Procesamiento del Lenguaje Natural, Revista nº 19, Septiembre de 1996

UNIFICACION CONSTRUCTIVA, ESTRATEGICA, CON COMPARTICION DE ESTRUCTURAS Y POST-COPIA

José F. Quesada
Centro Informático Científico de Andalucía (CICA)
Avda. Reina Mercedes, s/n (41014) Sevilla
josefran@cica.es

In NLP, unification is usually described as a computational sink. During the last years, research has concentrated in elucidating the reasons of such a problem. Some authors have identified process such as early copying, over copying or redundant copying. The paper proposes a novel model based on constructive unification, that avoids all copying problems. The general constructive model can be expanded using different techniques like strategic unification, sub-structure sharing or post-copy in order to accomodate to different situations.

La unificación es la operación computacionalmente más costosa para un sistema de PLN. Durante los últimos años se han identificado las causas de este coste, y en la literatura especializada son comunes las referencias a la pre-copia, sobre-copia o copia redundante. En esta línea, el artículo investiga estos problemas y propone un modelo constructivo de unificación de rasgos que evita los tres procesos de copia indicados. El modelo permite la incorporación de la unificación estratégica a través de la ordenación de los rasgos de las estructuras en función de la probabilidad de error obtenida en un proceso de aprendizaje. Desde el punto de vista de su utilización en un sistema real, la unificación constructiva es en realidad una estrategia híbrida, que permite la incorporación de las técnicas de compartición de estructuras y de post-copia para su adaptación a diferentes situaciones.

1.- Los Problemas de la Unificación: Pre-Copia, Sobre-Copia y Copia Redundante

Este trabajo asume la corriente de investigación sobre algoritmos eficientes de unificación en Procesamiento del Lenguaje Natural (PLN), representados fundamentalmente por Pereira 1985, Karttunen y Kay 1985, Wroblewsky 1987, Kogure 1990, Tomabechi 1991 o Emele 1991.

En todos estos trabajos se aborda el problema computacional que han generado los formalismos gramaticales basados en unificación. Este reto se suele enunciar con una cifra bastante significativa: en la mayoría de los sistemas de PLN basados en unificación más del 90% del tiempo total de cómputo es invertido en la operación de unificación. Hay incluso trabajos que hablan de un 98%.

En nuestro análisis partiremos de las nociones básicas de unificación en PLN (Shieber 1986). El primer algoritmo que cualquiera que haya trabajado en este campo habrá intentado implementar es el que se ha denominado "ingenuo", debido a que ignora todos los problemas de eficiencia en su concepción e implementación.

El procedimiento "ingenuo" de unificación recibe dos estructuras como entrada y devuelve bien un valor de error si las dos estructuras son incompatibles (no unificables), o bien una nueva estructura resultado de la unificación de las dos iniciales. El principal problema de este modelo es que modifica destructivamente las estructuras de entrada tanto en el caso de éxito como en el de fracaso de la uni-

ficación. Por tanto, si interesa preservar la información de entrada para posibles usos posteriores, es necesario realizar una copia de cada estructura antes de comenzar el proceso de unificación. Pero esto es una carga computacional nada despreciable. Wroblewsky (1987) denominó a este problema "Pre-Copia" (*Early Copying: The argument DAGs are copied before unification is started*).

Otro problema que Wroblewsky encuentra en los modelos ingenuos o destructivos de unificación es el que denomina "Sobre-Copia" (*Over Copying: Copies are made of both DAGs, and then these copies are ravaged by the unification algorithm to build a new result DAG*).

En su trabajo, Wroblewsky propone un algoritmo de unificación no destructiva que se basa fundamentalmente en la idea de copia incremental. Este modelo elimina la pre-copia y logra evitar (aunque no siempre) la sobre-copia.

Para Kogure (1990) este algoritmo sigue presentando un grave problema, al que califica de "Copia Redundante" (*redundant copying*):

> *However, the problem with his method (Wroblewsky's method) is that a unification result graph consists only of newly created structures. This is unnecessary because there are often input subgraphs that can be used as part of the result graph without any modification, or as sharable parts between one of the input graphs and the result graph. Copying sharable parts is called redundant copying. (Kogure 1990)*

La copia redundante se evita mediante técnicas de compartición de estructuras, cuyo modelo paradigmático se encuentra en el ya clásico trabajo de Pereira en 1985. No obstante, el modelo de datos de Pereira genera un coste adicional de acceso a los datos que Kogure reduce mediante la técnica SLING (*strategic lazy incremental copy graph*).

