Módulo de generación en JULIETTA

Gabriel Amores Carredano
Departamento de Lengua Inglesa
Universidad de Sevilla
e-mail: gamores@sevax4.us.es

0. Introducción

En anteriores presentaciones (Amores Carredano 1993) he descrito el prototipo de traducción automática (TA) JULIETTA, un sistema basado en la gramática léxico funcional (LFG) implementado en Prolog.

En esas ocasiones nos ocupamos de describir la arquitectura general del sistema, su cobertura lingüística y el algoritmo de transferencia.

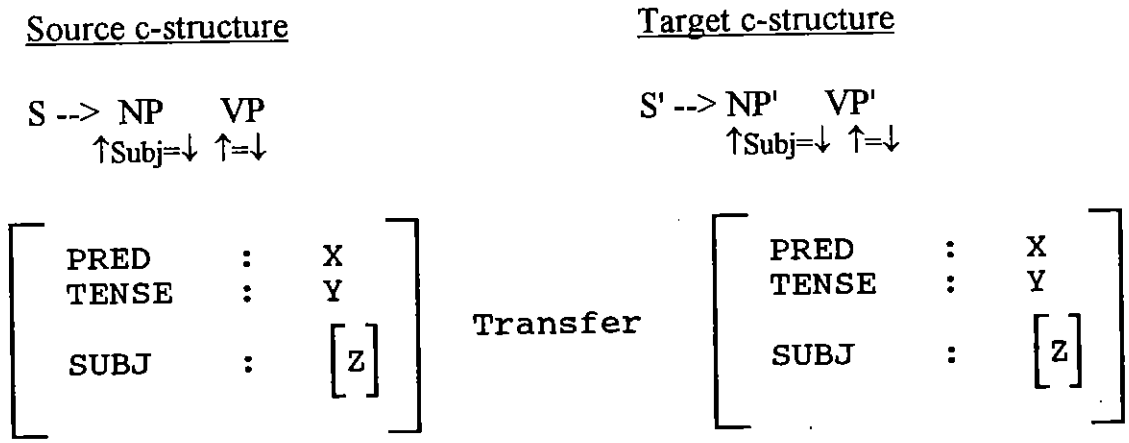
En este artículo nos ceñiremos al módulo de generación.

1. Fases de la generación

Dentro de la arquitectura general del sistema, el módulo de generación toma una representación de la oración en lengua destino y genera la correspondiente cadena de palabras.

En el caso de JULIETTA, la representación que se obtiene tras la transferencia es una estructura funcional (f-structure en LFG) para el español. De esta estructura funcional, siguiendo el camino inverso del análisis, se debe generar primero la estructura de constituyentes que le corresponde (c-structure en LFG) y después la cadena de palabras.

La figura 1 ilustra esta derivación.



Source f-structure Target f-structure

ANALYSIS ------> TRANSFER -----> GENERATION

figura 1

2. Generación desde estructuras funcionales

El concepto de generación en el Procesamiento del Lenguaje Natural varía de una aplicación a otra. En sistemas de TA que siguen el enfoque de transferencia se asume que la selección léxica ya ha tenido lugar durante la transferencia y que la labor del generador consiste en la generación morfológica y ordenación de los constituyentes siguiendo una gramática de la lengua destino.

El desarrollo de algoritmos de generación para gramáticas basadas en unificación es un campo relativamente reciente, ya que casi todas las implementaciones se habían ceñido a problemas de análisis y/o transferencia.

Por lo que se refiere a LFG, Momma y Dörre (1987), Netter y Wedekind (1986), Wedekind (1986), y Gates y otros (1989) proponen algoritmos de generación desde estructuras funcionales, mientras que Calder y otros (1989) y Shieber y otros (1990) ofrecen ofrecen un algoritmo general de generación aplicable a gramáticas de unificación. Asimismo, Ruiz Antón (1993) hace una propuesta de generación desde estructuras funcionales aplicando ideas de la gramática funcional.

Nuestro algoritmo sigue la propuesta de Momma y Dörre (1987) y Gates y otros (1989).

Podemos definir el problema de la generación como el *descubrimiento de un árbol de análisis* para una estructura funcional dada como insumo (**input**). Este proceso encierra varios problemas.

Primero, la generación es, por definición, no determinística, ya que se puede generar un número infinito de árboles posibles para una estructura funcional dada.

