UN ANALIZADOR SINTACTICO EFICIENTE PARA FRASES DE SIMBOLOS ALTERNATIVOS.

J. Oncina, F. Casacuberta, J. M. Benedi.
Dpto. de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia.

RESUMEN

Los analizadores sintácticos generales para gramáticas independientes del contexto son algoritmos de alta complejidad temporal, y por ello de difícil utilización en algunas aplicaciones. En este trabajo se han realizado dos extensiones de un algoritmo mas eficiente que los clásicos, introducido recientemente, para gramáticas estocásticas y para frases de símbolos alternativos. El objetivo final de estos trabajos es su utilización en Reconocimiento Automático del Habla (RAH).

1.-INTRODUCCION

Las Gramáticas Independientes del Contexto (GIC) son modelos matemáticos muy utilizados para la representación del conocimiento en el tratamiento del Lenguaje Natural (Tomita,85,86,87) (Barr,81), en muchos sistemas de Reconocimiento de Formas (Gonzalez,78) (Fu,82) (Cheng,86) (Juhola,86) (Mohr,86) y particularmente en Reconocimiento Automático del Habla (Pierrel,81) (Levinson,85) (Deroulaut,86) (Tomita,85,86,87) (Nakagawa,87) (Ney,87). Sin embargo, su utilización presenta varios problemas. En primer lugar, la obtención de las reglas de las gramáticas se realiza fundamentalmente, de forma manual, ya que son pocos los métodos existentes para la Inferencia Gramatical Automática de lenguajes independientes del contexto. Este hecho se analiza en otro trabajo (Marco,88). En segundo lugar, la complejidad temporal de los analizadores sintacticos generales para gramática independiente del contexto es $O(n^3)$ (Aho,72) (Graham,80) y no parece que pueda existir un algoritmo cuya complejidad sea inferior a $O(n^{2.81})$ (Valiant,75), por lo cual, no es aconsejable la utilización de dichos analizadores para frases cuya longitud sea grande. No obstante, para ciertas clases de gramáticas independientes del contexto existen analizadores sintácticos mas eficientes, por ejemplo, si las gramáticas no son ambigüas, tenemos un analizador cuya complejidad es $O(n^2)$ (Aho,72) y si es LR, entonces existen algoritmos de $O(n)$ (Aho,72) (Tomita,85,86,87). Mucha gramáticas independientes del contexto no son LR, por lo que los correspondientes analizadores LR no pueden aplicarse, no obstante, algunas o muchas son "casi" LR por lo que Tomita en (Tomita,85,86,87) propuso una modificación de estos analizadores con el objeto que admitiesen cualquier GIC de forma que si se "parecía" a una gramática LR, entonces su complejidad era

---

Trabajo parcialmente subvencionado por el proyecto "Reconocimiento Automático del Habla" PA85-0086 del CAICYT.

ligeramente por encima de la lineal. En la practica Tomita demostró que su algoritmo es, en promedio, mas eficiente que los analizadores sintácticos generales (Tomita,86).

En tercer lugar, la extensión de los analizadores mencionados para el análisis corrector de errores, tan necesario en Reconocimiento de Formas (Fu,82) (Gonzalez,78) hace que estos sean extremadamente costosos.

Varios autores, han propuesto un modelo alternativo (Don,85) (Casacuberta,87) (Nishimura,87) para la representación de un objeto ruidoso o con deformaciones, basados en la idea de que cada primitiva puede ser etiquetada con mas de un símbolo alternativo, con o sin una medida de evidencia asociada.

En este trabajo propondremos dos modificaciones del algoritmo de Tomita, una para que pueda analizar una frase de símbolos alternativos, que de hecho representa a un conjunto de frases que alternativamente podían etiquetar la representación del objeto analizado. No confundir esta aproximación con la propuesta por el propio Tomita (Tomita,86) en una aplicación en Reconocimiento Automático de Habla, en el que ciertas categorias lingüisticas pueden tener a la vez diversos significados y actuando en cierta forma como "comodines". La otra modificación permite la utilización de gramáticas estocasticas, y por tanto , permite obtener la probabilidad de generación de la forma analizada.

