UN COMPILADOR DE LFG Y SU APLICACIÓN AL EUSKARA

J. Carlos Ruiz Antón (ISS, Barcelona)
Joseba Abaitua (Fujitsu, Barcelona)
J. Ramón Zubizarreta (UPV-EHU, Donostia)

Este artículo presenta un sistema denominado LFG-LAB para el desarrollo de descripciones gramaticales en LFG, aplicado a una sintaxis del euskara como parte de un sistema de diálogo orientado al objeto, que incluye modelación del diálogo, actos de habla, base de conocimiento, etc. (Zubizarreta, tesis doctoral en preparación).[1]

LFG-LAB permite al usuario introducir reglas y léxico en una notación equivalente a la descrita en Kaplan y Bresnan (1982). La gramática resultante se compila a un programa de PROLOG, que puede procesarse directamente siguiendo una estrategia de análisis ascendente.

El artículo se configura de la siguiente manera: la sección 1 introduce los rudimentos de la LFG.[2] La sección 2 presenta algunas de las soluciones adoptadas en la implementación del sistema (tratamiento paralelo de estructura-c y estructura-f, unificación, estrategias de análisis). La sección 3 muestra una sesión típica de trabajo con LFG-LAB.

1. ¿Por qué LFG?

La Gramática Léxica Funcional (*Lexical Functional Grammar*, o LFG) (Bresnan 1982) nació como fruto de la colaboración entre la lingüista Joan Bresnan, de tradición generativista, y el informático Ron Kaplan, uno de los pioneros en

[1]Nuestra gramática de euskara (v. apéndice) cubre un cuerpo experimental de diálogos extraído de un modelo que consta de dos agentes con diferente conocimiento del dominio y capacidades físicas diversas. El experimento trata de generar un entorno que provoque la comunicación con naturalidad. El agente humano pretende alcanzar sus objetivos produciendo actos de habla de naturaleza informativa o inquisitiva (Searle 1969, Austin 1962, Allen & Litman 1987). El sistema final incluirá un componente morfológico, basado en los trabajos de Carroll y Abaitua (1987).

parsing mediante ATNs y charts. El empeño de ambos fue la elaboración de una teoría gramatical que facilitara la descripción lingüística de lenguas tipológicamente diversas, en un formalismo computacionalmente tratable.

La LFG pertenece a la familia de las gramáticas de unificación. Básicamente, todas ellas son gramáticas de Estructura de Frase, con una operación de unificación. En los formalismos de este género, los objetos lingüísticos se representan bajo la forma de estructuras complejas de rasgos. Estas estructuras se construyen inductivamente mediante reglas declarativas que combinan elementos parcialmente especificados (Shieber 1986).

La LFG ofrece la ventaja de ser un formalismo equilibrado entre sus exigencias teóricas y su versatilidad práctica (cf. Yasukawa 1983, Frey 1985). Permite establecer generalizaciones gramaticales adecuadas manteniendo su tratabilidad computacional. Además, la adopción de un modelo gramatical tan extendido como la LFG ofrece la ventaja de acceder directamente a un importante cuerpo de descripción e investigación casi directamente aplicable al sistema computacional desarrollado (Cf. Abaitua 1988 para el euskara).

La LFG asigna dos estructuras a las oraciones. Una representa la configuración superficial de constituyentes (**estructura-c**). La otra describe las dependencias funcionales (**estructura-f**). Las estructuras-c se generan por medio de una gramática de estructura de frase, cuyas reglas están anotadas con ecuaciones funcionales con las que se construyen las estructuras-f. Las ecuaciones funcionales también aparecen en las entradas léxicas.

Por ejemplo, para analizar una oración como (1):

(1) txakurrak egunkaria ahoan zekarren
perro-subj diario-obj boca-loc traía
'el perro traía el diario en la boca'

Necesitamos una gramática con la regla inicial:[3]

(2) O --> *X / (u:d:caso=d)
SV / (u=d).

