Integración de Herramientas para el Análisis Automático de los Lenguajes Naturales.

Autor: David Cabrero Souto
Director: Manuel Vilares Ferro

El presente trabajo ha sido realizado usando LaTeX y vi.
¡Toma ya !

Con estas razones perdía el pobre caballero el juicio, y desvelábase por entenderlas y desentrañarles el sentido, que no se lo sacara ni las entendiera el mesmo Aristóteles, si resucitara para sólo ello.

[Miguel de Cervantes]

Agradecimientos

En su momento, dijo Ortega y Gasset, "*yo soy yo y mi circunstancia*". De la misma manera, considero la presente tesis de licenciatura fruto de mi trabajo y de su circunstancia. Por todo ello, quisiera mostrar mi agradecimiento a todos aquellos que han influenciado esta circunstancia.

Especial mención merece el doctor M. Vilares, encargado de dirigir mi trabajo, a pesar de lo difícil de la tarea, revisarlo, y ofrecerme buenos consejos, no obstante mi reiterada "insistencia" en algunos casos, ...

Mi agradecimiento también es para Miguel Alonso, Víctor Gulías, Jorge Graña, J.J., y ese largo etcetera de personas que nunca son mencionadas por la pereza o descuido del autor, a las que pido mis más sinceras disculpas.

Por último mi más sincero agradecimiento es para mi familia y para dos personas que aún no habiendo intervenido directamente en la realización de esta tesina, les considero los más directos responsables de la misma: mi padre y mi madre.

Índice General

Capítulo 1

Introducción.

1.1 Carácter multidisciplinar del trabajo

El carácter multidisciplinar del trabajo presentado viene dado por la necesidad de sintetizar y coordinar el conocimiento proveniente de las áreas de la lingüística, las matemáticas y la informática.

La primera será la encargada de describir los fenómenos y principios que rigen el *lenguaje natural*[1] y las teorías en las que se engloban. Sobre la segunda recae la responsabilidad de establecer los modelos formales a partir de los cuales la tercera desarrollará mecanismos capaces de extraer automáticamente la información lingüística contenida en los textos que deseemos analizar.

Por suerte o por desgracia, en ninguno de los campos mencionados existe una única teoría que abarque todas las posibles soluciones, no obstante resulta bastante común partir de una clasificación de la información lingüística como la presentada en la figura 1.1. Esta distribución debe entenderse como una distribución de carácter difuso. Como botón de muestra, baste significar que muchos autores consideran que parte de la información semántica se debe determinar en un nivel morfológico, al considerar que las palabras se distribuyen en una jerarquía semántica, o simplemente resaltar la polémica existente acerca de que información es relevante o no en cada nivel.

No obstante, en el trabajo que ha continuación se presenta, se ha optado por separar, en diferentes herramientas, el análisis morfológico del análisis sintáctico, integrando posteriormente ambas, de modo que los resultados obtenidos por una primera fase de análisis, puedan ser usados en la siguiente.

1.2 Situación del trabajo

El trabajo expuesto en la presente memoria ha sido realizado dentro de los proyectos de procesamiento del lenguaje natural, *GA*LENA[2] y *X*IADA[3]. Ambos presentan una misma estructura, diferenciándose principalmente en que el primero trata el *Castellano* y el segundo el *gallego*. En estos proyectos trabajan los siguientes grupos:

[1] De manera intuitiva, el lenguaje que usamos las personas para comunicarnos entre nosotros.

[2] Generador de Analizadores de los LEnguajes NAturales

[3] Xerador de Analizadores DA lingua galega.

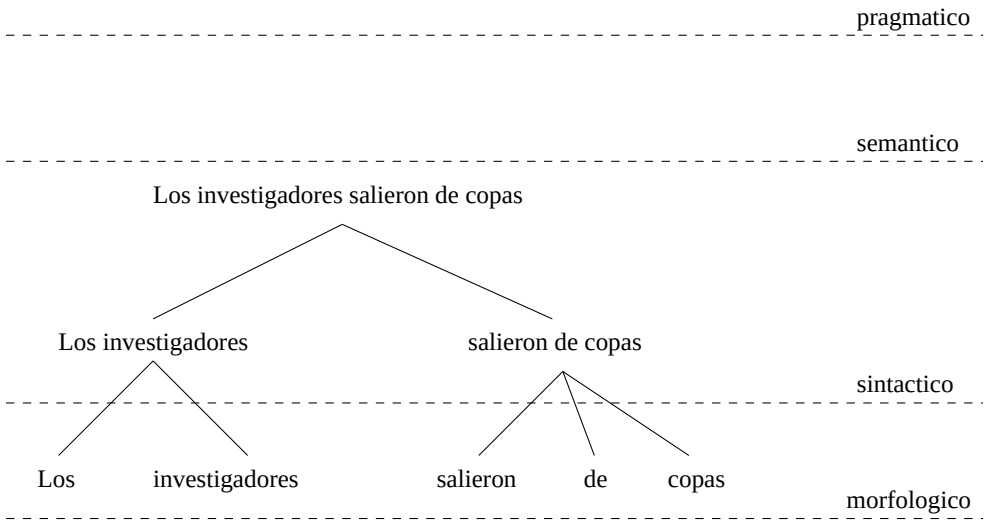


Figura 1.1: Información lingüística en diferentes niveles.

- Grupo *COLE* [4], Universidade da Coruña.
- Grupo de Sintaxis del Español, Universidad de Santiago de Compostela.
- Departamento de Filología española, Teoría de la Literatura y Lingüística General, Universidad de Vigo.
- Centro Ramón Piñeiro para a Investigación en Humanidades, Santiago de Compostela.

En lo concerniente al análisis sintáctico, se establece una estrecha colaboración con el proyecto *Atoll*, INRIA[5], Francia, Ecole Nationale Supérieure des Télécommunications de Paris y Ecole Polytechnique Dederale de Lausanne.

1.3 Propósito y guía de la memoria

El objetivo de la presente memoria es describir el trabajo realizado dentro de los proyectos *GA*LeNAy *X*IADA, poniendo un mayor énfasis en aquellas partes donde la aportación del autor ha sido más relevante. El resto de capítulos se organizan de la siguiente manera:

El capítulo 2 está dedicado al *analizador morfológico*. El creciente interés demostrado en el uso de modelos de *autómatas finitos* [65] para este tipo de analizadores parece indicar un camino correcto. Aunque la eficacia de estos modelos, tanto en lo referente a la velocidad como a la compactación de los resultados, constituye un aspecto francamente positivo, su debilidad es su falta de flexibilidad descriptiva, principal ventaja de los enfoques orientados a gramáticas [62]. Para solventar este problema, se ha desarrollado un formalismo descriptivo, al estilo de los analizadores de dos niveles,

[4] *CO*mpiladores y *LE*nguajes.
[5] Institut National de Recherche en Informatique et en Automatique

basado en la factorización de los fenómenos morfológicos descritos, y que puede ser transformado de forma automática en un modelo de autómatas finitos.

El capítulo 3 es una introducción al análisis sintáctico. En él se describen las *gramáticas de cláusulas definidas* [57], ampliamente utilizadas en el *procesamiento del lenguaje natural*. También en este capítulo, se describe la integración del analizador morfológico con el sintáctico, centrándose en cómo se realiza dicha integración.

Por su parte el capítulo 4 se centra en el formalismo operacional desarrollado para el analizador sintáctico. Las técnicas clásicas basadas en estrategias descendentes con retroceso como las utilizadas en PROLOG muestran problemas de eficiencia, completud, y terminación. Para sobreponerse a estos problemas se decidió basar el modelo operacional en los *Autómatas Lógicos a Pila*, LPDA [46], poniendo especial énfasis en la reducción y compactación del *espacio de búsqueda* para asegurar tanto la eficacia computacional, como la completud y terminación de los esquemas de evaluación.

El capítulo 5 presenta una amplia gama de resultados, así como su interpretación y comparación con otros sistemas.

En el capítulo 6 se exponen las conclusiones sobre el trabajo realizado y las mejoras y ampliaciones que restan por realizar.

Por último en los apéndices A, y B se detallan algunos aspectos introducidos al hablar del formalismo operacional, como los *autómatas LR*, *programación dinámica* o el *algoritmo de Earley*. Por su parte, el apéndice C está dedicado a la distribución de los recursos desarrollados para facilitar su uso y acceso.

Capítulo 2

Análisis morfológico.

El análisis morfológico se presenta como el primer paso en el tratamiento automático de los lenguajes naturales. Su objetivo es la etiquetación y clasificación de las palabras en categorías sintácticas y la posterior eliminación de las posibles ambigüedades aparecidas en este proceso.

2.1 Introducción.

En adelante usaremos el término *morfología computacional* para referirnos a la parte de la lingüística computacional dedicada al estudio de:

1. Las unidades que forman las palabras y como se relacionan entre sí.
2. La teoría de las categorías léxicas (sustantivo, adjetivo, verbo, ...).

Tradicionalmente los componentes morfológicos se definen como las unidades mínimas del análisis gramatical: *morfemas*. Las *palabras* se forman combinando diferentes morfemas mediante dos procesos principales:

- *Flexión*, consistente en la unión de morfemas flexivos:
  - *conjugación* o *flexión verbal*, y
  - *declinación* o *flexión nominal*.
- *Derivación*, o unión de morfemas derivativos, también conocida por formación de palabras.

Podemos establecer la diferenciación entre ambos procesos en base a los siguientes factores:

- La flexión no cambia la parte del discurso[1] de la palabra, al contrario que la derivación. Por ejemplo, tras aplicar el morfema de plural -s a la palabra chico, para formar chicos, ésta sigue siendo un nombre, sin embargo si derivamos estupidez a partir de estúpido, hemos pasado de un nombre a un adjetivo.

[1] *part–of–speech*, en la terminología anglosajona.

- Frecuentemente la flexión es requerida por un contexto sintáctico concreto, cosa que no ocurre con la derivación. Como ejemplo podemos observar la siguiente frase en Latín: Puer Ciceronem laudat, en este caso Puer adopta la forma de nominativo porque funciona como sujeto y Ciceronem la de acusativo puesto que desempeña el papel de objeto directo. Por otra parte en Everybody desires happiness vemos como cualquier nombre puede ser objeto directo sin importar la forma en que ha sido derivado.

- Puesto que la inflexión es a menudo requerida por la sintaxis, si un lenguaje marca una cierta categoría inflexional, entonces debe marcar esa categoría en todas las palabras apropiadas. Por contra el razonamiento equivalente no es aplicable a las formas derivadas. Volviendo al ejemplo del latín, todos los nombres deben tener una forma de acusativo. Por último, para ejemplificar el caso de la derivación, baste decir que aunque algunos adjetivos puedan derivar nombres como en el caso de happy, happiness; no es cierto que este fenómeno se repita en todos los casos.

Partiendo de estos procesos se han desarrollado diferentes modelos para describir la estructura morfológica de las palabras. Según Hockett [53]:

- *Palabras y paradigma.* Las palabras se clasifican por categorías. Cada *categoría* se caracteriza por un *paradigma* o grupo de palabras que sirve como modelo[2]. Las palabras se componen de lexemas y terminaciones.

- *Elementos y colocación.* Las palabras se segmentan en morfemas relacionados únicamente por su simple sucesión.

- *Elementos y proceso.* Los morfemas se clasifican en léxicos y flexivos o derivativos. En virtud de los procesos morfológicos los segundos modifican a los primeros, dando lugar a la *forma superficial*[3]. *forma gráfica*

Llegados a este punto, podemos definir nuestro objetivo como la creación de sistemas *software* capaces de analizar palabras, descomponiéndolas en sus constituyentes morfológicos, i.e. morfemas.

2.2 El caso del Castellano y del Gallego.

Tanto el Castellano como el Gallego son dos lenguajes con una gran variedad de procesos morfológicos, especialmente no concatenativos, derivados de su origen del Latín. A la hora de elaborar reglas que describan estos procesos flexivos nos encontramos con las siguientes dificultades:

- Un paradigma verbal realmente complicado, con 108 formas flexionadas en Castellano, incluyendo tiempos verbales simples y compuestos, y 65 formas en Gallego.

[2]Por ejemplo para la primera conjugación: amar
[3]Es decir, la palabra tal y como la escribimos. También usaremos el término *forma gráfica*

- Frecuentes irregularidades tanto en la raíz verbal como en sus terminaciones. Hasta un 30% de los verbos se clasifican como irregulares, con verbos que presentan hasta 7 raíces diferentes, como es el caso del verbo hacer: hac–er, hag–o, hic–e, ha–ré, hiz–o, haz, hech–o. En total 42 grupos de verbos irregulares en Gallego y 38 en Castellano.

  Por otra parte existen verbos extremadamente irregulares que no pueden ser encajados en ningún modelo de conjugación; como ir o ser.

- La posible inclusión de pronombres enclíticos, que pueden añadirse, hasta un máximo de tres en Castellano y cuatro en Gallego, a las formas verbales: tráe–te–me–lo, déu–lle–lo.

  En el caso del Gallego los pronombres se pueden contraer: vaichemo buscar, me + o = mo.

- Presencia de los llamados verbos defectivos, que carecen de ciertas formas. Es el caso de los verbos meteorológicos: nevar, llover, ..., del verbo soler que carece de tiempos compuestos o de otros más peculiares como abolir que carece de algunas formas del presente de indicativo y todas las del presente de subjuntivo y de la segunda persona del singular de imperativo.

- Duplicaciones en los participios, por ejemplo; impreso e imprimido.

- La casuística en la inflexión del género incluye una veintena de casos diferentes para el Castellano, 32 para el Gallego. Esta casuística va desde aquellas palabras que presentan un género gramatical o un género heredado hasta aquellas que presentan la misma forma tanto para el masculino como para el femenino, (azul).

- Por otra parte, en el caso del número llegamos a la decena de esquemas flexivos en Castellano, y trece en Gallego, incluyendo nombres y adjetivos con formas alternativas del plural (bambús, bambúes), nombres con la misma forma para el plural y el singular (crisis), palabras que sólo se usan en su forma de singular (estrés) o del plural (matemáticas).

- La aparición de alomorfos[4] al añadir sufijos de género y número (luz, luc–es).

  En el Gallego este problema se ve agravado debido a las contracciones (da + lles + o = dallelo *o* dalles lo, a + os = ós).

2.3 Formalismo descriptivo.

El formalismo elegido se basa en el modelo *palabras y paradigma* introducido en la sección 2.1. A la hora de definir las diferentes clases de palabras usaremos el concepto de *categorías sintácticas* (ver apéndice C). Tradicionalmente se distinguen diez categorías básicas:

[4]Cada una de las variantes de un morfema en función de un contexto y significado idénticos: -s y -es son *alomorfos* del plural en Castellano.

sustantivo	*adjetivo*	*pronombre personal*
adverbio	*verbo*	*determinante*
preposición	*conjunción*	*numeral*
interjección		

Además, en el sistema *GA*LENA se han establecido otras dos categorías, *marcas de puntuación* y *categorías periféricas*, donde se incluyen acrónimos, abreviaciones, signos, formulas, palabras extranjeras y otros elementos que no pueden ser incluidos en otros grupos.

Cada categoría puede a su vez dividirse en varias *sub-categorías* o *tipos* y éstos a su vez en *sub-sub-categorías* o *subtipos*. Un ejemplo es la subclasificación de la categoría de los sustantivos en *nombres propios* y *nombres comunes*.

2.3.1 Los paradigmas.

Por desgracia, la división en categorías ofrecida hasta el momento no es suficiente para poder identificar una palabra que actúe de modelo para el resto de las palabras pertenecientes a su misma categoría. Como ya se ha dicho, sólo para la formación del género existen, en Castellano, veinte formas diferentes por lo que resulta imposible, por ejemplo, encontrar un sustantivo que sirva de modelo en la formación del género de todos los demás sustantivos.

categoría	*tipo*
adjetivo	*sin tipo*
artículo	*sin tipo*
adverbio	exclamativo, modificador, nuclear, nuclear y modificador, interrogativo, relativo
periférico	palabra extranjera, fórmula, símbolo, abreviatura, sigla, otros
conjunción	coordinada, subordinada
sustantivo	común, propio
demostrativo	*sin tipo*
indefinido	*sin tipo*
interjección	*sin tipo*
interrogativo	*sin tipo*
numeral	cardinal, ordinal, partitivo, múltiplo
pronombre personal	tónico, proclítico átono, enclítico átono
preposición	*sin tipo*
posesivo	*sin tipo*
relativo	*sin tipo*
verbo	*sin tipo*
marca de puntuación	punto, coma,punto y coma, dos puntos, comillas, interrogación, exclamación, paréntesis, puntos suspensivos

Figura 2.1: Ejemplo de categorización para el Castellano.

Para solucionar este problema, cada categoría se divide en *grupos*[5] de tal manera que cada uno de ellos sí se rige mediante un paradigma cuyo comportamiento obedece

[5]Para más detalles de la implementación de los grupos ver [30].

a un conjunto sencillo de reglas. Sirva como ejemplo el grupo de los sustantivos que forman el masculino singular añadiendo -o a la raíz, el masculino plural añadiendo -os, -a para el femenino singular y -as para el femenino plural; como ocurre con chico, chicos, chica, chicas. El caso extremo lo constituyen los verbos como ir y ser a cuyos grupos sólo pertenecen ellos mismos.

De esta manera la tarea de seleccionar una palabra que actúe como representante del grupo, dado que todas las palabras se rigen por el mismo paradigma, es irrelevante. La situación es similar a la planteada a la hora de seleccionar un representante en una clase de equivalencia clásica. En la figura 2.3.1 vemos la clasificación completa aplicada al caso del Castellano.

2.3.2 Las etiquetas

Otra cuestión importante es: ¿qué información debe ofrecer el analizador morfológico? Evidentemente no es suficiente con conocer únicamente la *categoría sintáctica* de cada palabra.

La respuesta a la pregunta diferirá en función del objetivo que nos planteemos a la hora de diseñar el analizador, pero en cualquier caso tendremos que definir el *juego de etiquetas*[6] que empleará. Una etiqueta no es más que una n-tupla donde se almacena toda la información relevante al análisis de una palabra. Normalmente la definición de la etiqueta varía según la categoría sintáctica a la que se aplique.

A cada una de estas componentes de la etiqueta la denominaremos *atributo* a fin de emplear los mismos términos que se usarán al hablar del formalismo descriptivo del *analizador sintáctico*.

Cuando podamos analizar una palabra de varias maneras, tendremos varias etiquetas asociadas. Véase como ejemplo la etiquetación de la palabra "sobre":

verbo	presente de subjuntivo, tercera persona singular, lema:sobrar
verbo	presente de subjuntivo, primera persona singular, lema:sobrar
sustantivo	común, masculino singular, lema:sobre
preposición	lema:sobre

Cuando, además de obtener todas las etiquetas posibles para cada palabra, el sistema es capaz de determinar cual es la correcta en un contexto concreto y rechazar las demás, usaremos el término *etiquetador*.

En el caso del sistema *GA*LENA, nos interesa recuperar todo la información que más tarde será utilizada por el analizador sintáctico. Para ello se definió un juego de etiquetas inspirado en la propuesta de EAGLES[7] [23, 24, 52]. El juego de etiquetas para el caso del Gallego se puede ver en la figura 2.2.

En la figura 2.3 podemos observar otro ejemplo de juego de etiquetas, en este caso para el Castellano. Nótese su parecido con el juego de etiquetas descrito para el Gallego, debido a su origen latino común.

[6] Algunos autores usan este término para referirse al conjunto de todos los valores posibles que pueden tomar las etiquetas.

[7] `http://www.ilc.pi.cnr.it/EAGLES/home.html`

Campo	Valores	
Palabra	*La forma presente en el texto de entrada.*	
Lema	*La forma canónica de la palabra.*	
Categoría	Adjetivo	*Sin tipo.*
	Adverbio	Exclamativo, modificador, nuclear, relativo, interrogativo *y* nuclear y modificador.
	Artículo	*Sin tipo.*
	Conjunción	Coordinada *y* subordinada.
	Demostrativo	*Sin tipo.*
	Indefinido	*Sin tipo.*
	Interjección	*Sin tipo.*
	Interrogativo	*Sin tipo.*
	Numeral	Cardinal, ordinal, partitivo *y* múltiple.
	Categoría periférica	Palabra extranjera, formula, símbolo, abreviación, acrónimo *y* otros.
	Preposición	*Sin tipo.*
	Pronombre Personal	Tónico, proclítico átono *y* enclítico átono.
	Posesivo	*Sin tipo.*
	Marca de Puntuación	Punto, coma, dos puntos, punto y coma, guión, comillas, signo de interrogación abierto/cerrado, signo de exclamación abierto/cerrado, paréntesis abierto/cerrado *y* puntos suspensivos.
	Relativo	*Sin tipo.*
	Sustantivo	Común *y* propio.
	Verbo	*Sin tipo.*
Subtipo	Determinado, no determinado *y* ambos.	
Género	Masculino, femenino, ambos, neutro *y* no aplicable.	
Número	Singular, plural, ambos *y* no aplicable.	
Grado	Comparativo *y* no aplicable.	
Persona	Primera, segunda, tercera, primera y tercera *y* no aplicable.	
Caso	Nominativo, acusativo, dativo, acusativo y dativo, caso preposicional, *y* nominativo y caso preposicional.	
Tiempo verbal	Presente, pretérito, co-pretérito, futuro, post-pretérito *y* no aplicable.	
Modo	Indicativo, subjuntivo, imperativo, infinitivo, gerundio *y* participio.	

Figura 2.2: Juego de etiquetas para el Gallego

2.3.3 Las reglas

Como ya se ha dicho, por cada paradigma existen una serie de reglas que lo definen. Por ejemplo para el grupo `G1` de los sustantivos:

$$\text{masculino} = \text{raíz} + \texttt{-o}$$
$$\text{femenino} = \text{raíz} + \texttt{-a}$$
$$\text{singular} = \text{raíz} + \text{género}$$
$$\text{plural} = \text{raíz} + \text{género} + \texttt{-s}$$

Una vez identificados y definidos todos los paradigmas, sólo resta indicar que palabras pertenecen a que paradigmas, indicando su raíz. Si decimos que `chico` pertenece al grupo G1 de los sustantivos, y su raíz es `chic`, podemos aplicar las reglas pertinentes para deducir todas su posibles formas derivadas: `chico`, `chica`, `chicos`, `chicas`.

2.4 Formalismo operacional.

A la hora de escoger un formalismo operacional que implemente el formalismo descriptivo presentado, deseamos que sea:

- *Compacto*, de otra manera, dado el gran número de fenómenos flexivos y la riqueza de vocabulario de los *lenguajes* que nos ocupan (Castellano y Gallego), el espacio necesario para almacenar el analizador sería difícilmente abordable por los computadores actuales.
- *Rápido*, por la misma razón y dada la enorme ambigüedad presente en todo *lenguaje natural*, el tiempo empleado en realizar todas las computaciones supone otro de los límites a tener en cuenta.

categoría	*atributos relevantes*
adjetivo	lema
adverbio	tipo, lema
artículo	lema, subtipo, género, número
conjunción	tipo, lema
demostrativo	lema
indefinido	lema
interjección	lema
interrogativo	lema
numeral	tipo, lema
periférico	tipo, lema
preposición	lema
pronombre personal	tipo, lema, género, número, persona
posesivo	lema
marca de puntuación	tipo
relativo	lema
sustantivo	tipo, lema, género, número
verbo	lema, número, persona, tiempo verbal, modo

Figura 2.3: Ejemplo de juego de etiquetas para el Castellano.

Recientemente se ha mostrado un renovado interés en los modelos basados en autómatas finitos [44, 62, 63, 65] para el diseño de etiquetadores y analizadores morfológicos, precisamente debido a la eficacia y lo compacto de su representación.

Puesto que una palabra puede tener asociadas más de una etiqueta y deseamos reconocer todas, la elección es un modelo basado en *autómatas finitos no deterministas*. Los primeros obstáculos a salvar encontrados son:

i. El modelo de autómatas finitos posee un poder descriptivo menor que el enfoque orientado a gramáticas [62] en el que se basa el formalismo descriptivo usado.

ii. La *compilación* y posterior *recompilación* del autómata a partir de las reglas gramaticales indicadas mediante el formalismo descriptivo, tal y como ha sido

propuesto en [44, 63], no asegura la compartición de los caminos lingüísticamente relacionados y por tanto no asegura la obtención de una buena compactación.

iii. Limitar el *no determinismo* del autómata resultante, de manera que la velocidad de análisis no se vea penalizada.

iv. Aislar en la medida de lo posible los conceptos de la tecnología empleada del trabajo específico del lingüista. El último punto se trata en el apéndice C.

La solución adoptada parte de la división del autómata en *mini autómatas* especializados, cada uno de los cuales reconoce un fenómeno morfológico concreto. De esta manera se evita la multiplicación, dentro del autómata resultante, de caminos lingüísticamente equivalentes. A *grosso modo*, por cada *grupo* tendremos un *mini autómata*, equivalente a las reglas que definen su paradigma, para reconocer las desinencias aplicadas a cada palabra, y un autómata mayor para reconocer las raíces de dichas palabras[8].

La acción conjunta de estos autómatas es la que da lugar al análisis de una palabra concreta. La figura 2.4 muestra un ejemplo de mini autómata para el grupo G1 de los sustantivos, mencionado en ejemplos anteriores.

