [gato];[casa];[auto].
```

```
oracion(oracion(Sujeto,Predicado)) --> sujeto(Sujeto,G,N),
                predicado(Predicado,G,N).

sujeto(sujeto(Pron),G,s) --> pronombre_personal(Pron,G).
sujeto(sujeto(Art,Sust),G,N) -->
        articulo(Art,G,N),sustantivo(Sust,G,N).

sujeto(sujeto(Pron,Conj,Suj),m,p) -->
        pronombre_personal(Pron,G),
        conjuncion(Conj),
        sujeto(Suj,_,_).

sujeto(sujeto(sujeto(Art,Sust),Conj,Suj),m,p) -->
        articulo(Art,G,N),sustantivo(Sust,G,N),
        conjuncion(Conj),
        sujeto(Suj,_,_).


articulo(art(fs,la),f,s) --> [la].
articulo(art(fs,una),f,s) --> [una].
articulo(art(fp,las),f,p) --> [las].
articulo(art(fp,unas),f,p) --> [unas].
articulo(art(ms,el),m,s) --> [el].
```

```
articulo(art(ms,un),m,s) --> [un].
articulo(art(mp,los),m,p) --> [los].
articulo(art(mp,unos),m,p) --> [unos].


sustantivo(sust(ms),m,s) --> [gato];[arbol].
sustantivo(sust(mf),m,f) --> [gatos];[arboles].
sustantivo(sust(fs),f,s) --> [gata];[mesa].
sustantivo(sust(fp),f,p) --> [luces];[gatas].

pronombre_personal(pron(f,X),f) -->

        {X=cecilia;X=soledad},[X].
pronombre_personal(pron(m,X),m) -->
           {X=miguelito;X=donald},[X].

conjuncion(conj(X)) --> {X=y;X=o},[X].

predicado(predicado(Verbo,Comp),G,N) -->
                  verbo(Verbo,G,N),complemento(Comp).

complemento(complmento(Comp)) --> complemento_directo(Comp).
complemento(complmento(Comp1,Comp2)) -->
```

```
    complemento_directo(Comp1),
    complemento_circunstancial(Comp2).

complemento_circunstancial(compcirc(Prep,Suj)) -->
    preposicion(Prep),sujeto(Suj,_,_).

preposicion(prep(X)) --> {X=a;X=sobre;X=bajo;X=de},[X].


complemento_directo(compdir(Suj)) --> sujeto(Suj,_,_).

verbo(verb(X,s),_,s) -->
      {member(X,[tiene,come,duerme,salta,es])},[X].
verbo(verb(X,p),_,p) -->
      {member(X,[tienen,comen,duermen,saltan,son])},[X].
```

- Así, se puede construir un árbol de *parse* a partir de nuestra especificación:

```
?- oracion(X,[cecilia,y,soledad,duermen,el,arbol,sobre,la,mesa],[]).

X = oracion(sujeto(pron(f, cecilia),
                   conj(y),
                   sujeto(pron(f, soledad))),
            predicado(verb(duermen, p),
                      complmento(compdir(sujeto(art(ms, el), sust(ms))),
                                 compcirc(prep(sobre),
                                          sujeto(art(fs, la),
                                                 sust(fs))))))
```

Otro ejemplo

- Puede resultar aún más interesante, construir un sistema que “aprenda” nueva información, a través de sentencias en lenguaje natural.

- El ejemplo que se muestra a continuación muestra un programa que es capaz de construir reglas a partir de información que se le entrega en lenguaje natural.

```
oracion -->
        sujeto_generico(Clase,Cond,Var),
        predicado(Pred,Var),
        {write('Agrego a la base de datos:'),
         nl,
         (Cond=true ->
              write((Pred:-Clase));
              write((Pred:-Cond,Clase))
         )}.

oracion -->
        sustantivo_propio(Nombre),
        predicado(Pred,Nombre),
```

```
        {write('Agrego a la base de datos:'),nl,write((Pred))}.

predicado(Pred,Var) -->
        verbo_intransitivo(Verb),
        {Pred=..[Verb,Var]}.

predicado(Pred,Var) -->
        verbo_transitivo(V),
        sustantivo(S),
        {Pred=..[V,S,Var]}.

predicado(Pred,Var) -->
        verbo_transitivo(V),
        adverbio(A),
        {Pred=..[V,A,Var]}.

predicado(Pred,Var) --> [es],[un],nombre_clase(C),{Pred=..[C,Var]}.

verbo_intransitivo(V) -->
        [V],
        {member(V,[ladra,vuela,corre,juega])}.

verbo_transitivo(V) -->
```

```
        [V],
        {member(V,[come,tiene,sabe,conoce,cuida])}.

sujeto_generico(Clase,Cond,Var) -->
        [todo],nombre_clase(X),
        {
         Clase=..[X,Var],
         Cond=true
        }.

sujeto_generico(Clase,Cond,Var) -->
        [todo],nombre_clase(X),
        [que],condiciones(Cond,Var),
        {
         Clase=..[X,Var]
        }.


condicion(V) --> verbo_intransitivo(V).

condiciones((Cond),Var) -->
        condicion(F),
        {
```

```
         Cond=..[F,Var]
        }.


condiciones((Cond,Conds),Var) -->
        condicion(F),
        ([y];[y],[que]),
        condiciones(Conds,Var),
        {
         Cond=..[F,Var]
        }.


sustantivo_propio(X) -->
        [X],
        {member(X,[ana,luisa,juan])}.

sustantivo(S) -->
        [S],
        {member(S,[alas,papas,canas,plumas,vida,auto])}.

adverbio(A) -->
        [A],
```

```
        {member(A,[bien,mal,rapidamente,lentamente])}.


nombre_clase(X) -->
        [X],
        {member(X,[animal,perro,canario,avestruz,ave,
                   libro,mueble,mesa,jugador,empedernido,
                   hombre,mujer])}.
```

- El siguiente es un ejemplo de ejecución:

```
?- oracion([ana,ladra],[]).
Agrego a la base de datos:
ladra(ana)


?- oracion([luisa,es,un,canario],[]).
Agrego a la base de datos:
canario(luisa)
```

```
?- oracion([juan,come,papas],[]).
Agrego a la base de datos:
come(papas, juan)

?- oracion([todo,animal,que,vuela,es,un,ave],[]).
Agrego a la base de datos:
ave(_G288):-vuela(_G288), animal(_G288)

?- oracion([todo,avestruz,come,rapidamente],[]).
Agrego a la base de datos:
come(rapidamente, _G258):-avestruz(_G258)

?- oracion([todo,animal,que,ladra,y,que,corre,es,un,perro],[]).
Agrego a la base de datos:
perro(_G318):- (ladra(_G318), corre(_G318)), animal(_G318)
```