Julia Díaz
Julio Gonzalo
Pilar Rodríguez-Marín
IIC /UAM

El Formalismo Semántico en la Interfaz de Lenguaje Natural de Prógenes.

Abstract

El presente artículo describe los mecanismos semánticos que ponen en relación el enunciado, en castellano, de un problema de matemáticas o física y su representación en el lenguaje formal Prógenes.

Aunque inspirados en el formalismo semántico de UCG (gramática Categorial de Unificación), la distancia entre sus principios generales y la aplicación práctica en nuestro dominio requiere su adaptación y modificación. En Prógenes, para incorporar las modificaciones se sigue un paradigma de representación orientada a objetos. Dentro de él, no sólo se adopta una red de herencia en la que realizar el léxico, lo que es ya una práctica común en las corrientes lingüísticas actuales, sino que, además, se modifica la forma de especificar la gramática, que se define como una gramática por defecto sobre clases y no sobre constituyentes ni en forma de principios universales. Surgen así los conceptos de regla por defecto y grado de especificidad de una gramática.
La generalidad de estos principios permite considerarlos como una estrategia para ajustar un formalismo general a un tarea específica, introduciendo las excepciones de forma modular.

Introducción.

Prógenes (Castells et al. 92) es un sistema capaz de resolver problemas de matemáticas[1] enunciados en lenguaje natural, tal y como los resolvería un estudiante. Una interfaz de lenguaje natural extrae el significado del enunciado en castellano (Díaz y Rodríguez-Marín 91), expresándolo en el lenguaje formal Prógenes (Gonzalo, Rodriguez-Marín y Díaz 92) Un módulo de resolución natural es capaz de dar respuesta al problema a partir de su expresión formal. Ambos módulos, así como los conceptos y la sintaxis del lenguaje formal, se apoyan en una serie de bases de conocimiento correspondientes a distintas materias.

Un ejemplo:

```
Hallar la pendiente de la recta $3x+2y+6=0$ y su intersección con el eje $X$
```

daría lugar, tras pasar por la interfaz, a:

```
(ACTION (pvalues
          (NUM (slope
                  (LINE (line (LINEAR-EQUATION $3x+2y+6=0$)))))
          (SET (intersection
                  (LINE (line (AXIS X-axis)))
                  (LINE (line (LINEAR-EQUATION $3x+2y+6=0$)))))))
```

La expresión anterior, así como cada subexpresión, va precedida por su *tipo* (en mayúsculas). Más adelante veremos cómo esta información es crucial en el proceso de parsing. Por su parte, el módulo de resolución tiene como entrada esa misma expresión, una vez eliminados los tipos:

[1] En general, conocimientos formalizables de forma matemática, como Mecánica, Astronomía, etc.

```
(pvalues (slope (line $3x+2y+6$))
         (intersection (line X-axis)
                       (line $3x+2y+6=0$)))
```

El módulo de deducción automática funciona como un intérprete de S-expresiones construido sobre Lisp. La evaluación del enunciado da lugar, en este caso, a:

```
(pvalues 3/2
         (point -2 0))
```

La Interfaz de Lenguaje Natural de Prógenes puede ser descrita en torno a tres ideas principales:

Por un lado, el sustrato teórico es deudor de UCG (gramática categorial de unificación, Calder et al. 88). Se trata de un formalismo que sintetiza los fundamentos de la familia de gramáticas categoriales (Oehrle et al. 88), de la Discourse Representation Theory (Kamp 81) y de la familia de formalismos de Unificación (Shieber 86). El resultado es un sistema de fundamentos muy sencillos muy apropiado para ser implementado. Su formalismo de representación semántica, Indexed Language (InL, Zeevat et al. 87), es especialmente adecuado para nuestro dominio.

Por otro, la representación del léxico se basa en (Flickinger et al. 85). El léxico Prógenes está jerarquizado en una red de herencia por defecto. Cada palabra, así como cada parte de la oración, está implementada como una estructura tipada recursiva de pares atributo-valor (*typed feature structures*, ver por ejemplo Carpenter 92a)

Por último, la gramática está concebida como un conjunto de reglas jerarquizadas que operan por defecto. El comportamiento no monótono que muestran los mecanismos por defecto no ha sido, hasta ahora, considerado por los sistemas de representación lingüística dotados de mecanismos de herencia (ALE (Carpenter 92b), TFS (Emele y Zajac 90)) para soslayar sus inconvenientes computacionales. El funcionamiento de las reglas gramaticales en Prógenes es un intento por evaluar las virtudes y defectos de la especificación gramatical por defecto, así como por detallar las condiciones teóricas en las que ésta puede operar.

En este artículo nos centraremos en los aspectos semánticos de la interfaz de lenguaje natural. Veremos la relación entre léxico, semántica y base de conocimientos. También veremos como se puede clasificar el léxico según su papel semántico. Por último, examinaremos parte de las reglas semánticas que se usan en la interfaz.

Las expresiones semánticas

Los valores de los atributos semánticos de nuestro léxico son denominados *expresiones semánticas*. La figura 1 ilustra su sintaxis, donde:

Tipo: Desempeña un papel similar a los índices de InL, y puede representar bien el tipo de un descriptor o los tipos de los argumentos. Mientras que los índices de InL respetan un orden parcial (object, mass, event, ...), los tipos del lenguaje Prógenes están estructurados en una jerarquía como grafos acíclicos orientados y reflejan estructuras matemáticas. Es más, en cualquier momento del análisis debemos saber no sólo el tipo de la expresión completa sino también los de los argumentos relacionados.

Descriptor: Representa la posible traducción de una palabra escrita en lenguaje natural al lenguaje formal de Prógenes, pero también denota los nombres de las funciones utilizadas en el módulo de resolución. En InL se corresponde con la palabra principal del cuerpo de la fórmula.