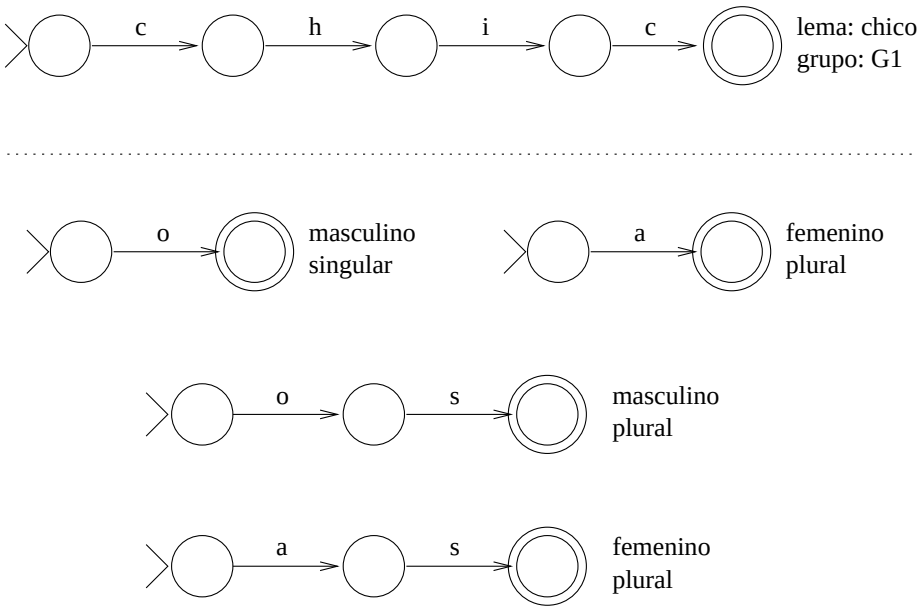


Figura 2.4: Mini autómatas para chico/a/os/as.

La compartición de caminos dentro del autómata conlleva, además, la reducción del no determinismo del mismo, puesto que no se duplican caminos que representan el mismo fenómeno lingüístico.

[8]Actualmente el tamaño de este autómata da lugar a tiempos de compilación excesivos, por lo que se está realizando una nueva implementación empleando *árboles de búsqueda*. La ventaja de este último tipo de estrategia, es que permite una compilación incremental, a la vez que reduce el tamaño del autómata de forma ostensible.

Por último, para solucionar el primer problema, se crea una herramienta software capaz de crear los *mini autómatas* necesarios a partir de las reglas y la lista de raíces de las palabras de los paradigmas. Nos referiremos a este proceso como la *compilación del autómata.*

2.4.1 Desambiguación.

El formalismo operacional descrito hasta el momento nos permite obtener un analizador capaz de asignar todas las etiquetas válidas para una palabra concreta. Sin embargo no es capaz de discernir, dentro de un contexto determinado, cual de esas etiquetas es la correcta. Nos referiremos a este proceso de selección de la etiqueta correcta como *desambiguación.*

Para solucionar el problema de la desambiguación han sido propuestas dos aproximaciones fundamentales:

- *Desambiguación por reglas.* Aplicamos un conjunto de reglas que determinan que etiquetas son correctas y cuales no en función de las palabras precedentes y subsiguientes.

  Esta tarea también puede ser llevada a cabo por, o en colaboración con, un analizador sintáctico. El analizador considerara todas las etiquetas y desechará aquellas que no den lugar a un análisis sintácticamente correcto.

- *Desambiguación estadística.* Un modelo estadístico basado en un *proceso de Markov* selecciona entre todas las etiquetas la que tiene la mayor probabilidad de ser la correcta. La probabilidad depende de las n palabras precedentes.

  Debido a su complejidad, el modelo estadístico ha de ser estimado. A este proceso se le conoce como *entrenamiento del desambiguador.*

En cualquier caso, ambos métodos pueden ser usados conjuntamente.

Capítulo 3

Introducción al análisis sintáctico.

A la hora de construir un sistema capaz de analizar sintácticamente frases pertenecientes a un lenguaje natural, hemos de cumplir dos objetivos principales:

1. *Describir el lenguaje* que queremos analizar. Para lograr este objetivo deberemos, antes de nada, definir un formalismo que nos permita realizar la descripción. La elección en el presente trabajo ha sido una gramática de tipo declarativo. El porque de tal elección frente a la posibilidad de las gramáticas de tipo procedural[1] es una cuestión que se considera ampliamente superada y no se tratará aquí, simplemente mencionar, a modo de apunte, que se busca tanto la independencia de la implementación final del analizador, como la mayor claridad y representatividad del formalismo.

2. *Generar una implementación*, es decir un sistema que sea capaz de analizar una frase en función de la gramática descrita en el paso anterior. En este caso hemos de definir un *formalismo operativo* que nos permita realizar dicha implementación. Como primera característica hemos de buscar que el paso de la información contenida en el formalismo descriptivo al formalismo operacional sea automática, de forma que se puedan aislar convenientemente la primera parte, propia de los expertos en lingüística, de la segunda, propia de los expertos en ingeniería informática.

El *formalismo operacional* será discutido en el siguiente capítulo, centrándose el presente en el *formalismo descriptivo.*

3.1 Los formalismos descriptivos

Definiremos una *gramática declarativa* como un conjunto de sentencias , expresadas de un modo lógico. Normalmente se usa el término *formalismo gramatical* para referirse al lenguaje lógico usado para expresar una gramática declarativa. Cada formalismo impone restricciones al tipo de sentencias en que se pueden expresar las gramáticas, y por tanto el poder descriptivo de las mismas. También se determinará el conjunto de formalismos operacionales que se pueden emplear en su análisis.

[1] Principalmente las *Redes de Transición Aumentadas* (*ATN*) [56].

Como ya se ha mencionado, una gramática declarativa se forma construyendo un conjunto de sentencias. Los formalismos actuales basan la creación de éstas en la descomposición de las *categorías gramaticales*[2] en componentes conocidos como *rasgos*[3] o *atributos*, y la aplicación de restricciones lógicas sobre ellos. De ahí que la mayoría de estos formalismos reciban el nombre de *Formalismos Gramaticales Basados en Restricciones*, *Basados en Unificación* [68, 71, 76], *Basados en Categorías Complejas* [68], o *Basados en Rasgos* [28, 98].

Dicho de otra manera, desde el punto de vista lingüístico, existen diversas clases de información asociada a una frase. Una de ellas es la relativa a la precedencia y dominancia entre categorías lingüísticas. Otra viene dada por medio de funciones gramaticales como *sujeto* u *objeto*, importantes para la construcción de una representación del significado de la frase. Otros tipos de información son la concordancia en género y número entre partes de la oración, la asignación de casos en lenguajes con sistemas de ese tipo (como el Ruso) o la asociación de predicados a las palabras de una oración.

A la hora de clasificar los distintos formalismos, podemos seguir varios criterios, a continuación se desarrollan brevemente algunos de ellos.

3.1.1 Jerarquía de Chomsky

En la figura 3.1 vemos las diferentes clases de lenguajes según la jerarquía de Chomsky, junto con el formalismo operacional asociado y el número de pilas que usa. Como se puede ver, los formalismos actuales están orientados a describir lenguajes situados entre la clase de lenguajes independientes del contexto, LICs, y la clase de lenguajes sensibles al contexto.

Esta clase de lenguajes se subclasifica a su vez, en orden de complejidad creciente, en: *lenguajes suavemente sensibles al contexto* [29, 37, 54, 60, 77], *lenguajes indexados* [34] y el resto de lenguajes que no recibe ningún nombre específico.

3.1.2 Basadas en reglas vs. basadas en principios

Se distinguen dos grandes clases de formalismos diferenciados por la forma en que las categorías gramaticales se combinan para formar estructuras sintagmáticas válidas:

- *Gramáticas basadas en reglas:* cada regla indica de que manera una estructura sintagmática compleja se divide en otras más sencillas. Algunos ejemplos son las *gramáticas léxico-funcionales* (GLFS) [9], y las *gramáticas de cláusulas definidas* (GCDS) [16].

- *Gramáticas basadas en principios:* para cada gramática se describe un conjunto de principios que deben cumplir todas las estructuras sintagmáticas de la gramática. Unas pocas reglas indican como formar los objetos sobre los que se aplican los principios. El primer ejemplo de este tipo de aproximación es la *teoría de la rección y el ligamiento*[4] [11, 12, 13].

[2] A, *grosso modo*, partes del discurso (part–of–speech *en inglés*).
[3] *features* en la literatura anglosajona.
[4] Del inglés, *Government and Binding*

Tipo 0, o recursivamente enumerables	Lenguajes dependientes del contexto	Lenguajes independientes del contexto	Lenguajes regulares
Máquinas de Turing	Autómatas linealmente acotados	Autómatas a pila	Autómatas finitos
Dos pilas independientes	n pilas dependientes	Una pila	Sin pila
	Gramáticas de cláusulas definidas Gramáticas de Estructura Sintagmática guiada por el núcleo Gramáticas de núcleo Gramáticas de adjunción de árboles	Gramáticas independientes del contexto	Expresiones regulares

Figura 3.1: Jerarquía de Chomsky. Ejemplos de gramáticas asociadas a cada clase de lenguaje.

Podemos imaginar esta clasificación como un espacio unidimensional en uno de cuyos extremos se encuentran la teoría puramente lingüística, mientras que en el otro encontramos las teorías matemático-computacionales. Las gramáticas basadas en principios tienden hacia el primer extremo, mientras que las basadas en reglas tienden al extremo contrario.

Por desgracia, cuanto más se aproxime una formalismo a la teoría lingüística, más fácil será describir un lenguaje natural, pero más ineficaces, computacionalmente, serán los formalismos operacionales correspondientes. Por esta razón, la solución adoptada por algunos autores pasan por crear una herramienta capaz de transformar un gramática descrita en un formalismo basado en principios a una gramática equivalente usando un formalismo basado en reglas más eficiente computacionalmente. Ver por ejemplo como en [27] se detalla el uso de técnicas de deducción parcial para transformar una *Gramática de Estructura Sintagmática guiada por el Núcleo* (*HPSG*) [58] en una Gramática de Cláusulas Definidas.

3.1.3 Gramáticas basadas en restricciones

En [35], Jaffar y Lassez introducen la idea de programación lógica con restricciones. Esta idea es generalizada por Höhfeld y Smolka en [33] en un esquema de programación lógica con restricciones que permite tratar varios formalismos basados en restricciones dentro de un marco unificado.

Los formalismos basados en restricciones se pueden clasificar en base a dos preguntas: ¿qué tipo de restricciones se usan? y ¿cómo se procesan?

En un primer grupo a considerar, las restricciones se definen como condiciones de igualdad entre términos lógicos de primer orden. La combinación de restricciones se realiza unificando los términos. En general estos sistemas parten del trabajo realizado por Robinson [64] sobre la unificación de términos en la década de los sesenta. Los ejemplos más representativos de este tipo de formalismos son los derivados de PROLOG,

y entre ellos las GCDs que se discutirán en detalle en subsiguientes apartados.

El otro gran grupo lo conforman las *gramáticas basadas en rasgos.* En ellos los términos lógicos como estructuras de datos son sustituidos por estructuras de rasgos. Los primeros desarrollos en esta dirección son las *gramáticas de unificación funcional* [39, 40, 41], *gramáticas léxico-funcionales* [9], el formalismo empleado en el sistema PATR-II [70], además de los trabajos realizados sobre los fundamentos formales [7, 36, 38, 66].

Las estructuras de rasgos pueden ser formalizadas como modelos de restricciones lógicas basados en la igualdad de caminos[69, 71] o, de manera más intuitiva, como matrices de pares atributo-valor dónde un valor puede ser a su vez otra matriz atributo-valor, o como funciones complejas sobre conjuntos de rasgos. Dado un conjunto de rasgos y sus etiquetas correspondientes, y sus valores, una estructura de rasgos es una función parcial de dichos rasgos a sus valores. Estas funciones se suelen representar como matrices, como se puede ver en la figura 3.2. Normalmente se usan etiquetas del tipo $\boxed{1}$ para indicar que una misma subestructura es compartida.

$$\begin{bmatrix} \text{num} & = & \text{sg} \\ \text{pers} & = & \text{ter} \end{bmatrix} \quad \begin{bmatrix} \text{cat} & = & \textit{sn} \\ \text{concord} & = & \begin{bmatrix} \text{num} & = & \text{sg} \\ \text{pers} & = & \text{ter} \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} \text{cat} & = & o \\ \text{sujeto} & = & \begin{bmatrix} \text{cat} & = & \text{sn} \\ \text{concord} & = & \boxed{1}\,[\text{ter, sing}] \end{bmatrix} \\ \text{núcleo} & = & \begin{bmatrix} \text{cat} & = & \text{sv} \\ \text{concord} & = & \boxed{1} \end{bmatrix} \end{bmatrix}$$

Figura 3.2: Ejemplos de estructuras de rasgos.

Como se puede ver la principal diferencia entre ambos grupos consiste en que en los basados en rasgos las subestructuras se identifican explícitamente por medio de una etiqueta, en lugar de por su posición dentro del término lógico.

A partir de estas definiciones básicas se han realizado diversas extensiones de los formalismos basados en restricciones, con el objeto de lograr una mejor adecuación al tratamiento del lenguaje natural. Algunos ejemplos son los siguientes:

- *Dominios finitos:* Fueron introducidos en el lenguaje CHIP [32]. Una variable de dominio finito puede tener un conjunto finito de valores. El resultado de la unificación de dos variables de dominio finito es la intersección de sus posibles valores. Si resulta en un conjunto vacío, entonces la unificación falla.

- *Descripción de conjuntos:* Las descripciones de conjuntos y las restricciones sobre conjuntos se usan frecuentemente en descripciones lingüísticas. Recientemente han sido formalizados, [48].

- *Restricciones de precedencia lineal:* Su uso más obvio es la modelización de los

fenómenos relativos al orden de las palabras. Otro uso es la descripción de las relaciones semánticas de precedencia temporal. Los fundamentos lógicos se describen en [49].

- *Restricciones de guardia:* Son restricciones cuya ejecución se pospone hasta que la precondición de su aplicabilidad se satisface. Se pueden usar para asignar objetivos a las variables. Han sido usadas en la implementación de principios HPSG y para integrar restricciones morfológicas en gramáticas HPSG [50, 75].

- *Lógica booleana:* Introducida en PROLOG III [15].

3.2 Gramáticas de Cláusulas Definidas

De lo expuesto hasta el momento podemos concluir que los formalismos gramaticales desarrollados a partir de modelos lógicos basados en restricciones son adecuados para el tratamiento del lenguaje natural. A favor de ellos podemos argumentar que su forma de manipular y representar el conocimiento es afín con las formas habituales de las teorías lingüísticas y que por lo tanto el coste de desarrollo en un sistema de procesamiento del lenguaje natural, *PLN*, frecuentemente es inferior al que conllevan otros formalismos.

A la hora de decantarse por alguno de los formalismos gramaticales existentes nos hemos decidido por las GCDs, entre otras, por las siguientes razones:

- Debido a su disponibilidad y simplicidad existe una cantidad enorme de ejemplos disponibles.

- Por su poder descriptivo, que puede ser comparado con el de las máquinas de Turing en el sentido de que la misma gramática sirve tanto para describir el lenguaje, como para reconocerlo o generarlo (interpretándola como un programa lógico). De hecho, algunos autores las consideran como un mero "*azúcar sintáctico*" para escribir programas lógicos en el estilo de PROLOG [56, 72].

- Es posible traducir gramáticas escritas en formalismos como HPSG o GAAs[5] a una GCD equivalente.

- Se pueden ver como una extensión natural de las GICs de forma que se puede aprovechar todo el conocimiento existente acerca del reconocimiento de las mismas.

A continuación se describen de manera más detallada las GCDs [56], que ha sido el formalismo elegido para la realización del trabajo descrito en la presente memoria.

3.2.1 Aspectos formales relevantes

Señalaremos dos aspectos principales:

[5] Gramática de Adjunción de Arboles.

- Uso de reglas independientes del contexto para definir las relaciones de precedencia y dominancia entre las categorías gramaticales. De hecho, como ya se ha mencionado anteriormente, se puede pensar en una GCD como una extensión de una GIC, considerando cada término en un dominio infinito.
- Codificación de información lingüística mediante categorías complejas[6]. Las estructuras de rasgos se representan mediante términos lógicos.
- Uso de la operación de unificación para resolver las restricciones.

3.2.2 Definiciones previas

Partimos de la existencia de los siguientes conjuntos:

$\mathcal{X}$ es un conjunto numerable y ordenado de variables.

$\mathcal{F}$ es un conjunto finito de símbolos de función.

Una *constante* se define como:

- un número: 0, 1, 9999, 69.8, ...
- un carácter: `a`, `b`, `z` , ...
- un nombre: `void`, `singular`, ...
- una cadena de caracteres: '`meu`', '`oiches`', ...; o
- la lista vacía: `[]`

Por convención, los nombres de variable comienzan por una letra mayúscula, mientras que los nombres de constante han de comenzar con una letra en minúsculas.

Tanto las variables, como las constantes se consideran *términos simples*, frente a los *términos compuestos*. Los términos compuestos se forman a partir de un símbolo de función o *funtor*, también llamado *funtor principal* del término, y uno o más términos, simples o compuestos, llamados *argumentos*. Nos referiremos al número de argumentos como la *aridad* del término compuesto.

Un funtor se caracteriza por su *firma*, que consiste en la unión de su nombre y aridad. Los argumentos se colocan a continuación del nombre, separados por comas y encerrados entre paréntesis. Por ejemplo el funtor de nombre `punto3D` y aridad 3, con las variables `X`, `Y` y `Z` como argumentos se escribe: `punto3D(X,Y,Z)`, y su firma es: `punto3D/3`.

Los términos compuestos también pueden representarse gráficamente por medio de un árbol cuya raíz es el nombre del funtor y cuyos hijos son sus argumentos. Ver a modo de ejemplo la figura 3.3.

Nos referiremos a los términos simples o compuestos como *términos lógicos*. Los términos compuestos así descritos tienen una funcionalidad puramente constructiva y no representan ninguna relación lógica, como el ejemplo `punto3D(X,Y,Z)`, y también reciben el nombre de *funciones*.

[6] Categorías + rasgos.

s(fn(Santiago),fv(verbo(rueda),fn(det(una),sust(película))))

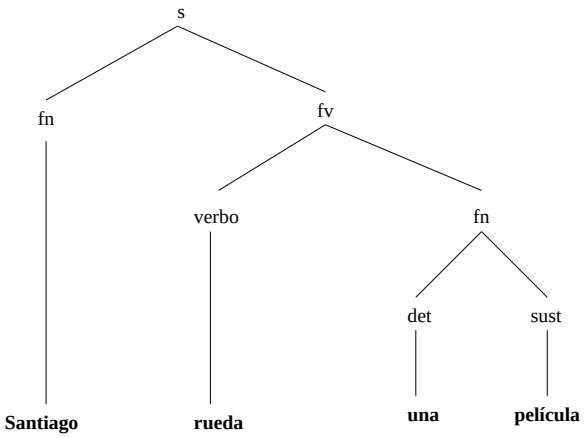


Figura 3.3: Representación gráfica de un término compuesto.

Existe, además, un tipo de término compuesto que recibe un tratamiento especial: las listas. Efectivamente podemos pensar en una lista como un funtor de aridad dos cuyos argumentos son el primer elemento y el resto de la lista. Así, la lista compuesta por los números 1, 2 y 3 se escribiría:

$$\cdot(1, \cdot(2, \cdot(3, [])))$$

Pero como ya se ha dicho, debido a su frecuencia de uso, las listas reciben un tratamiento especial pudiéndose escribir de manera más cómoda:

$$[1, 2, 3]$$

$$[1|[2, 3]]$$

Los símbolos '[' y ']' marcan, respectivamente, el principio y final de la lista y '|' separa el primer elemento del resto.

Otro concepto importante es el de *substitución*. Una substitución Θ es una lista de pares variable/término lógico. El resultado de aplicar una substitución Θ a un término lógico T es $T\Theta$, y se obtiene tomando para cada par (X_i, t_i) de Θ y substituyendo todas las ocurrencias de X_i en T por t_i. Por ejemplo:

$$T = f(X, 3, g(Y)), \quad \Theta = \{X/1, Y/h(Z)\}, \quad T\Theta = f(1, 3, g(h(Z)))$$

Es importante tener en cuenta que cada paso de la aplicación de una substitución es independiente de los demás. Por lo tanto el resultado de aplicar $\{X/Y, Y/2\}$ a $f(X, Y)$ es $f(Y, 2)$ y no $f(2, 2)$.

Por otra parte, diremos que Θ es un *unificador* de T_1 y T_2 sii $T_1\Theta = T_2\Theta$. En tal caso, también diremos que T_1 y T_2 son *unificables*. Por ejemplo, los términos

$$T_1 = p(A, 3, f(Y)) \quad T_2 = p(B, C, f(g(Z)))$$

son unificables mediante cualquiera de las siguientes substituciones:

$$\Theta_1 = \{A/5, B/5, C/3, Y/g(Z)\}$$
$$\Theta_2 = \{B/A, C/3, Y/g(0), Z/0\}$$
$$\Theta_3 = \{A/B, C/3, Y/g(Z)\}$$

Cuando dos términos son unificables, existe un unificador particular Θ llamado el *unificador más general*, que denotaremos $\mathrm{mgu}(T_1, T_2)$. La propiedad que ha de cumplir este unificador es que para cualquier otro unificador Θ' exista una substitución σ tal que $\Theta' = \Theta\sigma$.

En el ejemplo anterior, el unificador más general es $\Theta_3 = \{A/B, C/3, Y/g(Z)\}$. En efecto:

$$\Theta_1 = \Theta_3\,\{B/5\}$$
$$\Theta_2 = \Theta_3\,\{B/A, Z/0\}$$

El concepto de unificación también se conoce como *correspondencia*[7]. El algoritmo por excelencia para calcular de forma automática el unificador más general de dos términos es el *algoritmo de unificación de Robinson*[64].

El siguiente concepto es el de *subsumción*. Diremos que un término A *subsume* a otro término B, o que B *es subsumido* por A, si existe una substitución σ tal que $B = A\sigma$. Notaremos esta relación de la siguiente manera: $A \preceq B$. Algunos ejemplos de subsumciones:

$$f(A,B) \preceq f(\texttt{toto}, C), \qquad f(A,B)\,\{A/\texttt{toto}, B/C\} = f(\texttt{toto}, C)$$
$$X \preceq \texttt{cualquiera}, \qquad X\,\{X/\texttt{cualquiera}\} = \texttt{cualquiera}$$
$$p(3, f(Z)) \preceq p(3, f(2)), \qquad f(3, f(Z))\,\{Z/2\} = p(3, f(2))$$

3.2.3 Definición

Podemos pensar en las GCDs como un "azúcar sintáctico" para los programas lógicos [56] o como una extensión de las GICs. Debido a la importancia que tendrá a la hora de describir el formalismo operacional del sistema *GA*LENA, definiremos las GCDs como una extensión de las GICs.

Una GCD es una 5-upla $\mathcal{D} = (\mathcal{X}, \mathcal{F}, \mathcal{P}, \Gamma, \overset{\circ}{p})$ donde:

- $\mathcal{X}$ es un conjunto numerable y ordenado de variables.
- $\mathcal{F}$ es un conjunto finito de símbolos de función.
- $\mathcal{P}$ es un conjunto finito de símbolos de predicado.
- Γ es un conjunto finito de $n+1$ *cláusulas* γ_k de la forma $A_{k,0} : -\eta_k$ donde la *cabeza* $A_{k,0}$ es un *átomo*, y el *cuerpo* η_k es un conjunto finito de átomos $A_{k,1}, \ldots, A_{k,n_k}$. Puede haber más de una cláusula con la misma cabeza.
- $\overset{\circ}{p}$ es el predicado inicial.

[7] En la literatura anglosajona, *matching*.

$\mathcal{P}$ equivale al alfabeto de símbolos no terminales de una GIC. El concepto de no terminal se generaliza asociando a cada categoría sintáctica una aridad n, $n \geq 0$, y una n-tupla de términos lógicos. Los no terminales aumentados de esta forma, se usarán para construir los *predicados* de la gramática.

Como se puede ver, la construcción de los predicados es equivalente a la de las funciones. No obstante, los predicados sí que representan una relación lógica entre sus argumentos, por ejemplo: `padre("Juan","Pepe")`, puede representar que `Pepe` es el padre de `Juan`.

Remplazando cada no terminal de la forma descrita, las reglas de producción de una GIC dan lugar a las cláusulas Γ de la GCD. El concepto de derivación se generaliza de la siguiente manera: Sean α una secuencia de predicados y terminales, $r : p(t_1, \ldots, t_n) \rightarrow \beta$ una cláusula perteneciente a la gramática y $r' : p(t'_1, \ldots, t'_n) \rightarrow \beta'$ una nueva cláusula que se obtiene renombrando todas las variables de r de forma que no coincidan sus nombres con las variables que aparecen en α.

Diremos que α *deriva directamente* α' *a través de* r sii:

1. $\alpha = \alpha_1\ p(s_1, \ldots, s_n)\ \alpha_2$.
2. Existe $\Theta = \mathrm{mgu}(p(s_1, \ldots, s_n), p(t'_1, \ldots, t'_n))$.
3. $\alpha = (\alpha_1\ \beta'\ \alpha_2)\theta$.

Una derivación mediante un conjunto de cláusulas $R = \{r_1, r_2, \ldots\}$ es una *secuencia* $\alpha_0 \ldots \alpha_n$, $n \geq 0$, donde, para todo $i = 0, \ldots, n-1$, α_i deriva inmediatamente α_{i+1} a través de r_i.