[3]. Este y los siguientes ejemplos respetan la notación de representación de reglas propia de LFG-LAB (que está condicionada por la sintaxis de los términos en Prolog). Esta notación se corresponde con la de Kaplan & Bresnan 1982. De esta forma, las metavariables '↑' y '↓' (que se refieren respectivamente al nudo madre de la regla y a la hija a la que se encuentra asociada la ecuación) se trascribirán respectivamente como 'u' (abreviatura de *up*) y 'd' (por *down*). Las secuencias de rasgos se representan por medio del operador ':'. Así la regla (2) es transcripción literal de

La regla (2) indica que una oración puede estar formada por una serie de elementos X (nominales y adverbiales), seguidos por un elemento verbal, SV. La ecuación funcional (u=d) indica que la estructura funcional parcial que tenga asignada el verbo en el léxico lo será a su vez de toda la oración. Esto es, establece que el verbo es el núcleo funcional de la oración. Por otro lado, la ecuación (u:d:caso=d) establece que cada elemento X identifica su subestructura según el valor que le corresponda por su caso morfológico.[4]

Un elemento terminal lleva aparejadas ecuaciones funcionales como:

```
(3)  zekarren = V /  (u:pred = traer -{subj,obj},
                      u:temp = pasado,
                      u:subj:num = singular,
                      u:obj:num  = singular).
```

La notación empleada en (3) se puede representar también en forma de estructuras complejas de rasgos:

(4) a. zekarren

$$\begin{bmatrix} \text{PRED} & \text{'traer'} \\ \text{TEMP} & \text{pasado} \\ \text{SUBJ} & \begin{bmatrix} \text{NUM} & \text{singular} \end{bmatrix} \\ \text{OBJ} & \begin{bmatrix} \text{NUM} & \text{singular} \end{bmatrix} \end{bmatrix}$$

b. txakurrak

$$\begin{bmatrix} \text{PRED} & \text{'perro'} \\ \text{CASO} & \text{subj} \\ \text{NUM} & \text{singular} \end{bmatrix}$$

c. egunkaria

$$\begin{bmatrix} \text{PRED} & \text{'diario'} \\ \text{CASO} & \text{obj} \\ \text{NUM} & \text{singular} \end{bmatrix}$$

[4] En LFG el procedimiento se conoce como codificación **no configuracional** de funciones gramaticales. Es

d. ahoan

```
[ PRED 'boca'
  CASO loc
  NUM  singular ]
```

Combinando las subestructuras de (4) mediante la regla (2) se obtiene la estructura-F (5) para la oración (1):

(5)

```
[ PRED 'traer'
  TEMP pasado
  SUBJ [ PRED 'perro'
         NUM  singular
         CASO subj ]
  OBJ  [ PRED 'diario'
         NUM  singular
         CASO obj ]
  LOC  [ PRED 'boca'
         NUM  singular
         CASO loc ] ]
```

El procedimiento para obtener (5) ha sido la unificación de subestructuras funcionales. Estas se describen mediante ecuaciones funcionales del tipo

(6) (METAVARIABLE:ATRIB = VALOR)

donde ATRIB es un elemento atómico o una secuencia (en inglés *path*) de atributos. Para el concepto de metavariable, v. nota 3.

Para elaborar gramáticas complejas, LFG-LAB dispone de un amplio sistema de tipos de ecuaciones:

• *Negación*

(7) (~METAVARIABLE:ATRIB = VALOR) o (~METAVARIABLE:ATRIB)

subordinadas no personales llevan asociada la ecuación (~u:fin).

• *Constricción*

(8) (METAVARIABLE:ATRIB == VALOR)

Estas ecuaciones comprueban la presencia de un rasgo en una subestructura, sin añadirlo. Puede usarse, por ejemplo, para señalar que el sujeto de ciertos verbos debe ser animado: (u:subj:clase == anim).

• *Pertenencia a conjunto*

(9) (d in u:CONJUNTO)

Estas ecuaciones son necesarias para dar cuenta de la necesidad de un número múltiple de adjuntos modificando a un predicado y para tratar las construcciones coordinadas (cf. 11).

Fuera del léxico, las ecuaciones pueden agruparse en series disyuntivas, mediante la notación

(10) (ECUACION ; ECUACION)

Esta solución establece diversas alternativas para un mismo nudo. Por ejemplo, la regla (11) (de Kaplan y Bresnan, pág. 228) determina que un SP se puede asignar como complemento o adjunto de una oración:

```
(11) O --> ...,
           SP / ( u:d:caso = d ;
                  d in u:adjuntos ),
           ...
```

Pueden consultarse más ejemplos en la gramática de euskara del apéndice.