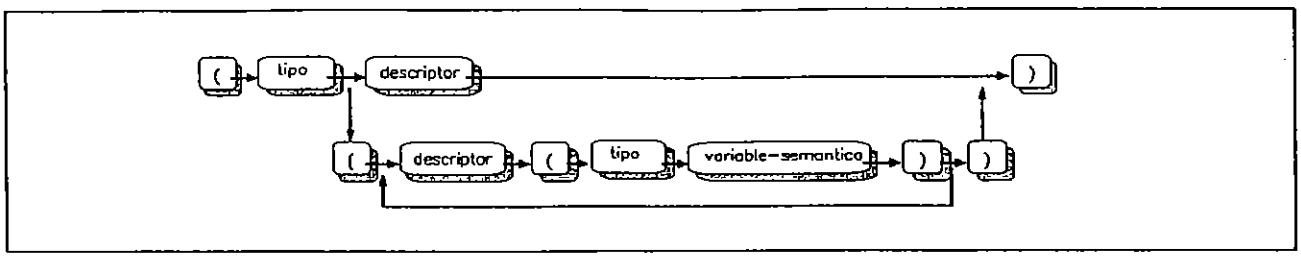

Figura 1. Sintaxis de las expresiones semánticas.

Variable-semantica: Está formada por el símbolo @ seguido de cualquier letra. Son instanciadas durante el análisis y, a su vez, su valor puede ser otra expresión semántica tan complicada como sea necesario.

La sintaxis descrita es general y puede representar:

- El valor de los atributos semánticos en el diccionario. Por ejemplo para la palabra *recta* es:

  ```
  (LINE (line (LINEAR-EQUATION @1)))
  ```

- Análisis semánticos parciales, como en *"... pasa por el punto P(3,1) ..."*, representado por la siguiente expresión formal:

  ```
  (BOOL (in (SET (point 3 1)) (SET @s)))
  ```

  Esta expresión es obtenida utilizando los métodos de combinación que serán descritos más adelante.

- El significado de la oración completa. Así, la oración *"Escribir la ecuación de la recta que pasa por el punto* **P(3,1)** *y que es paralela a* **3x+2y+3=0**" es representada mediante la siguiente expresión formal

```
(ACTION
  (find
    (LINE
      (pthe
         (LINE (: (VAR %X) (LINE line)))
         (BOOL (and
                 (BOOL (in (POINT (point (POINT 3 1))) (LINE %X)))
                 (BOOL (parallel (LINE %X)
                                 (LINE (line
                                         (LINEAR-EQUATION (== (++ (++ (** 3 x) (** 2 y)) 3) 0)
                                         ))))))))
```

Sin embargo no todas las palabras desempeñan un papel activo en la Base de Conocimiento (KB), sólo aquellas que tienen traducción. En la siguiente sección exponemos una breve clasificación atendiendo a la relación entre la KB y nuestro léxico.

Clasificación semántica del léxico Prógenes

En la sección anterior describimos el aspecto y la sintaxis de nuestro lenguaje formal. En esta sección veremos cómo se construyen expresiones bien enunciadas teniendo en cuenta la información de la KB y las restricciones impuestas por el lenguaje formal.

Como hemos mencionado, nuestra KB es el nexo entre la interfaz en lenguaje natural y el módulo de resolución, puesto que contiene el conocimiento de los conceptos matemáticos involucrados y el metaconocimiento requerido para resolver los problemas. Como veremos, también contiene, de forma implícita, las restricciones necesarias para enunciar expresiones correctas. Otro tipo de información

contenida en la KB son las relaciones entre objetos, representadas en forma declarativa o procedural. Veamos con un poco más de detalle el tipo de descripciones que podemos encontrar en la KB, teniendo en cuenta algunas categorías sintácticas de las palabras:

- Nombres y adjetivos: Estas palabras tienen significado en el dominio y se corresponden con los **constr**uctores en la KB. Este tipo de elemento está constituido por entidades complejas y estructuradas. Todos tienen un tipo característico y un número arbitrario de atributos describiendo los elementos del concepto. Por ejemplo, la definición en nuestra KB del nombre *muelle* es

```
(constr (spring (stiffness NUM)
                (hanging-position POINT)
                (tension NUM))
        MUELLE
        ())
```

Las tres primeras líneas definen no sólo el tipo de los argumentos, sino también la definición implícita de otros conceptos como la *rigidez*(stiffness) y la *tensión* de un muelle. La cuarta línea denota el tipo asignado al concepto y, a partir de la quinta, información propia del módulo de resolución en aquellos casos en los que sea necesario. La palabra siguiente a *constr*, en este caso *spring*, representa la traducción de la palabra enunciada en lenguaje natural al lenguaje formal. Aunque en este caso coincide, no siempre es así.

La extracción de la definición semántica de *muelle* es general, automática y respeta la sintaxis enunciada en la sección anterior:

```
(SPRING (spring (NUM @n) (POINT @p)
                (NUM @n)))
```

- Los verbos son también palabras con significado que normalmente desempeñan acciones especificadas en las metafunciones de nuestra KB. Las metafunciones son aplicaciones del conjunto de constructores de la KB en sí mismo. Aunque su sintaxis es similar a la de los constructores, su representación formal siempre es única. La siguiente es la definición del verbo **hallar** en la KB:

```
(fun (hallar (object OBJECT))
     NUM
     ()
     (- (third (cdadr (position object)))))
```