Cada derivación produce una substitución $\theta = \theta_1 \ldots \theta_n$, donde los θ_i son los m.g.u. implicados en las derivaciones inmediatas. A su vez, el *árbol de derivación* se construye aplicando θ a las cláusulas implicadas en la derivación.

En adelante, siguiendo el estandard de PROLOG, usaremos [`texto`] para representar una cadena de símbolos terminales.

Ejemplo 3.1:

A continuación se ilustra el concepto de derivación mediante un sencillo ejemplo sobre al siguiente gramática:

r_0 : `frase(s(NP,VP))` $\rightarrow$ `np(NP,N)`, `vp(VP,N)`.
r_1 : `np(np(barbara),singular)` $\rightarrow$ [Barbara].
r_2 : `vp(vp(deslumbra),singular)` $\rightarrow$ [deslumbra].

Podemos hacer la siguiente derivación usando $R = \{r_0, r_1, r_2\}$:

α_0 : `frase(X)`.
r'_1 : `frase(s(NP,VP))` $\rightarrow$ `np(NP,N)`, `vp(VP,N)`.
θ_1 : $\{X/$`s(NP,VP)`$\}$

α_1 : `np(NP,N)`, `vp(VP,N)`.
r'_2 : `vp(vp(deslumbra),singular)` $\rightarrow$ [deslumbra].
θ_2 : $\{VP/$`vp(deslumbra)`$, N/$`singular`$\}$

α_2 : `np(NP,singular)`, [deslumbra].
r'_3 : `np(np(barbara),singular)` $\rightarrow$ [Barbara].
θ_3 : $\{NP/$`np(barbara)`$\}$

α_3 : [Barbara], [deslumbra].

Como resultado de aplicar R obtenemos la substitución:

$$\theta = \theta_1\theta_2\theta_3 = \{\ X/\texttt{frase(np(barbara),vp(deslumbra))}, VP/\texttt{vp(deslumbra)}, \\ N/\texttt{singular}, NP/\texttt{np(barbara)}\}\ .$$

Después de aplicar cada derivación inmediata se obtienen los siguientes árboles de derivación:

T_0

`frase(X)`

T_1

`frase(s(NP,VP))`
|
`np(NP,N),vp(VP,N)`

T_2

`frase(s(np(NP),vp(deslumbra)))`
|
`np(np(NP),singular)`
`vp(vp(deslumbra),singular)`
|
`np(np(NP),singular)`, [deslumbra]

T_3

`frase(s(np(barbara),vp(deslumbra)))`
|
`np(np(barbara),singular)`
`vp(vp(deslumbra),singular)`
|
`np(np(barbara),singular)`, [deslumbra]
|
[Barbara], [deslumbra]

□

3.2.4 Esqueleto independiente del contexto

A continuación pasamos a describir la noción de esqueleto independiente del contexto. Este concepto será usado con profusión a la hora de desarrollar el formalismo descriptivo empleado.

Tal y como se han presentado las GCDs, como una extensión de las GICs, es evidente que para cada GCD existe una GIC subyacente, de la que diremos que es su *esqueleto independiente del contexto.*

Esta se obtiene, típicamente, eliminando los argumentos y quedándonos sólo con los símbolos de predicado. También es común aplicar algún refinamiento adicional, como añadir a cada símbolo un número que indique su aridad [83].

Ejemplo 3.2:

Si aplicamos lo expuesto a la gramática del ejemplo 3.1 obtenemos:

Gramática de cláusulas definidas	Esqueleto indep. del contexto
`frase(s(NP,VP))` → `np(NP,N)`, `vp(VP,N)`.	`frase` → `np vp`
`np(np(barbara),singular)` → [Barbara].	`np` → 'Barbara'
`vp(vp(deslumbra),singular)` → [deslumbra].	`vp` → 'deslumbra'

□

3.2.5 Interpretación

Ejemplo 3.3:

Antes de nada, veamos como podemos interpretar alguna de las cláusulas usadas en el ejemplo 3.1:

```
frase_nominal(Numero, Genero) ⟶ determinante(Numero, Genero),
                                sustantivo(Numero, Genero).
```

la interpretamos como:

> *Una frase nominal se puede reescribir como un determinante y un sustantivo, siempre y cuando coincidan los valores asignados a las variables* ***Numero****, y* ***Genero****.*

O la cláusula:

```
sustantivo(singular, femenino) ⟶ ''copa''.
```

> *Un sustantivo cuyos atributos son* ***singular*** *y* ***femenino*** *se puede expandir en la cadena de caracteres copa.*

Esta notación es equivalente a:

```
sustantivo(singular, femenino) ⟶ [copa].
```

□

De los ejemplos expuestos se sigue más claramente como una GCD puede ser considerada como una GIC dónde cada símbolo no terminal se ha aumentado añadiéndole una serie de atributos o argumentos.

El primer efecto notable que observamos al aumentar una GIC para obtener una GCD, es que el conjunto de interpretaciones de los símbolos de la gramática pasa de un conjunto finito a otro potencialmente infinito. El segundo efecto es aumentar el poder expresivo mediante el mecanismo de restricciones.

Como ya se ha dicho el mecanismo de restricciones descansa sobre el proceso de unificación. Este proceso nos asegura que no se expandirá ninguna cláusula que no cumpla las condiciones que impone el mecanismo de unificación sobre la ella.

Mientras que en una GIC el análisis de una frase equivale a la construcción de una derivación expandiendo los símbolos no terminales mediante las reglas que conforman la gramática hasta conseguir convertir el axioma en la frase de entrada, en una GCD surge el concepto de *instanciación*.

Al realizar una derivación en una GIC, para expandir un símbolo no terminal, es suficiente con encontrar una regla cuya parte izquierda coincida con el símbolo. En el caso de las DCGs, buscaremos una cláusula cuya cabeza unifique con el predicado que queremos expandir, y aplicaremos el unificador[8] al cuerpo de la cláusula, antes de expandirlo.

De esta forma, las restricciones sobre los elementos de la gramática y la forma en como se combinan, se definen en función de los argumentos de los predicados.

Ejemplo 3.4:

Una vez más, tomamos la gramática del ejemplo 3.1, para demostrar como el mecanismo de restricciones aumenta el poder descriptivo de la gramática. Partimos de la cláusula:

```
frase_nominal(Numero, Genero) ⟶ determinante(Numero, Genero),
                                 sustantivo(Numero, Genero).
```

Nos dice que una frase nominal se puede reescribir como un determinante y un sustantivo, siempre y cuando coincidan en número y género. Añadámosle tres nuevas cláusulas a la gramática que nos permitan analizar las frases: "las copas" y "la copas".

```
determinante(singular,femenino) ⟶ [la].
determinante(plural,femenino) ⟶ [las].
sustantivo(plural,femenino) ⟶ [copas].
```

A continuación describiremos de forma simplificada el proceso de análisis, con el objeto de mostrar como actúan las restricciones impuestas. En el primer caso podemos realizar la siguiente secuencia de reescrituras:

```
frase_nominal(Numero, Genero) ⇒  determinante(Numero, Genero),
                                 sustantivo(Numero, Genero)
```

Si queremos reescribir `determinante(Numero, Genero)` debemos unificarlo con la cabeza de alguna de las cláusulas:

```
determinante(Numero, Genero) sustantivo(Numero, Genero) ⇒
determinante(plural, femenino) sustantivo(plural, femenino) ⇒
''las'' sustantivo(plural, femenino)
```

Por último expandimos `sustantivo(plural, femenino)`:

```
''las'' sustantivo(plural, femenino) ⇒ ''las'' ''copas''
```

Si intentamos analizar la segunda frase:

[8] Es decir, *instanciamos* los predicados de la cláusula.

`adjetive(abs)`	$\longrightarrow$	"`red`"
`adjetive(comp)`	$\longrightarrow$	"`redder`"
`adjetive(sup)`	$\longrightarrow$	"`reddest`"
`adjetive(abs)`	$\longrightarrow$	"`sad`"
`adjetive(comp)`	$\longrightarrow$	"`sadder`"
`adjetive(sup)`	$\longrightarrow$	"`saddest`"

Tabla 3.1: Entradas léxicas para los adjetivos `red` y `sad`.

```
frase_nominal(Numero, Genero) ⇒
determinante(Numero, Genero) sustantivo(Numero, Genero) ⇒
determinante(singular, femenino) sustantivo(singular, femenino) ⇒
``la'' nombre(singular, femenino)
```

Y no podemos seguir el análisis, lo cual significa que la frase no pertenece al lenguaje descrito por la gramática. Nótese que de no haber impuesto la restricción de concordancia en número y género, la frase hubiese sido reconocida como válida.

□

3.3 Integración del analizador morfológico con el analizador sintáctico

Existen varias razones que justifican la separación entre análisis morfológico y análisis sintáctico:

- Conseguir una mayor modularidad del sistema.
- Las GCDs son extremadamente artificiosas para describir los procesos morfológicos, especialmente en el caso de lenguajes fuertemente flexivos. Estos fenómenos se simulan fácilmente en entornos operativos más simples y por lo tanto más eficaces.

  Aún en el caso de lenguajes con un escaso número de fenómenos flexivos, resulta engorroso tener que manejar una cláusula por cada una de las formas de cada palabra, tal y como se deduce de la tabla 3.1.
- Reutilizar el trabajo realizado en el desarrollo del analizador morfológico descrito en el capítulo 2.

Podríamos pensar en desarrollar una extensión de las GCDs de forma que nos permitiese factorizar las entradas léxicas convenientemente, al estilo de las gramáticas de afijos[21], dónde se permite incluir expresiones regulares en la parte derecha de las reglas. Sin embargo, de esta manera:

- La gramática resultante no será tan sencilla como desearíamos al mezclar información sintáctica y morfológica.

- Estamos creando un nuevo formalismo, cuando lo que nos interesa es aprovechar uno ya existente.
- No solucionamos el problema de la eficiencia del analizador resultante.

La solución adoptada en el sistema *GA*LENAconsiste en aprovechar el trabajo realizado en el analizador morfológico. Dicho analizador es capaz de asociar cada una de las palabras de la frase de entrada con su etiqueta correspondiente. Las etiquetas contendrán en cualquier caso la categoría sintáctica de la palabra y una serie de atributos tales como género, número, tiempo verbal, ... que varían en función de la categoría.

El siguiente paso consiste en extender los símbolos terminales al igual que los no terminales. Las categorías sintácticas jugarán el mismo papel que los símbolos de predicado. Cada uno de los terminales se corresponderá con una de las etiquetas generadas por el analizador morfológico.

En un principio se podría pensar en ordenar los atributos de las etiquetas y crear una correspondencia directa entre su posición dentro de la etiqueta y dentro del *predicado terminal* correspondiente. Véase el siguiente ejemplo:

Palabra	*Categoría sintáctica*	*Tiempo verbal*	*Modo verbal*	*Persona*	*Número*
"sobre"	`Verbo`	`Presente`	`Subjuntivo`	`Tercera`	`Singular`

`verbo(T,M,P,N,L)`, $\Theta = \{$`T/presente`, `M/subjuntivo`, `P/tercera`, `N/singular`$\}$

Sin embargo, se plantean dos problemas:

- Es incómodo tener que usar la tupla entera `verbo(T,M,P,N,L)`, aunque no sean necesarios todos los atributos.
- Cada vez que cambie la definición de las etiquetas, tenemos que cambiar la gramática. Esto propicia la aparición de errores y es contrario al objetivo de modularidad que nos habíamos propuesto.

Para evitarlo, simplemente añadimos a cada argumento una referencia al atributo de la etiqueta. En la notación empleada, el argumento se separa mediante dos puntos (:) del nombre del atributo en cuestión, por ejemplo: `verbo(T:Tiempo,P:Persona)`.

En la figura 3.4 se muestra la definición de las etiquetas para la categoría `verbo`. En la primera parte de la definición se especifica el nombre de la categoría y a continuación los atributos que posee, indicando el nombre, el tipo y como queremos que sea referido en los mensajes del analizador sintáctico. Esta última información se suministra en diferentes idiomas. Como se puede ver, también es posible indicar que atributos toman valores diferentes cuando existe una ambigüedad en la interpretación de la palabra, y cuales no, precediendo su tipo por las palabras `LIST OF`. Por último se enumeran los valores que pueden tomar los atributos indicando también su nombre en diferentes idiomas. En la figura 3.5 tenemos una GCD en la que se usa entre otras la etiqueta descrita: `verbo(P:Palabra,Num:Numero)`.

```
%category VERBO
  STRING              Palabra         "Word" "Palabra";
  LIST OF VALUES      Modo            "Mode" "Modo",
                      Tiempo_verbal   "Verbal tense" "Tiempo verbal",
                      Persona         "Person" "Persona",
                      Contador_clit   "Clit counter" "Contador de cliticos";
  LIST OF STRINGS     Lemma           "Lemma" "Lema";

%value INDICATIVO     "Indicative"    "Indicativo"
%value SUBJUNTIVO     "Subjunctive"   "Subjuntivo"
%value IMPERATIVO     "Imperative"    "Imperativo"
%value INFINITIVO     "Infinitive"    "Infinitivo"
%value GERUNDIO       "Gerund"        "Gerundio"
%value PARTICIPIO     "Participle"    "Participio"
```

Figura 3.4: Ejemplo de especificación de la interfaz léxico-sintáctico.

```
s(esp(Arbol))                →  frase(Arbol).
frase(fr(Arbol1,Arbol2))     →  fn(Arbol1,Num), fv(Arbol2,Num).
frase(fr(Arbol1,Arbol2))     →  frase(Arbol1), fp(Arbol2).
fn(fn(n(P)),Num)             →  nombre(P:Palabra,Num:Numero).
fn(fn(pr(P)),Num)            →  pronombre(P:Palabra,Num:Numero).
fn(fn(det(P1),n(P2)),Num))   →  determinante(P1:Palabra,Num:Numero),
                                nombre(P2:Palabra,Num:Numero).
fn(fn(Arbol1,Arbol2),Num)    →  fn(Arbol1,Num),fp(Arbol2).
fp(fp(prep(P),Arbol)         →  preposicion(P:Palabra), fn(Arbol,Num).
fv(fv(verbo(P),Arbol),Num)   →  verbo(P:Palabra,Num:Numero),
                                fn(Arbol,Num2).
```

Figura 3.5: Gramática de ejemplo para la estructura sintagmática del Castellano.

Capítulo 4

Formalismo operacional para el análisis sintáctico.

El formalismo operacional desarrollado para el sistema *GA*LENA comienza con el trabajo desarrollado por Vilares en sus tesis doctoral [92]. Fruto de este trabajo, es el sistema ICE[1]. ICE es capaz de generar analizadores sintácticos que reconocen GICs sin restricciones, aplicando además un mecanismo de análisis incremental.

Posteriormente, ICE ha sido extendido de manera que los analizadores generados, además de las GICs son capaces de reconocer GCDs [57]. En lo que sigue nos referiremos a esta extensión de ICE como ICE extendido.

Al basar la construcción del segundo modelo en el primero, se han usado los mismos fundamentos teóricos:

- Programación dinámica aplicada a los autómatas de pila,
- Autómatas LALR, y
- El algoritmo de Lang [45], que generaliza la construcción de Earley [25].

añadiendo además los principios provenientes de los *autómatas lógicos de pila*, ALPs, [46, 47, 97].

El funcionamiento de ICE y de ICE extendido se explica ampliamente tanto en [92], como en [2, 3, 4, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 89, 90, 93, 94, 96]. Por su parte, en los apéndices A, y B se introducen brevemente los autómatas LALR, la programación dinámica y el algoritmo de Earley clásico.

El resto del capítulo se estructura como sigue: en primer lugar se introducen los ALPs, a continuación, sobre la base de los mismos, se explican las características propias de ICE extendido, programación dinámica, control estático, análisis ascendente, y por último se presentan una serie de ejemplos ilustrativos de su funcionamiento.

4.1 Autómatas lógicos de pila

La apariencia sintáctica de las cláusulas de Horn, y por tanto de las GCDs, es muy similar a la de las gramáticas independientes del contexto, tal y como se mostró en

[1] Incremental Compiler Enviroment

el capítulo 3. De hecho, uno de los propósitos del lenguaje PROLOG, que contribuyo en gran medida a su popularización, fue generalizar las GICs que se habían mostrado insuficientes para el tratamiento de la estructura sintáctica del lenguaje natural [14, 16].

Sin embargo, y sorprendentemente, ha habido relativamente pocos intentos de aplicar, basándose en las similitudes de ambos formalismos, los resultados obtenidos en el ámbito de los lenguajes independientes de contexto a las gramáticas de cláusulas de Horn. Una de las aproximaciones más interesantes es la adoptada por Pereira y Warren en [57] para la extensión del algoritmo de Earley a un esquema de evaluación lógica completo, sin usar símbolos funcionales.

Otra aproximación, desarrollada inicialmente por Lang [46, 47] y posteriormente por Villemonte de la Clergerie [97], extiende la noción de autómata de pila para dar lugar a los ALPs. Un ALP es, en esencia, un autómata de pila no determinista que almacena átomos lógicos y substituciones en la pila, y usa la unificación a la hora de aplicar las transiciones.

En este momento la propiedad más importante de los ALPs es que para cualquier programa de cláusulas definidas, existe un ALP que computa las mismas respuestas extensionalmente, tal y como se prueba en [46].

Definición

Un autómata lógico de pila es una 6-tupla $\mathcal{A} = (\mathcal{X}, \mathcal{F}, \Sigma, \Delta, \$, \$_f, \Theta)$ donde:

$\mathcal{X}$ es un conjunto enumerable y ordenado de variables.

$\mathcal{F}$ (respec. Δ), es un conjunto finito de símbolos de función (respec. de predicado).

$\$$ (respec. $\$_f$), es el *predicado inicial* (respec. *predicado final*).

Θ es un conjunto finito de transiciones.

Las transiciones en Θ son de tres tipos:

transiciones horizontales:	$B \longmapsto C$
transiciones de empilamiento:	$B \longmapsto CB$
transiciones de depilamiento:	$BD \longmapsto C$

donde B,C y D son átomos construidos con Δ, $\mathcal{F}$ y $\mathcal{X}$. En lo que sigue nos referiremos a ellos como literales, átomos-Δ o simplemente átomos.

□

El funcionamiento intuitivo de las transiciones es como sigue: una transición horizontal intercambia el elemento en la cima de la pila, una transición de empilamiento añade un elemento a la pila, mientras que una de depilamiento retira los dos elementos de la cima e introduce uno nuevo.

El predicado inicial $\$$ sólo puede aparecer como primer elemento de la pila. Por lo tanto, no es más que un elemento usado como marca de final.

Los elementos de la pila son los pares `literal`(A)/`substitucion`(s) y se notarán $A.s$. Usaremos $d_1 d_2 \ldots d_n \xi$ para representar una pila donde los d_i son los n primeros pares de la parte superior y ξ el resto de la pila.

Una *configuración* del ALP está compuesta únicamente por los contenidos de la pila. Una *pila inicial*, y por lo tanto una *configuración inicial*, es de la forma $\$.[]$. Puesto que para una configuración dada pueden existir varias transiciones aplicables, un ALP es un autómata no determinista. Usaremos, además, las siguientes notaciones:

$\xi \underset{\tau}{\vdash\!\!\!-\!\!\!-} \xi'$, aplicando la transición τ a ξ obtenemos ξ' en un solo paso.

$\xi \overset{n}{\vdash\!\!\!-\!\!\!-} \xi'$, pasamos de ξ a ξ' con n transiciones. $\xi = \xi_0 \vdash\!\!\!-\!\!\!- \xi_1 \vdash\!\!\!-\!\!\!- \ldots \vdash\!\!\!-\!\!\!- \xi_n = \xi'$.

$\xi \overset{*}{\vdash\!\!\!-\!\!\!-} \xi'$, indica, como cabe esperar, el cierre transitivo reflexivo de $\vdash\!\!\!-\!\!\!-$.

Si ξ_0 es la configuración inicial y $\xi_0 \overset{*}{\vdash\!\!\!-\!\!\!-} \xi$, diremos que ξ es *derivable*.

4.1.1 Reglas para la aplicación de las transiciones

A la hora de aplicar una transición, se ha de tener en cuenta que:

- Una transición horizontal $B \longmapsto C$ es aplicable sobre una pila $A.u\xi$ si y sólo si A y B son unificables, i.e. existe $s = \text{mgu}(A, B)$. El resultado de aplicar la transición es la pila $Cs.us\,\xi$.
- Una transición de empilamiento $B \longmapsto CB$ es aplicable sobre $A.u\,\xi$ si y sólo si A y B son unificables, $s = \text{mgu}(A, B)$. El resultado es $Cs.s\,A.u\,\xi$.
- Una transición de depilamiento $DB \longmapsto C$ es aplicable sobre una pila $A.u\,A'u'\,\xi$ si y solo si el par $< A, A'u >$ es unificable con $< B, D >$, $s = \text{mgu}(< A, A'u >, < B, D >)$. Obtenemos la pila $Cs.u'us\,\xi$.

Un ALP puede terminar con éxito cuando alcanza una *configuración final*, es decir, una pila conteniendo el predicado final encima del predicado inicial.

Ya que los ALP son no deterministas, puede haber más de una computación que termine con éxito en diferentes configuraciones finales, que no termine o que termine en una configuración no final, es decir, que *fallen*. La *respuesta* del ALP será el conjunto de todas las configuraciones finales posibles y se representará *respuesta*(A).

Ejemplo 4.5:

A continuación se muestra un ejemplo sencillo del funcionamiento de un ALP. El autómata viene dado por:

$$\begin{array}{lll} \Theta = \{ & q'(X) \longmapsto p'(f(X))\,q'(X) & (\tau_1) \\ & p'(f(a)) \longmapsto p''(f(a)) & (\tau_2) \\ & p''(X)\,q'(X) \longmapsto q''(X) & (\tau_3) \quad \} \end{array}$$

$$\$ = q'$$

$$\$_f = q''$$

Aplicando las transiciones τ_1, τ_2 y τ_3 vamos desde la configuración inicial hasta la final pasando por:

$$\boxed{q'(X).[\,]} \overset{\tau_1}{\vdash} \boxed{\begin{array}{c} p'(f(X)).[\,] \\ \hline q'(X).[\,] \end{array}} \overset{\tau_2}{\vdash} \boxed{\begin{array}{c} p'(X).[X \to a] \\ \hline q'(X).[\,] \end{array}} \overset{\tau_3}{\vdash} \boxed{q''(a).[X \to a]}$$

□

4.1.2 Compilación del autómata

Hasta ahora hemos evitado deliberadamente, hablar de la *estrategia de ejecución*[2], puesto que, en principio, no se impone ningún tipo de restricción sobre cual de ellas utilizar.

No obstante, la estrategia seleccionada determinará la manera en que se realiza el proceso de *compilación del ALP*. Dicho proceso consiste en obtener, a partir de un programa de cláusulas definidas, un ALP equivalente por extensión.

Nos referiremos a cada proceso como un *esquema de compilación*. A continuación se expondrán varios de ellos. Introduciremos los predicados $\nabla_{k,i}(t_k)$ entre los literales de cada cláusula $\gamma_k : A_{k,0} \to A_{k,1} \ldots A_{k,n_k}$. El significado de estos literales depende del esquema de compilación elegido. En general indica que parte de la cláusula ya ha sido analizada y cual resta por analizar, t_k es el vector de variables que aparecen en la cláusula; su funcionalidad es la de asegurar que la información proveniente de los argumentos de los predicados manipulados en el pila se transmite correctamente durante la aplicación de las transiciones.

Esquema descendente

En esencia, el mecanismo descrito es el mismo que se emplea en los evaluadores PROLOG. La idea es seleccionar sistemáticamente de izquierda a derecha cada uno de los literales del cuerpo de la cláusula que queremos refutar. A su vez, estos literales así seleccionados se convierten en el siguiente subojetivo a demostrar. El primer paso es añadir una cláusula de índice 0 de la forma

$$\text{pregunta}(\vec{X}) \longmapsto A_{0,1} \ldots A_{0,n_0}$$

donde $\vec{X}$ es el conjunto de variables que aparecen en la(s) pregunta(s) y los literales $A_{0,i}$ la(s) pregunta(s). La pregunta será el predicado inicial y ∇_{0,n_0} el predicado final. El esquema viene definido por las siguientes transiciones:

1. $\$ \longmapsto \nabla_{0,0}(t_0)\$$
2. $\nabla_{k,i}(t_k) \longmapsto A_{k,i+1}\nabla_{k,i}(t_k)$, $\quad \forall\, \gamma_k$ y $\forall i$ t.q. $0 \leq i < n_k$
3. $A_{k,0} \longmapsto \nabla_{k,0}(t_k)$, $\quad \forall \gamma_k$

[2]Las principales estrategias son la ascendente, y la descendente, aunque veremos otras como la deducción Earley.

4. $\nabla_{k,n_k}(t_k)\nabla_{k',i}(t_{k'}) \longmapsto \nabla_{k',i+1}(t_{k'})$[3], $\forall\gamma_k, \gamma_{k'}$ y $\forall i$ t.q. $0 \leq i < n_{k'}$ [4]

El predicado final, ∇_{0,n_0}, se corresponde con el final del cuerpo de la "cláusula de consulta". Intuitivamente, el significado de cada transición es como sigue:

1. *Inicialización*: Intentamos refutar el cuerpo de la pregunta, i.e. γ_0.
2. *Selección del siguiente subobjetivo más a la izquierda*: La presencia de $\nabla_{k,i}(t_k)$ indica que los primeros i literales a la izquierda de la cláusula γ_k han sido refutados. El siguiente paso consiste en intentar refutar el literal $A_{k,i+1}$ que ocupa la posición $i+1$.
3. *Selección de la cláusula γ_k*: Pasamos a considerar el cuerpo de γ_k como una secuencia de nuevos subobjetivos, tal y como indica el literal $\nabla_{k,0}(t_k)$.
4. *Vuelta a la cláusula $\gamma_{k'}$*: La presencia del literal $\nabla_{k,n_k}(t_k)$ en la pila indica que todos los literales del cuerpo de la cláusula γ_k han sido sucesivamente refutados. Volvemos a la cláusula $\gamma_{k'}$ y pasamos de $\nabla_{k',i}(t_k)$ a $\nabla_{k',i+1}(t_{k'})$ puesto que $A_{k',i+1}$ ha sido refutado como una instancia de la cabeza de γ_k.

