APROXIMACIÓN METAGRAMATICAL SINTÁCTICO SEMÁNTICA DE LA COORDINACIÓN

Palomar, M; Moreno, L; Lopez, V.

Departamento de Sistema Informáticos y Computación
Universidad Politécnica de Valencia.

RESUMEN

En este trabajo se aborda el tratamiento, a través de la lógica, de la Coordinación en frases del Lenguaje Natural. El formalismo utilizado para expresar la gramática del lenguaje es la Gramática Lógico Modular, propuesta por M.McCord. La Coordinación (construcciones gramaticales con conjunciones y, o, pero ...) es uno de los fenómenos del Lenguaje Natural más difícil de tratar, tanto a nivel lingüístico como computacional. Este fenómeno implica a un número muy amplio de componentes gramaticales más los problemas de la elipsis (o reducción de elementos coordinados) y la interacción entre coordinación y cuantificación. En este trabajo presentamos un mecanismo metagramatical que trata el fenómeno de la coordinación más los problemas de la elipsis. El trabajo esta desarrollado en BIM-Prolog, sobre un SUN 3/80.

1. INTRODUCCIÓN

El presente estudio se encuentra desarrollado dentro de un Sistema capaz de responder consultas en Lenguaje Natural a una Base de Datos. El sistema procesa consultas en Lenguaje Castellano que serán evaluadas en una Base de Datos geográfica, puede verse en [MORENO,92a],[MORENO,92b] y [MOLINA,92]. Este trabajo se centra en el estudio y tratamiento de la coordinación, utilizando como formalismo la Gramática Lógico Modular (GLM) [MCCORD,85] fácilmente compiladas a Clausulas de Horn.

El Sistema presentado consta de cuatro apartados principales que son tratados en el trabajo y son los siguientes:

1.- El formalismo para gramáticas Lógicas, presentado por M.McCord.
2.- Un preprocesador de reglas gramaticales que transforma las reglas GLM en clausulas de Horn.
3.- Un interprete para GLM encargado de realizar el Análisis Sintáctico, construyendo el árbol sintáctico.
4.- Un componente de Interpretación Semántica que genera formas Lógicas a partir del árbol sintáctico generado en el punto anterior, donde trataremos ademas el problema de la elipsis.

En un árbol sintáctico, cada nodo contiene no solo información sintáctica, sino también un termino llamado item semántico, que determina la construcción del nodo a la forma lógica de la sentencia. A través de sus items semánticos, los hijos actúan como modificadores del nodo. De este modo no se soluciona el fenómeno de la coordinación; para resolverlo se requiere unas posibilidades metalógicas con las que Prolog no cuenta, por lo tanto vamos a utilizar un interprete para GLM escrito en Prolog que no haga necesaria su compilación y nos permita tratar dicho fenómeno de la coordinación. En los siguientes apartados veremos como definimos este interprete así como el preprocesador utilizado para traducir las reglas de la gramática.

Además veremos, el componente de interpretación semántica que genera formas lógicas a partir de árboles sintácticos y que resuelve el problema de la elipsis. Se resuelven los problemas puntuales como la concordancia en genero y número entre múltiples elementos coordinados y el tratamiento de los nombres propios en casos de coordinación.

La GLM por sí misma no menciona la coordinación. El interprete facilita el tratamiento para el manejo de las conjunciones; Entonces cuando aparece un conjunción en una sentencia,

A X conj Y B

comienza un proceso que inicia la vuelta atrás en la historia del análisis para analizar Y en paralelo a X. B se analizara una vez analizados los elementos coordinados, mediante el estado en el que se encontraba el sistema cuando fue interrumpida por la conjunción.