Una de las ideas más interesantes del trabajo de Kogure es la noción de copia estratégica: "*The method treats features tending to fail in unification first*".

Emele (1991) aborda estos mismos problemas y presenta un modelo al que denomina LIC (*lazy-incremental copying*), que combina las técnicas de copia incremental (*incremental copying*), copia "perezosa" (*lazy copying*) y compartición de estructuras (*structure sharing*) para eliminar los problemas de pre-copia, copia redundante, y en la mayoría de los casos evitar la sobre-copia.

La Figura 1 muestra como conclusión de esta sección las relaciones entre problemas y técnicas usados por diferentes algoritmos (extraída de Emele 1991).

ENFOQUES	PROBLEMAS			TECNICAS		
	Pre-Copia	Sobre-Copia	Copia Redundante	Copia Incremental	Copia Perezosa	Compartición de Estructuras
ingenuo	si	si	si	no	no	no
Pereira 85	no	no	no	no	no	si
Karttunen/Kay 85	no	no	si	no	si	si
Karttunen 86	no	no	si	no	no	no
Wroblewsky 87	no	si	si	si	no	no
Godden 90	no	no	si	no	si	si
Kogure 90	no	si	si	no	si	si
Emele 91	no	si	no	si	si	si

Fig 1: Problemas y Tecnicas en la Unificación

2.- Unificación Constructiva

Si se analiza detalladamente el proceso de unificación se puede observar como éste realmente no introduce nueva información. Parafraseando la famosa máxima de la Física, podemos decir que *la unificación no crea ni destruye información, únicamente la transforma*. Sin embargo, todos los algoritmos de unificación son en mayor o menor medida destructivos. En mi opinión, esto se produce porque las estructuras de datos en las que se basan dichos algoritmos no logran representar adecuadamente las diferencias entre estructura y contenido en una estructura de rasgos.

A continuación se presenta un nuevo modelo de unificación, al que se ha denominado de unificación constructiva.

2.1.- Estructuras de Datos

El sistema constructivo completo, tanto reversible como no reversible, se basa en las siguientes cinco estructuras. Explicaremos estas estructuras usando como ejemplo la estructura *E2* que aparece al final de esta misma sección y su representación gráfica (Figura 4).

La estructura *FR* es el nodo raíz de una lista de atributos. En la Figura 4, la estructura número 50 es de tipo *FR*. Una estructura *FR* apunta a una lista de atributos (estructura *FL*) mediante el campo *cn*, o bien, este campo quedará invalidado si la estructura ha sido "dereferenciada" hacia otra estructura también del tipo *FR*, valor que almacenará el campo *dr*. La situación actual estará controlada por el campo *rt*. En la Figura 4, las estructuras 50, 53, 58 y 63 usan el campo *cn*, mientras que las 66 y 69 usan el campo *dr*, permitiendo así un tratamiento uniforme de la correferencia. Finalmente el campo *pc*, se usa en la implementación del algoritmo de post-copia para evitar los problemas de copia-redundante que aparecen en el algoritmo de Emele.

Estructura	Campo	Tipo	Comentario
FR			Feature Root
	rt	int	root type
	dr	FR	dereference
	cn	FL	content
	pc	FR	post copy

La estructura FL es una lista dinámica de estructuras *FV*. Permite por tanto la especificación de una estructura de rasgos, cada uno de los cuales se almacena como una estructura de tipo *FV*. Las estructuras 51, 56 y 61 de la Figura 4 son un ejemplo de lista recursiva de estructuras *FL*.

FL			Feature List
	lm	FV	element
	nx	FL	next

La estructura *FV* contiene la información específica de un atributo, incluyendo la etiqueta de dicho atributo (*ft*), el valor usado por dicho campo como parámetro de ordenación para la unificación estratégica (*st*). El campo *vt* (tipo de valor) controla si el atributo es atómico (*tm*) o bien complejo (*cm*). La estructura 52 es un ejemplo de atributo complejo que apunta a la subestructura 53, mientras que la número 55 es un ejemplo de estructura simple.