Esquema ascendente

En este caso, los literales son seleccionados de derecha a izquierda y por lo tanto el literal $\nabla_{k,i}(t_k)$ indica que el resto del cuerpo de la cláusula γ_k, a partir de la posición i, ya ha sido probado. La presencia de un átomo formado a partir de un predicado de la gramática, indica que ya ha sido probado, justo al contrario que en el caso anterior.

Para simplificar la descripción de las transiciones, usaremos M para denotar un átomo formado aplicando un vector de nuevas variables a cualquier predicado del autómata. También haremos la misma consideración sobre la cláusula γ_0 que hicimos para el esquema ascendente. Por lo tanto, el predicado final pasa a ser $\nabla_{0,0}$.

1. $M \longmapsto \nabla_{k,n_k}(t_k)M$, $\forall\gamma_k, \forall M$
2. $\nabla_{k,i}(t_k)A_{k,i} \longmapsto \nabla_{k,i-1}(t_k)$, $\forall\gamma_k, \forall i, 0 < i \leq n_k$
3. $\nabla_{k,0}(t_k) \longmapsto A_{k,0}$, $\forall\gamma_k$

La explicación informal de las transiciones es como sigue:

1. *Selección de una cláusula*: Sea cual se el literal M en la cima de la pila, elegimos de manera arbitraria una cláusula γ_k cuya cabeza deseamos probar. Para ello comenzamos por probar todos los literales de su cuerpo, de derecha a izquierda: $\nabla_{k,n_k}(t_k)$.
2. *Reducción de un literal*: Puesto que hemos probado todos los literales de γ_k después del i-ésimo, $\nabla_{k,i}(t_k)$, y también el i-ésimo, $A_{k,i}$ podemos reducir la pila y pasar a $\nabla_{k,i-1}(t_k)$.

[3] Si $k = k'$ renombramos las variables en $t_{k'}$ como si estuviésemos usando dos variantes distintas de la cláusula γ_k.

[4] Nótese que no necesitamos definir la transición para todos los triples k,k', e i posibles, bastará con aquellos en que la cabeza de γ_k unifique con el literal $A_{k',i+1}$.

3. *Terminación de la prueba de la cabeza de la cláusula* γ_k: El literal $\nabla_{k,0}(t_k)$ indica que todos los literales del cuerpo de γ_k han sido probados, por lo tanto $A_{k,0}$ está probado y podemos reemplazarlo en la pila.

Sobre este esquema, se ha de remarcar que la forma en que se construyen las transiciones del primer tipo, dará lugar a muchas acciones inútiles, es decir que no llevan a ninguna solución, sin embargo se ha decido no introducir ninguna modificación para mantener la simplicidad del ejemplo. Más tarde se verá como se soluciona este problema en el caso de *GA*LENA.

Deducción de *Earley*

Por último se presenta un esquema de compilación que extiende la mecánica transicional del algoritmo de Earley. Antes de introducir las transiciones, para cada predicado P de la gramática definimos los predicados P' y P'', y para cada átomo $A_{k,i}$ conteniendo el símbolo de predicado P definimos $A'_{k,i}$ y $A''_{k,i}$ reemplazando P por P' y P'' respectivamente.

Intuitivamente $A'_{k,i}$ representa la pregunta sobre un átomo y $A''_{k,i}$ la repuesta. Los predicados inicial y final corresponden, respectivamente a $A'_{0,0}$y $A''_{0,0}$. El significado de los predicados $\nabla_{k,i}$ es el mismo que en el esquema descendente. Las transiciones son:

1. $\$ \longmapsto A'_{0,0}\$$
2. $A'_{k,0} \longmapsto \nabla_{k,0}(t_k)A'_{k,0}$ $\quad \forall\gamma_k$
3. $\nabla_{k,i}(t_k) \longmapsto A'_{k,i+1}\nabla_{k,i}(t_k)$ $\quad \forall\gamma_k$ y $\forall i : 0 \leq i < n_k$
4. $\nabla_{k,n_k}(t_k)A'_{k,0} \longmapsto A''_{k,0}$ $\quad \forall\gamma_k$
5. $A''_{k,i+1}\nabla_{k,i}(t_k) \longmapsto \nabla_{k,i+1}(t_k)$ $\quad \forall\gamma_k$ y $\forall i : 0 \leq i < n_k$

La explicación de las transiciones es la siguiente:

1. *Transición inicial.*

2. Elegimos la cláusula γ_k para probar el objetivo situado en la cima de la pila. Se ha de verificar que la cabeza $A_{k,0}$ de γ_k subsume al literal en la cima de la pila.

3. Añadimos a la pila la siguiente pregunta $A'_{k,i+1}$ sobre los literales de la cláusula que está siendo refutada. $\nabla_{k,i}(t_k)$ nos indica que los i primeros literales ya han sido refutados.

4. La presencia de $\nabla_{k,n_k}(t_k)$ significa que γ_k ha sido probada y por lo tanto hemos obtenido $A''_{k,0}$ como repuesta a $A'_{k,0}$.

5. Puesto que hemos refutado los i primeros literales de γ_k, tal y como indica $\nabla_{k,i}(t_k)$, y hemos obtenido la respuesta al siguiente, $A''_{k,i+1}$, indicamos que hemos refutado los $i+1$ primeros: $\nabla_{k,i+1}(t_k)$.

Ejemplo 4.6:

A continuación veremos el resultado de aplicar los esquemas de compilación descritos a una gramática de tamaño mínimo formada por una sola cláusula,

$$\gamma_1 : p(f(a)).$$

y la pregunta $p(X)$. La gramática expandida resultante será:

γ_0 : pregunta(X):-$p(X)$.
γ_1 : $p(f(a))$.

y los autómatas:

Esquema descendente

τ_1 : pregunta$(X) \mapsto \nabla_{0,0}(X)$ pregunta(X)
τ_2 : $\nabla_{0,0}(X) \mapsto p(X)\nabla_{0,0}(X)$
τ_3 : pregunta$(X) \mapsto \nabla_{0,0}(X)$
τ_4 : $p(f(a)) \mapsto \nabla_{1,0}()$
τ_5 : $\nabla_{0,1}(X)\nabla_{0,0}(X') \mapsto \nabla_{0,1}(X')$
τ_6 : $\nabla_{1,0}()\nabla_{0,0}(X) \mapsto \nabla_{0,1}(X)$

Esquema ascendente

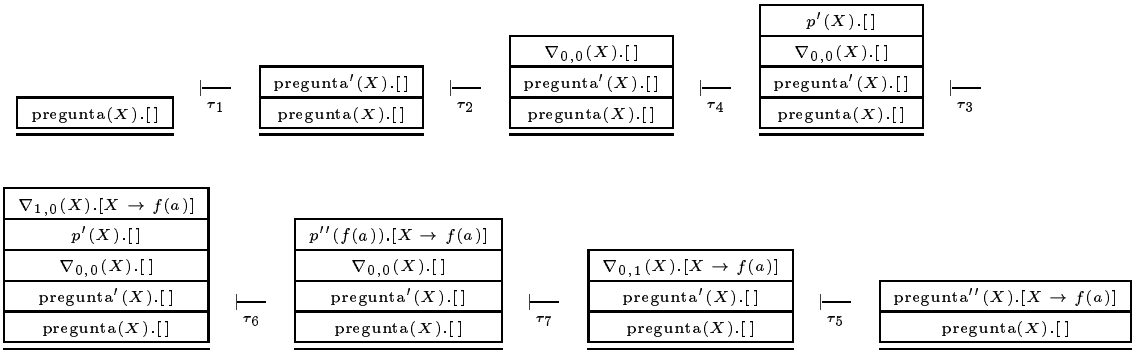
τ_1 : pregunta$(X) \mapsto \nabla_{0,1}(X)$ pregunta(X)
τ_2 : pregunta$(X) \mapsto \nabla_{1,0}(X)$ pregunta(X)
τ_3 : $p(X) \mapsto \nabla_{0,1}(X)\ p(X)$
τ_4 : $p(X) \mapsto \nabla_{1,0}()\ p(X)$
τ_5 : $p(f(a)) \mapsto \nabla_{0,1}(X)\ p(f(a))$
τ_6 : $p(f(a)) \mapsto \nabla_{1,0}()\ p(f(a))$
τ_7 : $\nabla_{0,1}(X)\ p(X) \mapsto \nabla_{0,0}(X)$
τ_8 : $\nabla_{0,0}(X) \mapsto$ pregunta(X)
τ_9 : $\nabla_{1,0}() \mapsto p(f(a))$

Deducción de Earley

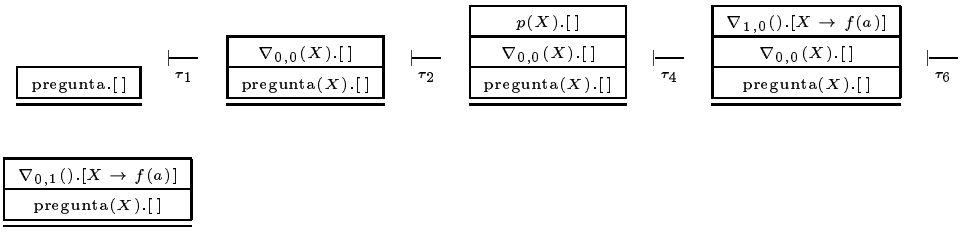
τ_1 : pregunta$(X) \mapsto$ pregunta$'(X)$ pregunta(X)
τ_2 : pregunta$'(X) \mapsto \nabla_{0,0}(X)$ pregunta$'(X)$
τ_3 : $p'(f(a)) \mapsto \nabla_{1,0}()\ p'(f(a))$
τ_4 : $\nabla_{0,0}(X) \mapsto p'(X)\ \nabla_{0,0}(X)$
τ_5 : $\nabla_{0,1}(X)$ pregunta$'(X) \mapsto$ pregunta$''(X)$
τ_6 : $\nabla_{1,0}()\ p'(f(a)) \mapsto p''(f(a))$
τ_7 : $p''(X)\nabla_{0,0}(X) \mapsto \nabla_{0,1}(X)$

Estos autómatas darán lugar, a su vez, a las siguientes secuencias de configuraciones:

Deducción de Earley

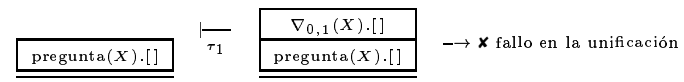


Esquema descendente

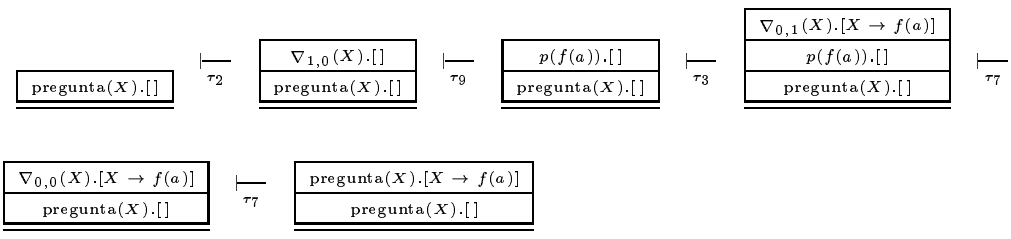


Esquema ascendente

Para la estrategia ascendente, ya desde la configuración inicial, es posible aplicar dos transiciones. La primera no nos lleva a ninguna solución:



En cambio si aplicamos τ_2 y τ_9, volvemos a tener dos opciones: τ_4, que tampoco da lugar a una derivación válida, y τ_3. Con esta última el análisis sí que tiene éxito:



□

4.2 Reducción del espacio de búsqueda

La difícil determinización del análisis, tanto morfológico como sintáctico hace que el espacio de búsqueda pueda alcanzar una complejidad inabordable, incluso exponencial en la longitud del texto analizado. Por lo tanto, se hace imprescindible su reducción, con el fin de ganar espacio útil desde el punto de vista computacional, y acelerar el proceso de cálculo. Las técnicas empleadas en ICE extendido para este fin son:

- La utilización de entornos dinámicos que aseguren una representación lo más compacta posible de los estados, favoreciendo la compartición de configuraciones y evitando a la vez la generación de cálculos redundantes.
- La aplicación de un mecanismo de control que evite la transición hacia estados inviables y por lo tanto cálculos inútiles.

4.2.1 Entornos dinámicos

La definición del entorno dinámico no es más que la aplicación de los concepto de programación dinámica a los autómatas de pila, tal y como se puede ver en el apéndice B. Para aplicar las técnicas de programación dinámica [8], debemos definir:

- Un dispositivo donde almacenar las soluciones parciales.
- Una estructura de datos adecuada para representarlas[5].
- La definición de una relación de subsumción para evitar el almacenamiento y cálculo de resultados duplicados. Al mismo tiempo, evitando la repetición de cálculos, se permite la terminación del proceso de cálculo.
- Un orden equitativo de selección para el tratamiento de los items, capaz de garantizar la completud operacional.

En nuestro caso usaremos *items* para representar las configuraciones intermedias en el proceso de interpretación del ALP. Teniendo en cuenta que diferentes configuraciones pueden compartir cálculos intermedios, cuanto más compacta sea la representación de los items, mayor será la compartición. Cada item es, por lo tanto, una representación "condensada" de una configuración del ALP, y el conjunto de todos los items posibles define el espacio de búsqueda.

Definición

Para un ALP, $\mathcal{A} = (\mathcal{X}, \mathcal{F}, \Sigma, \Delta, \$, \$_f, \Theta)$, un entorno dinámico es un par $(\Re, Op)$, donde:

- $\Re$ es una relación de equivalencia sobre las configuraciones del autómata. Llamaremos *items* a las clases de equivalencia. Notaremos:
  - La clase de ξ como $\overline{\xi}$.

[5] En nuestro caso, los *items* o *estados*

  - El conjunto de items como $It_{A,\Re}$, o simplemente It, cuando no hay ambigüedad.

- Op es un operador de la forma:

$$\begin{array}{rccl} Op: & \Theta & \rightarrow & \{It^+ \rightarrow It^+\} \\ & \tau & \rightsquigarrow & Op(\tau) : It^n \rightarrow It^m \end{array}$$

  donde n y m dependen de la naturaleza de las transiciones, concretamente representan, respectivamente, el número de items sobre los que se aplica la transición y el número de items que se generan.

  Además, se han de verificar las siguientes condiciones:

  - *Compatibilidad*: Todas las computaciones en el ALP tienen su contrapartida en el entorno dinámico.

$$\forall \tau \in \Theta,\ \xi \in Dom(\tau), \left\{ \begin{array}{l} \exists I_{1\ldots n},\ \text{t.q. } (\overline{\xi}, I_1, \ldots, I_n) \in Dom(Op(\tau)) \\ Op(\tau)(\overline{\xi}, I_1, \ldots, I_n) = \tau(\xi) \end{array} \right.$$

    donde $Dom(\tau)$ representa el dominio para $\tau \in \Theta$.

  - *Completud*: Todas las configuraciones finales en el ALP tienen su contrapartida en el entorno dinámico.

$$\forall \xi,\ \text{t.q} \stackrel{*}{\vdash\!\!\!-} \quad \Rightarrow \quad \stackrel{*}{\overline{\vdash}\!\!\!-}$$

  - *Corrección*: Todas las configuraciones finales en el entorno dinámico tienen su contrapartida en el ALP.

$$\forall I,\ \text{t.q.} \stackrel{*}{\overline{\vdash}\!\!\!-} I \quad \Rightarrow \quad \exists \xi,\ \text{t.q.} \stackrel{*}{\vdash\!\!\!-} \xi \text{ y } \overline{\xi} = I$$

Intuitivamente, $\overline{\vdash}\!\!\!-$ corresponde a la aplicación de una transición en el entorno dinámico.

□

El entorno dinámico estandard, llamado S^T, se define por:

- $\Re$: Para cualquier configuración ξ, $\overline{\xi} = \xi$.

- El operador Op es la identidad.

Intuitivamente S^T equivale al ALP original. Cada clase de equivalencia está formada por un único elemento; una configuración del ALP.

En la práctica, sólo se consideran dos entornos dinámicos, S^2 y S^1. Este último es el que mejor se adapta a los objetivos de ICE e ICE extendido.

El entorno S^1, introducido por Villemonte de la Clergerie en [97] en lógica de Horn y por Vilares en [92], se caracteriza por *items* de la forma $[A.u] := \{A.u\xi\}$, mientras que en S^2 son de la forma $[A.u\, A'.u'] := \{A.uA'.u'\xi\}$, donde $A'u = A'u'$, ver [73].

En cuanto a la relación de subsumción, se define como la subsumción entre items, $A \preceq B$ en S^1, $AA' \preceq BB'$ en S^2. Es decir si $[A.u] \preceq [A'.u']$ ambos estados representan la misma solución parcial[6] y por lo tanto se deben almacenar una única vez en la tabla de objetos, y no es necesario repetir los cálculos sobre $[A.u]$ en $[A'.u']$, o viceversa. El mismo razonamiento se aplica al caso de S^2.

Si tomamos como *espacio de búsqueda* el conjunto de todos los items que se podrían generar para una gramática dada, el espacio de búsqueda generado en S^1 es mucho menor que en S^T, y también es menor que en S^2. Esto se puede ver más claro en el ejemplo de la figura 4.1.

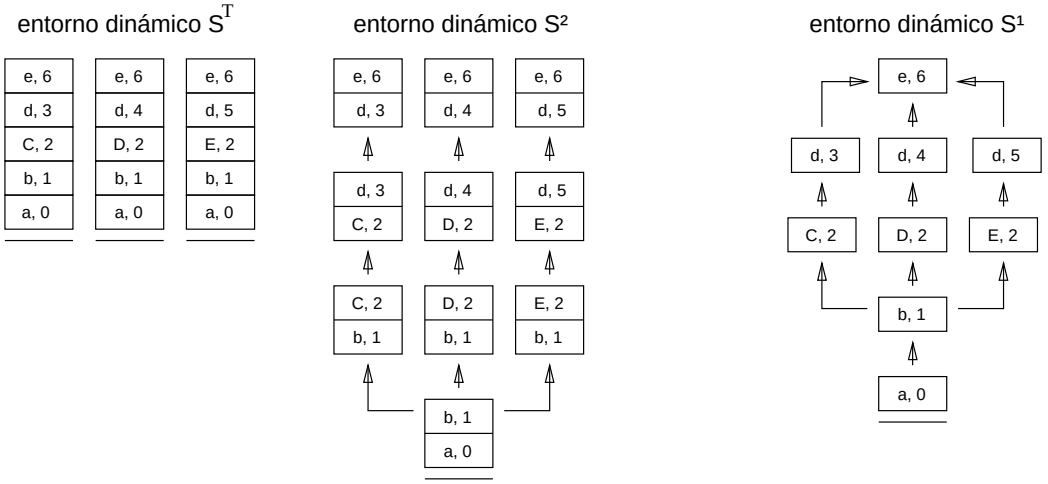


Figura 4.1: Compartición de estructuras en diferentes entornos dinámicos.

A pesar de que el entorno S^1 presenta una mayor compactación que S^2, tiene el problema de no proporcionar información acerca del contexto sintáctico, por lo que es necesario en las transiciones que extraen elementos de la pila, prever un mecanismo capaz de solventar esta limitación.

Concretamente el problema se plantea a la hora de definir la equivalencia de las transiciones de depilamiento[7]. Para solventar el problema se aplica un nuevo concepto: las *transiciones dinámicas*.

En esta línea, definimos el operador Op de la siguiente manera:

- *Caso horizontal*: $B \longmapsto C$ $Op(B \longmapsto C)([A.u]) = [C\sigma.u\sigma]$, donde $\sigma = \text{mgu}(A, B)$

- *Caso extracción*: $BD \longmapsto C$ $Op(BD \longmapsto C)([A.u]) = \{D\sigma \longmapsto C\sigma\}$, donde $\sigma = \text{mgu}(A, B)$, y $D\sigma \longmapsto C\sigma$ es una *transición dinámica* generada por la extracción.

  La nueva transición es aplicable tanto a la configuración resultante como a todas aquellas todavía no generadas, que comparten el mismo entorno sintáctico.

[6] En realidad, probablemente uno de ellos representará una solución más general.

[7] Recordemos que son de la forma: $BD \longmapsto C.$, y que cada estado sólo contiene información del elemento en la cima de la pila.

- *Caso inserción*: $B \longmapsto CB\ Op(B \longmapsto CB)([A.u]) = [C\sigma.\sigma]$, donde $\sigma = \text{mgu}(A, B)$

Ejemplo 4.7:

En este ejemplo se ilustra el uso de las transiciones dinámicas. Supongamos las siguientes cláusulas creadas a partir del conocido lenguaje de Dyck:

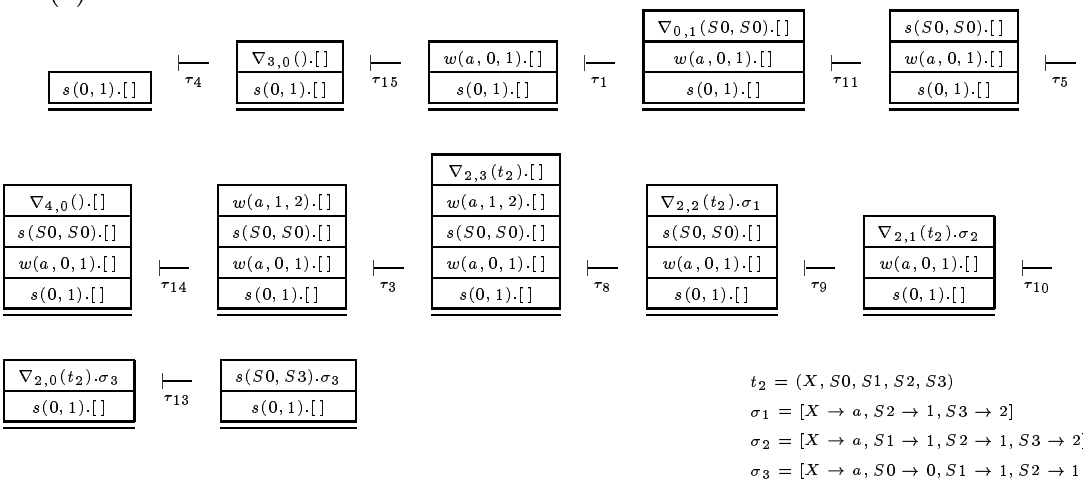
$$\begin{array}{lll}
\gamma_0: & s(S0,S0). & \\
\gamma_1: & s(S0,S2) & \rightarrow\ s(S0,S1),\ s(S1,S2). \\
\gamma_2: & s(s0,S3) & \rightarrow\ w(X,S0,S1),\ s(S1,S2),\ w(X,S2,S3). \\
\gamma_3: & w(a,0,1). & \\
\gamma_4: & w(a,1,2). &
\end{array}$$

Aplicando una estrategia ascendente se obtiene el siguiente conjunto de transiciones:

$$\begin{array}{ll}
\tau_1: & M \mapsto \nabla_{0,0}(S0)\ M \\
\tau_2: & M \mapsto \nabla_{1,2}(S0,S1,S2)\ M \\
\tau_3: & M \mapsto \nabla_{2,3}(X,S0,S1,S2,S3)\ M \\
\tau_4: & M \mapsto \nabla_{3,0}()\ M \\
\tau_5: & M \mapsto \nabla_{4,0}()\ M \\
\tau_6: & \nabla_{1,2}(S0,S1,S2)\ s(S1,S2) \mapsto \nabla_{1,1}(S0,S1,S2) \\
\tau_7: & \nabla_{1,1}(S0,S1,S2)\ s(S0,S1) \mapsto \nabla_{1,0}(S0,S1,S2) \\
\tau_8: & \nabla_{2,3}(X,S0,S1,S2,S3)\ w(X,S2,S3) \mapsto \nabla_{2,2}(X,S0,S1,S2,S3) \\
\tau_9: & \nabla_{2,2}(X,S0,S1,S2,S3)\ s(S1,S2) \mapsto \nabla_{2,1}(X,S0,S1,S2,S3) \\
\tau_{10}: & \nabla_{2,1}(X,S0,S1,S2,S3)\ w(X,S0,S1) \mapsto \nabla_{2,0}(X,S0,S1,S2,S3) \\
\tau_{11}: & \nabla_{0,0}(S0,S1) \mapsto s(S0,S0) \\
\tau_{12}: & \nabla_{1,0}(S0,S1,S2) \mapsto s(S0,S2) \\
\tau_{13}: & \nabla_{2,0}(X,S0,S1,S2,S3) \mapsto s(S0,S3) \\
\tau_{14}: & \nabla_{3,0}() \mapsto w(a,0,1) \\
\tau_{15}: & \nabla_{4,0}() \mapsto w(a,1,2)
\end{array}$$

Partiendo de la pregunta $s(0,1)$, podemos generar en el entorno S^T las dos derivaciones que se muestran a continuación:

(1)



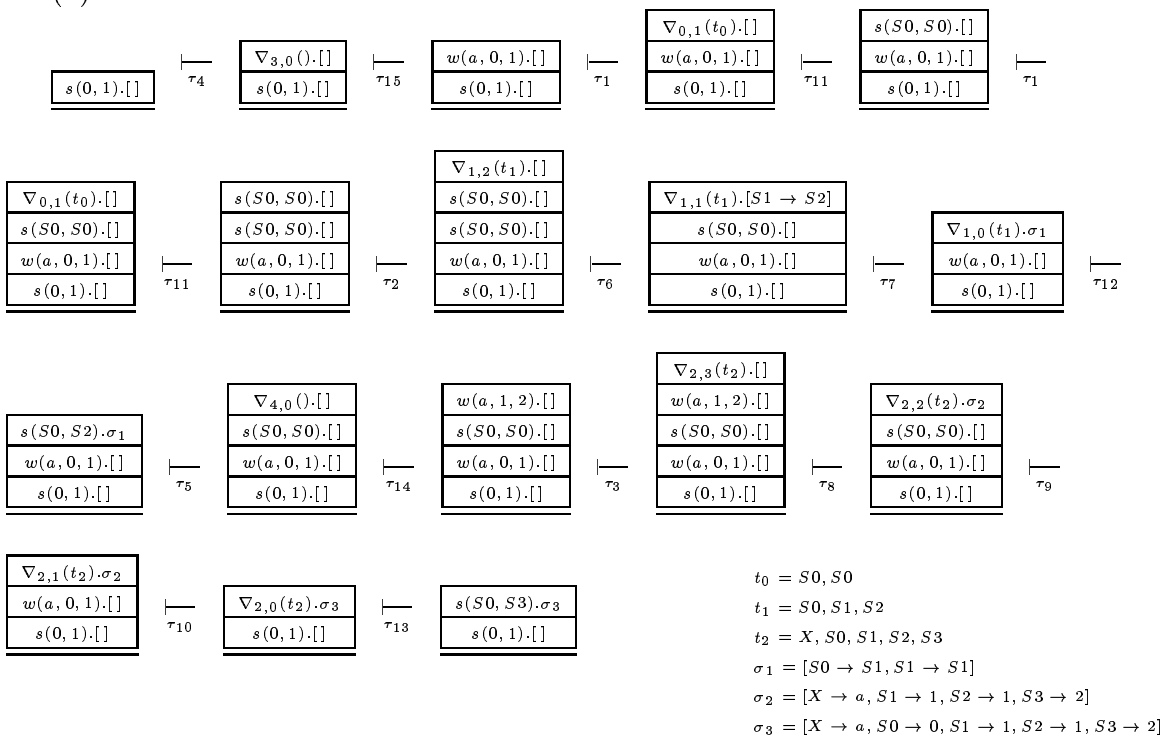
$t_2 = (X, S0, S1, S2, S3)$

$\sigma_1 = [X \rightarrow a, S2 \rightarrow 1, S3 \rightarrow 2]$

$\sigma_2 = [X \rightarrow a, S1 \rightarrow 1, S2 \rightarrow 1, S3 \rightarrow 2]$

$\sigma_3 = [X \rightarrow a, S0 \rightarrow 0, S1 \rightarrow 1, S2 \rightarrow 1, S3 \rightarrow 2]$

(2)



Obsérvese que, por un lado, existen muchas más posibles derivaciones, algunas producen un análisis válido y otras no. Para simplificar el ejemplo sólo se muestran dos, tanto en S^T como en S^1. Por otra parte, tanto las configuraciones del inicio como del final de ambas derivaciones, son comunes por lo que debemos esperar que sólo se calculen una vez en S^1.