Así nuestro sistema analizaria la siguiente frase del siguiente modo:

«Cada hombre y cada mujer comen una manzana»

el árbol generado de forma muy simplificada y reordenado sería:

```
nombre( cada hombre)
      conj(y)
            nombre(cada mujer)
      verbo(comen)
      nombre(una manzana)
```

donde la fase de análisis sería igual que en una frase simple hasta encontrar una conjunción, en este caso «y», en ese momento guarda en la pila «cada hombre», y suspende el análisis de la frase, para buscar el segundo elemento coordinado de igual función sintáctica y estructura jerárquica, en este caso «cada mujer». Una vez detectado continua el análisis en el punto donde lo suspendio anteriormente. Se desinstancian las variables de los múltiples (en este caso dos) elementos coordinados para controlar la concordancia en genero y número. Con todo esto nos genera un árbol sintáctico donde aparecen los items semánticos necesarios para la siguiente fase de análisis. Este árbol es recogido por el analizador semántico que se encarga de generar la forma lógica después de reordenar y clasificar el árbol según los items propios y no propios de la conjunción, que apareceran en las entradas léxicas. Así nos generara para el ejemplo la siguiente forma lógica:

```
todo(hombre(X),ex(manzana(Y),comer(X,Y)))&
todo(mujer(X),ex(manzana(Y),comer(X,Y)))
```

Como se observa, «comen una manzana» estará dentro del rango del sintagma nominal coordinado, y este se duplicara. Así podemos observar en otro ejemplo la utilidad de una eficiente reordenación teniendo en cuenta la prioridad de los cuantificadores,

«Un hombre y una mujer comen cada manzana»

La reordenación mueve el sintagma nominal universalmente cuantificado a la izquierda del sintagma nominal coordinado cuantificado universalmente, entonces lo único que se duplica en este caso seria «comen». La forma lógica completa sería:

```
todo(manzana(Y),ex(hombre(X),comer(X,Y)) &
            &ex(mujer(X),comer(X,Y))
```

En las secciones siguientes revisaremos estos tratamientos según han sido procesados en los módulos correspondientes al Anexo.

Ambito gramatical abordado por nuestro sistema.

Según el estudio lingüístico realizado sobre la coordinación en los trabajos [LOPEZ,92] y [GERRER,92], hemos reducido el conjunto de posibles elementos coordinados, para centrar el problema al subconjunto que a continuación se detalla.

En primer lugar el sistema aborda tanto coordinación oracional como coordinación sintagmatica; la coordinación oracional rebasa los limites de una gramática oracional y funciona como un fenomeno textual, mientras en la coordinación sintagmatica hemos centrado el problema en los siguientes casos posibles:

El sistema trata oraciones coordinadas copulativas con el conector «y», las coordinadas disjuntivas a través del conector «o», más el caso especial de yuxtaposición conectado con la «,».

Los elementos coordinados seran los siguientes, sintagmas nominales, nombres propios, pronombres, nucleos nominales, sintagmas nominales, nucleos verbales, adjetivos, adverbios y comparativos en oraciones copulativas; que cubren un amplio campo de la coordinación. El tratar más posibilidades, seria tan sencillo como modificar la gramática utilizada definiendo los posibles elementos coordinados, lo que llevaria a definir nuevos no terminales fuertes, pero este es un tema no propio del presente trabajo.

2. INTERPRETE DE REGLAS DE LAS GLM.

Como se ha comentado anteriormente, hemos desarrollado un preprocesador encargado de traducir la gramática GLM a una forma más conveniente para su computación mediante el metainterprete. Este preprocesador actúa del siguiente modo:

Dada una regla GLM,

A ===> B

esta se procesa en un término tal como,

rule (NT,B1)

donde NT es el no terminal A, B1 es el resultado de convertir B en una lista. Por tanto la lista B1 estará formada por no terminales, terminales, items semánticos y objetivos Prolog. Este proceso se puede ver en el Anexo.