Su representación en lenguaje formal sería:

```
(TYPE (hallar (TYPE @t)))
```

- Los determinantes tienen una semántica muy específica que nos permite resolver adecuadamente los complementos preposicionales así como resolver referentes dentro y fuera de la oración. Dichos determinantes son representados en la KB como **Binding functions**. Este grupo también engloba todas aquellas funciones que realicen asignación de tipo, propiedad utilizada para la resolución de referentes. La definición en la KB de dichas *binding functions* es:

```
(bndfun (the (obj OBJECT) (bool BOOL))
        OBJECT
        ()
        (if bool (condition (eval bool)))
            (eval (move-= = = obj)))
```

En este contexto, el tipo *OBJECT* es similar al tipo *TYPE* de nuestra jerarquía. La representación de este conocimiento es:

```
(TYPE (the (TYPE (: (VAR %X) (TYPE @t)))
           (BOOL @b)))
```

Obsérvese que el código escrito en negrita no puede ser deducido a partir de la KB utilizando el mecanismo natural utilizado con los constructores y metafunciones, pero es necesario para que se pueda ejecutar la acción de los determinantes, que se describirá en los siguientes apartados. Una vez que el tipo TYPE ha sido instanciado, mantiene el mismo valor en todas las expresiones y además a la variable %X se le asigna el valor que tenga la variable semántica @t.

- Las preposiciones y los adverbios sólo desempeñan un papel sintáctico durante el análisis y su semántica depende del contexto. En nuestro lenguaje formal este caso es representado como (TYPE @t) donde TYPE es el supremo de nuestra jerarquía, a partir de la cual heredan el resto de los tipos.

Las expresiones matemáticas (en nuestro caso, fórmulas TeX) son el elemento más característico del dominio. La interfaz está dotada de un analizador semántico específico que traslada la notación TeX a expresiones Prógenes, y extrae de ellas la información necesaria para integrarlas en la traducción del texto: variables que se introducen, tipos, comportamiento matemático... Las fórmulas pueden funcionar como sintagmas nominales ("Hallar $f(g(x))$"), como modificadores de sintagma nominal ("La curva $y=f(x)$"), como cláusulas ("Sabiendo que $f'(0)=2$")... A menudo, la mayor parte de la información semántica se encuentra en las fórmulas (como en "Sea $f(x)=x\%2$."), así como la introducción de referentes del discurso ("Hallar los $x\in (0,2)$ tales que ..."). El lenguaje de las fórmulas TeX y su interacción con este subconjunto del castellano son temas complejos que no tienen cabida en este artículo.

Mecanismos de combinación semántica

Hasta ahora, se ha descrito cómo se relaciona la representación semántica con la base de conocimientos, el lenguaje formal y el léxico. A continuación mostraremos cuáles son los mecanismos que permiten combinar expresiones semánticas. Para ello, debemos introducir el concepto de regla orientada a objetos, relacionado estrechamente con nuestro léxico jerarquizado, pero que presenta alguna novedad respecto a otros sistemas basados también en representaciones jerárquicas, como HPSG.

Herencia y gramática

Los principios de programación orientada a objetos (OOP) están convirtiéndose en un nuevo paradigma dentro de la comunidad informática (Meyer 88). Al mismo tiempo, la mayoría de los formalismos lingüísticos objeto de estudio en la actualidad son orientados a objetos en un cierto sentido. Todos ellos se centran en la representación de los objetos lingüísticos básicos: palabras y signos en general. Desde que se desarrolló el concepto de *hierarchical lexicon* (Flickinger et al. 85), una de las prioridades de la investigación ha sido la de desarrollar y estudiar los marcos formales para representar objetos lingüísticos y las relaciones entre ellos.

No sorprende, pues, encontrar teorías linguisticas (de alto nivel) y herramientas de descripción linguística (de bajo nivel) cuyo sustrato descriptivo se basa en *feature structures* (fs) y en redes de herencia.

Quizá el ejemplo más notable de formalismo computacional del segundo tipo es TFS (Emele y Zajac 90). En él, tanto gramática como léxico se representan mediante una red de herencia de *feature structures* tipadas (tfs). Una red de herencia de tfs puede expresar desde una DCG hasta una gramática HPSG (Pollard y Sag 93). Las tfs superiores de la jerarquía expresan principios o restricciones generales sobre los objetos linguisticos que heredan de ellas. Las tfs más bajas de la jerarquía son entradas del léxico.

Uno de los rasgos decisivos de una red de herencia es que presente, o no, monotonicidad (Daelemans et al. 92). Los formalismos que preservan la monotonicidad tienen propiedades muy estimables desde un punto de vista lógico-computacional. Sin embargo, la expresividad y la concisión de las redes que presentan herencia multiple y herencia por defecto (ambas no-monotónicas) las hacen

muy adecuadas desde un punto de vista lingüístico. Su principal problema es que presentan muchas dificultades de especificación cuando se usan conjuntamente con la unificación de feature structures en las que hay valores compartidos (*reentrancy*), como se pone de manifiesto en (Bouma 92).

En el formalismo de Prógenes hemos intentado sacar partido de un léxico organizado en una red de herencia por defecto, minimizando sus defectos. El utilizar directamente un mecanismo de unificación como el de UCG sobre nuestro dominio tiene dos inconvenientes: el primero, la poca eficiencia. El segundo, los conflictos que pueden surgir al aplicarlo sobre nuestra red de herencia por defecto. Para ello hemos tomado dos medidas: la primera, eliminar de los objetos la compartición de valores. La segunda, hemos expresado los mecanismos de combinación gramatical como un conjunto de *reglas por defecto* binarias aplicables sobre las clases de la jerarquía (la llamaremos OOG). La especificación de las clases sobre las que se aplica cada regla es similar a una gramática libre de contexto. La diferencia estriba en que los no-terminales clásicos han sido sustituidos por clases. Las reglas especificadas sobre clases superiores de la jerarquía serán aplicables casi siempre, y aquellas escritas sobre las clases inferiores solo seran aplicables en casos particulares. Al aplicar la gramática sobre un par de objetos, encontraremos en general varias reglas válidas, y la más específica anulará el efecto del resto. Este comportamiento esencialmente no-monotónico diferencia esencialmente este mecanismo del usado en UCG o HPSG. Definiendo reglas por defecto sobre las clases superiores de la jerarquía debe reducirse drásticamente el tamaño de las gramáticas, frente a una CFG. De esta forma podemos, en principio, preservar gran parte de la generalidad de la unificación. Además, podemos resolver situaciones excepcionales de una forma modular y económica, definiendo reglas específicas que anulen a las generales cuando sea preciso.