Obtenemos los items equivalentes a las configuraciones de la primera parte de las derivaciones aplicando las operaciones:

$$\boxed{s(0,1).[]} \overset{Op(\tau_5)}{\longrightarrow} \boxed{\nabla_{3,0}().[]} \overset{Op(\tau_{15})}{\longrightarrow} \boxed{w(a,0,1).[]} \overset{Op(\tau_1)}{\longrightarrow} \boxed{\nabla_{0,1}(t_0).[]} \overset{Op(\tau_{11})}{\longrightarrow} \boxed{s(S0,S0).[]}$$

Sobre el último item generado, aplicaremos $Op(\tau_5)$ y $Op(\tau_1)$, siguiendo la primera y segunda derivación realizadas en S^T. Para la primera opción obtenemos:

$$\overset{Op(\tau_5)}{\longrightarrow} \boxed{\nabla_{4,0}().[]} \overset{Op(\tau_{14})}{\longrightarrow} \boxed{w(a,1,2).[]} \overset{Op(\tau_3)}{\longrightarrow} \boxed{\nabla_{2,3}(t_2).[]} \overset{Op(\tau_8)}{\longrightarrow} \begin{cases} d_1 \equiv \left\{ w(X,S2,S3) \longmapsto \nabla_{2,2}(t_2) \right\} \\ \boxed{\nabla_{2,2}(t_2).\sigma_1} \end{cases}$$

$$\boxed{\nabla_{2,2}(t_2).\sigma_1} \overset{Op(\tau_9)}{\longrightarrow} \begin{cases} d_2 \equiv \left\{ s(S1,S2).\sigma_1 \longmapsto \nabla_{2,1}(t_2).\sigma_1 \right\} \\ \boxed{\nabla_{2,1}(t_2).\sigma_2} \overset{Op(\tau_{10})}{\longrightarrow} \begin{cases} d_3 \equiv \left\{ w(X,S0,S1).\sigma_2 \longmapsto \nabla_{2,0}(t_2).\sigma_2 \right\} \\ \boxed{\nabla_{2,0}(t_2).\sigma_3} \end{cases} \end{cases}$$

$$\boxed{\nabla_{2,0}(t_2).\sigma_3} \overset{Op(\tau_{13})}{\longrightarrow} \boxed{s(S0,S3).\sigma_3}$$

$t_2 = X, S0, S1, S2, S3$
$\sigma_1 = [X \to a, S2 \to 1, S3 \to 2]$
$\sigma_2 = [X \to a, S1 \to 1, S2 \to 1, S3 \to 2]$
$\sigma_3 = [X \to a, S0 \to 0, S1 \to 1, S2 \to 1, S3 \to 2]$

Hasta el momento la tabla de objetos contiene los siguientes items:

1	$s(0,1).[\,]$	2	$\nabla_{3,0}().[\,]$	3	$w(a,0,1).[\,]$	4	$\nabla_{0,1}(t_0).[\,]$	5	$s(S0,S0).[\,]$	6	$\nabla_{4,0}().[\,]$
7	$w(a,1,2).[\,]$	8	$\nabla_{2,3}(t_2).[\,]$	9	$\nabla_{2,2}(t_2).\sigma_1$	10	$\nabla_{2,2}(t_2).\sigma_1$	11	$\nabla_{2,1}(t_2).\sigma_2$	12	$\nabla_{2,0}(t_2).\sigma_3$
13	$\nabla_{2,0}(t_2).\sigma_3$	14	$s(S0,S3).\sigma_3$								

y transiciones dinámicas:

$$d_1 \equiv \left\{ w(X,S2,S3) \longmapsto \nabla_{2,2}(t_2) \right\}$$
$$d_2 \equiv \left\{ s(S1,S2).\sigma_1 \longmapsto \nabla_{2,1}(t_2).\sigma_1 \right\}$$
$$d_3 \equiv \left\{ w(X,S0,S1).\sigma_2 \longmapsto \nabla_{2,0}(t_2).\sigma_2 \right\}$$

La otra opción era aplicar $Op(\tau_1)$ sobre el item 5. El resultado es $\nabla_{0,1}(t_0).[\,]$, es decir el item 4.

$$\boxed{s(S0,S0).[\,]} \overset{Op(\tau_1)}{\longrightarrow} \boxed{\nabla_{0,1}(t_0).[\,]}$$

En principio el proceso de análisis terminaría aquí. Sin embargo, se han omitido opciones a la hora de tratar el item 5 para seguir el proceso que se llevo a cabo en S^T. También es posible aplicar $Op(\tau_2)$ sobre este mismo item.

$$\boxed{s(S0,S0).[\,]} \overset{Op(\tau_2)}{\longrightarrow} \boxed{\nabla_{1,2}(t_1).[\,]} \overset{Op(\tau_6)}{\longrightarrow} \begin{cases} d_4 \equiv \left\{ s(S1,S2).[\,] \longmapsto \nabla_{1,1}(t_1).[\,] \right\} \\ \boxed{\nabla_{1,1}(t_1).\sigma_4} \overset{Op(\tau_7)}{\longrightarrow} \begin{cases} d_5 \equiv \left\{ w(X,S0,S1).\sigma_4 \longmapsto \nabla_{1,0}(t_1).\sigma_4 \right\} \\ \boxed{\nabla_{1,0}(t_1).\sigma_5} \overset{Op(\tau_{12})}{\longrightarrow} \end{cases} \end{cases}$$

$$\boxed{\nabla_{1,0}(t_1).\sigma_5} \overset{Op(\tau_{12})}{\longrightarrow} \boxed{s(S0,S2).\sigma_5}$$

$$\sigma_4 \equiv [S1 \to S2]$$
$$\sigma_5 \equiv [S1 \to S2, S0 \to S2]$$

$s(S0,S2).\sigma_5$ comparte el mismo contexto sintáctico que el item 5 y por lo tanto le podemos aplicar la transición dinámica d_2.

$$\boxed{s(S0,S2).\sigma_5} \overset{Op(d_2)}{\longrightarrow} \boxed{\nabla_{2,1}.\sigma_2}$$

$\nabla_{2,1}.\sigma_2$ es igual al item 11, por lo tanto el análisis para aquí. La tabla de objetos queda:

1	$s(0,1).[\,]$	2	$\nabla_{3,0}().[\,]$	3	$w(a,0,1).[\,]$	4	$\nabla_{0,1}(t_0).[\,]$	5	$s(S0,S0).[\,]$	6	$\nabla_{4,0}().[\,]$
7	$w(a,1,2).[\,]$	8	$\nabla_{2,3}(t_2).[\,]$	9	$\nabla_{2,2}(t_2).\sigma_1$	10	$\nabla_{2,2}(t_2).\sigma_1$	11	$\nabla_{2,1}(t_2).\sigma_2$	12	$\nabla_{2,0}(t_2).\sigma_3$
13	$\nabla_{2,0}(t_2).\sigma_3$	14	$s(S0,S3).\sigma_3$	15	$\nabla_{1,2}(t_1).[\,]$	16	$\nabla_{1,1}(t_1).\sigma_4$	17	$\nabla_{1,0}(t_1).\sigma_5$	18	$s(S0,S2).\sigma_5$

y las transiciones dinámicas:

$$d_1 \equiv \left\{ w(X,S2,S3) \longmapsto \nabla_{2,2}(t_2) \right\}$$
$$d_2 \equiv \left\{ s(S1,S2).\sigma_1 \longmapsto \nabla_{2,1}(t_2).\sigma_1 \right\}$$
$$d_3 \equiv \left\{ w(X,S0,S1).\sigma_2 \longmapsto \nabla_{2,0}(t_2).\sigma_2 \right\}$$
$$d_4 \equiv \left\{ s(S1,S2).[\,] \longmapsto \nabla_{1,1}(t_1).[\,] \right\}$$
$$d_5 \equiv \left\{ w(X,S0,S1).\sigma_4 \longmapsto \nabla_{1,0}(t_1).\sigma_4 \right\}$$

□

En relación a los inconvenientes, S^1 es incompatible con los esquemas de análisis descendente [92, 97], debido a la falta de información acerca del contexto sintáctico. Un esquema de evaluación de este tipo puede llegar a predecir más información de la que luego puede recuperar, con lo que la corrección del proceso de cálculo no está asegurada. Este no es el caso que nos ocupa, pues el nuestro será un esquema ascendente.

4.2.2 Sincronización al estilo del algoritmo de Earley clásico

Se pretende simplificar:

- El proceso de tabulación de los items.
- El manejo de las transiciones dinámicas.

Para ello los items generados durante el proceso de análisis se dividen en conjuntos de items, también llamados *itemsets*. Al igual que en el algoritmo de Earley clásico [26], existe en itemset por cada elemento de la cadena de entrada.

Además, asociamos a cada predicado A una referencia al itemset al que pertenece, es decir la posición de lectura en la cadena de entrada cuando fue generado, también llamado puntero actual, y un *puntero de retroceso*, que será la posición en la cadena de entrada del primer símbolo terminal derivado por A, en la forma:

$$A^{posición,retroceso}$$

Decimos que el proceso de análisis se sincroniza a nivel de itemsets, puesto que, hasta que no se generan todos los items pertenecientes al itemset i, no se pasa a procesar el itemset $i+1$. En adelante, nos referiremos al itemset que estamos tratando en un momento dado del análisis como el *itemset actual*.

Intuitivamente, la idea es simple. Recordemos que la relación de subsumción que definíamos, nos indicaba cuando dos items representaban la misma solución parcial, por lo tanto items pertenecientes a diferentes itemsets no se pueden subsumir puesto que reconocen partes diferentes de la cadena de entrada. Consecuentemente, en cada etapa del proceso de análisis la gestión de la tabla de objetos se limita al itemset actual.

Desde el punto de vista práctico, en los experimentos realizados con diversas gramáticas, el número medio de items en cada itemset, en relación al número total de items se situaba en un rango de 0.22%, en el mejor de los casos, a 33.33%, en el peor, siendo el valor medio 6.41%. Cabe resaltar, que los peores resultados se obtuvieron para cadenas de entrada pequeñas, 3 y 4 símbolos terminales, y que ha medida que aumentaba su tamaño, disminuía la relación.

En lo tocante a las transiciones dinámicas, podemos aplicar los mismos razonamientos expuestos para el caso de los items.

4.2.3 Control estático

Para evitar los conflictos de evaluación entre términos del ALP, echaremos mano del concepto de *ventana*[8], ampliamente utilizado en la determinación de los lenguajes algebraicos.

En el caso de ICE extendido, la ventana se calcula directamente a partir del esqueleto independiente del contexto, $\mathcal{G}^f$, de la GCD, $\mathcal{G}$. El esquema de evaluación se define por el conjunto de transiciones que se presenta a continuación. Para el modo de reducción:

$$\begin{array}{lllll}
1. & A^{it,bp}_{k,n_k} & \left\{lookahead^f_k(A^f_{k,0})\right\} & \longmapsto & \nabla^{it,it}_{k,n_k}(\vec{T}_k) \quad A^{it,bp}_{k,n_k} \\
2. & \nabla^{it,r}_{k,i}(\vec{T}_k) & A^{r,bp}_{k,i} & \longmapsto & \nabla^{it,bp}_{k,i-1}(\vec{T}_k), \quad i \in [1,n_k] \\
3. & \nabla^{it,bp}_{k,0}(\vec{T}_k) & & \longmapsto & A^{it,bp}_{k,0}
\end{array}$$

Para el modo de lectura:

$$\begin{array}{lllll}
4. & A^{it,bp}_{k,i} & \left\{follow^f_1(A^f_{k,i}), n, it\right\} & \longmapsto & A^{n,it}_{k,i+1} \quad A^{it,bp}_{k,i},\ i \in [0,n_k) \\
5. & A^{it,bp}_{k,0} & \left\{first^f_1(A^f_{m,0}), n, it\right\} & \longmapsto & A^{n,it}_{m,1} \quad A^{it,bp}_{k,0} \\
6. & \$^{it,bp} & \left\{first^f_1(\Phi), n, it\right\} & \longmapsto & E^{n,t}_{\$} \quad \$^{it,bp}
\end{array}$$

En cualquier caso, las notaciones $first^f_1$, $follow^f_1$, y $lookahead^f_k$ representan, respectivamente, los conceptos $first_1$, $follow_1$, y $lookahead_k$ para el esqueleto independiente del contexto. Representaremos por $A^f_{k,i}$ el término obtenido en $\mathcal{G}^f$ a partir de $A_{k,i}$.

La posición n referencia la posición siguiente a la actual de lectura *it* cuando el símbolo reconocido no es ε, e *it* en otro caso. De nuevo $\nabla_{k,i}$ indica que, para la cláusula γ_k, todos los literales situados a partir de la posición i han sido analizados. Brevemente, la interpretación de las transiciones es como sigue:

1. *Selección de una cláusula*: Cuando la condición $E_k = \left\{lookahead^f_k(A^f_{k,0})\right\}$ se verifica, seleccionamos la regla γ_k si un literal A_{k,n_k} está en la cima de la pila. En ese caso, insertamos $\nabla_{k,n_k}(\vec{T}_k)$ para indicando que restan por reconocer todos los literales en la parte derecha de la cláusula.

   Esta transición es un refinamiento de la que veíamos en el caso del autómata lógico con una estrategia ascendente pura. El mecanismo de control estático impone una restricción que nos asegura que no se intentarán aquellas alternativas de las que se sabe *a priori* que darán lugar a un análisis válido.

2. *Reducción en la parte derecha de una cláusula* γ_k: La presencia de $\nabla_{k,i}(\vec{T}_k)$ en la cima de la pila indica que todos los literales en la parte derecha de γ_k situados más allá de la posición i, han sido ya reconocidos. En ese momento, todas las pilas que contengan al literal $A_{k,i}$ justo debajo de su cima pueden incrementar la posición del literal ∇, indicando de esta forma el avance del proceso de análisis en la parte derecha de la regla.

3. *Reducción de la parte izquierda de una cláusula* γ_k: El literal $\nabla_{k,0}(\vec{T}_k)$ indica que todos los literales en la parte derecha de la regla γ_k han sido reconocidos.

[8] También, símbolo adelantado, del inglés *lookahead*.

Reemplazar el literal de posición por el literal $A_{k,0}$ que constituye la parte izquierda de la regla, indicando de esta forma que ha sido reducido.

Tanto esta transición como la anterior son equivalentes a las que ya vimos en el caso de un esquema ascendente puro.

4. *Inserción de literales*: El literal $E_{k,i}$, correspondiente a la lectura de una categoría léxica, se inserta en la pila.

5. *Inserción del primer literal de la parte derecha de una cláusula*: El literal $E_{m,1}$ es insertado en la pila, marcando el inicio del reconocimiento de la parte derecha de la cláusula γ_m.

6. *Inserción inicial*: El predicado inicial marca el inicio del proceso de análisis mediante acciones de inserción exclusivamente.

En consecuencia el esquema descrito es ascendente, aunque admite la consideración de mecanismos predictivos de carácter estático a través de la condición de control opcional. El mecanismo de control actualmente implementado en ICE extendido, se corresponde a la adaptación de la dinámica de un autómata LALR(1) clásico.

Dado que tanto $first_1$, $follow_1$, y $lookahead_k$ pueden ser calculados *a priori*, el mecanismo de control se implementa de manera estática, presentando las siguientes ventajas:

- Desde el punto de vista computacional, el coste del cálculo de los hechos que componen el conjunto de condiciones de control no afecta a la operación del analizador sintáctico.

- El dominio determinista de la familia LALR(1) es amplio, lo cual se traduce en un tratamiento eficiente del determinismo local[9]. Ello conlleva una mayor eficacia del sistema.

- El fenómeno de división de estados, típico de las estrategias de determinación mediante predicción estática, se mantiene dentro de límites razonables.

  Consecuentemente, se minimizan las distinciones entre contextos sintácticos que resultan irrelevantes. El impacto en la compartición de estructuras y cálculos resulta mínimo.

4.3 Ejemplo de análisis

A continuación se presenta un ejemplo del proceso de análisis. La gramática de ejemplo es la proporcionada en *GATE*[10] para las estructuras sintagmáticas del inglés.

La gramática agrupa un gran número de reglas, 317, por lo que se incluye en el apéndice D, y a continuación se listan solamente las cláusulas implicadas en el proceso de análisis:

[9] En la práctica, el procesos es determinista durante buena parte del análisis.

[10] `http://www.dcs.shef.ac.uk/research/groups/nlp/gate/`

γ_{20}	$\mathrm{np}(A,B,C,D,E,F)$	$\rightarrow$	$\mathrm{np}(G,B,C,H,I,J),\mathrm{prepp}(K,L,M,N,O,P).$
γ_{23}	$\mathrm{np}(A,B,C,D,E,F)$	$\rightarrow$	$\mathrm{n}(G,B,C).$
γ_{24}	$\mathrm{np}(A,B,C,D,E,F)$	$\rightarrow$	$\mathrm{names\_np}(G,H,I,J,K,L).$
γ_{27}	$\mathrm{np}(A,B,C,D,E,F)$	$\rightarrow$	$\mathrm{qual}(G,H,I,J,K,L),\ \mathrm{n}(M,B,C).$
γ_{35}	$\mathrm{np}(A,B,sing,C,D,E)$	$\rightarrow$	$\mathrm{np}(F,G,H,I,J,K),\ \mathrm{n}(L,B,sing).$
γ_{50}	$\mathrm{prepp}(A,B,C,D,E,F)$	$\rightarrow$	$\mathrm{in}(G),\mathrm{np}(H,I,J,K,L,M).$
γ_{67}	$\mathrm{qual}(A,B,C,D,E,F)$	$\rightarrow$	$\mathrm{names\_np}(G,H,I,J,K,L).$
γ_{160}	$\mathrm{names\_np}(A,B,C,D,E,F)$	$\rightarrow$	$\mathrm{pn}(G,H,I).$
γ_{173}	$\mathrm{organ\_names\_np}(A,B,C,D,E,F)$	$\rightarrow$	$\mathrm{organ\_names\_np2}(G,H,I,J,K,L).$
γ_{181}	$\mathrm{organ\_names\_np2}(A,B,C,D,E,F)$	$\rightarrow$	$\mathrm{names\_np}(G,H,I,J,K,L).$
γ_{294}	$\mathrm{n}(A,B,C)$	$\rightarrow$	$[\mathrm{n}(A,B,C)].$
γ_{295}	$\mathrm{pn}(A,B,C)$	$\rightarrow$	$[\mathrm{pn}(A,B,C)].$
γ_{303}	$\mathrm{in}(A)$	$\rightarrow$	$[\mathrm{in}(A)].$

La frase de entrada a analizar es:

licence of australian futures trader

que se corresponde con el siguiente análisis léxico:

licence	$\mathrm{n}(\mathrm{licence},3,\mathrm{plur})$
of	$\mathrm{in}(\mathrm{noroot})$
australian	$\mathrm{pn}(\mathrm{australian},3,\mathrm{sing})$
futures	$\mathrm{n}(\mathrm{futures},3,\mathrm{plur})$
trader	$\mathrm{pn}(\mathrm{trader},3,\mathrm{sing})$

El análisis comienza con la configuración

$$\boxed{[\$^{0,0},0].[\,]}$$

Para no extender excesivamente la exposición, se ha tratado de simplificar el tratamiento de los argumentos de los predicados, por considerar más interesante los mecanismos de control estático y sincronización. Así escribiremos

$$\mathrm{n}(\mathrm{licence},3,\mathrm{plur}),\quad \text{en lugar de } \mathtt{n}(A,B,C).[A \rightarrow \mathtt{licence}, B \rightarrow \mathtt{3}, C \rightarrow \mathtt{plur}]$$

siempre que no haya ambigüedad. Igualmente, la estrategia de evaluación puede ser expresada de la siguiente manera, más breve, añadiendo a cada item el estado del control estático a partir del cual fue generado:

1. $[A^{it,bp}_{k,n_k},st] \longmapsto [\nabla^{it,it}_{k,n_k}(t_k),st]\,[A^{it,bp}_{k,n_k},st]$ $\quad\left\{\mathrm{acc}(st,w_{it}) = \mathrm{reducir}(\gamma^f_k)\right\}$
2. $[\nabla^{it,r}_{k,i}(t_k),st_1]\,[A^{r,bp}_{k,i},st_2] \longmapsto [\nabla^{it,bp}_{k,i-1}(t_k),st_2]$ $\quad\{\mathrm{acc}(st_2,w_{it}) = \mathrm{desplazar}(st_1)\}\,, i \in [1,n_k]$
3. $[\nabla^{it,bp}_{k,0}(t_k),st] \longmapsto [A^{it,bp}_{k,0},st]$
4. $[A^{it,bp}_{k,i},st_1] \longmapsto [A^{it+1,it}_{k,i+1},st_2]\,[A^{it,bp}_{k,i},st_1]$ $\quad\{\mathrm{acc}(st_1,w_{it}) = \mathrm{desplazar}(st_2)\}\,, i \in [0,n_k)$
5. $[\$^{0,0},0] \longmapsto [A^{0,0}_{k,0},st]\,[\$^{0,0},0]$ $\quad\{\mathrm{acc}(0,w_0) = \mathrm{desplazar}(st)\}$

donde acc(st, w_i) es la acción determinada por el mecanismo de control estático[11] para el estado st y el símbolo terminal correspondiente a la posición i de la cadena de entrada. Estas acciones pueden ser:

- reducir(γ_k^f), reducir la regla del esqueleto independiente del contexto extraída de la cláusula γ_k, o bien,
- desplazar(st), desplazar el control al estado st.

Si nos fijamos en las condiciones impuestas a las transiciones, veremos que por la construcción del autómata LALR(1), que implementa el mecanismo de control, al aplicar una transición del tipo 1 sobre A_{k,n_k}, la pila contendrá los literales necesarios, $\nabla_{k,i}$ y $A_{k,i}$, para aplicar las pertinentes transiciones de tipo 2 hasta obtener un literal de la forma $\nabla_{k,0}$ sobre el que aplicar una transición de tipo 3.