A continuación vemos el interprete que también puede verse en el Anexo. Este sigue las nociones de [DAHL,83], aunque difiere de él, en cuanto que no maneja las listas de extraposición que no permite las GLM, pero si procesa los items semánticos aunque de modo diferente, pues en las GLM aparecen en el cuerpo de las reglas. El procedimiento de más alto nivel es el «parse», que analiza una cierta cadena de entrada de tipo «nonterminal» y devuelve su árbol de análisis en «syn». Actúa como sigue:

```
parse(String,Nonterminal,Syn):-prs([Nonterminal],String,[Syn],nil,nil),!.
```

donde «parse» llama al procedimiento base del análisis del siguiente modo:

```
prs(BL,X,Mods,Par,Mer)
```

«prs» toma el cuerpo de una regla «BL» y lo analiza contra una subcadena de la cadena de entrada «X», devolviendo una subestructura de análisis «Mods». Los dos siguientes argumentos «Par» y «Mer» son las pilas «parent» y «merge». La pila «parent» sirve como pila de recursión en el trabajo normal de análisis y en la vuelta atrás necesaria para analizar la coordinación. La pila «merge» sirve para detener el análisis de la segunda cláusula coordinada y regresar a la situación del instante en que se encontró la conjunción.

Así la regla «prs» es la encargada de detectar la aparición de una conjunción en la frase introducida.

```
prs([],[D|X],Mods,Par,Mer):-demon(D,[],X,Mods,Par,Mer).
```

de esto se encarga «demon» que detecta si el siguiente terminal a analizar de la cadena de entrada es o no una conjunción, como puede verse a continuación:

```
demon(D,BL,X,Mods,Par,Mer) :-
                    conjunction(D,Cat,Sem),
                    backup(Par,Mods,[syn(Cat,[Sem|Mods0])],[NT|_],Par1),
                    concordance(NT),
                    unboundnt(NT,NT1),
                    prs([NT1],X,Mods0,nil,merge(BL,Par1,Mer)).
```

Si es una conjunción llama a «backup» que vuelve atrás en el análisis utilizando la pila «parent». Si no lo es, continua el análisis normal. El procedimiento «backup» es el encargado de detectar cual es la categoría gramatical que se coordina, empezando por la última analizada en el momento de encontrar la conjunción. Si el intento de analizar la segunda cláusula coordinada como esa última categoría fracasa, se satisfacen trivialmente los no terminales de ese nivel que queden por analizar, se instancia a lista vacía la lista de modificadores para ese nivel, y se sube un nivel más arriba. Así hasta que se tiene éxito. En ese instante se añade un nodo conjunción al árbol de análisis del que colgara la segunda cláusula. Las reglas para este procedimiento serian:

```
backup(parent(BL,Mods,Par),Mods0,Mods0,BL,parent(BL,Mods,Par)).
backup(parent([_|BL],Mods,Par),[],Mods0,BL1,Par1) :-
                    satisfied(BL),
                    backup(Par,Mods,Mods0,BL1,Par1).
```

Después, si se satisface «backup», el procedimiento «demon» llama a un nuevo predicado llamado «concordance», que se encarga de solucionar los problemas de concordancia entre sujeto y verbo en casos de coordinación. El modo de solucionar este problema es el de desinstanciar las variables encargadas de controlar la concordancia en genero y número. Este predicado es el siguiente:

```
concordance(nombrepropio(_,_,_,t)) :- !.
concordance(nombrepropio1(_,_,_,t)) :- !.
concordance(pronombre(_,_,_,t)) :- !.
concordance(nucnom(_,_,_,t)) :- !.
concordance(sn(_,_,_,t)) :- !.
concordance(snint(_,_,_,t)) :- !.
```

Posteriormente llama a «unboundnt», procedimiento de vital importancia que determina cuales son las categorías gramaticales que es posible coordinar, restringidas según el apartado del estudio lingüístico. Una de estas reglas, la encargada de coordinar dos oraciones, es la siguiente:

```
unboundnt(s(T_orac),s(T_orac)) :-!.
```

Para terminar «demon» llama a «prs» con la categoría gramatical a coordinar, apilando en la pila «merge» la situación del análisis en ese instante.