Parte de los conceptos necesarios para adaptar las reglas gramaticales a una red de herencia se encuentran en los principios de Programación Orientada a Objetos (Booch 90). Dentro de este paradigma, los objetos se organizan jerárquicamente de la misma forma en la que los signos lingüísticos se organizan en un léxico jerarquizado. Se pueden definir funciones que actúan sobre estos objetos. Pero, al contrario que en programación convencional, estas funciones están formadas por *métodos*. Los métodos se diferencian entre sí por las clases de los argumentos sobre los que se aplican. Se definen independientemente unos de otros, y nunca son llamados directamente. Cuando se llama a una función, las clases de los argumentos, junto con la red de herencia, deciden qué método será usado. El conjunto de los métodos aplicables está formado por todos los métodos definidos sobre clases compatibles con las de los argumentos. Dentro de este conjunto, el método usado finalmente es el más específico (a este proceso se le llama *dynamic binding* en OOP).[2]

En la interfaz de Prógenes, una regla gramatical es el equivalente lingüístico de un método. En particular, al concepto de método con las restricciones y características propias de Common Lisp Object System (CLOS)(Steele 90), que es el lenguaje usado para su implementación. Definimos reglas gramaticales que son operativas sobre ciertas clases de signos. Estas reglas se agrupan en *reglas genéricas*. Cuando se aplica una regla genérica sobre dos signos, sus clases determinan qué regla será usada. De esta forma no hay ya necesidad de definir reglas sobre cada par posible de clases. El diseño jerárquico nos permite definir reglas por defecto que sean siempre - o casi siempre - aplicables. Reglas específicas sustituirán a las anteriores en situaciones excepcionales.

Existen, además, motivaciones computacionales para adoptar este sistema. Los fenómenos complejos del lenguaje natural -como la coordinación o las dependencias a larga distancia- son tratados dentro de las implementaciones mediante dispositivos extragramaticales. Cierto número de ellos es *demon-based* (Haugeneder 92). En (Huang 83), por ejemplo, una subgramática especifica de la coordinación se dispara sólo cuando se la requiere. Dentro de una aproximación orientada a objetos, cada regla puede ser vista como un *demon*. Al ser definida, cada regla adquiere automáticamente el grado de

[2] Decidir qué método es el más específico no es siempre una tarea simple. Las dificultades son similares a aquellas que se encuentran al manejar redes con herencia múltiple o herencia por defecto (Selman and Levesque 89).

especificidad adecuado. Además, cualquier procedimiento particular puede ser integrado dentro de la gramática de una forma totalmente modular. Finalmente, los lenguajes de programación orientados a objeto permiten implementar este tipo de gramáticas de forma trivial.

Quizás el punto más delicado de esta aproximación es el de seleccionar la regla que se usará en cada momento entre las aplicables. En determinadas ocasiones, entran en conflicto reglas con el mismo grado de especificidad, y la elección es arbitraria. Parte de nuestro trabajo se ha centrado en la definición teórica de las condiciones de prioridad entre reglas, buscando una axiomatización que limite los procedimientos, arbitrarios en mayor o menor medida, de los lenguajes de programación orientados a objeto que permiten herencia por defecto.

A continuación vamos a describir los métodos que llevan a cabo los mecanismos de combinación semántica.

Reglas semánticas en Prógenes.

Llamaremos sign, como en HPSG, a la clase superior de la jerarquía de objetos. Por herencia, todas las entradas del léxico y todos los análisis parciales pertenecen a esta clase. Sus atributos principales son tres: spelling, category (cat) y semantics (sem). El atributo category tiene como valor el papel sintáctico del objeto en forma categorial (vg., en el caso de un determinante sería NP/NP). El atributo semantics contiene el valor semántico del objeto, según lo descrito en párrafos anteriores.

Las reglas sintácticas, semánticas y globales del sistema se aplican sobre un par contiguo y ordenado de objetos para dar otro objeto, del que son sus hijos.

A continuación describiremos una parte sustancial de la gramática que rige la combinación semántica.

Regla por defecto

Dentro de las reglas semánticas, una de ellas se puede aplicar siempre que ambos objetos pertenezcan a la clase *sign*. Como siempre se da este extremo, la regla es siempre aplicable. Sólo cuando existan reglas más específicas no será disparada . En nuestro entorno, juega el papel que la unificación tiene en UCG. Pero ese es el único mecanismo del que dispone UCG, lo que dificulta en ese formalismo el tratamiento de fenómenos complejos o específicos. Nuestras reglas por defecto tienen también cierto paralelismo con cierto tipo de reglas en HPSG, ya que ambas se definen sobre estructuras tipadas de pares atributo-valor. Una diferencia menor es que en HPSG el orden se especifica de forma distinta (mediante reglas de "Linear Precedence"). Una diferencia mayor es que las reglas de HPSG, expresadas en forma de restricciones, deben cumplirse siempre; no se contempla en la teoría la posibilidad de que sean incumplidas por el hecho de que otras reglas más específicas tomen prioridad sobre éstas.