FV			Feature Value
	ft	string	feature name
	st	int	strategic value

	vt	int	value type
	tm	string	atom
	cm	FR	complex

Una estructura *FD* contiene la información necesaria para deshacer cualquiera de los cambios producidos sobre una estructura durante la unificación. Para ello, se recuerda el tipo de estructura sobre el que se ha producido el cambio (*st*), el campo que ha variado para esta estructura, la dirección de memoria de la estructura (*pt*) y el valor antiguo (*ld*). *FD* está diseñada asimismo como una lista recursiva que se auto-enlaza mediante el campo *nx*.

FD			Feature Desunification
	st	int	structure type
	fl	int	field
	pt	pointer	structure address
	ld	pointer	old value
	nx	FD	next

Finalmente, una estructura *FS* es el núcleo de información que manipula el algoritmo de unificación. Dicho algoritmo recibe dos estructuras FS como entrada y en caso de que la unificación sea correcta genera una nueva estructura *FS* a su salida. Esta nueva estructura indica mediante el campo *fr* el nodo raíz de la estructura de rasgos obtenida tras la unificación, en *dn* registra la lista de acciones de desunificación necesarias para obtener las estructuras originales, y en *f1* y *f2* almacena los punteros a dichas estructuras. De esta forma, la desunificación de una estructura *FS*, aplica las instrucciones registradas en dn y devuelve los punteros *f1* y *f2*.

FS			Feature Structure
	fr	FR	feature root
	dn	FD	desunification
	f1	FS	previous structure
	f2	FS	previous structure

En la estructura *FR*, los campos *dr* y *cn* funcionan como una unión y el valor actual es determinado por el contenido del campo *rt*. En la estructura *FV* sucede algo similar, los campos *tm* y *cm* funcionan como una unión controlada por el campo *vt*.

En la Figura 2 se muestran los iconos que se usarán en la representación gráfica de las estructuras de rasgos.

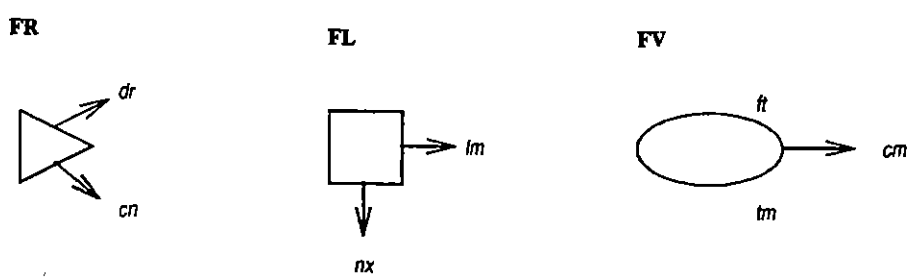

Fig 2: Iconos para las Estructuras de Datos

Para mostrar el comportamiento del algoritmo de unificación constructiva utilizaremos las estructu de rasgos que presenta Worblewsky como uno de los casos para los cuales su algoritmo no consi resolver los problemas de sobre-copia. Se trata de las estructuras E1 y E2 que mostramos a continu ción:

$$E1 = \begin{bmatrix} x & [a\ b] \\ y & [c\ d] \\ z & \begin{bmatrix} p[1] & [e\ f] \\ q \rightarrow 1 & \end{bmatrix} \end{bmatrix}$$

$$E2 = \begin{bmatrix} x[1] & [a\ b] \\ y[2] & [c\ d] \\ z & \begin{bmatrix} p \rightarrow 1 \\ q \rightarrow 2 \end{bmatrix} \end{bmatrix}$$

Las Figuras 3 y 4 contienen la representación correspondiente a las estructuras E1 y E2, respectivamente. Para facilitar la manipulación de dichas representaciones se ha numerado cada uno de los componentes de cada estructura.