Es igualmente posible marcar como opcionales ecuaciones y series de ecuaciones, mediante el operador '&'. Cuando fallan, estas ecuaciones no invalidan el nudo al que se asocian, y pueden utilizarse, por tanto, para introducir información defectiva en las estructuras-F.

Por último, LFG-LAB dispone de la posibilidad de abreviar por medio de **macros** las secuencias de ecuaciones que suelen repetirse en las entradas léxicas y reglas

gramaticales.[5] Por ejemplo:

(12) @absga = (u:caso=abs, u:num=sing, u:animado=(+)).

Podemos eliminar en cualquier regla la referencia a estas ecuaciones, substituy
dolas por la referencia a la macro. Así, la entrada léxica del substantivo *sem*
puede simplificarse como:

(13) semea = n / (u:pred=hijo, @absga).

Las macros pueden contener a su vez referencias a otras macros.

La ventaja más importante que ofrece el uso de macros es simplificar grandemer
las reglas y las entradas léxicas, faciitando asf el mantenimiento y actualización
la gramática.

2. Objetivos de LFG-LAB

2.1. Análisis con BUP

El desarrollo de LFG-LAB ha estado guiado por varias consideraciones teóricas
prácticas a la vez. La principal ha sido desarrollar un sistema para la escritura
gramáticas LFG que fuera puramente declarativo e independiente de la impleme
tación. Asf, LFG-LAB separa de forma tajante los datos (reglas gramaticales
entradas léxicas) y los procedimientos que los utilizan en el parser. Con
consecuencia, el mantenimiento de las gramáticas resulta mucho más simple q
si hubieran sido escritas directamente en Prolog (DCG) o Lisp.

Interesaba además que el sistema fuera aplicable a lenguas tipológicamen
diversas. Por ello se decidió utilizar el parser ascendente (*Bottom Up Parser*: BU
de Matsumoto *et al.* (1983, 1985). Esta estrategia es la más adecuada para trat
las construcciones recursivas a la izquierda (propias de las lenguas de núcleo
final, como el euskara o el japonés), sin caer en bucles infinitos.

En BUP las reglas se expresan como relaciones entre la primera hija de una reg
y los otras hijas y la madre. Para comparar una regla de BUP con una regla
estructura de frase (EF), considérese el siguiente par de reglas:

(a) Regla EF: O --> SN, SV.
(b) Regla BUP: SN => goal(SV), O.

5 –

La primera regla puede parafrasarse como, "si quieres encontrar una O, busca un SN seguido de SV", mientras que la segunda se leería, "encontrar un SN nos hace esperar un SV, en cuyo caso, el constituyente entero será O".

Un aspecto importante de BUP es que combina, en cada etapa del análisis, facetas de tratamiento ascendente y descendente. Esto significa que cuando se invoca una llamada a **goal** (v. 2), podremos comprobar que es coherente con lo que ya sabemos del constituyente buscado. A tal fín, se consulta una tabla predefinida de alcanzabilidad (cf. Kay 1980). En BUP esta tabla tiene la forma de una serie de relaciones **link**, que determinan, para una categoría dada, qué otras categorías pueden ser su primer constituyente.

Por último, las consideraciones de eficiencia aconsejaron definir un parser adaptado a la idiosincrasia del Prolog y que redujera al máximo la carga de interpretación inherente al operar con reglas declarativas. Esto se logró incluyendo procedimientos de evaluación parcial en el compilador (Takeuchi y Furukawa 1985; v. Pereira y Shieber 1986, §6.4).

2.2. Panorámica de LFG-LAB

El análisis en LFG consiste tradicionalmente en el método denominado de "sobregeneración y filtrado": en una primera etapa, se construyen todas las estructuras-c correspondientes a una expresión. De ellas surgen a continuación las estructuras-f por la resolución de las ecuaciones funcionales. Entonces algunas estructuras-c quedarán eliminadas al fallar la resolución. Este procedimiento es computacionalmente poco eficiente, y no sorprende que la mayoría de las implementaciones reales de LFG (Yasukawa 1984, Block y Hunze 1986 y otros) prefieran un procesamiento paralelo e incremental de la estructura-c y la estructura-f, descartando los análisis incorrectos en cuanto son detectados. LFG-LAB utiliza la misma técnica: cuando aplica una regla y reconoce una de las hijas, resuelve inmediatamente las ecuaciones que tiene asociadas. Si el procedimiento falla, se abandona la regla en favor de otra alternativa. Obsérvese que esta estrategia reduce la estructura-c a un mero auxiliar del parser.