La regla para dos objetos *sign sign* se inspira en la conversión λ de los formalismos Montagovianos:

La semántica global de dos objetos s1 y s2 de tipo signo se obtiene reemplazando por el valor semántico de s2 la variable semántica más cercana cuyo tipo sea compatible con el de s2.

Por ejemplo, la combinación de `pendiente:(NUM (slope (LINE @1)))` con `de+la+recta+$y=x$:(LINE (line (LINEAR-EQUATION $y=x$))` da lugar a `pendiente+de+la+recta+$y=x$: (NUM (slope (LINE (line (LINEAR-EQUATION $y=x$))))`.

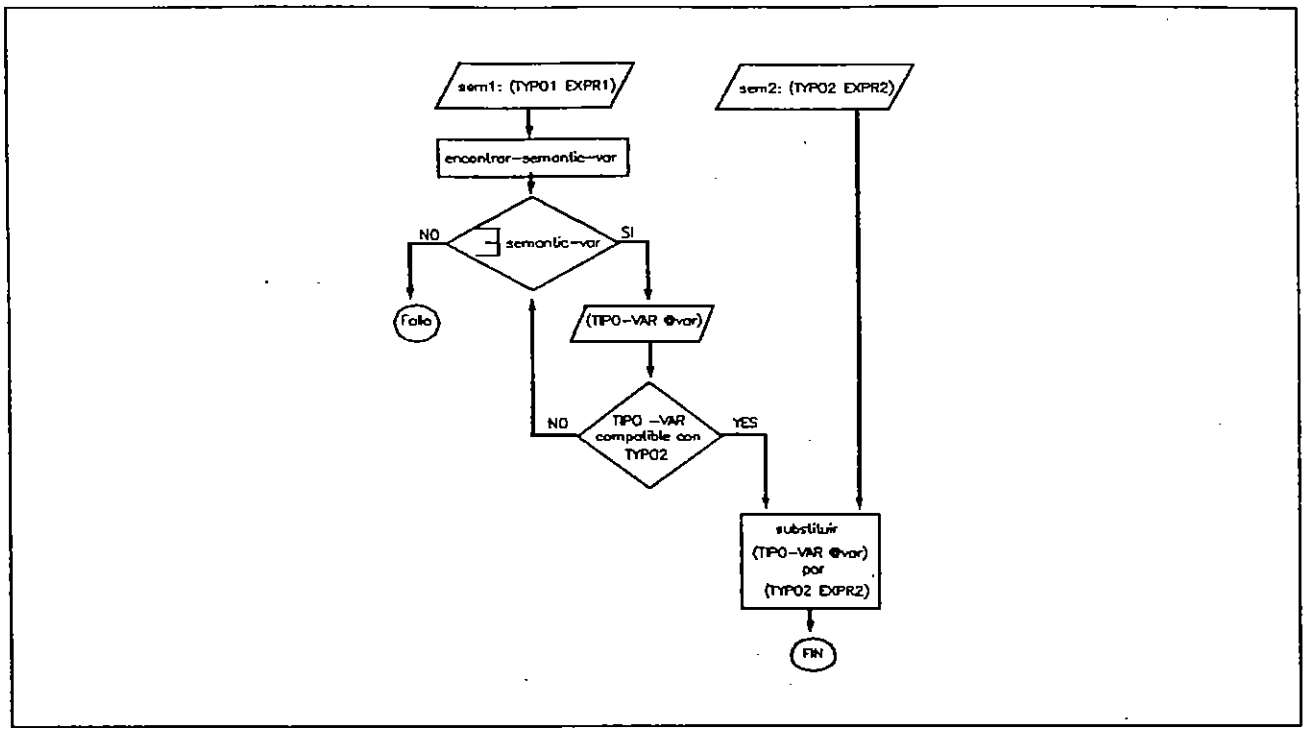

Figura 2. Combinación semántica por defecto.

Determinantes

La acción del determinante no puede ser descrita mediante el algoritmo general. Un determinante puede tener dos funciones distintas sobre un nombre, según su alcance. En una de ellas coinciden denotación e intensión:

```
La+recta+$y=x$ :
 (LINE (line (LINEAR-EQUATION $y=x$)))
```

En la otra son distintas:

```
La+recta+que+pasa+por+p(3,2)+y+q(4,1):
 (LINE (pthe (LINE (: (LINE %X) (LINE LINE)))
             (BOOL (and (BOOL (in (POINT p(3,2))
                                  (LINE %X)))
                        (BOOL (in (POINT q(4,1))
                                  (LINE %X)))))))
```

Por ello existe un método *article noun* que da dos semánticas posibles para el análisis de, vg., la y recta:

```
recta:
(LINE (line (LINEAR-EQUATION @1)))

la+recta:
(LINE (line (LINEAR-EQUATION @1)))
(LINE (pthe (LINE (: (LINE %X) (LINE LINE))
            (BOOL @b)))
```

Durante el proceso de análisis, la expresión correspondiente a `la+recta+que+pasa+por+$p(3,2)$+y+$q(4,1)$` adquirirá, en virtud de la regla general, la semántica

```
 (LINE (pthe (LINE (: (LINE %X) (LINE LINE)))
             (BOOL (and (BOOL (in (POINT p(3,2))
                                  (LINE @1)))
                        (BOOL (in (POINT q(4,1))
                                  (LINE @1)))))))
```

Esta expresión difiere de la correcta en que hay dos variables sin instanciar. El proceso de encontrar sus referentes se realiza al final del análisis. Por estar en el entorno de denotación de la variable %X, que tiene tipo LINE, se tomará (LINE %X) como el referente adecuado para las dos expresiones (LINE @1).