2.2.- Estructuras Ordenadas y Unificación Estratégica con Entrenamiento

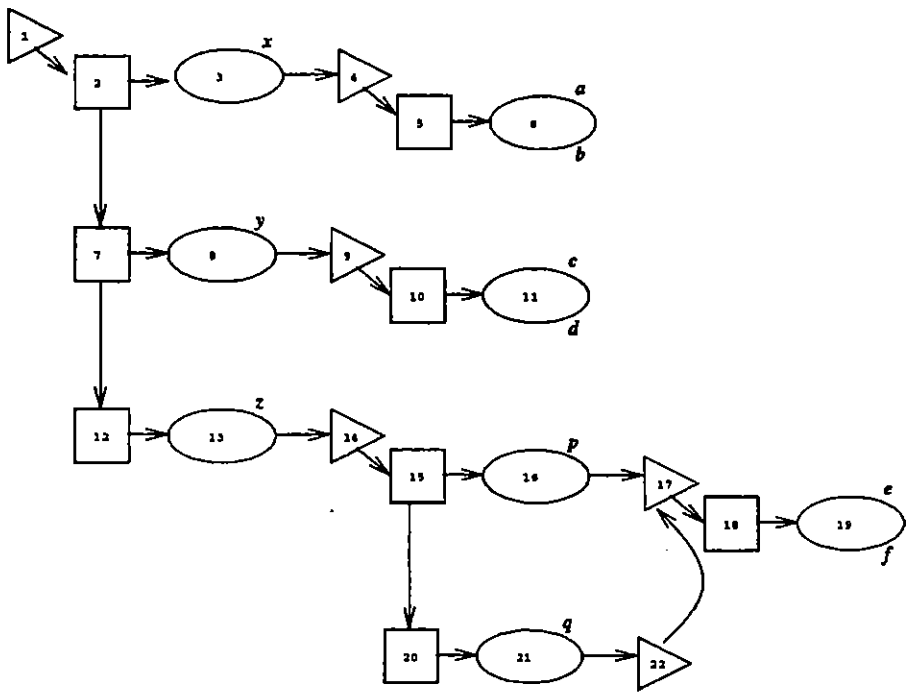

Fig 3: Estructura E1

Todos los algoritmos comentados a lo largo de este trabajo seleccionan en una primera fase el conjunto de arcos compartidos por las dos estructuras que van a ser unificadas y el conjunto de los arcos específicados de cada una de éstas. Aunque inofensivas a primera vista, estas operaciones introducen una considerable carga computacional en el algoritmo. De hecho, si tenemos en cuenta que en proce-

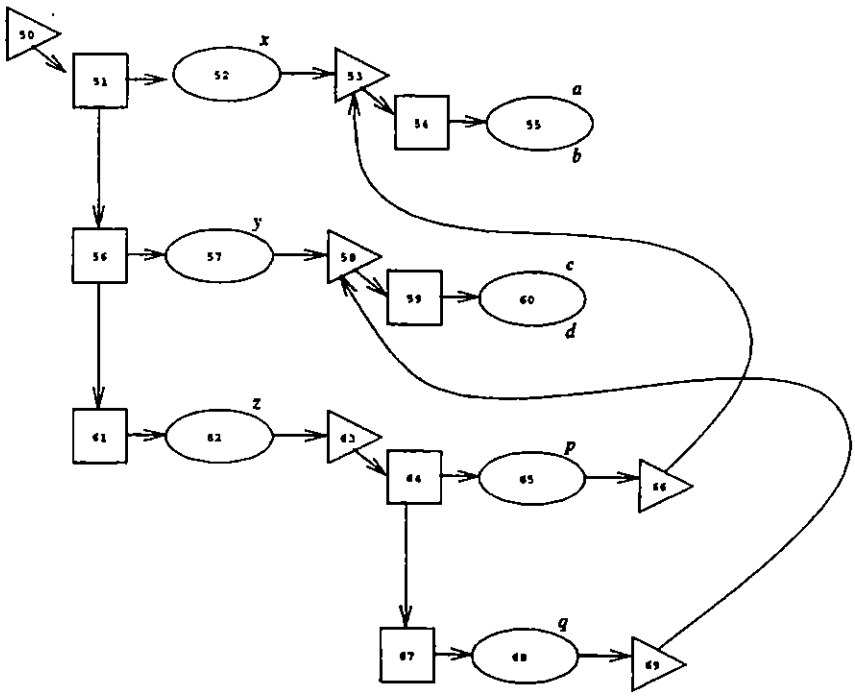

Fig 4: Estructura E2

sos reales de unificación, el número de elementos (arcos) de ambas estructuras es bastante similar, el proceso inicial de clasificación de los arcos es una operación de orden n^2, donde n es el número de elementos de las estructuras de entrada.