En el caso de que todos los predicados de la cláusula implicada, γ_k, fuesen de aridad 0, el proceso equivaldría a aplicar una reducción mediante la regla independiente del contexto γ_k^f. Esto es cierto por la condición impuesta al comienzo del proceso.

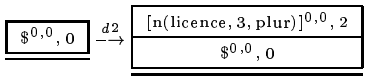
En el caso de que alguno de los predicados participantes en el proceso de reducción contenga argumentos, se ha de comprobar, además, por cada transición aplicada, que se cumplen las condiciones referentes a las substituciones tal y como se explicó en la definición de ALP.

En resumen, la idea es obviar todas estas transiciones intermedias en el proceso de reducción, y representarlas como un único paso, en el ejemplo que nos ocupa, para simplificar la exposición.

Para el caso de las acciones de desplazamiento se aplicará un razonamiento análogo. Por lo tanto se evitará representar los predicados $\nabla_{k,i}$ y los pasos intermedios de las acciones de reducción, indicando solamente las acciones que tendrían lugar sobre el esqueleto independiente del contexto.

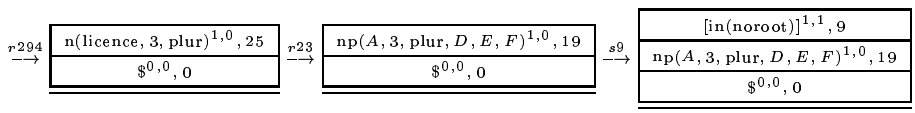
En el primer itemset el símbolo adelantado es n y sólo se pueden aplicar una acción de desplazamiento al estado 2, $d2$.

Itemset 0



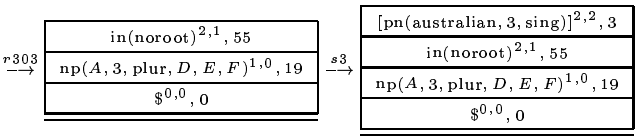
El los itemsets 2 y 3, los símbolos adelantados son *in* y *pn*, y se aplican varias acciones como puede ser reducir la cláusula 294: $r294$.

Itemset 1



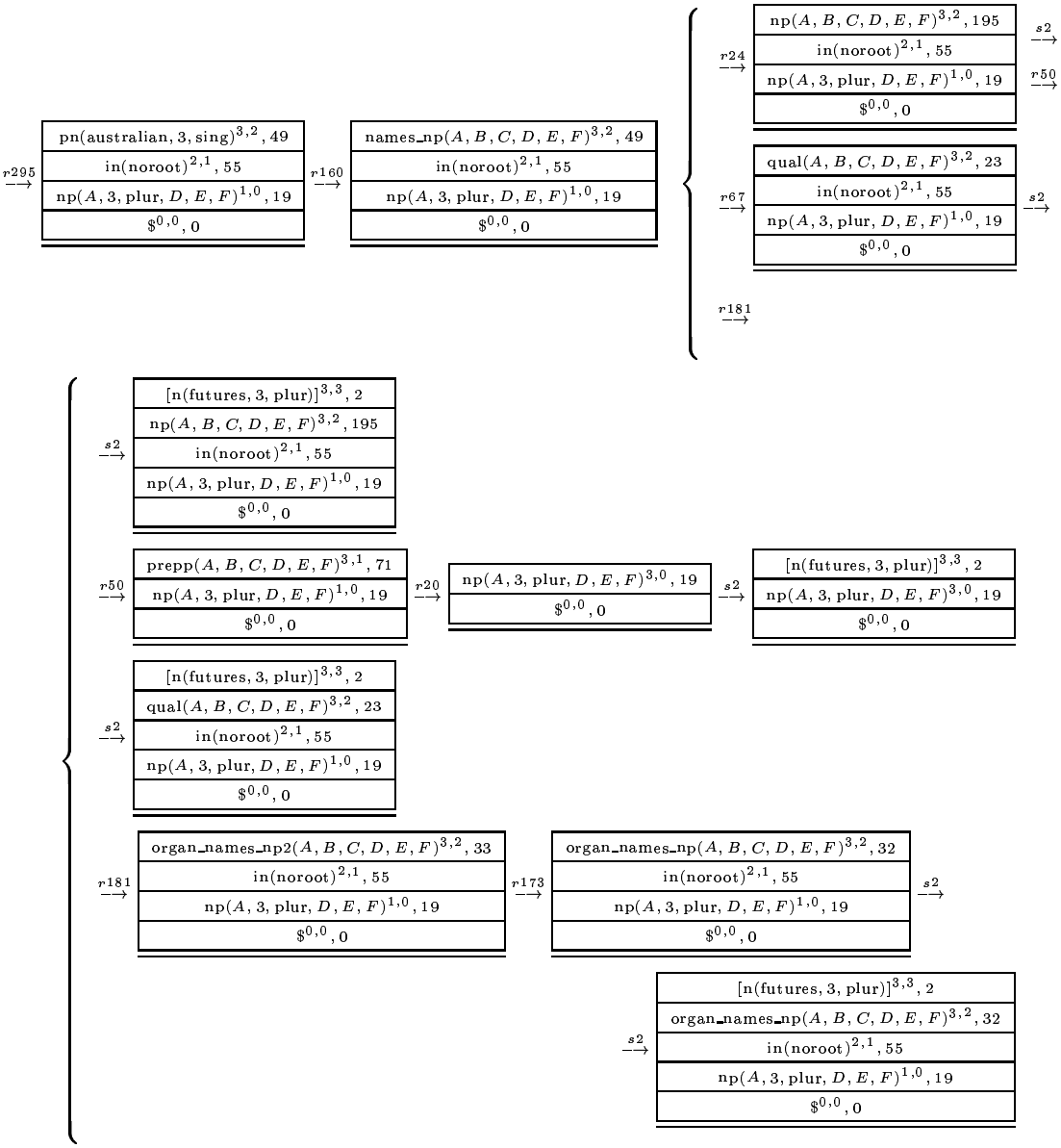
[11] En la implementación actual de ICE e ICE extendido, un autómata LALR(1).

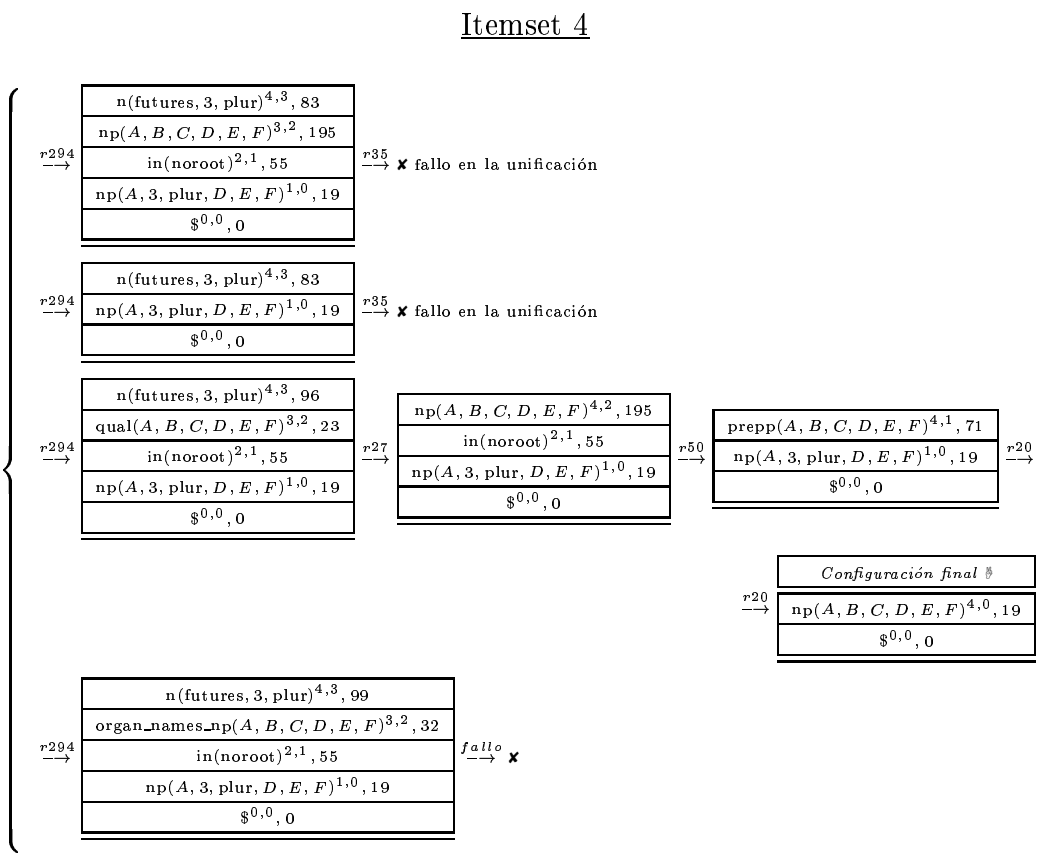
Itemset 2



En el itemset 3, ya aparece la primera ambigüedad. Si estubiesemos usando un entorno dinámico S^1, las cuatro configuraciones obtenidas al final del cálculo del itemset 3 se unirían en el mismo item $[[n(futures, 3, plur)]^{3,3}, 2]$. Lo mismo ocurre en el itemset 4 con el item $[[n(futures, 3, plur)]^{4,3}, 83]$.

Itemset 3




Itemset 4
n(futures, 3, plur)^{4,3}, 83
np(A, B, C, D, E, F)^{3,2}, 195
in(noroot)^{2,1}, 55
np(A, 3, plur, D, E, F)^{1,0}, 19
^{0,0}, 0
r294
r35
fallo en la unificación
n(futures, 3, plur)^{4,3}, 83
np(A, 3, plur, D, E, F)^{1,0}, 19
^{0,0}, 0
r294
r35
fallo en la unificación
n(futures, 3, plur)^{4,3}, 96
qual(A, B, C, D, E, F)^{3,2}, 23
in(noroot)^{2,1}, 55
np(A, 3, plur, D, E, F)^{1,0}, 19
^{0,0}, 0
r294
r27
np(A, B, C, D, E, F)^{4,2}, 195
in(noroot)^{2,1}, 55
np(A, 3, plur, D, E, F)^{1,0}, 19
^{0,0}, 0
r50
prepp(A, B, C, D, E, F)^{4,1}, 71
np(A, 3, plur, D, E, F)^{1,0}, 19
^{0,0}, 0
r20
Configuración final
r20
np(A, B, C, D, E, F)^{4,0}, 19
^{0,0}, 0
n(futures, 3, plur)^{4,3}, 99
organ_names_np(A, B, C, D, E, F)^{3,2}, 32
in(noroot)^{2,1}, 55
np(A, 3, plur, D, E, F)^{1,0}, 19
^{0,0}, 0
r294
fallo

Capítulo 5

Resultados experimentales.

En este capítulo se introduce e interpretan algunos resultados experimentales obtenidos con las herramientas descritas. En primer lugar, y puesto que ya se habló de las ventajas del mecanismo de control estático aplicado al análisis sintáctico, tanto en ICE como en ICE extendido, se mostrará mediante un sencillo ejemplo la parte negativa del mismo. En cualquier caso, los inconvenientes son mínimos, y las ventajas los superan con creces.

A continuación, se muestran algunas medidas realizadas sobre el analizador morfológico que corroboran la idea de que el mecanismo operacional elegido es sumamente compacto y eficiente. Por último, se han realizado idénticos experimentos en ICE, ICE extendido y otros sistemas desarrollados por otros autores con la finalidad de comparar diferentes estrategias de análisis sintáctico, *ascendente*, *descendente*, ... y analizar las diferencias entre usar o no entornos de programación dinámica.

5.1 Control estático vs. compartición óptima

Los resultados presentados se obtuvieron analizando la gramática de *Dyck*:

$$\begin{array}{lll} \gamma_0: & s(\text{nil}). & \\ \gamma_1: & s(s(T_1, T_2)) & \rightarrow s(T_1),\ s(,T_2). \\ \gamma_2: & s(s(X, T, X)) & \rightarrow [X],\ s(T),\ [X]. \end{array}$$

En la figura 5.1 observamos dos árboles de análisis: el primero corresponde al resultado obtenido por ICE extendido y el segundo al resultado que se obtendría en el hipotético caso de que la compartición de cálculos y estructuras fuese la máxima posible.

En el caso ideal, las flechas hacia arriba indican la compartición del nodo referenciado. En el caso real, la compartición se expresa por medio de pares *etiqueta-del-nodo*$(\#n)$, $\#n$. Para indicar porque no se alcanza la compartición máxima se ha añadido a cada nodo información acerca del estado del *item* que lo genero. Esta información se extrae directamente de la figura 5.2, a partir de las configuraciones de la pila durante el proceso de análisis.

Comparando ambos árboles, vemos que aquellos nodos que respecto al caso de máxima compartición están duplicados, han sido generados por distintos *items*, cuya única diferencia consiste en el estado en que fueron generados. La conclusión inmediata

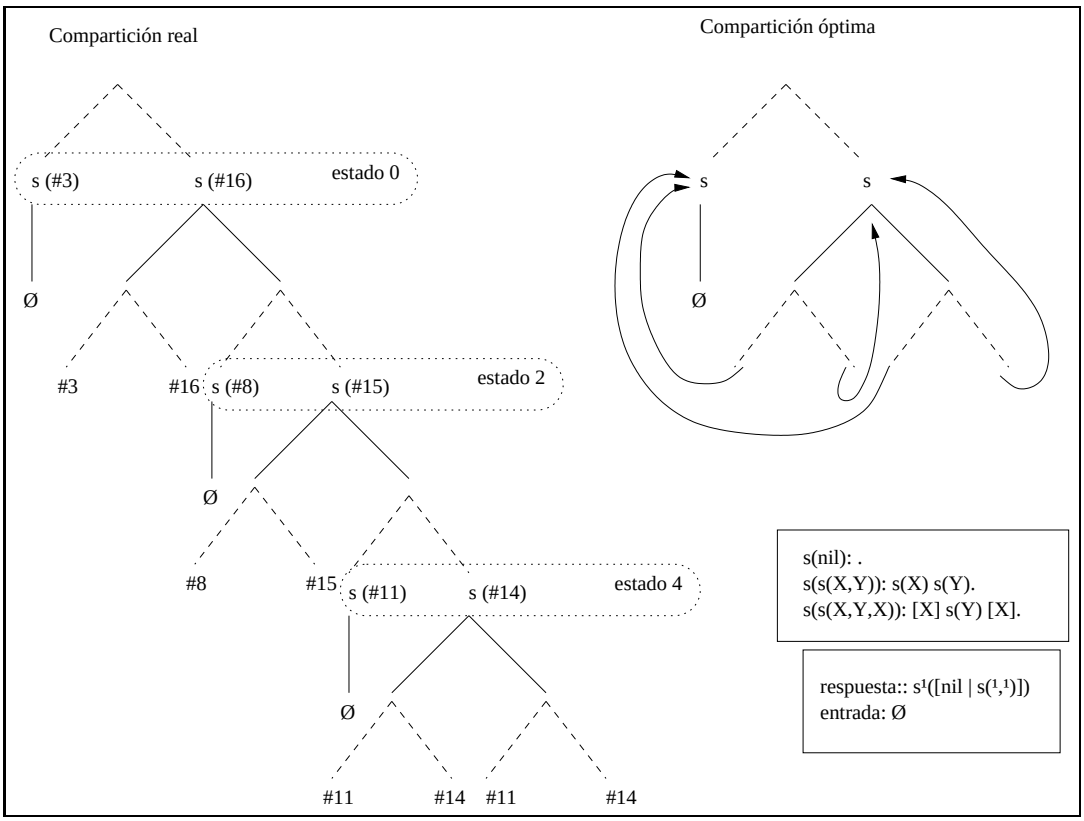


Figura 5.1: Comparación entre la compartición óptima y la real.

es que eliminando el control estático, i.e. autómata $LALR$, podemos alcanzar el mayor grado de compartición posible. Sin embargo, también resulta evidente que dicha decisión daría lugar a una pérdida de eficiencia general al prescindir el mecanismo que guía el proceso de análisis, eliminando cálculos inútiles y determinizando el proceso.

En resumen, hemos de aceptar un compromiso entre el uso de un mecanismo estático que guíe el proceso de análisis, reduciendo el problema de la explosión combinatoria debida a la ambigüedad de la gramática elegida, y una compartición óptima, aún a costa de seguir muchas ramas de análisis que *a priori* se determina que no son correctas.

5.1.1 Detalles de la implementación del analizador morfológico.

La implementación del formalismo operacional se describe con más detalle en [30]. En [91, 95] se describen así mismo facilidades de traza y depuración incorporadas al analizador.

Para la implementación se eligió la herramienta FLEX [55], debidamente modificada y adaptada para el análisis y la etiquetación del lenguaje natural [5, 30].

Actualmente la versión desarrollada incorpora más de 12.000 lemas[1] distintos, tanto para el castellano como para el gallego, y el autómata resultante consta de más de 163.354 estados (135.000 en gallego). Por termino medio, el añadir un nuevo lema

[1] Nótese que hablamos de lemas, no de palabras.

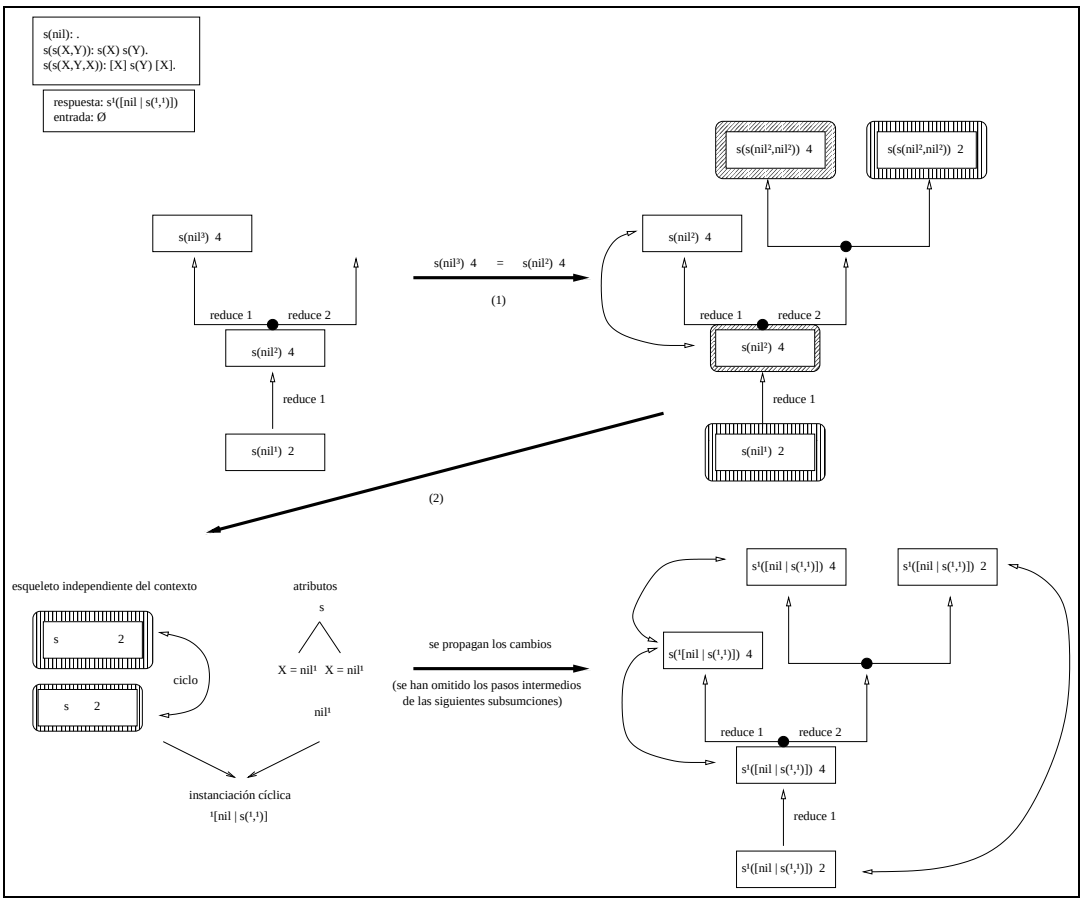


Figura 5.2: Proceso de análisis para la cadena de entrada vacía.

supone incrementar en dos el número de estados.

El fichero ejecutable resultante ocupa aproximadamente dos mega-bytes, lo que da una idea del grado de compactación alcanzado. En cuanto a la velocidad, el analizador es capaz de etiquetar 25.000 palabras en menos de siete segundos en una *ALPHAstation 500/400*, tal y como se muestra en la figura 5.3.

5.2 Comparación con otras aproximaciones

Las gramáticas basadas en la unificación han sido y son objeto de estudio dentro de la lingüística computacional a la hora de desarrollar entornos de análisis sintáctico. A pesar de ello, no es común encontrar estudios prácticos sobre las técnicas empleadas y qué aproximaciones se adaptan mejor a cada caso.

En este apartado[2], se intentará justificar la elección del entorno de programación dinámica S^1 realizada dentro de ICE e ICE extendido, siempre dentro del análisis sintáctico empleando gramáticas de cláusulas definidas.

Siguiendo la terminología empleada en [57], podemos distinguir entre analizadores

[2]Parte de los resultados han sido expuestos en [88].

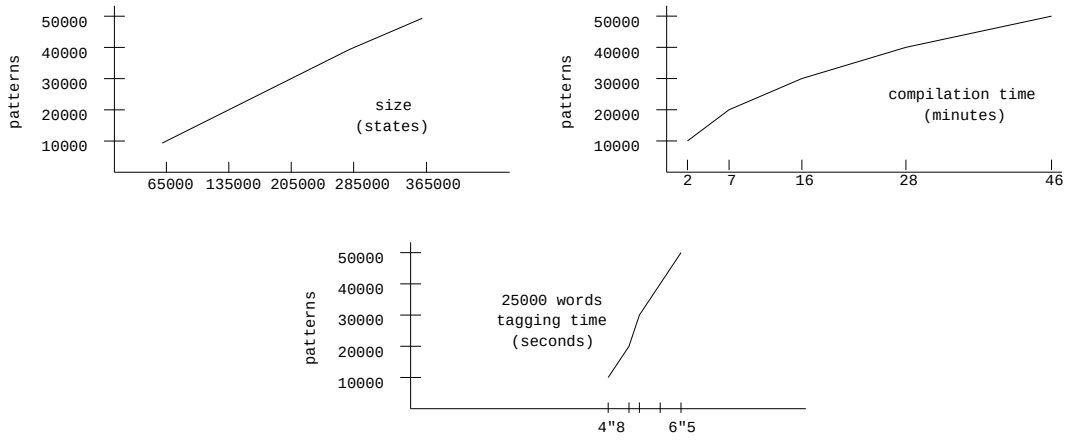


Figura 5.3: Resultados experimentales del analizador morfológico.

sintácticos inmediatos y diferidos. Estos últimos se caracterizan por producir, en una primera fase, algún tipo de esqueleto preliminar sobre el cual se filtran los análisis inválidos empleando una estructura más fina basada en la unificación.

En el caso de los analizadores sintácticos inmediatos, las restricciones se aplican directamente como parte del proceso de análisis. ICE extendido se sitúa en este grupo.

En general, los analizadores inmediatos presentan el problema de explosión de contextos sintácticos, mientras que en los diferidos, se produce un crecimiento exponencial del número de ambigüedades en la primera fase.

Los resultados que se presentan han sido obtenidos considerando las gramáticas de cláusulas definidas, y comparándolos con los resultados obtenidos en ICE extendido, para los analizadores inmediatos. En cuanto a los analizadores diferidos, puesto que el esqueleto usado normalmente es una GIC, compraremos los resultados con los obtenidos para ICE, limitándonos al análisis de la GIC que conforma el esqueleto de la gramática.

5.2.1 Algoritmos de análisis

Los algoritmos de análisis para las gramáticas independientes del contexto, se pueden entender como un caso particular de las gramáticas de cláusulas definidas, donde todos los predicados tienen aridad 0 y la operación de unificación es en realidad la función identidad:

Transición	Análisis diferido	Análisis inmediato
$B \longmapsto C(A) = E$	$A = B,\ E = C$	$E = C\sigma,\ \sigma = \mathrm{mgu}(A, B)$
$BD \longmapsto C(AA') = E$	$A = B,\ A' = D,\ E = C$	$E = C\sigma,\ \sigma = \mathrm{mgu}(AA', BB')$
$B \longmapsto CB(A) = EB$	$A = B,\ E = C$	$E = C\sigma,\ \sigma = \mathrm{mgu}(A, B)$

Para describir los algoritmos empleados, podemos recuperar los esquemas de compilación introducidos en el capítulo 4, y recopilados en la figura 5.4.