Coordinación de constituyentes

Consideremos el siguiente problema:

```
"Hallar la recta que pasa por p(3,2) y q(4,1)"
```

Existen, al menos, tres semánticas posibles para esta oración, provenientes de la ambigüedad que introduce la conjunción:

```
(A) "Hallar la recta que pasa por p(3,2) y q(4,1)"
(ACTION (find
          (LINE (pthe (LINE (: (VAR %X) (LINE LINE)))
                      (BOOL (in (LINE %X)
                                (POINT  (and (POINT p(3,2))
                                             (POINT q(4,1)))))))))


(B)"Hallar la recta que pasa por p(3,2) y pasa por q(4,1)"
 (ACTION (find
          (LINE (pthe (LINE (: (VAR %X) (LINE LINE)))
                      (BOOL (and
                              (BOOL (in (VAR %X) (POINT p(3,2))))
                              (BOOL (in (VAR %X) (POINT q(4,1)))))))))


 * (C)"Hallar la recta que pasa por p(3,2) y hallar la recta
      que pasa por q(4,1)"
 (ACTION (and
          (ACTION (find
                   (LINE (pthe (LINE (: (VAR %X) (LINE LINE)))
                               (BOOL (in (LINE %X) p(3,2))))))))
          (ACTION (find
                   (LINE (pthe (LINE (: (VAR Y) (LINE LINE))))
                               (BOOL (in (LINE %Y) p(4,1)))))))))
```

Las semánticas de los constituyentes coordinados ("p(3,2) y q(4,1)") no coinciden con las semánticas coordinadas en la expresión global correcta (que corresponderían a la semántica de "pasa por p(3,2) y pasa por q(4,1)").

Este ejemplo llama la atención sobre algunas de las dificultades que se plantean al analizar estructuras coordinadas. Se necesita una gramática para la coordinación que cumpla tres requisitos: uno, ser capaz de dar cuenta de las diferencias entre coordinación sintáctica y coordinación semántica; dos, ser capaz de evitar la explosión combinatoria de análisis usual de la coordinación (en el ejemplo anterior, esa condición se traduce en producir sólo la opción (B)); y tres, no interferir en el funcionamiento del resto de la gramática.

Para conseguirlo, hemos definido en Prógenes clases y métodos que aplican propiedades semánticas distributivas de la coordinación restringidas por un principio de *coordinación completa*. Veamos en qué consisten.

En nuestro corpus, las expresiones semánticas que pueden ser coordinadas en una expresión completa son siempre predicados (acciones y condiciones).[3] Esto quiere decir que, independientemente de lo que coordinemos sintácticamente (que puede ser casi cualquier cosa), la interpretación semántica será finalmente una coordinación de acciones o condiciones. En la jerarquía de tipos semánticos, los predicados pueden ser de tipo ACTION (vg. "Hallar la recta..") o BOOLEAN (vg. "pasa por p(4,1)") Diremos que una coordinación cuyo tipo pertenezca a alguno de esos dos es *completa*. En caso contrario, será *incompleta*.

Adoptaremos el siguiente principio:

El alcance de una coordinación será siempre el de la minima coordinación completa.

De acuerdo con este principio, (B) es el único análisis posible para el ejemplo anterior. En (A), la coordinación no ha sido completada. En (C), la coordinación es completa, pero no mínima.

Las coordinaciones, completas o incompletas, presentan la propiedad de distribución semántica por la derecha. Esto quiere decir que, al combinar su semántica con la de un constituyente que esté a su derecha, la semántica del otro se insertará en cada uno de los miembros coordinados. Cuando una coordinación es incompleta, presenta además la propiedad de distribución semántica por la izquierda. Esta simple distinción nos ha permitido obtener una sola interpretación en todos los casos tratados hasta ahora.[4]

En una gramática libre de contexto, necesitaríamos, al menos, cuatro reglas distintas, según la coordinación estuviera a la izquierda o a la derecha y según fuese completa o incompleta. Aunque la regla de combinación a la derecha sea la misma para las dos, es necesario especificarla para ambos tipos de no terminales : coordinaciones completas e incompletas.

Una gramática orientada a objeto ofrece más flexibilidad. En nuestro caso definimos una clase *coordination* y una subclase de ésta, *unfinished-coordination*.

Escribiremos una regla para clases *sign unfinished-coordination* que regule el comportamiento especial de las incompletas por la izquierda. En virtud de la jerarquía, sobre las que sean del tipo más general *coordination* se podrá aplicar la regla general *sign sign* ya explicada. Escribiremos otra para clases *coordination sign* que será aplicable por igual a objetos *coordination* y *unfinished-coordination*, ya que la segunda es subclase de la primera.

Supuesto que las coordinaciones mostraran algún tipo de comportamiento especial frente a objetos de cierta clase, se podrían escribir reglas adicionales que sustituirían automáticamente a estas cuando la jerarquía de clases así lo indicara.

Veamos brevemente como se realiza en la práctica lo descrito en los últimos párrafos:

Definimos las siguientes clases de signos: *Conjunction*, *Pre-coordination* y *Coordination*, que son subclases de *sign*, y *unfinished-coordination*, que es una subclase de *coordination*.

A continuación se describen brevemente las reglas relacionadas con estas clases.

- *sign conjunction*

---

[3] Sería interesante intentar extender esta condición a dominios más generales.

[4] Excluimos de la discusión casos como "triángulo con vértices a,b y c", que son resueltos de forma sencilla sin interferir con los casos aquí estudiados. Tampoco mencionaremos el tratamiento de coordinaciones con más de dos elementos, que solo requiere la generalización de los métodos aquí descritos.