Para evitar este coste, el algoritmo que presentamos utiliza estructuras ordenadas. Desde un punto de vista teórico, esta condición no supone ninguna limitación, puesto que resulta evidente la adaptación del algoritmo a estructuras no ordenadas. Desde un punto de vista práctico, podemos suponer dos situaciones. La primera de ellas aparecería si tuviesemos que unificar dos estructuras no ordenadas usando este algoritmo. En este caso, el coste de la ordenación es similar al coste provocado por la clasificación de los arcos compartidos y específicos. En la segunda situación, podemos considerar que estamos aplicando el algoritmo en un entorno de PLN, en concreto en un modelo gramatical basado en unificación. Lo único que será necesario realizar es un proceso previo de ordenación de las estructuras de rasgos del léxico, lo cual sólo se debe hacer una vez durante toda la vida de dicho léxico. Este proceso se puede considerar tiempo de compilación y no de ejecución. Una vez realizado este proceso de ordenación el algoritmo de unificación se encarga de mantener ordenadas las estructuras que va generando.

En el algoritmo que presentaremos a continuación se asume que las estructuras están ordenadas según el nombre del atributo, es decir, según el campo *ft* de la estructura *FV*. No obstante, el modelo de datos permite un proceso previo de entrenamiento que acumularía los valores relativos a la probabilidad de fracaso de la unificación en el campo *st* de la estructura *FV*. Si en lugar de ordenar las estructuras en función del nombre, se ordenan en función de este nuevo valor, se habrá obtenido un sistema de unificación estratégica.

Desde el punto de vista de la implementación, la unificación estratégica ha de tener en cuenta el siguiente criterio: Dados cuatro atributos (a,b,c,d) con probabilidades (.25, .50, .50, .75) respectivamente, los atributos con una misma probabilidad deberán modificar su valor de forma que nunca dos atributos posean el mismo valor, y no se altere el orden relativo de la ordenación inicial. En este caso se prodrían asignar los valores (.25, .49, .51, .75).

```
FUNCTION CUnif (FS fs1, FS fs2)
        newfr = CU(fs1.fr,fs2.fr,fd)
        IF (newfr)     // Unificacion Correcta
                newfs = CreateFS()
                newfs.fr = newfr
                newfs.dn = fd
                newfs.f1 = fs1
                newfs.f2 = fs2
                RETURN newfs
        ELSE           // Unificacion Erronea
                DesCU(fd)      // Des-unificacion
                RETURN null
        ENDIF
ENDFUNCTION

FUNCTION CU (FR fr1, FR fr2, FD fd)
        fr1 = Dereference (fr1)
        fr2 = Dereference (fr2)
        IF (fr1.cn.lm.ft <= fr2.cn.lm.ft)
                newfr = fr1
                oldfr = fr2
        ELSE
                newfr = fr2
                oldfr = fr1
        ENDIF
        newfl = newfr.cn
        oldfl = oldfr.cn
        newfd = CreateFD("FR","cn",oldfr,oldfr.cn)
        fd = UnionFD(fd,newfd)
        oldfr.dr = newfr
        unify = 1
        WHILE (unify AND oldfl)
                IF (oldfl.lm.ft == newfl.lm.ft)
                        prelm = newfl.lm
                        newfl.lm.cm = CU(newfl.lm.cm,oldfl.lm.cm,returnfd)
                                                // Llamada Recursiva
                        fd = UnionFD(fd,returnfd)
                        IF (Unification failed)
                                unify = 0
                        ELSE IF (newfl.lm.cm != prelm.cm)
                                newfd = CreateFD("FV","cm",prelm,prelm.cm)
                                fd = UnionFD(fd,newfd)
                        ENDIF
                        oldfl = oldfl.nx
                ELSE IF (newfl.nx.lm.ft <= oldfl.lm.ft)
                        newfl = newfl.nx
                ELSE
                        newfd = CreateFD("FL","nx",newfl,newfl.nx)
                        fd = UnionFD(fd,newfd)
                        intfl = newfl.nx
                        newfl.nx = oldfl
                        newfl = oldfl
                        oldfl = intfl
                ENDIF
        ENDWHILE
        IF (!unify)
                RETURN null
        ELSE
                RETURN newfr
        ENDIF
ENDFUNCTION
```

Fig 5: Unificación Constructiva

2.3.- El Algoritmo de Unificación Constructiva

El algoritmo recibe como entrada dos estructuras tipo *FS* y devuelve a su salida bien un valor nulo, si la unificación falla, o bien una nueva estructura *FS* que es el resultado de la unificación. Durante este proceso, no se copia ninguna información de las estructuras de entrada.