La segunda regla de «prs» tiene que ver con la utilidad de la pila «merge». En el instante del análisis en que lo que falta por analizar de la segunda cláusula coordinada y la lista BL apilada en «merge» coinciden, se satisface trivialmente todo lo que quede por analizar de esa segunda cláusula y se continua el análisis normal, de forma que los elementos que restan por analizar son compartidos por ambas cláusulas coordinadas. Esto se consigue a través de la compartición de variables que tiene lugar al hacerse coincidir las listas mencionadas. La siguiente seria la segunda regla de «prs»:

```
prs(BL,X,Mods,Par,merge(BL,Par1,Mer)):-
                    cutoff(Par),
                    prs(BL,X,Mods,Par1,Mer).
```

La tercera regla se encarga de procesar los items semánticos que aparecen en el cuerpo de las regla GLM, incorporándolas a la estructura de análisis que haya en ese momento. esta regla sería la siguiente:

```
prs([Op-P|Bl],X,[Op-P|Mods],Par,Mer):-
                    prs(BL,X,Mods,Par,Mer).
```

La cuarta regla consume un terminal de la cadena de terminales de entrada. La quinta regla procesa las listas vacías que aparecen como terminales en las reglas GLM. La sexta regla procesa los objetivos Prolog de las reglas de la gramática. La tercera y quinta regla son necesarias por estar sometidos a una GLM, que no serian necesarias en una Gramática de Estructura Modificada (GEM) como utiliza V. Dahl en su trabajo sobre el «Tratamiento de la Coordinación» para un Lenguaje Ingles [DAHL,83]. Estas en resumen son las principales reglas que se encargan de manejar el fenómeno de la coordinación. Estas reglas son las siguientes:

```
prs([[W]|BL],[W|X],Mods,Par,Mer):-
                    prs(BL,X,Mods,Par,Mer).
prs([[]|BL],X,Mods,Par,Mer):-
                    prs(BL,X,Mods,Par,Mer).
prs([{B}|BL],X,Mods,Par,Mer):- !,B,
                    prs(BL,X,Mods,Par,Mer).
```

El descenso de un nivel en el análisis lo realiza la séptima regla de «prs» que obtiene el cuerpo de la regla encabezada por el primer no terminal de la lista «BL» a través de los predicados «rule» almacenados por el preprocesador GLM en la base de hechos, y apila el resto del análisis de ese nivel en la pila «parent» mediante la llamada al procedimiento «prspush». Esta regla sería la siguiente:

```
prs([NT|BL],X,Mods,Par,Mer):-
                    rule(NT,Body),
                    prspush(Body,NT,BL,X,Mods,Par,Mer).
```

Si el «Body» de la regla es vacío, entonces «prspush» continua normalmente con el análisis. Pero, por el contrario, si «Body» no es vacío, se añade el nodo correspondiente al no terminal que encabeza la lista «BL» (incluido ese no terminal) y la variable que hace referencia al resto de la estructura de análisis de este nivel en la pila «parent», y se continua con el análisis un nivel más abajo.

El conjunto de Metareglas utilizadas para el tratamiento de las coordinadas puede verse en conjunto en el Anexo.

3. INTERPRETACIÓN SEMÁNTICA

La interpretación semántica de un árbol sintáctico de GLM se realiza en dos fases fundamentales.

La primera de ellas trata heurísticamente con el problema del rango o alcance de los cuantificadores, que surge debido a las discrepancias que existen entre relaciones sintácticas de superficie y pretendidas relaciones semánticas. El modo de tratarlas es a través de una reordenación y transformación del árbol generado en el análisis sintáctico en otro árbol con las relaciones de los modificadores; en principio, se utiliza la reordenación y transformación del árbol sintáctico utilizado por el sistema de consulta en las oraciones simples [MORENO,92a] y [MORENO,92b].