La operación básica del parser es la **unificación** (Kay 1985, Shieber 1986, Knight 1989) de dos estructuras-f.
La unificación de Prolog no resulta adecuada para nuestros propósitos, ya que opera sobre términos de aridad fija, mientras que las estructuras-f son per se de extensión variable y orden interno indefinido. Esto obliga a representar las estructuras-f como listas de rasgos, y a suplementar el mecanismo normal de unificación en Prolog con un nuevo predicado (**unify**) que realice la unificación de listas. En LFG-LAB adopta la definición de Gazdar y Mellish (1989):

```
unify( Ef, Ef ) :- !.

unify( [ Atributo:Valor | Ef1 ], Ef ) :-
        pathval( Ef, Atributo, Valor, Ef2),
        unify( Ef1, Ef2 ).
```

La base de este procedimiento de unificación es la idea de listas incompletas o abiertas (esto es, listas cuya cola es una variable anónima). La unificación de dos estructuras-f se produce insertando en ambas estructuras los rasgos que faltan respecto a la otra, y unificando las variables en PROLOG. Cuando los rasgos son complejos la unificación tiene lugar recursivamente.
El predicado **pathval** verifica la presencia de cierto rasgo en la estructura-f o lo añade, si no está presente. Por ejemplo, obsérvese la conducta de **pathval** sobre la estructura X = [num:pl, caso:abs | _]:

```
?- pathval(X, caso, B, _).

        B = abs

?- pathval(X, animado, +, _).

        X = [ num:pl, caso:abs, animado:+ | _ ]
```

Para tratar la unificación de constricciones, negación y conjuntos, ha sido preciso extender el procedimiento. Las constricciones se tratan como los demás rasgos, aunque su valor es una estructura de PROLOG con la forma c(**Estructura_F**, **Marca**). La marca se instancia en caso de unificación (v. *infra*, a); al final del análisis, las constricciones que contengan marcas sin instanciar son descartadas.

De forma similar, las ecuaciones negativas se compilan como rasgos cuyo valor es una estructura **u**(V, NEG), que no podrá unificar con un rasgo cuyo valor sea análogo a NEG. En otro caso, el procedimiento tendrá éxito (v. *infra*, b).

La unificación de conjuntos se efectúa por medio de un predicado **append_c** (una versión destructiva de **append** que funciona con listas abiertas, y añade en la cola anónima de la primera lista el contenido de la segunda) (v. *infra*, c).

a.
```
unify(X,c(X,'+')) :- !.
unify(c(X,'+'),X) :- !.
```

b.
```
unify( X, u(Y,N) ) :- var(Y),X==N,!,fail.
unify( u(Y,N), X ) :- var(Y),X==N,!,fail.
unify(X, u(X,_)) :- !.
unify( u(X,_), X) :- !.
```

c.
```
unify({X},{Y}) :- !,append_c(X,Y).
```

términos: permite al usuario pasar por alto la estructura completa del término y la posición que cierto rasgo ocupa en su interior (Kay 1985, pág. 235). Estos problemas son muy comunes cuando se pretende desarrollar sistemas extensos en un formalismo como las DCGs.

La verificación de consistencia de las estructuras-F se lleva a cabo durante el proceso de análisis, haciendo un uso exhaustivo de la unificación.[6] La idea consiste en expandir la forma semántica de los predicados -que da cuenta de su subcategorización funcional- por una serie de rasgos positivos (para las funciones exigidas) y negativos (para las restantes funciones argumentales). Por ejemplo, la compilación de la entrada léxica para *ikusi* (14) produce la cláusula de Prolog que se expone a continuación:

(14) ikusi = n / (u:pred=ikusi - {subj,obj}).