Para el ejemplo que venimos usando, la estructura *FS* asociada con *E1*, contiene el valor 1 en el campo *fr*, y un valor nulo en el resto de campos. La estructura *FS* para *E2* contiene el valor 50 en *fr* y valores nulos en el resto de campos.

La Figura 5 contiene el pseudocódigo para el algoritmo completo de unificación constructiva.

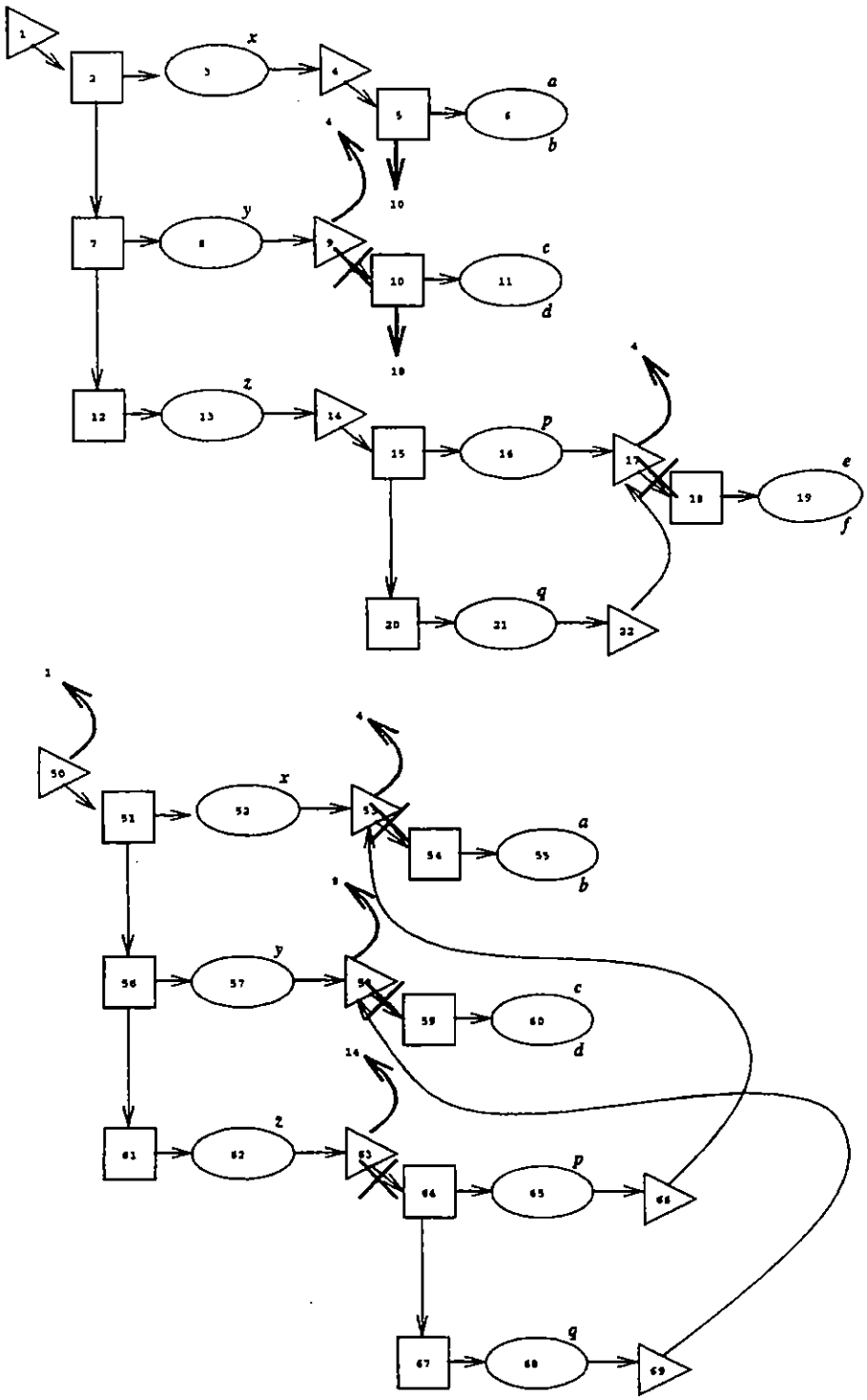

Fig 6: Estado de las Estructuras E1 y E2 tras la Unificación

El proceso de unificación de las estructuras *E1* y *E2* presentadas en las Figuras 3 y 4 genera una nueva estructura *FS*, cuyo valor para *fr* será el mismo *fr* de *E1*, los valores de *f1* y *f2* son los correspondiente a *E1* y *E2*, y el valor de dn es una nueva estructura que contiene la siguiente secuencia de instrucciones de desunificación (elementos de *FD*):

```
FR,cn,50,51
FR,cn,53,54
FR,cn,58,59
FR,cn,63,64
FR,cn,17,18
FL,nx,5,null
FL,nx,5,18
FL,nx,10,null
```

Cada una de estas entradas indica el tipo de estructura, el campo en que ha sido modificada, el punt a dicha estructura y el valor antiguo que poseía dicho campo en dicha estructura.

Las modificaciones sobre las estructuras originales se presentan en la Figura 6 mediante trazos gr sos.

3.- Unificación Reversible con Compartición de Estructuras y No Reversible con Post-Copia

Manteniendo siempre el modelo de unificación constructiva, el modelo de datos permite dos compo tamientos diferentes tras el proceso de unificación:

• Unificación Reversible: Cualquier estructura *FS* obtenida por un proceso de unificación construct contiene toda la información necesaria como para obtener las estructuras que entraron al proceso de unific ción. Para ello recuerda las estructuras *FS* asociadas con dichas estructuras de entrada, así como la secuenc de operaciones de desunificación que devolverán los valores antiguos de las estructuras modificadas. Por co siguiente, el modelo constructivo es en sí mismo un mecanismo reversible que consigue el mayor grado compartición de estructuras, puesto que en ningún momento se copia ninguna subestructura.