2.-GRAMATICAS LR

Las gramáticas LR(k) son una clase de las GIC que tienen la propiedad de ser analizadas de forma determinista con un algoritmo ascendente. Para que esto sea posible se exige a las gramáticas que dada una derivación por la derecha podamos determinar el pivote de cada una de las formas sentenciales por la derecha y que no-terminal remplazará al pivote conociendo hasta como máximo los k términos siguientes a su pivote (Aho,72).

La parte central de los reconocedores diseñados especialmente para gramáticas LR(k), y que tienen una complejidad O(n), está constituida por un autómata de estados finitos. Dada una gramática de este tipo definida de la forma usual como una tupla (N, T, P, S), el autómata mencionado lo podemos describir mediante un par de funciones (Aho,72):

- GOTO : $E \times \Sigma \dashrightarrow E \cup \{ Error \}$
- ACTION : $E \times N^{*k} \dashrightarrow \{ 'Reduce\ r', 'Shift', 'Accept', 'Error' \}$

donde:

E: Conjunto de todos los estados del autómata.

Σ: T U N (conjunto de no terminales y terminales de la gramática)

T^{*k}: Todas las posibles cadenas de como mucho k terminales.

r: Indica una de las reglas de la gramática.

La función GOTO nos indicará cual es el siguiente estado del autómata y la función ACTION la operación a realizar.

3.-ANALIZADOR SINTACTICO DE TOMITA

Si la gramática no es LR(1) ocurrirá que la función ACTION será multivaluada, cuando esto sucede se dice que tenemos ambigüedades.

La idea del algoritmo (Alg.1) consiste que en cuanto nos encontremos con una ambigüedad, entonces, intentaremos seguir todas las posibilidades.

Ademas, ahora el resultado ya no tiene porque ser un arbol de derivación único sino que, como puede tratarse de una gramática ambigüa, puede haber varias posibilidades. El resultado del analisis ya no puede darse en una cadena que representa al árbol de derivación, sino que lo daremos en una estructura que llamaremos arboleda, y que definiremos como un grafo dirigido sin ciclos, <V,E>, tal que un vértice puede tener más de una lista de sucesores.

Sobre la arboleda las siguientes operaciones :

- SUBNODES(v) : Toma como argumento un vértice de V y devuelve el conjunto de todas las listas de sucesores de v.
- ADDSUBNODE(v,L) : Toma como argumento un vértice y una lista de sucesores y lo añade a la arboleda.

En esta arboleda cada vértice estará etiquetado con un símbolo de la grámatica, si está etiquetado con un terminal, la lista de sucesores será vacía, si es con un no-terminal, en la lista de sucesores indicarán, en orden, los vértices donde se desarrollan cada uno de los términos de su parte derecha, si tiene varias listas de sucesores distintas indicará que hay varias fomas de interpretar la cadena.

La pila del analizador la sustituiremos por un grafo dirigido que tendrá dos tipos de vértices, los E_vértices (E_V) que corresponderán a cuando en la pila apilábamos un estado, y los A_vértices (A_V) que correponderá a cuando apilábamos términos, solo que ahora contendrán punteros a la arboleda.

Además tendremos dos listas, la A que hará las funciones de la cabeza de la pila, y la Q que será donde iremos apuntando todos los nuevos vértices que aparecerán en cada paso.

4.-ALGORITMO DE TOMITA CON GRAMATICAS ESTOCASTICAS Y PARA FRASES DE SIMBOLOS ALTERNATIVOS

Para el caso de gramáticas estocásticas, las probabilidades de las frases se calculán según las siguientes reglas :

- En una arboleda la probabilidad asociada a un nodo con una sola lista de sucesores es el producto de la probabilidad de la regla aplicada y de las probabilidades asociadas a cada uno de los símbolos hijos.
- Si se tiene más de una lista de sucesores la probabilidad será la asociada a la suma de cada una de sus listas de sucesores.

En el algoritmo sólo habrá que cambiar la parte correspondiente al Shift y al Reduce que quedarán tal como se indica en Alg.2.