La semántica se obtiene aplicando el principio general, pero con el orden invertido. La clase del objeto resultante será *pre-coordination*. Por ejemplo:

```
sign                                    conjunction
p(3,2)                                  y
cat: NP                                 cat: (X/X)\X
sem: (POINT p(3,2))                     sem: (TYPE (and (TYPE @t)
                                                        (TYPE @t)))
```

da lugar a

```
pre-coordination
p(3,2)+y
cat:NP/NP
sem: (TYPE (and (POINT p(3,2))
                (TYPE @t)))
```

- *pre-coordination sign*

La semántica del objeto *sign* es embebida en la semántica de la pre-coordinación. Su tipo es el supertipo común más específico de los tipos de las dos expresiones coordinadas. La clase del objeto resultante depende de ese tipo: será *coordination* si su tipo es ACTION o BOOLEAN, y *unfinished-coordination* en caso contrario. Por ejemplo:

```
sign
LA+PENDIENTE+Y
cat: NP
sem:  (TYPE (and (NUM (slope (LINE @1)))
                 (TYPE @t)))

sign
LA+INTERSECCION+CON+EL EJE+X
cat: NP
sem:  (SET (intersection (AXIS X-AXIS)
                         (SET @s)))
```

dan lugar a

```
unfinished-coordination
LA+PENDIENTE+Y+LA+INTERSECCION+CON+EL+EJE+X
cat: NP
sem: (SET (and (NUM (slope (LINE @1)))
               (SET (intersection (AXIS X-AXIS)
                                  (SET @s)))))
```

- coordination sign

La semántica del objeto *sign* es embebida distributivamente en las dos expresiones coordinadas. El objeto resultante es de clase *coordination*. Como *unfinished-coordination* es subclase de *coordination*, el método se aplica sobre ambos.

Un ejemplo:

```
unfinished-coordination
LA+PENDIENTE+Y+LA+INTERSECCION+CON+EL+EJE+X
cat: NP
sem: (SET (and (NUM (slope (LINE @1)))
               (SET (intersection (AXIS X-AXIS)
                                  (SET @s)))))

sign
DE+LA+RECTA+$Y=3X+2$
cat: NP\NP
sem: (LINE (line (LINEAR-EQUATION $y=3x+2$)))

unfinished-coordination
LA+PENDIENTE+Y+LA+INTERSECCION+CON+EL+EJE+X+DE+LA+RECTA+$Y=3X+2$
cat: NP
sem: (SET (and (NUM (slope (LINE @1)))
               (SET (intersection (LINE (line (LINEAR-EQUATION $Y=3X+2$)))
                                  (LINE (line (AXIS X-axis)))))))
```

- sign unfinished-coordination

El resultado es una coordinación en la que cada uno de los elementos coordinados resulta de aplicar el método por defecto a la semántica del objeto *sign* junto con cada una de las semánticas coordinadas en el objeto *unfinished-coordination*. El objeto resultante será de clase *coordination* si su semántica es una coordinación completa, y *unfinished-coordination* si no es así.

Por ejemplo, aplicado a PASA y POR+P(3,2)+Y+Q(4,1) este método llega a:

```
coordination
PASA+POR+P(3,2)+Y+Q(4,1)
cat:  S\NP
sem:  (BOOL (and (BOOL (in (LINE @1) (POINT (3 2)))
                 (BOOL (in (LINE @1) (POINT (4 1))))))
```

En la figura 3 puede verse cómo se combinan los distintos métodos para analizar un enunciado que requiere estos métodos.

Referencias

(Booch 90) BOOCH,G. "Object Oriented Design With Applications", Benjamin/Cummings 1990.

(Bouma 92) BOUMA,G.: "Feature Structures and Nonmonotonicity" en *Computational Linguistics* ,vol 18-2, Junio 1992.

(Calder et al. 88) CALDER, J., KLEIN, E. and ZEEVAT, H.: "Unification Categorial Grammar: A concise, extendable grammar for Natural Language Processing ", COLING 88, Budapest 1988.

(Carpenter 92a) Carpenter,R.: *The Logic of Typed Feature Structures*, Cambridge University Press 1992.

(Carpenter 92b) Carpenter, R.: *ALE User's Guide* Report of the Laboratory for Computational Linguistics, Carnegie Mellon University 1992.

(Castells et al. 92) CASTELLS,P., DIAZ,J., GONZALO,J., MORIYON,R., RODRIGUEZ-MARIN,P., SAIZ,F. and TOBAR,M.J.: "A scientific Problem Solver with a Natural Language Interface", *Seventh International Symposium on Computer and Information Sciences: ISCIS VII*, Turquía 1992.

(Daelemans et al. 92) DAELEMANS,W. DE SMEDT,K. and GAZDAR,G.: "Inheritance in Natural Language Processing", *Computational Linguistics* ,vol 18-2, Junio 1992.

(Díaz y Rodríguez-Marín 91) DIAZ,J. y RODRIGUEZ-MARIN,P.: "Prógenes: La Interfaz de Lenguaje Natural", boletín de la SEPLN n. 11, 1991.