Considérese, por ejemplo, la siguiente estructura funcional parcial, de la cual se quiere generar la cadena *la presión sanguínea sistólica*. Si nuestro generador opera con prioridad de profundidad (**depth-first**), no hay manera de asegurar que obtendremos el resultado deseado ya que se pueden generar varios árboles de los que resultarían, al menos, las cadenas de palabras que se muestran a la derecha de la representación.

pred:presión

```
spec:el
mods:pred:sistólico
      temp:no
      tail:pred:sanguíneo
agr:gen:fem
   num:sing
```

presión, la presión, presión sanguínea, la presión sanguínea, presión sistólica, la presión sistólica, presión sanguínea sistólica, la presión sanguínea sistólica

figura 2

Este hecho descarta en principio un algoritmo con prioridad de profundidad, como ha sido demostrado también por Momma y Dörre (1987) y Shieber y otros (1990).

El segundo problema surge del tipo de representación que codifica una estructura funcional en LFG. La estructura funcional carece de información categorial y de información acerca del orden en que aparecen los constituyentes. Se supone que este tipo de información, al ser más dependiente de la lengua, no debe estar presente en la estructura funcional y se descarta durante el análisis. Esto ya fue apuntado por Kaplan y otros (1989) como argumento para defender un enfoque de **co-descripción** para sistemas de TA basados en LFG.

Otras gramáticas basadas en unificación, como la **Functional Unification Grammar** (FUG), (Kay 1985), obvian este problema incluyendo información categorial y de precedencia lineal en la estructura funcional[1].

Así, por ejemplo, la estructura funcional parcial anterior tendría el aspecto siguiente en una representación basada en FUG.

```
pred:presión
pattern:<spec,pred,mods>
cat:noun
spec:el
mods:pred:sistólico
        pattern:<tail,pred>
        cat:adj
        temp:no
        tail:pred:sanguíneo
            cat:adj
 agr:gen:fem
     num:sing
```

figura 3

[1] En FUG, el atributo **pattern** se usa para indicar precedencia lineal. Es interesante hacer notar que FUG fue la única gramática de unificación con una aplicación directa a la TA. Tal vez la inclusión de esta información en la estructura funcional se viera necesaria como un requisito impuesto por la TA.

Wedekind (1986) y Netter y Wedekind (1986), también en un entorno de TA, proponen una representación de un solo nivel (frente a los dos niveles tradicionales en LFG) para solventar el problema.

Las estructuras de su versión consisten en estructuras funcionales aumentadas con información adicional acerca de los símbolos derivados y su orden lineal. A continuación proponen un concepto de **derivación** que puede usarse tanto en modo análisis como en modo generación, con idea de que las gramáticas sean reversibles. Si se usan en generación, la estructura funcional insumo lidera el proceso y las reglas libres de contexto se satisfacen si las ecuaciones anotadas propias de LFG están realizadas con material no vacío en la estructura parcial que se está generando.

Sin embargo, no mencionan nada acerca del no determinismo del proceso de generación, ya que sólo comprueban la existencia de valores no vacíos (ya sean de tipo atómico o complejo) en los rasgos relevantes.

Momma y Dörre (1986) solucionan el problema imponiendo un algoritmo con prioridad de extensión (**breadth-first**) a la satisfacción del proceso de generación. Así, en vez de proceder simplemente en modo descendente (**top-down**), el algoritmo ha de comprobar que **toda la información** contenida en la estructura funcional se ha consumido en el proceso. De esta forma se asegura que se intenta primero la regla de generación más larga.

3. Nuestra propuesta

Si aplicamos estas ideas a nuestro ejemplo de estructura funcional parcial, el generador observará la presencia de un especificador (atributo **spec**), un nombre (atributo **pred** en este caso) y un modificador complejo (atributo **mods**), y tratará de aplicar la regla que tiene en cuenta a los tres, y sólo a los tres, en dos llamadas recursivas.

En JULIETTA, cada regla de generación toma como primer argumento una estructura funcional y devuelve como salida (**output**) la porción de árbol que esta regla genera. Por ejemplo, las reglas que se necesitan para generar el ejemplo anterior serían las siguientes.

```
generate(EstructuraF,sn(Art,Stbar)) :-
      ( member(spec:_,EstructuraF)
      ; member(quant:_,EstructuraF)
      ; member(det:_,EstructuraF)
      ; member(dem:_,EstructuraF) ),
      generate_rs(EstructuraF,Art,EstructuraF1),
      generate_stbar(EstructuraF1,Stbar).
```

```
generate_stbar(F,stbar(st(St),Sadj)) :-
        getargs(EstructuraF,List),
        perm(List,[agr,X]),
        member(X,[mod,mods,vmod]),
        member(X:Mod,EstructuraF),
        \+ member(stayput:yes,Mod),
        member(agr:A,EstructuraF),
        gener_sadj([agr:A|Mod],Sadj),
        delete(X:Mod,EstructuraF,EstructuraF1),
        reduce(EstructuraF1,st,St).
```

figura 4

La primera regla comprueba la presencia de cualquier especificador, y aplica recursivamente la regla de generación de sintagmas determinantes (**generate_rs/2**). Este predicado devuelve la estructura funcional original sin el especificador, y **generate_stbar/2** se aplica para obtener un sintagma nominal de un nivel jerárquico inferior.