La segunda fase toma el árbol transformado y lo traduce a una forma lógica. Los modificadores hacen su trabajo de modificación a través de sus items semánticos. En esta fase se resuelve el problema de la elipsis en coordinación utilizando los items semánticos relacionados. El objetivo fundamental de todo módulo para el tratamiento de la elipsis sera el de obtener a partir de la sentencia introducida en el sistema (estructura superficial), una representación del significado de la frase (forma lógica) que sea adecuada y donde cada elemento ocupe el orden preciso (estructura profunda), incluido el elemento elidido, que se ha de recuperar. Para ello existen diversos enfoques que dependen del entorno de trabajo en el que nos movamos, es decir, si pretendemos analizar el problema gramaticalmente o metagramaticalmente. Al tratarlo metagramaticalmente, la solución que adoptamos es que la reconstrucción de la estructura profunda de la oración, se haga en en esta fase de análisis.

El procedimiento que se encarga de ello es «translate» que trabaja únicamente con los componentes semánticos de los nodos del árbol sintáctico y que a continuación se describe su principal funcionamiento.

```
translate(Syn,LogForm1):-
        transmod(Syn,l-t,l-LogForm),
        simplify(LogForm,[],LogForm1).
```

Un item semántico puede combinarse con un segundo item semántico para producir un tercer item semántico. «translate» utiliza estas operaciones de combinación de manera recursiva para producir la forma lógica de un árbol. El procedimiento auxiliar «transmod», que es el que en realidad realiza la recursión, produce items semánticos completos como traducciones, no solamente formas lógicas.

«transmod» trabaja del siguiente modo, los hijos (modificadores) de un nodo N del árbol, se traducen recursivamente a items semánticos, y estos items modifican de manera acumulativa al item semántico de N, actuando el más a la izquierda como modificador más exterior, es decir, conforme se dibuja el árbol, de abajo a arriba.

Así, el corazón del proceso de traducción esta en las reglas que dicen como los items semánticos pueden combinarse con otros items semánticos. Hay reglas para el procedimiento trans(Sem0,Sem1,Sem2) que dicen que Sem0 se combina (modifica) con Sem1 para producir Sem2. Estas reglas son las siguientes, que son las utilizadas para tratar la coordinación:

```
trans(_Sem,_C*_D-_P,cbase1(_Sem,_C,_D)-_P) :- !.
trans(cbase1(_Sem,_C,_D)-_P,Op-_Q,cbase2(_Op,_Sem,_C,_D,_R)-true) :- !,
                        and(_P,_Q,_R).
trans(_Op-_P,cbase2(_Op1,_Sem,_C,_D,_B)-_P1,
        cbase2(_Op2,_Sem,_C,_D,_B)-_P2):- !,trans(_Op-_P,_Op1-_P1,_Op2-_P2).
trans(cbase2(_Op,_Sem1,_C,_D,_B)-_P,_Sem2,_Op1-_B) :- !,
            trans(_Op-_P,Sem2,_Op1-_C),
            trans(_Sem1,_Sem2,_Op1-_D).
```

además del predicado «and» necesario en la utilización de las reglas anteriores,

```
and(t,Q,Q):- !.
and(P,t,P):- !.
and(P,Q,(P&Q)).
```

Con todo ello se puede realizar una implementación resolviendo el problema de la reconstrucción profunda y obteniendo ls forma lógica en función de los items semánticos propios y no propios de la coordinación.

4. CONCLUSIONES

Como se comento anteriormente el fenómeno de la coordinación puede tratarse de modo gramatical, disponiendo de una gramática apropiada y desarrollada para este tratamiento, o de modo metagramatical independiente de las reglas de la gramática que se utilicen. En el estudio realizado en [LOPEZ,92] y [GERRER,92], se discuten todos estos aspectos, decantandonos por un procesamiento metagramatical por disponer de un sistema interactivo de consulta y evaluación el cual utiliza una gramática Lógico Modular.

En este trabajo hemos presentado un procesamiento metagramatical de la coordinación independiente de la gramática que se utilice; esto es, no es necesario manipular el conjunto de reglas de la gramática utilizadas para las frases simples, sino que apuntamos tres módulos que estan por encima de la gramática.

Además nuestro sistema cuenta con una caracteristica interesante, la cual el interprete de las GLM manipula las pilas de manera que se permite la coordinación multiple, y su correspondiente coordinación en genero y número con el resto de elementos coordinados.