En el caso del análisis de frases de símbolos alternativos, la entrada ya no es una cadena sino una cadena de conjuntos de posibles símbolos. Ahora, tendremos que añadir, a la lista de los vértices activos, que símbolos son los que pueden ir a continuación, esto implica unos cambios en el algoritmo primero que está reflejado en Alg.3.

```
Algoritmo :
     Entrada : Las dos funciones GOTO y ACTION de una GIC y la cadena de entrada z ∈ Σ*
     Salida : si z ∈ L(G) en r un puntero a la cabeza del árbol de derivación almacenado en
                    la Arboleda, sino r = NIL.
     Variables : Γ : Grafo; T : Arboleda; A,U : {x/ x ∈ V };
                 Q : {<v,s>/ v ∈ E_V,s ∈ E } ; R : {<v,p>/ v ∈ E_V,p ∈ P }
Método :
          añadir $ al final de z
          Γ := φ
          crear en Γ un vértice v0 etiquetado E0
          r := NIL ; T := φ ; U := E0
          para todo i ∈ [ 1..long(z)]
               a := Sacar Primer término de z
               A := U ; R, Q := φ
               repetir
                    si A / = φ entonces ACTOR
                    sino si R / = φ entonces REDUCER
               hasta que A = φ y R = φ
               SHIFTER
          finpara
          devuelve r : la raíz de la arboleda
 ACTOR
     Borrar un elemento v de A
     para todo α ∈ ACTION( ESTADO( v ), a )
          si α = 'accept' entonces r := v
          sino si α = 'shift s' entonces add < v, s > en Q
               sino si α = 'reduce p' entonces add < v, p > en R
     finpara
REDUCER
     sea < v, p > ∈ R
     N := LEFT( p ); Y := { y / existe un camino de longitud 2*|p| entre v y y }
     para todo y ∈ Y
          L:=(SIMBOLO(z1),...,SIMBOLO(z|p|)) y z1..z|p| son los A_vértices en el camino de v a y
          X:={x ∈ Y / L':=(SIMBOLO(z1),...,SIMBOLO(z|p|)) donde z1..z|p| son los A_vértices en
                        el camino de v a x y L' = L } ; Y := Y - X
          para todo x ∈ X
               s := GOTO( ESTADO( x )); W := { w ∈ X / GOTO( ESTADO( w )) = s ); X := X - W
               si Existe u ∈ U / ESTADO( u ) = s entonces
                    si Existe un camino de u a un A_vértice z / SUCESORES(z) = W entonces
                         ADDSUBNODE(SIMBOLO(z),L)
                    sino crear un nodo n en T etiquetado N
                         ADDSUBNODE( n, L )
                         crear en Γ un A_vértice z etiquetado n y un arco de u a z; añadir u en A
                         para todo w ∈ W : crear en Γ un arco de z a w finpara
               sino crear en T un vértice etiquetado N
                         ADDSUBNODE(n,L)
                         crear en Γ un E_vértice u, un A_vértice z y un arco de u a z
                         para todo w ∈ W : crear en Γ un arco de z a w finpara
                         añadir u a A y u a U
          finpara
     finpara
SHIFTER
     U := φ
     crear en T un vértice n etiquetado a
     para todo s / Existe v ( < v, s > ∈ Q )
          crear en Γ un E_vértice w etiquetado s, un A_vértice x etiquetado n y un arco de w a x
          apuntar w en U
          para todo v / < v, s > ∈ Q : crear un arco de x a v finpara
     finpara
```