La estrategia con prioridad de extensión (**breadth-first**) opera de la siguiente manera: el predicado **getargs/2** devuelve una lista con todos los valores no atómicos en la estructura funcional. En el ejemplo anterior instanciaría **List** a la lista [mods,agr]. **perm(ute)/2** comprueba que estamos tratando con los valores pertinentes, ya que los rasgos en la estructura funcional no vienen en un orden pre-establecido. Se puede asimismo observar que esta regla se utiliza para generar sintagmas nominales modificados por otro tipo de material, concretamente **mod**, **mods** o **vmod**.

De hecho existe cierto isomorfismo entre la estructura de constituyentes que se generará y el tipo de rasgo con el que estamos tratando. Así, **pobj**, **pmod**, **padj** y **padj1** generan todos sintagmas preposicionales, y así sucesivamente.

Finalmente, si todos los predicados tuvieron éxito, se hace una llamada al generador morfológico para sintetizar la forma apropiada del sustantivo, verbo, adjetivo, etc.
Nótese que los predicados más costosos computacionalmente hablando (la recursión sobre reglas de generación y el generador morfológico) sólo se invocan si se cumplen las restricciones impuestas a la regla de generación.

4. El generador morfológico

El generador morfológico está controlado por el predicado **reduce/3**. Éste toma como insumo una lista de pares atributo-valor y genera el lexema correspondiente teniendo en cuenta la categoría deseada. Un ejemplo de llamada sería el siguiente

```
?- reduce([pred:niño,agr:[gen:masc,num:plur]],st,L).
L = niños
yes
```

La versión actual del generador morfológico sintetiza todas las formas regulares de los verbos, sustantivos y adjetivos. Las irregularidades han de ser declaradas como excepciones a la base de datos en Prolog. No obstante, no es necesario contar con un diccionario en lengua destino. Toda la información léxica necesaria para la generación se encuentra a la salida de la transferencia.

Por último, una vez que la regla superior **top_generate/3** ha generado el nodo más alto de la gramática, este término se pasa a una gramática DCG del español, que convertirá directamente el árbol de análisis en una lista de palabras. Esto es posible gracias al caracter reversible de los predicados en Prolog.

Referencias

Amores Carredano, J.G. (1993) "JULIETTA: Un prototipo de TA basado en LFG" Procesamiento del Lenguaje Natural **13**, pp. 305-317

Calder, J., M. Reape y H. Zeevat (1989) "An Algorithm for Generation in Unification Categorial Grammar" 4th Conference of the European Chapter of the ACL. Manchester, pp. 233-240.

Gates, D., K. Takeda, T. Mitamura, L. Levin y M. Kee. (1989) "Analysis and Generation Grammars." Machine Translation **4 (1)** Special Issue on Knowledge-Based Machine Translation. pp. 53-66.

Kay, M. (1985) "Parsing in Functional Unification Grammar." En Dowty, D.R., L. Karttunen and A. Zwicky eds. Natural Language Parsing. Cambridge University Press, Cambridge, pp. 251-278.

Momma, S. y J. Dörre (1987) "Generation from f-structures." En Klein and van Benthem eds. Categories, Polymorphism and Unification. Centre for Cognitive Science, University of Edinburgh, Edinburgh, 1987, pp. 148-167.

Netter, K. y J. Wedekind (1986) "An LFG-based Approach to Machine Translation." Proceedings of IAI-MT86, Dudweiler, Germany, pp. 199-209.

Ruiz Antón, J.C. (1993) "Un tratamiento funcional de la síntesis (en traducción automática)" Procesamiento del Lenguaje Natural **13**, pp. 349-359.

Shieber, S., F.C.N. Pereira, G. van Noord y R.C. Moore (1990) "Semantic-Head-Driven Generation" Computational Linguistics **16 (1)**, pp. 30-42.

Wedekind, J. (1986) "A Concept of Derivation for LFG." En Coling 86. Bonn, pp. 487-489.