BIBLIOGRAFÍA

[ABRAMS, 89a] Abramson, H. y Dahl, V.(1989)
Logic Grammars,
Symbolic Computation Series.
Springer-Verlag.

[ALLEN, 87] Allen,J. (1987)
Natural Language Understanding.
Benjaming Cummings series in Computer Science.

[DAHL, 81] Dahl, V. (1981)
Traslating Spanish into Logic through Logic.
American Journal of Computational Linguistics, vol. 7, n. 3.

[DAHL,83] Dahl, V. y McCord, M. (1983)
Treating Coordination in Logic Grammars.
American Journal of Computational Linguistics

[GAZDAR, 89] Gazdar,G.y Mellish,C. (1989)
Natural Language Processing
Addison-Wesley.

[MCCORD, 85] McCord,M. (1985)
Modular Logic Grammars.
Proc. Association of Computational Linguistics.

[MCCORD, 82] McCord,M. (1982)
Using Slots and Modifiers in Logic Grammars for Natural Language.
Artificial Intelligence, vol. 18

[MCCORD, 89a] McCord, M.(1989)
A New Version of the Machine Translation System LMT.
Literary and Linguistic Computing, vol. 4, n.3.

[MCCORD, 89b] McCord, M. (1989)
Design of LMT: A Prolog-Based Machine Translation System.
Computational Linguistics, vol. 15, n. 1.

[MCCORD, 89c] McCord, M.(1989)
LMT
Proc. MT SUMMIT II, Munich.

[MORENO,92] Moreno, L. y Palomar, M.(1992)
Semantic Constraints in a Syntactic Parser.
en Database and Expert Systems Applications
Springer-Verlag (pendiente de publicación)

[PALOMA, 91] Palomar, M. , Moreno, L. y Pascual, A. (1991)
Semantic Interpretation of Natural Language in PROLOG:Logical Forms.
Proc. Database and Expert Systems Applications
Karagiannis, D. Ed. (3-211-82301-8)
Springer-Verlag

[PALOMA, 90] Palomar, M. y Moreno, L. (1990)
Syntactic and Morphologic Analysis of Natural Language Expressions.
Proc. Artificial Intelligence Application & Neural Networks
Hamza, M.H. Ed. (0-88986-152-8)
Acta Press

[VANCAN, 86] Van Caneghem,M y Warren,D eds. (1986)
Logic Programming and its applications.
Ablex Publishing Coorporation.

[WALKER, 89] Walker,A y otros eds. (1989)
Knowledge Systems and Prolog.
Addison-Wesley.