(15)
```
word( ikusi, n, X ) :-
        pathval(X,pred,ikusi,_),
        pathval(X,subj,_,_),
        pathval(X,obj,_,_),
        pathval(X,obj2,u,_),
        pathval(X,comp,u,_),
        pathval(X,xcomp,u,_).
```

(Obsérvese que el valor 'u' de al serie de atributos funcionales en (6) impide la unificación con cualquier otro valor, y equivale, por tanto, a la negación absoluta de la presencia de la correspondiente función gramatical.)

Los casos de movimiento (o más propiamente, de dependencias no locales, como las que se dan entre un pronombre relativo o interrogativo extrapuesto y su posición controlada) se tratan en LFG-LAB mediante una versión del mecanismo de **incertidumbre funcional** de Kaplan y Zaenen (1989). Básicamente consiste en definir todas las secuencias de atributos de estructura-F que pueden vincular la estructura del elemento extrapuesto y la del controlado. La secuencia se describe como una expresión regular de la forma **'FI .. FF'** (donde **FF** especifica las funciones finales que puede ocupar el elemento extrapuesto, y **FI** indica las funciones intermedias - generalmente COMP). Por ejemplo:

(16)
```
O  --> QU / (u:topic=d)
       SV / (u=d,
             d:comp* FG = u:topic).
```

(COMP* indica una secuencia no local de COMP; la abreviatura FG se utiliza para referirse a cualquier función final de la lengua en cuestión. Pude usarse también la fórmula {A,B,...,Z} para indicar una serie disyuntiva de funciones).

La implementación de la incertidumbre funcional utiliza dos listas, una de funciones intermedias (**LFI**) y otra de funciones finales (**LFF**). Consiste en un procedimiento especial que intenta primero adscribir el elemento desplazado en la predicación local, de acuerdo con las funciones de LFF. Si esto falla, el sistema busca una subestructura-F cuyo atributo sea una función de LFI, y aplica recursivamente el procedimiento, hasta dar con la posición controlada del elemento desplazado.

3. Detalles de implementación

LFG-LAB se encuentra implementado en C-Prolog (entorno UNIX) y en PROLOG-2 (entorno PC).

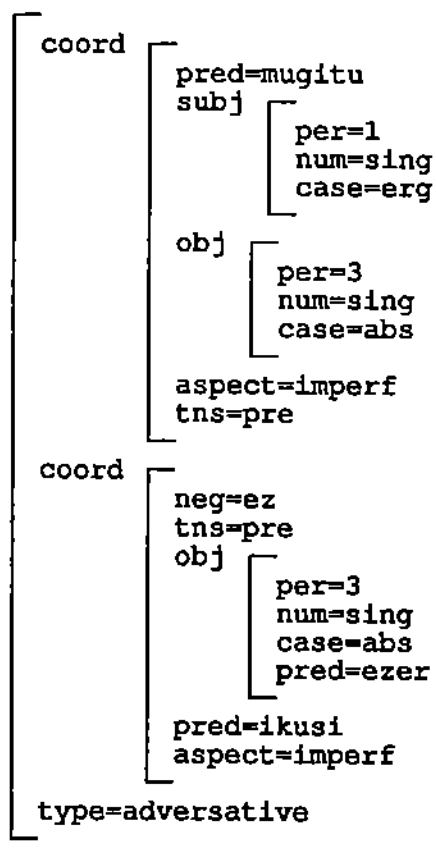
El resto de esta sección presenta una sesión típica de trabajo con LFG-LAB. Los tiempos referidos corresponden a la versión para UNIX, sobre SUN/4.

```
> argazki bat manualki atera-tzeko zer egin behar dut

Estructura-F:

┌
│ fin ┌
│     │ comp ┌
│     │      │ obj ┌
│     │      │     │ pred=argazki
│     │      │     │ det=bat
│     │      │     │ case=abs
│     │      │     │ num=sing
│     │      │     └
│     │      │ manner ┌
│     │      │        │ pred=manual
│     │      │        │ case=manner
│     │      │        └
│     │      │ pred=atera
│     │      │ aspect=perf
│     │      └
│     │ pred=fin
│     │ case=fin
│     └
│ focus ┌
│       │ pred=zer
│       │ case=abs
│       │ per=3
│       │ num=sing
│       └
│ obj ┌
│     │ pred=zer
│     │ case=abs
│     │ per=3
│     │ num=sing
│     └
│ pred=egin
│ subj ┌
│      │ per=1
│      │ num=sing
│      │ case=erg
│      └
│ aspect=imperf
│ modal=behar
└
```

Tiempo : 0.39 seg.