Para los tres esquemas elegidos, hemos empleado los siguientes sistemas, en el caso de los analizadores diferido, el entorno AGFL [51], el algoritmo de Earley clásico [26], y el sistema *GA*LENA, es decir ICE. Para los analizadores inmediato, la opción ha sido,

Esquema ascendente	
$\$ \longmapsto \nabla_{0,0}\ \$$	$\$ \longmapsto \nabla_{0,0}(t_0)\ \$$
$\nabla_{k,i} \longmapsto A_{k,i+1}\ \nabla_{k,i}$	$\nabla_{k,i}(t_k) \longmapsto A_{k,i+1}\ \nabla_{k,i}(t_k)$
$A_{k,0} \longmapsto \nabla_{k,0}$	$A_{k,0} \longmapsto \nabla_{k,0}(t_k)$
$\nabla_{k,n_k}\ \nabla_{k',i} \longmapsto \nabla_{k',i+1}$	$\nabla_{k,n_k}(t_k)\ \nabla_{k',i}(t'_k) \longmapsto \nabla_{k',i+1}(t'_k)$

Resolución de Earley	
$\$^{0,0} \longmapsto \nabla_{0,0}^{0,0}\ \$^{0,0}$	$\$^{0,0} \longmapsto \nabla_{0,0}^{0,0}(t_0)\ \$^{0,0}$
$A'^{it,it}_{k,0} \longmapsto \nabla^{it,it}_{k,0}\ A'^{it,it}_{k,0}$	$A'^{it,it}_{k,0} \longmapsto \nabla^{it,it}_{k,0}(t_k)\ A'^{it,it}_{k,0}$
$\nabla^{it,bp}_{k,i} \longmapsto A'^{it,it}_{k,i+1} \nabla^{it,bp}_{k,i}$	$\nabla^{it,bp}_{k,i}(t_k) \longmapsto A'^{it,it}_{k,i+1} \nabla^{it,bp}_{k,i}(t_k)$
$\nabla^{it,bp}_{k,n_k}\ A'^{bp,bp}_{k,0} \longmapsto A''^{it,bp}_{k,0}$	$\nabla^{it,bp}_{k,n_k}(t_k)\ A'^{bp,bp}_{k,0} \longmapsto A''^{it,bp}_{k,0}$
$A''^{it,bp}_{k,i+1}\ \nabla^{bp,r}_{k,i} \longmapsto \nabla^{it,r}_{k,i+1}$	$A''^{it,bp}_{k,i+1}\ \nabla^{bp,r}_{k,i}(t_k) \longmapsto \nabla^{it,r}_{k,i+1}(t_k)$

ICE	
$[A^{it,bp}_{k,n_k}, st] \longmapsto [\nabla^{it,it}_{k,n_k}, st]\ [A^{it,bp}_{k,n_k}, st]\ \{1\}$	$[A^{it,bp}_{k,n_k}, st] \longmapsto [\nabla^{it,it}_{k,n_k}(t_k), st]\ [A^{it,bp}_{k,n_k}, st]\ \{1\}$
$[\nabla^{it,r}_{k,i}, st_1]\ [A^{r,bp}_{k,i}, st_1] \longmapsto [\nabla^{it,bp}_{k,i-1}, st_2]\ \{2\}$	$[\nabla^{it,r}_{k,i}(t_k), st_1]\ [A^{r,bp}_{k,i}, st_1] \longmapsto [\nabla^{it,bp}_{k,i-1}(t_k), st_2]\ \{2\}$
$[\nabla^{it,bp}_{k,0}, st_1] \longmapsto [A^{it,bp}_{k,0}, st_2]\ \{3\}$	$[\nabla^{it,bp}_{k,0}(t_k), st_1] \longmapsto [A^{it,bp}_{k,0}, st_2]\ \{3\}$
$[A^{it,bp}_{k,i}, st_1] \longmapsto [A^{it+1,it}_{k,i+1}, st_2]\ [A^{it,bp}_{k,i}, st_1]\ \{4\}$	$[A^{it,bp}_{k,i}, st_1] \longmapsto [A^{it+1,it}_{k,i+1}, st_2]\ [A^{it,bp}_{k,i}, st_1]\ \{4\}$
$[A^{it,bp}_{k,i}, st_1] \longmapsto [A^{it+1,it}_{k,0}, st_2]\ [A^{it,bp}_{k,i}, st_1]\ \{5\}$	$[A^{it,bp}_{k,i}, st_1] \longmapsto [A^{it+1,it}_{k,0}, st_2]\ [A^{it,bp}_{k,i}, st_1]\ \{5\}$
$[\$^{0,0}, 0] \longmapsto [A^{0,0}_{0,0}, st]\ [\$^{0,0}, 0]\ \{6\}$	$[\$^{0,0}, 0] \longmapsto [A^{0,0}_{0,0}, st]\ [\$^{0,0}, 0]\ \{6\}$
	$1 \equiv \{\text{action}(st, \text{token}_{it}) = \text{reduce}(\gamma_k)\}$
	$2 \equiv \{\text{action}(st_2, \text{token}_{it}) = \text{shift}(st_1)\},\ i \in [1, n_k]$
	$3 \equiv \left\{\text{goto}(st_1, A_{k,0}) = st_2\right\}$
	$4 \equiv \left\{\text{action}(st_1, A_{k,i+1}) = \text{shift}(st_2)\},\ i \in [0, n_k)\right\}$
	$5 \equiv \left\{\text{action}(st_1, A_{k,0}) = \text{shift}(st_2)\right\}$
	$6 \equiv \{\text{action}(0, \text{token}_0) = \text{shift}(st)\}$
Análisis diferido	Análisis inmediato

Figura 5.4: Esquemas de compilación.

AGFL[3], una implementación del esquema de *Earley deduction* [57], y, finalmente, otra vez *GA*LENA, e ICE extendido.

Hemos elegido ICE e ICE extendido como el representante más eficiente de la familia de analizadores sintácticos de tipo *LR*, entre los que destacan SDF [31] y GLR [61], ambos basados en el algoritmo de Tomita [73].

5.2.2 El test realizado

Para realizar los tests empleamos la gramática para la estructura sintagmática del castellano, que hemos introducido en capítulos anteriores:

[3]Realmente AGFL [51] se basa en el concepto de gramática de afijos, que en el de gramática de unificación. Sin embargo, su comportamiento se puede asimilar al de un analizador de GCD dónde no se permiten los símbolos de función. De esta forma podemos tener en cuenta uno de los sistemas más conocidos.

```
s(esp(Arbol))                 →  frase(Arbol).
frase(fr(Arbol1,Arbol2))      →  fn(Arbol1,Num), fv(Arbol2,Num).
frase(fr(Arbol1,Arbol2))      →  frase(Arbol1), fp(Arbol2).
fn(fn(n(P)),Num)              →  nombre(P:Palabra,Num:Numero).
fn(fn(pr(P)),Num)             →  pronombre(P:Palabra,Num:Numero).
fn(fn(det(P1),n(P2)),Num))    →  determinante(P1:Palabra,Num:Numero),
                                 nombre(P2:Palabra,Num:Numero).
fn(fn(Arbol1,Arbol2),Num)     →  fn(Arbol1,Num),fp(Arbol2).
fp(fp(prep(P),Arbol)          →  preposicion(P:Palabra),
                                 fn(Arbol,Num).
fv(fv(verbo(P),Arbol),Num)    →  verbo(P:Palabra,Num:Numero),
                                 fn(Arbol,Num2).
```

Con esta gramática se han analizado el conjunto de cadenas de entrada:

$$\textit{Yo veo un padre}\,(\textit{de un hijo de un padre})^i, \quad i \geq 0$$

Evidentemente i es el número de veces que se repite "*de un hijo de un padre*". El número de análisis válidos, *c.f.* el número de ambigüedades, crece exponencialmente con i.

$$C_0 = C_1 = 1 \qquad y \qquad C_i = \left(\begin{array}{c} 2i \\ i \end{array} \right) \frac{1}{i+1}, \; si \; i > 1$$

El número de items empleados para realizar el análisis de la cadena descrita, por medio de las diferentes estrategias, para diferentes valores de i, se muestra el las tablas 5.5 y 5.6.

5.2.3 Comparación de diferentes estrategias

En la parte izquierda de la figura 5.7 se muestra el número de items generados durante el análisis diferido empleando estrategias ascendentes, *Galena*, descendentes, *DyALog* y el algoritmo de *Earley* clásico. En este caso, ICE se muestra ligeramente inferior debido principalmente a:

- El fenómeno distorsionador sobre la compartición de cálculos que introduce el control estático.
- En la gramática empleada, las predicciones realizadas durante un análisis descendente son acertadas en un elevado % de las ocasiones.

La parte derecha de la figura, muestra las mismas medidas en el caso de un análisis inmediato. En esta ocasión, los mecanismo de predicción de los analizadores han de tener en cuenta los argumentos de los predicados, puesto que se trata de un análisis inmediato. La consecuencia es que se realizan más predicciones de las necesarias y, por lo tanto, el número de items generados crece, quedando en clara desventaja respecto a ICE extendido. En ambos casos, los mecanismos basados en la resolución de *Earley* parecen situarse en una posición intermedia.

i	ICE	ICE S^T	DyALog	Earley	Earley S^T	AGFL
0	11	11	24	28	152	60
1	36	42	55	67	1152	621
2	78	101	98	118	3820	6885
3	136	188	153	181	9151	86599
4	210	303	220	256	18440	1166945
5	300	446	299	343	33295	16298861
6	406	617	390	442	55612	23284250
7	528	816	493	553	87575	3380144066
8	666	1043	608	676	131656	
9	820	1298	735	811	190615	
10	990	1581	874	958	267500	
11	1176	1892	1025	1117	365647	
12	1378	2231	1188	1288	488680	
13	1596	2598	1363	1471	640511	
14	1830	2993	1550	1666	825340	
15	2080	3416	1749	1873	1047655	
16	2346	3867	1960	2092	1312232	
17	2628	4346	2183	2323	1624135	
18	2926	4853	2418	2566	1988716	
19	3240	5388	2665	2821	2411615	
20	3570	5951	2924	3088	2898760	
21	3916	6542	3195	3367	3456367	

Figura 5.5: Número de items generados para el análisis diferido

Para poder hacerse una idea de la relación existente entre las dos partes de la figura 5.7, en la figura 5.8 se compara el número de items necesarios para realizar un análisis diferido o inmediato empleando una estrategia basada en la resolución de Earley.

5.2.4 Comparación de entornos dinámicos

En la parte izquierda de la figura 5.11 tenemos el número de items generados en ICE e ICE extendido. Recordemos que ambos trabajan en el entorno dinámico S^1, y que usamos ICE en el caso del análisis diferido e ICE extendido en el caso de análisis inmediato. Además, se muestra una estimación del número de items que se necesitarían, en ambos casos, para realizar el análisis empleando un entorno dinámico S^T.

Por su lado, la parte derecha de la figura muestra los resultados obtenidos por DyALog y Agfl, que trabajan en los entornos dinámicos S^2 y S^T, respectivamente.

Como era de esperar el número de items que se necesita generar durante el proceso de análisis en un entorno S^T es netamente superior a S^1 y S^2. comparan S^T,S^2 y S^1 en el caso de ICE e ICE extendido.

i	*GA*LENA	*GA*LENA S^T	DyALog	Earley deduction
0	11	11	28	26
1	38	45	71	78
2	82	108	248	228
3	142	199	1740	580
4	218	318	17849	1270
5	310	465	2466	
6	418	640	4368	
7	542	843		
8	682	1074		
9	838	1333		
10	1010	1620		
11	1198	1935		
12	1402	2278		
13	1622	2649		
14	1858	3048		
15	2110	3475		
16	2378	3930		
17	2662	4413		

Figura 5.6: Número de items generados para el análisis inmediato

5.2.5 Complejidad

En último lugar, se ha comparado cada sistema *contra* sí mismo, calculando, por cada incremento de la variable i el tanto por ciento de items nuevos que se generan. Los resultados los podemos ver en las figuras 5.9, 5.10, y 5.12.

Como se puede observar en las figuras, en los sistemas que hacen uso de las técnicas de programación dinámica, al incrementar la complejidad de la cadena de entrada, el aumento en el número de items generados durante el análisis es, proporcionalmente, inferior. Este es el resultado que se podía esperar *a priori*, puesto que, aunque al aumentar i se aumenta el número de análisis válidos de la cadena de entrada, éstos conservan una parte común. Debido al uso de las técnicas de programación dinámica, esta parte común no ha de ser calculada varias veces.

Por su parte, los sistemas en los que no se usa programación dinámica, cada nueva alternativa en el proceso de análisis supone repetir parte o la totalidad de los cálculos ya realizados, lo que lleva a una crecimiento exponencial del número de items generados.

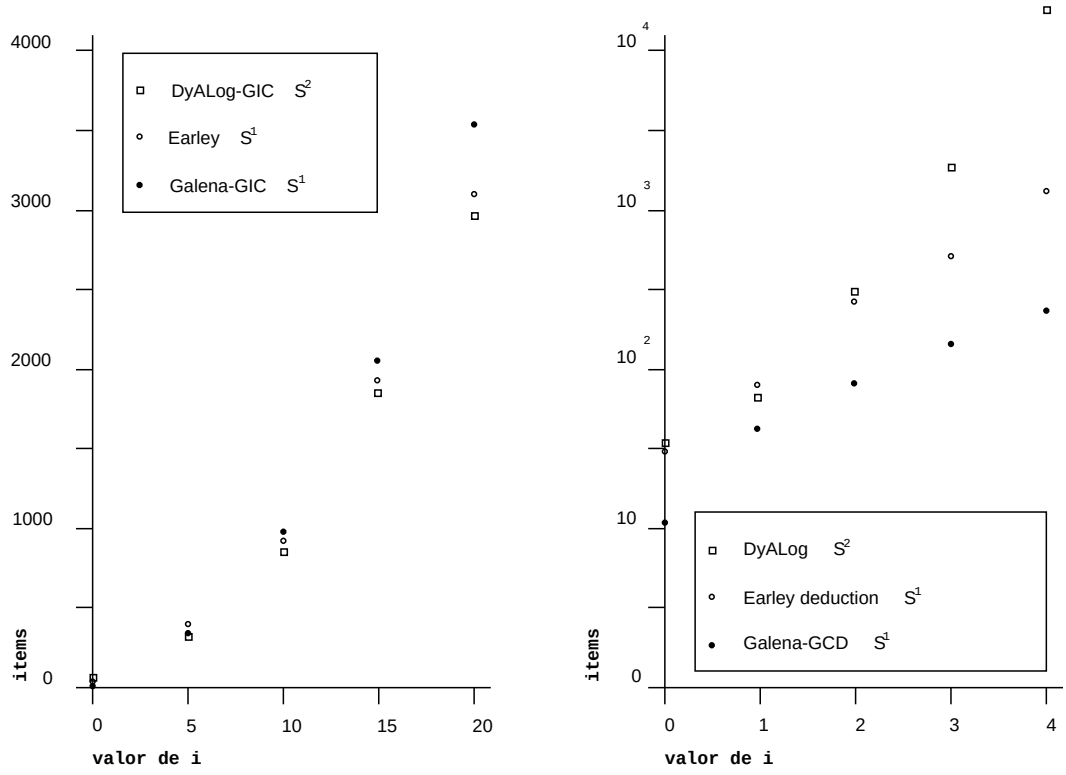


Figura 5.7: Diferentes estrategias para GICs y GCDs.

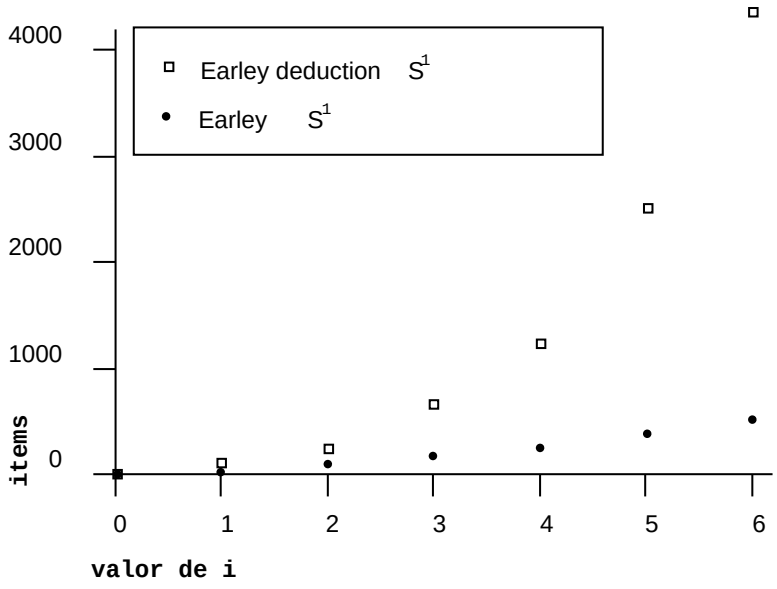


Figura 5.8: Comparación entre análisis diferido y inmediato.

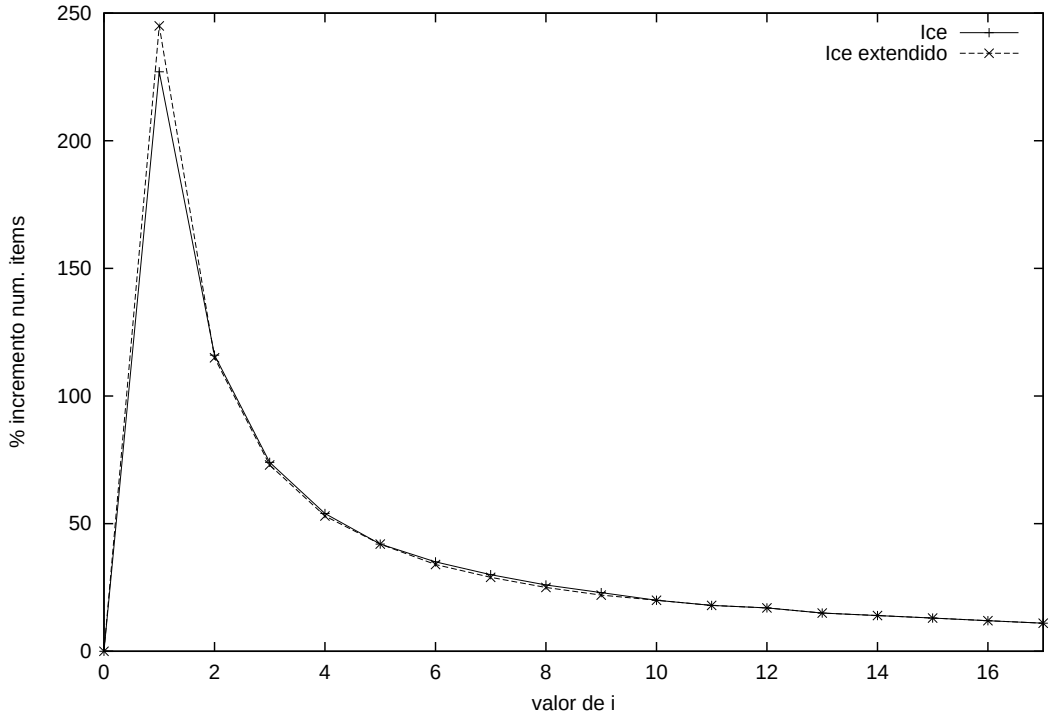


Figura 5.9: Crecimiento del número de items en ICE e ICE extendido.

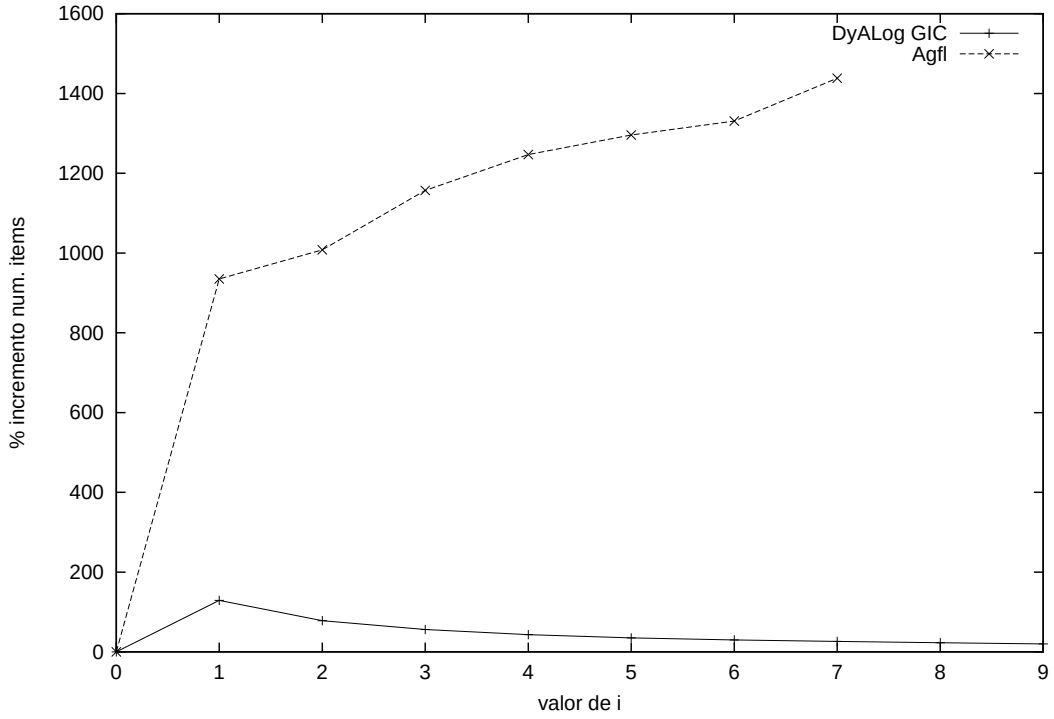


Figura 5.10: Crecimiento del número de items en DyALog y AGFL.

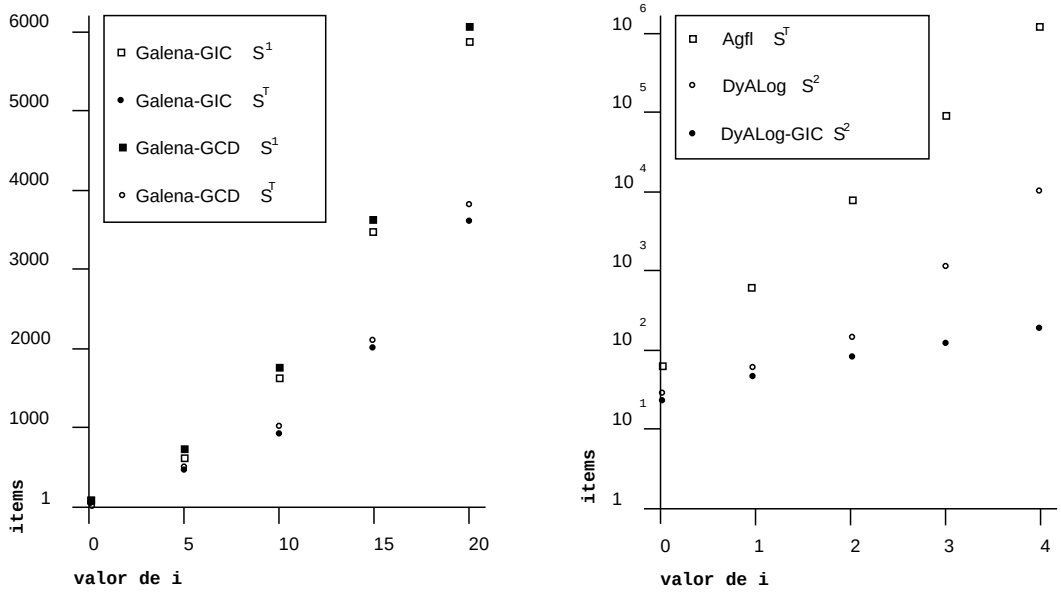


Figura 5.11: Diferentes entornos dinámicos.

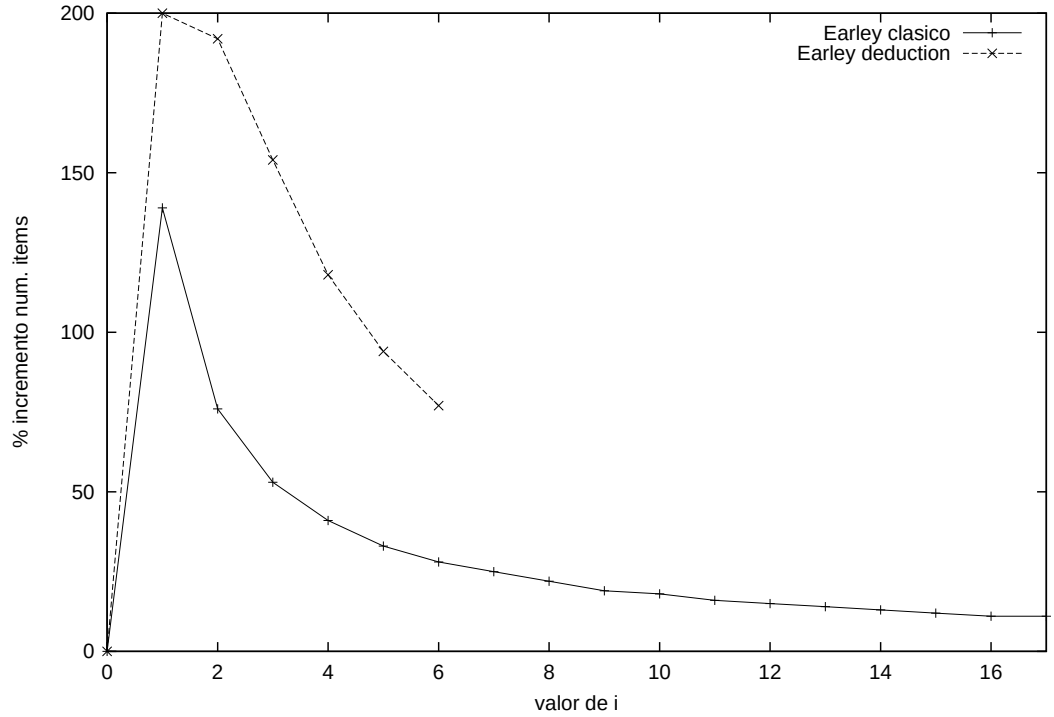


Figura 5.12: Crecimiento del número de items en *Earley*.