• Unificación No Reversible: Si tras un proceso de unificación constructiva que haya finalizado co éxito se realiza una copia de la estructura obtenida como resultado obtenemos un modelo que se puede cali car de unificación con post-copia, ya que la copia sólo se realiza cuando ha finalizado la operación de unific ción y se conoce con total seguridad que la unificación ha tenido éxito. Tras realizar la copia, se aplica procedimiento de desunificación obteniendo las estructuras de entrada.

Se trata por consiguiente de un modelo híbrido de unificación, en la línea de las propuestas de alguno autores, tales como Wroblewsky, quien indica en su artículo:

> *The author's experience suggests that the "perfect graph unification algorithm" may not exist, and is best thought of as a family of related algorithms optimized for different purposes. [Wroblewsky 1987]*

El modelo presentado de unificación constructiva permitiría de esta forma adaptarse a las distinta necesidades que aparecen en las diferentes fases de aplicación de la unificación en PLN. Conside rando el caso paradigmático de aplicación del algoritmo de unificación al árbol o bosque de análisi generado por un parser, podemos considerar varios casos bien diferenciados:

• Desde los símbolos terminales a los preterminales: En este caso siempre se usan estructuras de rasgo que provienen del léxico y que por tanto deberemos recuperar en su estado original. Así pues, en esta situa ción puede compensar realizar una unificación no reversible con post-copia.

• Nodos intermedios del árbol sin ambigüedad: En esta situación sólo cabe un posible análisis, con lo cual si éste falla, podemos considerar que el análisis completo del árbol o bosque es también incorrecto. Por tanto, será preferible aplicar la unificación no reversible con compartición de estructuras. Se trata de una mezcla de los dos modelos básicos comentados, que aprovecha la compartición de estructuras para evitar cualquier copia, y además al tratarse de un modelo no reversible evita almacenar la información correspondiente a la desunificación.

• Nodos intermedios ambigüos: En estos casos, podrá interesar usar el comportamiento reversible del algoritmo, puesto que tendremos que realizar distintos intentos. En función de que el modelo de análisis esté basado en backtracking o en análisis paralelo podremos aplicar unificación constructiva reversible con compartición de estructuras o bien no reversible con post-copia.