ANEXO

```
/* Preprocesador de reglas GLM */

readRules :- see('metgrr.ari'), teel(metrr1.ari'), repeat, read(Rule),
process(Rule).
process(endRules) :-write(endR),
                           write('.'),nl,seen,told,see('x.pro'),
                           repeat,read(Rule), proc(Rule).
process((Head ==> Body)) :-!,
                          makelist(Body,Body1),
                           write(rule(Head,Body1)),
```

```
                                write('.'),nl,fail.
process(Rule) :- write(_Rule),write(.),nl,fail.
proc(endR) :- !,seen.
proc(Rule) :- assertz(Rule),fail.
makelist((X:L),[X|L1]):- !,makelist(L,L1).
makelist([],[]):- !.
makelist(X,[X]).

/* Analizador Sintáctico */

pparse(String,Nonterminal,Syn) :-
            prs([Nonterminal],String,[Syn],nil,nil),!.
prs([],[D|X],Mods,Par,Mer) :-
            demon(D,[],X,Mods,Par,Mer).
prs(BL,X,Mods,Par,merge(BL,Par1,Mer)):-
            cutoff(Par),
            prs(BL,X,Mods,Par1,Mer).
prs([Op-P|Bl],X,[Op-P|Mods],Par,Mer):-
            prs(BL,X,Mods,Par,Mer).
prs([[W]|BL],[W|X],Mods,Par,Mer):-
            prs(BL,X,Mods,Par,Mer).
prs([[]|BL],X,Mods,Par,Mer):-
            prs(BL,X,Mods,Par,Mer).
prs([{B}|BL],X,Mods,Par,Mer):- !,B,
            prs(BL,X,Mods,Par,Mer).
prs([NT|BL],X,Mods,Par,Mer):-
            rule(NT,Body),
            prspush(Body,NT,BL,X,Mods,Par,Mer).
prs([[],X,[],parent([_|BL],Mods,Par),Mer):-
            prs(BL,X,Mods,Par,Mer).
prs([],[],[],nil,nil).
prspush([],_,BL,X,Mods,Par,Mer):- !,
            prs(BL,X,Mods,Par,Mer).
prspush(Body,NT,BL,X,[syn(NT,Mods1)|Mods],Par,Mer) :-
            prs(Body,X,Mods1,parent([NT|BL],Mods,Par),Mer).
cutoff(parent([_|BL],[],Par)) :-
            satisfied(BL),
            cutoff(Par).
cutoff(nil).
demon(D,BL,X,Mods,Par,mer) :-
            conjunction(D,Cat,Sem),
            backup(par,Mods,[syn(Cat,[Sem|Mods0])],[NT|_],Par1),
            concordance(NT),
            unboundnt(NT,NT1),
            prs([NT1],X,Mods0,nil,merge(BL,Par1,Mer)).
backup(parent(BL,Mods,Par),Mods0,Mods0,BL,parent(BL,Mods,Par)).
backup(parent([_|BL],Mods,Par),[],Mods0,BL1,Par1) :-
            satisfied(BL),
            backup(Par,Mods,Mods0,BL1,Par1).
satisfied([]) :- !.
satisfied([NT|BL]) :- rule(NT,[]), !,satisfied(BL).
concordance(nombrepropio(_,_,_,t)) :- !.
concordance(nombrepropio1(_,_,_,t)) :- !.
concordance(pronombre(_,_,_,t)) :- !.
concordance(nucnom(_,_,_,t)) :- !.
concordance(sn(_,_,_,t)) :- !.
concordance(snint(_,_,_,t)) :- !.
concordance(NT).
unboundnt(s(T_orac),s(T_orac)) :-!.
unboundnt(sn(Xres:G_N:Tdet,fld(I,TE),T,Conc),sn(Xres:_:_,_,T,_)) :- !.
unboundnt(nombrepropio(X:Res,G_N,Y:Rgs,Conc),nombrepropio(X:Res,_,_,_)) :- !.
```

```
unboundnt(nucnom(Xres,G_N,Huecos,Yrgs,Conc),nucnom(Xres,_,Huecos1,_,_))  :-
                    unboundlt(Huecos,Huecos1), !.
unboundnt(pronombre(Xres,G_N,Dtipo,Yrgs,Conc),pronombre(Xres,_,Dtipo,_,_)) :- !.
unboundnt(sv(Infl,Voz,E,X:G_N,T,Comp,Conc),sv(Infl,Voz,_,X:G_N,T,_,_)) :- !.
unboundnt(nucver(Voz,Infl,E,X,Huecos),nucver(voz,Infl,_,X,Huecos1)) :-
                    unboundnt(Huecos,Huecos1), !.
unboundnt(adjetivos(X,G_N),adjetivos(X¿G_N)) :- !.
unboundnt(adverbio(E,T),adverbio(E,T) :- !.
unboundnt(compser((X:Ras):G_N,Op),compser((X:Ras):G_N,Op)) :- !.
unboundnt(comppost((X:Ras):G_N,Op),comppost((X:Ras):G_N,Op)) :- !.
unboundnt(comppre2(X,_),comppre2(X,_)) :- !.
unboundlt([Hueco|Huecos],[Hueco1|Huecos1]) :-
                    unboundlt(Huecos,Huecos1),
                    unbound(Hueco,Hueco1).
unboundlt([],[]) :-.
unbound((X:Y:Z),(X:Y:_)).
```