ALG 1. Analizador sintáctico de Tomita

```
Algoritmo
     Entrada : Las dos funciones GOTO y ACTION de una GIC G y la cadena de entrada z ∈ Σ*
     Salida : si z ∈ L(G) en r un puntero a la cabeza del árbol de derivación
                  almacenado en la Arboleda, sino r = NIL.
     Variables : Γ : Grafo; T : Arboleda; A,U : {x/ x ∈ V };
                 Q : {<v,s>/ v ∈ E_V,s ∈ E } ; R : {<v,p>/ v ∈ E_V,p ∈ P }
     Método :
          añadir $ al final de z
          Γ := φ
          crear en Γ un vértice v0 etiquetado E0
          r := NIL ; T := φ ; U := E0
          para todo i ∈ [ 1..long(z)]
               a := Sacar Primer término de z
               A := U ; R, Q := φ
               repetir
                    si A / = φ entonces ACTOR
                    sino si R / = φ entonces REDUCER
               hasta que A = φ y R = φ
               SHIFTER
          finpara
          devuelve r : la raíz de la arboleda
ACTOR
     Borrar un elemento v de A
     para todo α ∈ ACTION( ESTADO( v ), a )
          si α = 'accept' entonces r := v
          sino si α = 'shift s' entonces add < v, s > en Q
               sino si α = 'reduce p' entonces add < v, p > en R
     finpara
REDUCER
     sea < v, p > ∈ R
     N := LEFT( p ); Y := { y / existe un camino de longitud 2*|p| entre v y y }
     para todo y ∈ Y
          L:=(SIMBOLO(z1),...,SIMBOLO(z|p|)) y z1..z|p| son los A_vértices en el camino de v a y
          P := Producto de las probabilidades de los símbolos de L * PROB_PROD(p)
          X := { x ∈ Y / L' := (SIMBOLO( z1 ),...SIMBOLO( z|p| )) donde z1.. z|p|
               son los A_vértices en el camino de v a x y L'|p| = L } ; Y := Y - X
          para todo x ∈ X
               s := GOTO( ESTADO( x )); W := { w ∈ X / GOTO( ESTADO( w )) = s }; X := X - W
               si Existe u ∈ U / ESTADO( u ) = s entonces
                    si Existe un camino de u a un A_vértice z / SUCESORES(z) = W entonces
                         ADDSUBNODE(SIMBOLO(z),L)
                         PROB(SIMBOLO(z)) := PROB(SIMBOLO(z)) + L
                    sino crear un nodo n en T etiquetado N y probabilidad P
                         ADDSUBNODE( n, L )
                         crear en Γ un A_vértice z etiquetado n y un arco de u a z; añadir u en A
                         para todo w ∈ W : crear en Γ un arco de z a w finpara
               sino crear en T un vértice etiquetado N y probabilidad P
                    ADDSUBNODE(n,L)
                    crear en Γ un E_vértice u, un A_vértice z y un arco de u a z
                    para todo w ∈ W : crear en Γ un arco de z a w finpara
                    añadir u a A y u a U
          finpara
     finpara
SHIFTER
     U := φ
     crear en T un vértice n etiquetado a y probabilidad 1
     para todo s / <v,s> ∈ Q
          crear en Γ un E_vértice w etiquetado s, un A_vértice x etiquetado n y un arco de w a x
          apuntar w en U
          para todo v / < v, s > ∈ Q : crear un arco de x a v finpara
     finpara
```