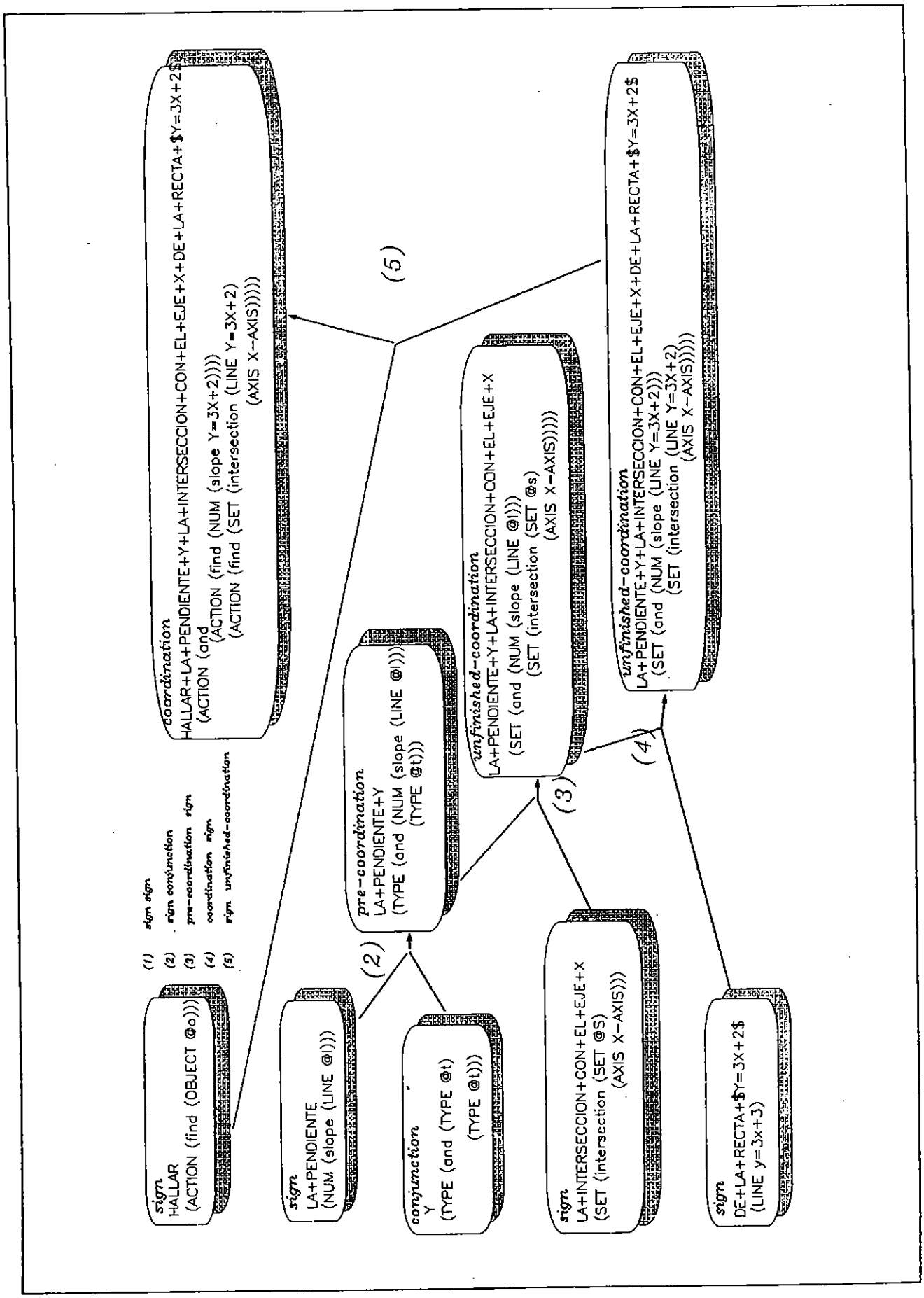

Figura 3. Métodos semánticos.

(Emele and Zajac 90) EMELE, M.C. y ZAJAC, R.: "Typed unification grammars", *Coling 90,* , vol.2, Helsinki 1990.

(Flickinger et al. 85) FLICKINGER, D., POLLARD, C. and WASOW, T: "Structure-sharing in Lexical Representation", in *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics* 262-267, University of Chicago, Chicago, 1985.

(Flickinger y Nerbonne 92) FLICKINGER,D. and NERBONNE,J.: "Inheritance and Complementation: a Case Study of Easy Adjectives and Related Nouns", *Computational Linguistics* vol. 18-3, 1992.

(Gonzalo, Rodriguez-Marín and Diaz 92) GONZALO,J., RODRIGUEZ-MARIN,P. and DIAZ,J.: "Formal Representation of Mathematical Texts", in *Les chemins du texte*, Slatkine-Champion, Géneve-Paris 1992.

(Haugeneder 92) HAUGENEDER, H.: "A Computational Model for processing coordinate structures: Parsing Coordination without grammatical specification", *ECAI 92*, Wiley & Sons 1992.

(Huang 83) HUANG,X.: "Dealing with conjunction in a machine translation environment", in Proc. *1st meeting of the European Chapter of the ACL*, 1983.

(Kamp 81) KAMP,H.: "A theory of truth and semantic representation", en *Formal Methods in the Study of Language*, vol. 136, Amsterdam, Mathematical Centre Tracks.

(Meyer 88) MEYER,B.: "Object-Oriented Software Construction", Prentice Hall, New York 1988.

(Oehrle et al. 88) OEHRLE,R., BACH,E. and WHEELER,D.(eds): "Categorial Grammars and Natural Language Structures", Reidel 1988.

(Pollard y Sag 93) POLLARD, C.J. and SAG,I.A.: "Head-Driven Phrase Structure Grammar" University of Chicago Press and CSLI publications, en prensa.

(Selman y Lebesque 89) SELMAN,B. y LEBESQUE,H.: "The tractability of path-based inheritance", IJCAI 89, 1989.

(Shieber 86) SHIEBER,S.M.: "An introduction to Unification-based Approaches to Grammar", University of Chicago Press 1986.

(Steele 90) STEELE,G.: "Common Lisp: The Language, Second Edition", Digital Press 1990.

(Zeevat et al. 87) ZEEVAT,H. KLEIN,E. y CALDER,J.: "Unification Categorial Grammar", en *Working Papers in Cognitive Science*, University of Edinburgh 1987.