Conclusión

En este artículo se presenta un modelo de unificación de estructuras de rasgos para el PLN basado en una estrategia constructiva que evita cualquier forma de copia durante el proceso mismo de la unificación.

El algoritmo permite la incorporación de un modelo de unificación estratégica que introduce la información sobre probabilidad de éxito o fracaso obtenida tras un proceso de entrenamiento del unificador.

El modelo constructivo realmente se puede concebir como una familia de algoritmos de unificación, por tanto es un modelo híbrido que puede usar la estrategia de compartición de subestructuras o la post-copia para adecuarse a diferentes situaciones.

Bibliografía

- Boyer, R.S.; J.S. Moore, J.S. 1972. "The sharing of structure in theorem proving programs". *Machine Intelligence*, 7.
- Carroll, J. 1994. "Relating Complexity to Practical Performance in Parsing with Wide-Coverage Unification Grammars". *CMP-LG e-print archive cmp-lg/9405033.*
- Dowding, J.; Moore, R.; Andry, F.; Moran, D. 1994. "Interleaving Sintax and Semantics in an Efficient Bottom-Up Parser". *Proceedings of the 32nd Annual Meeting of the ACL.*
- Emele, M.C. 1991. "Unification with Lazy Non-Redundant Copying". 29th Annual Meeting of the Association for Computational Linguistics.
- Godden, K. 1990. "Lazy Unification". *28th. Annual Meetinggg of the Association for Computational Linguistics.*
- Grosz, B; Sparck Jones, K.; Webber, B.L.. 1986. *Readings in Natural Language Processing.* Los Altos: Morgan Kaufmann.
- Karttunen, L.; Kay, M. 1985. "Structure sharing with binary trees". *Proceedings of the 23th Annual Meeting of the Association for Computational Liguistics.*
- Karttunen, L. 1986. *D-PATR: A Development Environment for Unification-Based Grammars.* Technical Repoort CSLI-86-61, Center for the Study of Language and Information.
- Kay, M. 1985. *Parsing in functional grammar.* En D. Dowty, L. Karttunen & A. Zwicky (eds): *Natural Language Parsing*, Cambridge, Cambridge University Press, 1985.
- Kaplan, R.; Bresnan, J. 1982. "Lexical-functional Grammar: A Formal System for Grammatical Representation". En J. Bresnan (ed): *The Mental Representation of Grammatical Relations*, Cambridge, Mass., MIT Press.
- Knight, K. 1989. "Unification. A Multidisciplinary Survey". *ACM Computing Surveys*, 21(1):93-124.
- Kogure, K. 1990. "Strategic Lazy Incremental Copy Graph Unification". *COLING-90.*
- Pereira, F.C.N. & D.H.D. Warren. 1980. "Definite Clause Grammars for Language Analysis- A Survey of the Formalism and a Comparison with Augmented Transition Network". *Artificial Intelligence*, 13:231-278, 1980.
- Pereira, F.C.N. 1985. "A structure-sharing representation for unification-based grammar formalisms". En *Proceedings of the 23th Annual Meeting of the Association for Computational Liguistics.*
- Pollard, C. 1985. "Phrase structure grammar without metarules". *Proceedings of the 4th West Coast Conference on Formal Linguistics.*
- Robinson, J.A. 1965. "A machine-oriented logic based on the resolution principle". *J. ACM*, 12:23-41, 1965.
- Ruiz, J.C. 1993. "GFU-LAB: Un sistema computacional para la co-descripción de la sintaxis y la semántica". *IX Congreso de Lenguajes Naturales y Formales*, 1993.
- Shiebert, S.M. 1986. *An Introduction to Unification-Based Approaches to Grammar.* Stanford, California, Center for the Study of Language and Information, CSLI Lecture Notes Series.
- Siekmann, J.H. 1990. "Unification Theory". En Claude Kirchner (ed). *Unification.* San Diego, CA, 1990,

Academic Press Inc.

- Tomabechi, H. 1991. "Quasi-Destructive Graph Unification". *29th Annual Meeting of the Association f Computational Linguistics.*
- Wroblewski, D. 1987. "Non-destructive graph unification". *Proceedings of the Conference on the Americ Association for Artificial Intelligence.*