> *Mugitzen dut baina ez da ezer ikusten*

Estructura-F:



Tiempo : 0.53 seg.

```
> Bale, orduan, goazen ikusten adierazten du-en ikur bakoitzak
Estructura-F:
[marker [type=confirmation]
 marker [type=deduction]
 vaux=joan
 pred=ikusi
 subj=u
 obj [mod [pred=adierazten
           subj [per=3
                 num=sing
                 case=erg]
           obj [per=3
                num=sing
                case=abs]
           aspect=imperf
           tns=pre
           mod=rel]
      head [pred=ikur
            det=bakoitza
            case=erg
            num=sing]
      case=abs]
 aspect=imperf]

Tiempo : 0.62 seg.
```

4. Conclusión

Hemos presentado LFG-LAB, un sistema para desarrollo y verificación de gramáticas desde el punto de vista del formalismo LFG. Sobre otras implementaciones de la teoría (Yasukawa, Eisele y Dörre), LFG-LAB ofrece varias ventajas: en primer lugar, estar integrada con un compilador a BUP que garantiza capacidad expresiva (al vencer la conocida limitación de las DCGs sobre las reglas recursivas a la izquierda), y lo que no es menos importante, eficiencia computacional. Además, LFG incluye una versión del mecanismo de incertidumbre funcional, y un tratamiento parcial -pero suficiente para la mayor parte de las necesidades- de ecuaciones negativas.

Referencias

J. Abaitua (1985) *An LFG Parser for Basque*. Universidad de Manchester. Tesis de MSc.
_____ (1988) *Complex Predicates in Basque: from Lexical forms to Functional Structures*. Universidad de Manchester, Tesis de doctorado.

J. Allen y D.J. Litman (1987) "A Plan Recognition Model for Subdialogues in Conversations". *Cognitive Science*, vol. 11.
J. Austin (1962) *How to do things with words*. J.O. Urmson, ed. Oxford University Press.
H.U. Block y R. Kunze (1986) "Incremental Construction of F-Structure in a LFG-parser". COLING.

J. Calder, E. Klein y H. Zeevat (1988) "Unification Categorial Grammar: A concise, extendable Grammar for Natural Language Processing", COLING, 1988, pp. 83-86.

J. Carroll y J. Abaitua (1987) "A Morphological Parser for Basque Verbs' Inflection". En *Proc. of the II World Basque Congress. International Conference on Artificial Intelligence*. Donostia, pp. 131-143.
A. Eisele y J. Dörre (1986) "A Lexical Functional Grammar System in Prolog", COLING.

W. Frey y U. Reyle (1983) "A PROLOG Implementation of Lexical Functional Grammar as a Base for a Natural Language Processing System". En *Proceedings of the Ist Meeting of the Association for Computational Linguistics*, Pisa.
W. Frey (1985) "Noun phrases in Lexical Functional Grammar". En V. Dahl y P. Saint-Dizier (eds.) *Natural Language Understanding and Logic Programming*, vol. I, North-Holland.

G. Gazdar y Ch. Mellish (1989) *Natural Language Processing in PROLOG*. Addison-Wesley.

R. Kaplan y J. Bresnan (1982) "Lexical-Functional Grammar: A Formal System for Grammatical Representation". En J. Bresnan (ed.) *The Mental Representation of Grammatical Relations*, MIT Press.
R. Kaplan y A. Zaenen (1989) "Long Distance Dependencies, Constituent Structure, and Functional Uncertainty". En M. R. Baltin y A. S. Kroch (eds.) *Alternative Conceptions of Phrase Structure*, The University of Chicago Press, pp. 17-42.

M. Kay (1985) "Unification in Grammar". En V. Dahl y P. Saint-Dizier (eds.) *Natural Language Understanding and Logic Programming*, vol. I, North-Holland.