ALG 2. Analizador Sintáctico con Gramáticas Estocásticas

```
Algoritmo
     Entrada : Las dos funciones GOTO y ACTION de una GIC G y la cadena de entrada z ∈ (2^Σ)*
     Salida : si z ∈ L(G) en r un puntero a la cabeza del árbol de derivación almacenado
                    en la Arboleda, sino r = NIL.
     Variables : Γ : Grafo; T : Arboleda; A : {<x,y>/ x ∈ V,y ∈ Σ }; U : {x/ x ∈ V };
                 Q : {<v,s>/ v ∈ E_V, s ∈ E } ; R : {<v,p>/ v ∈ E_V, p ∈ P }
     Método :
          añadir $ al final de z
          Γ := φ
          crear en Γ un vértice v0 etiquetado E0
          r := NIL ; T := φ ; U := E0
          para todo i ∈ [1..long(z)]
               con := Sacar Primera cadena de z
               A := U x con ; R, Q := φ
               repetir
                    si A / = φ entonces ACTOR
                    sino si R / = φ entonces REDUCER
               hasta que A = φ y R = φ
               SHIFTER
          finpara
          devuelve r : la raíz de la arboleda
ACTOR
     Borrar un elemento <v,a> de A
     para todo α ∈ ACTION(ESTADO(v),a)
          si α = 'accept' entonces r := v
          sino si α = 'shift s' entonces add <v,s,a> en Q
               sino si α = 'reduce p' entonces add <v,p,a> en R
     finpara
REDUCER
     sea <v,p,a> ∈ R
     C := { b ∈ Σ / <v,p,b> ∈ R }
     R := R - { <v,p,b> / b ∈ C }
     N := LEFT(p); Y := { y / existe un camino de longitud 2*|p| entre v y y }
     para todo y ∈ Y
          L := ( SIMBOLO( z1 ),... SIMBOLO( z|p| ) ) y z1 .. z|p|
               son los A_vértices en el camino de v a y
          X := { x ∈ Y / L' := (SIMBOLO( z1 ),...SIMBOLO( z|p| )) donde z1 .. z|p|
               son los A_vértices en el camino de v a x y L'|p| = L } ; Y := Y - X
          para todo x ∈ X
               s := GOTO( ESTADO( x )); W := { w ∈ X / GOTO( ESTADO( w )) = s }; X := X - W
               si Existe u ∈ U / ESTADO( u ) = s entonces
                    si Existe un camino de u a un A_vértice z / SUCESORES(z) = W entonces
                         ADDSUBNODE(SIMBOLO(z),L)
                    sino crear un nodo n en T etiquetado N
                         ADDSUBNODE( n, L )
                         crear en Γ un A_vértice z etiquetado n y un arco de u a z; añadir u en A
                         para todo w ∈ W: crear en Γ un arco de z a w finpara
                         para todo a ∈ C : añadir <u,a> a A finpara
               sino crear en T un vértice etiquetado N
                    ADDSUBNODE( n, L )
                    crear en Γ un E_vértice u, un A_vértice z y un arco de u a z ; añadir u a U
                    para todo w ∈ W : crear en Γ un arco de z a w finpara
                    para todo a ∈ C : añadir <u,a> a A finpara
          finpara
     finpara
SHIFTER
     U := φ
     para todo a / <v,s,a> ∈ Q
          crear en T un vértice n etiquetado a
          para todo s /  <v,s,a> ∈ Q
               crear en Γ un E_vértice w etiquetado s, un A_vértice x etiquetado n y un arco de w a x
               apuntar w en U
               para todo v / <v,s,a> ∈ Q : crear un arco de x a v finpara
          finpara
     finpara
```

ALG 3. Analizador Sintáctico para frases de símbolos alternativos

5.- CONCLUSIONES

En este trabajo se han presentado dos modificaciones del algoritmo de Tomita para el análisis sintáctico con GIC.

En la primera, se ha introducido los pesos de una GIC estocastica, y la segunda permite el análisis de frases con símbolos alternativos. Aunque estas dos modificaciones se presentan de forma independiente, en la práctica se unificarán, gracias a lo cual, podremos obtener la frase de máxima verosimilitud contenida en la frase de símbolos alternativos.

El motivo de desarrollo de estos algoritmos es el de su aplicación al RAH, ya que debido a su eficiencia computacional, permite utilizar las GIC como representación del conocimiento. En la actualidad se está implementando un algoritmo de inferencia de GIC (Chirathamjaree,80) que permitirá obtener gramáticas que representen los distintos patrones para los digitos castellanos, posteriormente se realizarán experimentos con otros diccionarios.

6.-BIBLIOGRAFIA

AHO,S.; ULLMAN,J. (1973). "The Theory of Parsing, Translation and Compiling". Vol. 1. Prentice Hall.

BARR,A.; FEIGENBAUM, A. (1981). "The Handbook of Artificial Intelligence". Vol. 1. Pitman Books.

CASACUBERTA,F; VIDAL,E.; BENEDI,J.M. (1987). "Interpretation of Fuzzy Data by Means of Fuzzy Rules with Applications to Speech Recognition". Int.J.Fuzzy sets and Systems. Vol. 23. pp 371-380.

CHENG,H.; FU,K. (1986). "Algorithm Partition and Parallel recognition of General Context-Free Languages using Fixed-Size VLSI Architectures" Pattern Recognition, Vol. 19. No.5, pp 361-372.

CHIRATHAMJAREE,C.; ACKROYD,M. (1980). "A method for the inference of non-recursive contex free grammars" Man_Machines Studies, 12, 379-387

DEROUAULT,A.M.; MERIALDO,B. (1986). "Natural Language Modeling for Phoneme-to-Text Transcription". IEEE Trans. on Pattern Analysis and Mach. Intel. Vol 8(6). pp 742-749.