K. Knight (1989) "Unification: A multidisciplinary Survey". *ACM Computing Surveys*, 21, 1.

Y. Matsumoto, H. Tanaka, H. Hirakawa, H. Miyoshi y H. Yasukawa (1983) "BUP: A Bottom-Up Parser Embedded in Prolog". *New Generation Computing*, 1.
Y. Matsumoto, M. Kiyono, y H. Tanaka (1985) "Facilities of the BUP Parsing System". En V. Dahl y P. Saint-Dizier (eds.) *Natural Language Understanding and Logic Programming*, vol. I, North-Holland.

F. Pereira y S. Shieber (1986) *PROLOG and Natural Language Analysis*. CSLI, Stanford.

P. Sells (1985) *Lectures on Contemporary Syntactic Theories*. CSLI, Stanford.

S. Shieber (1985) "Criteria for designing computer facilities for linguistic analysis". *Linguistics*, 23, pp. 189-211.
_____ (1986) *An Introduction to Unification-based approaches to Grammar*. CSLI, Stanford.

A. Takeuchi y K. Furukawa (1985) "Partial Evaluation of Prolog Programs and its Application to Metaprogramming". ICOT, Tokyo.

H. Yasukawa (1984) "LFG systems in PROLOG". En Proceedings of the 11th International Conference on Computational Linguistics (COLING)

Apéndice

Esta gramática de euskara trata actualmente una gama de fenómenos que incluye entre otros cláusulas relativas, negativas, coordinación y subordinación.

En las siguientes reglas la opcionalidad de los constituyentes se marca mediante el operador '&'. Los constituyentes sin ecuaciones funcionales asumen por defecto la ecuación (u=d).

```
    /*****************************
     **        Macros           **
     *****************************/

@mdv =  ( (u:focus:case=erg, u:comp .. subj = u:focus ) ;
          (u:focus:case=abs, u:comp .. (obj,subj) = u:focus) ;
          (u:focus:case=dat, u:comp .. obj2 = u:focus ) ;
          (u:focus:case=C, u:comp .. C = u:focus ) ).

func = [subj,obj,obj2,comp,xcomp].

    /*****************************
     **     Erregelak           **
     *****************************/

s --> pp, &pnt.

pp --> *sx,
        sv,
       *sx.

pp -->  sl / (u:d:case=d),
        sv,
       *sx.

pp -->  *sx,
         sv,
         sl / (u:d:case=d).

pp -->  cw / (d in u:marker),            /* barkatu */
        pp.

pp -->   pp / (d in u: coord),
         conj,
         pp / (d in u: coord).

sl --> s_inf / (u:comp=d),               /* atera */
       vsuf.                             /* tzeko */

/* sl -->    conj,
             s  / (u:comp=d). */

s_inf --> *sx,
           v.

s_rel --> *sx,
           sv,
           rel.

sx --> sn / ( (u:obj=d,d:case=abs) ;
              (u:subj=d,d:case=erg) ;
              (u:obj2=d,d:case=dat) ).

sx --> sadv / (u:d:case=d).              /* manualki */

sx --> sp  / (u:d:case=d).


sp -->  sn / (u:obj=d),                  /* pantaila */
         p.                              /* n */

sp -->  sa,                              /* pantaila-ren */
```

```
sn --> sa / (u:gen=d),                          /* pantaila  ren */
       sn.                                      /* MODE a */

sn --> s_rel / (u:mod=d),
       sn    / (u:head=d).


sn -->  sn   / (d in u: coord, d:case=u:case),
        conj,
        sn   / (d in u: coord, d:case=u:case).

sa -->  sn / (u:obj=d) ,                        /* pantaila */
      gen.                                      /* ren      */

sn -->   n,
      &det.

sadv --> adv.

sv -->   &gg  /  (u:focus=d, @mov),
         vl,                                    /* egin behar */
         &neg,                                  /* ez  */
         &aux.                                  /* dut */

sv -->   neg,                                   /* ez */
         aux,                                   /* dut */
         &sx,                                   /* ezer */
         vl.                                    /* egin behar */

sv -->   gg   / (u:focus=d, @mov),              /* zer */
         neg,                                   /* ez  */
         aux,                                   /* dut */
         vl.                                    /* egin behar */

vl -->  v,                                      /* atera */
      &vaux,                                    /* egin */
      &vmod.                                    /* nahi */

vl -->  vaux  / (u=d, d:vaux=joan),               /* goazen */
           v  / (u=d, d:aspect=imperf).           /* ikusten */
```