DON,H.S.; FU,K.S. (1985). "A Sintactic Method for Image Segmentation and Object Recognition" Pattern Recognition, Vol 18(1) pp 73-81

FU,K.S. (1982). "Syntactic Pattern Recognition and Applications". Prentice-Hall, New York.

GRAHAM,S.; HARRISON,M.; RUZZO,W. (1980). "An Improved Context-Free Recognizer", ACM Trans. on Prog. Lang. and Syst., Vol.2 No. 3, pp. 415-462.

GONZALEZ,R.C.; THOMASON,M.G. (1978). "Syntactic Pattern Recognition, an introduction". Addison-Wesley, Reading, Mass.

JUHOLA,M. (1986). "A Syntactic Method for Analysis of Saccadic Eye Movements". Pattern Recognition, Vol. 19, No. 5, pp 353-359.

LEVINSON,S. (1985). "Structural Methods in Automatic Speech Recognition". Proc. IEEE Vol 73 (1) 1625-1650.

MOHR,R. (1986). "Precompilation of Syntactical Descriptions and Knowledge Directed Analysis of Patterns". Pattern Recognition. Vol. 4. pp. 255-266.

NAKAGAWA,S. (1987). "Spoken Sentence Recognition by Time-Synchronous Parsing Algorithm of Context-Free Grammar". Proc. of the ICASSP-87. pp. 829-832.

NEY,H. (1987). "Dynamic Programming Speech Recogntion using a Context-Free Grammar". Proc. of the ICASSP-87, pp. 69-72.

NISHIMURA,M.; TOSHOKA,K. (1987). "HMM-based Speech Recognition using Multi-Dimensional Multi-Labeling". Proc. of the ICASSP-87. pp. 1163-1166.

PIERREL,J.M. (1981). "Etude et Mise on Oeuvre de Contraintes Linguistiques en Comprehension Automatique du Discours Continu". Tesis de Estado. Univ. de Nancy.

TOMITA,M. (1985). "An Efficient Context-Free Parsing Algorithm for Natural Languages". 9th Int. Joint Conf. on A.I.

TOMITA,M. (1986). "Efficient Parsing for Natural Language". Kluwer Academic Pub.

TOMITA,M.; CARBONELL,J. (1987). "The Universal Parser Architecture for Knowledge-Based Machine Translation". Doc. CMU-CMT-87-101. Center for Machine Translation. Carnegie Mellon Univ.

VALIANT,L. (1975). "General Context-Free Recognition in Less than Cubic Time". J. of Computer and System Sci. Vol. 10, pp. 308-315.

MOHR,R. (1986). "Precompilation of Syntactical Descriptions and Knowledge Directed Analysis of Patterns". Pattern Recognition. Vol. 4. pp. 255-266.

NAKAGAWA,S. (1987). "Spoken Sentence Recognition by Time-Synchronous Parsing Algorithm of Context-Free Grammar". Proc. of the ICASSP-87. pp. 829-832.

NEY,H. (1987). "Dynamic Programming Speech Recogntion using a Context-Free Grammar". Proc. of the ICASSP-87, pp. 69-72.

NISHIMURA,M.; TOSHOKA,K. (1987). "HMM-based Speech Recognition using Multi-Dimensional Multi-Labeling". Proc. of the ICASSP-87. pp. 1163-1166.

PIERREL,J.M. (1981). "Etude et Mise on Oeuvre de Contraintes Linguistiques en Comprehension Automatique du Discours Continu". Tesis de Estado. Univ. de Nancy.

TOMITA,M. (1985). "An Efficient Context-Free Parsing Algorithm for Natural Languages". 9th Int. Joint Conf. on A.I.

TOMITA,M. (1986). "Efficient Parsing for Natural Language". Kluwer Academic Pub.

TOMITA,M.; CARBONELL,J. (1987). "The Universal Parser Architecture for Knowledge-Based Machine Translation". Doc. CMU-CMT-87-101. Center for Machine Translation. Carnegie Mellon Univ.

VALIANT,L. (1975). "General Context-Free Recognition in Less than Cubic Time". J. of Computer and System Sci. Vol. 10, pp. 308-315.