Capítulo 6

Conclusiones y trabajo futuro.

A lo largo del presente trabajo se han descrito los esfuerzos realizados dentro del sistema *GA*LENA, en el desarrollo tanto de analizadores morfológicos como analizadores sintácticos. Estos esfuerzos han estado dirigidos a la creación de herramientas eficientes, manteniendo en todo momento el poder descriptivo de los formalismos empleados.

6.1 Eficiencia

En lo que respecta a los analizadores morfológicos la eficacia se garantiza por:

- El uso de un formalismo operacional basado en autómatas finitos.
- La limitación del no determinismo de los autómatas resultantes, mediante la compartición de caminos lingüísticos.

Los analizadores sintácticos, por su parte, presentan las siguientes características:

- Compartición de cálculos y estructuras, dentro de un entorno de programación dinámica.
- Gestión simplificada de la tabla de objetos, mediante una sincronización a nivel de itemset, al estilo de los algoritmos basados en la deducción de *Earley*.
- Eliminación de cálculos inútiles por medio de un mecanismo de control estático.
- Eliminación de los problemas de no terminación propios de las estrategias descendentes, al implementar una estrategia ascendente.

6.2 Otras características

Por razones de espacio, no se han tratado los problemas de no terminación en el análisis sintáctico debidos a la aparición de estructuras cíclicas. Este problema, presente en muchos otros sistemas, también es tratado en el sistema *GA*LENA [92, 84, 85, 87]. En ICE, tanto la detección como la representación de estas estructuras se realiza satisfactoriamente, tal y como se explica en [92].

ICE extendido se basa en la detección de ciclos de ICE sobre el esqueleto independiente del contexto para detectar este tipo de estructuras sobre los argumentos de los predicados. No obstante la detección se realiza satisfactoriamente, [84, 85, 87], la representación es una característica que resta por implementar y, por lo tanto, puede ser incluida en el apartado de trabajo futuro.

Otra característica que implementa ICE y que no se ha introducido en este trabajo, es la capacidad de realizar un *análisis incremental* [92], evitando repetir el análisis de las partes no afectadas cuando se modifica la cadena de entrada. Aunque los resultados obtenidos en ICE son trasplantables de manera inmediata al análisis diferido[1], no resulta fácil decidir si la incrementalidad puede ser extendida con éxito al análisis inmediato de las GCDs. Esta labor queda, igualmente, como trabajo futuro.

6.3 Futuros desarrollos

En primer lugar, y a pesar de la compartición de las estructuras de datos en memoria, la complejidad del análisis del lenguaje natural hace necesaria la inclusión de técnicas avanzadas de *recuperación de memoria*[2]. En el caso del analizador sintáctico, puesto que su comportamiento es muy cercano al de un autómata de pila, lo más indicado parece ser un mecanismo de recuperación de memoria *generacional* [6].

En cuanto al analizador morfológico, como ya se mostró en las gráficas del capítulo 5, a medida que crece el número de palabras que es capaz de reconocer, crece el tamaño del autómata resultante, y el proceso de compilación se vuelve más lento. Para evitar llegar a tiempos inaceptables, se ha comenzado a trabajar en un nuevo formalismo operacional mixto. Por una parte se mantienen los mini autómatas asociados a las desinencias de las palabras y por otro lado un árbol de búsqueda se encarga de reconocer las raíces. Al contrario que los autómatas empleados hasta el momento, los árboles binarios permiten una compilación incremental, reduciendo drásticamente el tiempo asociado a la inclusión de nuevas palabras. Por contra, se espera que la degradación de la eficacia del sistema final no sea significante. Los primeros resultados experimentales confirman estas expectativas.

Otro aspecto de interés que no ha sido tratado en el sistema *GA*LENA, es la *corrección automática de errores*, tanto a nivel morfológico, como sintáctico.

Una posible mejora adicional del analizador sintáctico consiste en el desarrollo de un mecanismo de control estático más minucioso. Algunas opciones son:

- Ampliando las nociones de *primero*, *siguiente* y *símbolo adelantado* de las GICs a las GCDs, tal y como ha sido descrito en [92].
- Aumentando el tamaño de la ventana, lo que nos lleva a generalizar el tratamiento LALR(k) a los ALPs.
- Introduciendo técnicas de control estático más complejas.

Estas estrategias no son exclusivas, sino complementarias. Se pueden combinar a voluntad, buscando el mejor resultado. En cualquier caso, la elección final debe situarse

[1] Basta con aplicar la incrementalidad el análisis del esqueleto de la gramática.
[2] Del inglés *garbage collector*.

en un punto de equilibrio entre los costes temporales y espaciales. En este punto, disponemos de algunas referencias coincidentes tanto en el campo de los lenguajes algebraicos como en el de la lógica de Horn [92, 97]:

- Los algoritmos ascendentes puros muestran mejores resultados desde el punto de vista de la compartición de cálculos y estructuras.
- Un nivel de predicción elevado en un algoritmo ascendente no garantiza un mejor comportamiento computacional. Si bien se reduce el espacio de búsqueda, también se favorece el fenómeno de división de estados [92], con una incidencia desfavorable en la calidad de compartición. Sería, por ejemplo, el caso de las técnicas de control inspiradas en el modelo LR.
- Las técnicas situadas entre los modelos ascendente puro y Earley [26] muestran los mejores resultados prácticos.

Por último, una extensión que resulta necesaria, es la inclusión de un modulo de *análisis semántico*. Esta parte será abordada dentro del proyecto ERIAL[3], cuyo comienzo es inmediato.

6.3.1 ERIAL

El proyecto ERIAL se cimenta sobre tres ejes principales:

- *Técnicas fundamentales*: principalmente técnicas de análisis semántico, sintáctico y morfológico.
- *Análisis de documentos*: desarrollo de bibliotecas electrónicas y análisis del contenido lingüístico de los textos.
- *Tecnologías de soporte*: tecnologías multimedia, en especial CD-ROM, e Internet.

En lo que atañe al análisis de documentos, las aplicaciones en PLN conciernen, generalmente, a dominios restringidos. En efecto, la complejidad del problema en el estado actual del arte, no permite un tratamiento práctico amplio. Por esto, la adquisición y evocación del universo del discurso son tareas indispensables. Estas tareas, normalmente, se conocen como la *modelización del dominio*. En este contexto, se han de considerar dos fases:

1. Automatización, en la medida de lo posible, de la extracción del universo del discurso a partir de conjuntos de enunciados. Como primer paso, es necesario poner en marcha técnicas lingüísticas de análisis distributivo.
2. Desarrollo de entornos en los que se apliquen las técnicas descritas.

   El primer dominio de aplicación tradicional son las interfaces en lenguaje natural para bases de datos, sobre todo en lo que se refiere a la captación del universo del discurso de los usuarios. No obstante, nuestro objetivo genérico alcanza también el tratamiento de hipertextos, documentos estructurados y bibliotecas electrónicas.

[3]Extracción y Recuperación de la Información Aplicando conocimiento Lingüístico.

Pasamos ahora a examinar, más en detalle, lo referente a la indexación semántica y jerárquica de documentos. El análisis y filtrado de documentos aparece hoy claramente como un objetivo mayor de la actividad industrial. Ello se debe en buena parte al desarrollo de Internet, que ofrece acceso a una masa cada vez mayor de información, a la vez que dificulta la identificación y selección de la información sobre la que se centra el interés del usuario. En este punto, el trabajo a desarrollar en ERIAL pasa por el desarrollo e implementación, sobre la base de un análisis morfológico y sintáctico, de herramientas capaces de extraer información semántica relevante, y complementar los mecanismos de deducción propios del análisis semántico.

En esta línea, gran parte de la información semántica puede ser recuperada en el momento del análisis morfológico y aprovechada para simplificar tanto el análisis sintáctico como de análisis semántico propiamente dicho. Una vez superada esta primera fase, mediante técnicas de análisis sintáctico superficial[4], podemos asociar a un documento dado lo que denominamos *objetos de discurso* jerarquizados según la estructura del documento, de los más generales a los más específicos. De este modo es permite un filtrado preciso de la información en la fase final que, esperamos, nos permita implementar un sistema eficaz de filtrado y búsqueda de documentos.

[4] *Shallow parsing* en terminología anglosajona.

Apéndice A

Análisis LR.

Los analizadores LR son capaces de reconocer un subconjunto de las GICs llamadas LR(k) construyendo la derivación más a la derecha. De ahí la R (*reverse*) en el nombre. La L (*left to right*) indica que la entrada se explora de izquierda a derecha.

Intuitivamente, el significado de la k es el siguiente. Si hemos reconocido la cadena de entrada $a_1 \dots a_n$ hasta a_i y podemos aplicar varias reglas de producción para continuar el análisis, examinando los siguientes k símbolos de entrada, $a_{i+1} \dots a_{i+k+1}$, podemos determinar cual de ellas es la correcta.

A.1 Análisis sintáctico LR básico

Un analizador sintáctico LR es del tipo desplazamiento-reducción, desarrollado originalmente para analizar lenguajes de programación, [1, 43]. En [74] se discuten varios aspectos relacionados con el análisis sintáctico del lenguaje natural.

Un analizador sintáctico LR es, básicamente, un autómata de pila, que incorpora un mecanismo estático de predicción. Tal y como se muestra en la figura A.1, consta de una pila, un conjunto finito de estados internos y un *puntero* que recorre la cadena de entrada de izquierda a derecha, avanzando un símbolo de cada vez. La pila se usa de una forma particular: los elementos de la pila son sucesiones de un símbolo de la gramática, y de un estado interno. El estado actual no es más que el estado en la cima de la pila. Sin embargo, la característica diferenciadora de un analizador sintáctico LR es la forma de las relaciones de transición, las tablas *acción* e *ir_a*.

La tabla *ir_a* asigna a cada par estado/símbolo (S, X) el siguiente estado, es decir, a qué estado debe transitar el autómata si se encuentra en un estado S y reconoce el símbolo X. Por su parte, la tabla *acción* hace corresponder a cada par estado/símbolo de anticipación[1] (S, Sim), una de las cuatro acciones básicas:

1. $aceptar(S, Sim)$: La frase de entrada se ha reconocido con éxito y se para el proceso de análisis.

2. $error(S, Sim)$: El análisis no es válido.

[1] El símbolo de anticipación no es más que el siguiente símbolo en la frase de entrada, es decir, el señalado por el puntero de entrada.

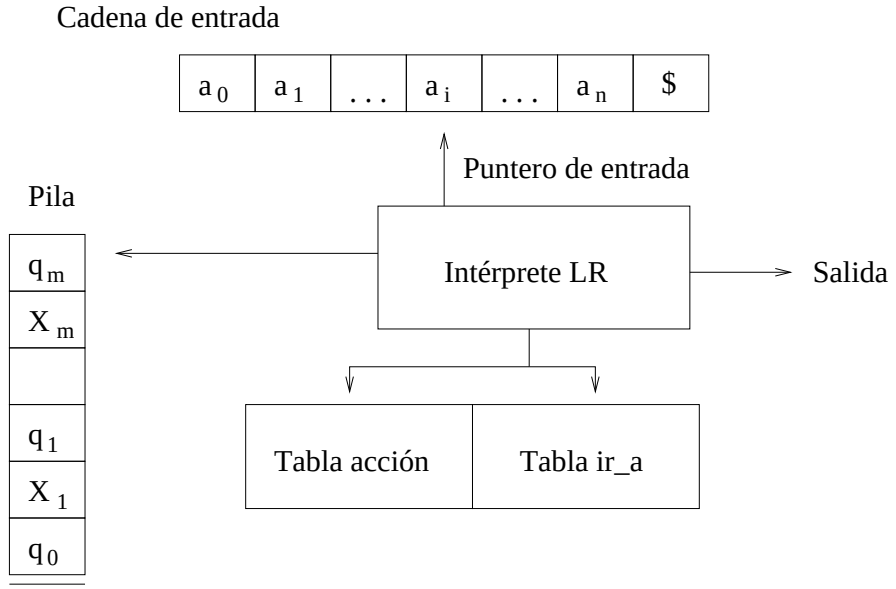


Figura A.1: Diagrama de un analizador sintáctico LR.

3. *desplazamiento*(S, Sim, S_2): Se avanza el puntero de entrada, reconociendo Sim, se introduce en la pila y S_2 también se introduce en la pila, pasando a ser el estado actual.

4. *reducción*(S, Sim, R): Por cada símbolo de la parte derecha de la regla R se extrae de la pila un par estado/símbolo, se examina el estado S_1 de la cima, y se añaden el símbolo LI[2] y el estado S_2 determinado por $ir\_a(S_1, LI, S_2)$, convirtiéndose éste en el estado actual.

Por el contrario, si a cada paso se pueden realizar varias de las acciones descritas, obtenemos un análisis no determinista, usualmente conocido como *análisis LR generalizado*.

La mezcla de prefijos se lleva a cabo en cada estado interno, correspondiente a un conjunto de reglas gramaticales parcialmente procesadas llamadas *items con punto*, donde las reglas contienen un punto que marca la posición actual del análisis. Por ejemplo, si tenemos las siguientes reglas:

```
FV → Verb
FV → Verb FN
FV → Verb FN FN
```

tendremos un estado conteniendo:

```
FV → Verb ·
FV → Verb · FN
FV → Verb · FN FN
```

Este estado corresponde al momento del análisis en que hemos reconocido un verbo `Verb`. A partir de él se ha de determinar cual de las tres reglas se debe aplicar.

[2]Lado Izquierdo, el símbolo en la parte izquierda de la regla de producción.

A.1.1 Ejemplo de análisis LR

Para desarrollar el siguiente ejemplo emplearemos la siguiente gramática de las estructuras sintagmáticas del Castellano:

(1)	`S` $\rightarrow$ `FN FV`	(6)	`FN` $\rightarrow$ `Det Sust`
(2)	`FV` $\rightarrow$ `Verb`	(7)	`FN` $\rightarrow$ `Pron`
(3)	`FV` $\rightarrow$ `Verb FN`	(8)	`FN` $\rightarrow$ `FN FP`
(4)	`FV` $\rightarrow$ `Verb FN FN`	(9)	`FP` $\rightarrow$ `Prep FN`
(5)	`FV` $\rightarrow$ `FV FP`		

A partir de la gramática se generan los estados internos de la figura A.3. Los cuales, a su vez, dan lugar a la tabla de la figura A.5. Las entradas del tipo $s2$ en la tabla de acciones se deben interpretar como *desplazamiento* al estado 2. Las acciones del tipo $r7$, como *reducir* por la regla 7. La entrada `acc` indica la acción *aceptar* y las entradas vacías *error*. Por último, el símbolo `eos` indica el fin de la frase de entrada. A destacar que en las entrada para los estados 11, 12 y 13 con el símbolo de anticipación `Prep` existen dos posibilidades. Esta ambigüedad se conoce con el nombre de *conflicto reducción-desplazamiento*. Ambas han de ser intentadas si queremos obtener todos los análisis posibles de la frase de entrada, o por el contrario diremos que no es posible obtener un analizador determinista.

Un ejemplo del proceso de análisis para la frase de entrada "`El alumno lee un libro`" es el que sigue:

Acción	Pila	Frase de entrada
Iniciar	[0]	↑`El alumno lee un libro`

Inicialmente se introduce el estado 0 en la pila. El símbolo ↑ representa el puntero de entrada. La siguiente acción, en el estado 0 y con el símbolo de anticipación `Det` (`El`) es *desplazamiento* al estado 2.

Acción	Pila	Frase de entrada
s2	[0,Det,2]	`El` ↑ `alumno lee un libro`

A partir de aquí la explicación de las siguientes acciones es similar.

Acción	Pila	Frase de entrada
s10	[0,Det,2,Sust,10]	`El alumno` ↑ `lee un libro`
r6	[0,FN,1]	`El alumno` ↑ `lee un libro`
s6	[0,FN,1,Verb,6]	`El alumno lee` ↑ `un libro`
s2	[0,FN,1,Verb,6,Det,2]	`El alumno lee un` ↑ `libro`
s10	[0,FN,1,Verb,6,Det,2,Sust,10]	`El alumno lee un libro` ↑
r6	[0,FN,1,Verb,6,FN,11]	`El alumno lee un libro` ↑
r3	[0,FN,1,FV,5]	`El alumno lee un libro` ↑
r1	[0,S,4]	`El alumno lee un libro` ↑
acc	[0,S,4]	`El alumno lee un libro` ↑

Estado	*Acción*						*Ir-a*			
	Det	*Sust*	*Prep*	*Pron*	*Verb*	*eos*	*FN*	*FP*	*S*	*FV*
0	s3			s4			2		1	
1						acc				
2			s8		s7			6		5
3		s9								
4			r7			r7				
5			s11			r1		10		
6			r8			r8				
7	s13			s14		r2	12			
8	s3			s4			15			
9			r6			r6				
10						r5				
11	s3			s4			16			
12	s3		s19	s4		r3	17	18		
13		s20								
14	r7		r7	r7		r7				
15			s8/r9			r9		6		
16			s8			r9		6		
17						r4				
18	r8		r8	r8		r8				
19	s13			s14			21			
20	r6		r6	r6		r6				
21	r9		s19/r9	r9		r9		18		

Figura A.2: Tabla del autómata LR(1).

A.1.2 Compilación de la tabla LR

El proceso de compilación de las tablas del autómata LR consiste en construir el conjunto de estados internos, o estados LR(0), y a partir de ellos derivar las acciones de desplazamiento, reducción y aceptación y las acciones *ir_a*, teniendo en cuenta la longuitud de la *ventana* considerada.

A partir de un estado S_1 se puede construir otro estado S_2 unido por la relación $ir\_a(S_1, Sim, S_2)$ o $desplazamiento(S_1, Sim, S_2)$. El proceso es como sigue:

1. Seleccionar todos los *items* del estado S_1 en los que el símbolo Sim está precedido por el punto, mover el punto inmediatamente después del símbolo e insertar el nuevo *item* en el estado S_2.

2. Construir el cierre no nuclear añadiendo todos los *items* no nucleares , con el punto al principio de la parte derecha. Estos *items* nuevos se crean a partir de las reglas gramaticales cuya parte izquierda coincide con algún símbolo a la derecha del punto en cualquiera de los *items* de S_2.

Por ejemplo el estado 1 de la figura A.3 se pueden construir a partir del estado 0

Estado 0

S' → · S
S → · FN FV
FN → · Det Sust
FN → · Pron
FN → · FN FP

Estado 1

S → FN · FV
FN → FN · FP
FV → · Verb
FV → · Verb FN
FV → · Verb FN FN
FV → · FV FP
FP → · Prep FN

Estado 2

FN → Det · Sust

Estado 3

FN → Pron ·

Estado 4

S' → S ·

Estado 5

S → FN FV ·
FV → FV · FP
FP → · Prep FN

Estado 6

FV → Verb ·
FV → Verb · FN
FV → Verb · FN FN
FN → · Det Sust
FN → · Pron
FN → · FN FP

Estado 7

FN → FN FP ·

Estado 8

FP → Prep · FN
FN → · Det Sust
FN → · Pron
FN → · FN FP

Estado 9

FV → FV FP ·

Estado 10

FN → Det Sust ·

Estado 11

FV → Verb FN ·
FV → Verb FN · FN
FN → FN · FP
FN → · Det Sust
FN → · Pron
FN → · FN FP
FP → · Prep FN

Estado 12

FV → Verb FN FN ·
FN → FN · FP
FP → · Prep FN

Estado 13

FP → Prep FN ·
FN → FN · FP
FP → · Prep FN

Figura A.3: Conjunto de estados LR(0).

avanzando el punto en los *items*

S → · FN FV
FN → · FN FP

para formar los *items*

S → FN · FV
FN → FN · FP

que constituyen el núcleo del estado 1. Los restantes *items* no nucleares se generan usando las reglas 2, 3, 4, 5 y 9.

Con este método se pueden inducir todos los estados a partir de un estado inicial generado usando un *item artificial* formado por un punto y el axioma de la gramática en la parte derecha y un símbolo nuevo no perteneciente a la gramática en la parte izquierda. En la figura A.3, es el item S' → · S del estado 0.

Las acciones del autómata *desplazamiento*, *ir_a* y *aceptar* se deducen automáticamente durante el procedimiento de inducción del conjunto de estados. Además, cualquier *item* con el punto al final de la parte derecha da lugar a una acción *reducción* por medio de la regla gramatical correspondiente. Sólo queda por determinar el símbolo de anticipación para las entradas *reducción*.

Estado	*Acción*						*Ir-a*			
	Det	*Sust*	*Prep*	*Pron*	*Verb*	*eos*	*FN*	*FP*	*S*	*FV*
0	s2			s3			1		4	
1			s8		s6			7		5
2		s10								
3	r7		r7	r7	r7	r7				
4						acc				
5			s8			r1		9		
6	s2/r2		r2	s3/r2	r2	r2	11			
7	r8		r8	r8	r8	r8				
8	s2			s3			13			
9	r5		r5	r5	r5	r5				
10	r6		r6	r6	r6	r6				
11	s2/r3		s8/r3	s3/r3		r3	12	7		
12	r4		s8/r4	r4	r4	r4		7		
13	r9		s8/r9	r9	r9	r9		7		

Figura A.4: Tabla del autómata SLR(1).

En este aspecto, un analizador LR(k) usará k símbolos de anticipación, que se calculan a partir de la tabla del autómata, llevando cuenta de que símbolos pueden suceder a aquel en que se reduce la regla en cuestión. En la figura A.2 se muestra la tabla de un autómata LR(1) calculado a partir de la gramática de las estructuras sintagmáticas del Castellano introducida anteriormente.

Por desgracia, el tamaño típico de una tabla de análisis LR(1) es mayor que para una LR(0). Para reducir su tamaño se presentan dos métodos alternativos de construcción de la tabla:

- *SLR(k)* (*Simple LR(k)* [20]): los símbolos de anticipación se calculan directamente a partir de la gramática, mirando que símbolos terminales pueden suceder al símbolo de la parte izquierda de la regla reducida.

  El número de estados es el mismo que para un analizador LR(0), sin embargo el dominio determinista es más restringido, y como resultado se obtiene un número de conflictos mayor.

  A modo de ejemplo, la tabla A.4 contiene el autómata SLR(1) para la gramática de ejemplo.

- *LALR(k)* (*LookAhead LR(k)*): en este caso los símbolos de anticipación se calculan de la misma manera que para un analizador LR(k). La diferencia estriba en la simplificación de los estados que se lleva a cabo uniendo en uno solo aquellos cuyos

items solamente difieren en los símbolos de anticipación. En [17, 19] se describe la técnica usada actualmente para la generación de conjuntos de símbolos de anticipación LALR(1).

En este caso, la reducción del tamaño de la tabla de análisis justifica plenamente una pequeña perdida en la calidad del determinismo. Desde un punto de vista más pragmático, esto se puede comprobar comparando las tablas A.2 y A.5.

Estado	*Acción*						*Ir-a*			
	Det	*Sust*	*Prep*	*Pron*	*Verb*	*eos*	*FN*	*FP*	*S*	*FV*
0	s2			s3			1		4	
1			s8		s6			7		5
2		s10								
3	r7		r7	r7	r7	r7				
4						acc				
5			s8			r1		9		
6	s2		r2	s3		r2	11			
7	r8		r8	r8	r8	r8				
8	s2			s3			13			
9						r5				
10	r6		r6	r6	r6	r6				
11	s2		s8/r3	s3		r3	12	7		
12			s8/r4			r4		7		
13	r9		s8/r9	r9	r9	r9		7		

Figura A.5: Tabla del autómata LALR(1).

Apéndice B

Programación dinámica y Earley.

La programación dinámica se basa en el principio de *divide y vencerás*. Es decir, para resolver un problema más complejo, lo dividimos en partes más simples, que suponemos son útiles para la resolución del problema original. Mediante el uso de las técnicas de programación dinámica se evita la posible repetición del proceso de resolución de estos subproblemas.

B.1 Condiciones de aplicación

En 1957 Bellman [8] introduce la programación dinámica para responder a los problemas de optimización en la búsqueda operacional. El ejemplo típico consiste en buscar la distancia mínima entre dos puntos en un grafo. Este problema se puede descomponer en subproblemas más simples y la solución final se obtiene combinando las soluciones obtenidas de los subproblemas.

Por lo tanto la programación dinámica responde esencialmente a la idea de descomponer un problema en subproblemas que se resuelven una única vez, cuyos resultados se almacenan en una tabla y, lo más importante, se reutilizan varias veces sin necesidad de volver a calcularlos. De hecho, el término *dinámica* se refiere a la noción de *tabulación* de la información en una tabla dinámica.

De todas formas, sería inocente pensar que podemos reducir la complejidad de cualquier algoritmo recursivo por medio de la tabulación. En efecto, Sedgewick [67] nos muestra la clase de algoritmos que recurren al principio de "divide y vencerás", en particular el caso de la búsqueda dicotómica, y a pesar del hecho de la descomposición del problema en subproblemas, el uso de la tabulación no hará más que aumentar el coste de los cálculos[1] sin que se reduzca su complejidad. Simplemente, los subresultados no se calculan más de una vez, con lo que no se evita su repetición.

No se conoce ningún modelo general necesario y suficiente para aplicar la programación dinámica, no obstante podemos pensar, al menos, en las siguientes condiciones de aplicación:

1. Una condición práctica para decidir la rentabilidad del uso de la programación dinámica es la presencia de numerosos subcálculos idénticos, siendo estos más

[1] Téngase en cuenta que a la complejidad del algoritmo hay que sumar el coste de la gestión de la tabla.

costosos de calcular que de memorizar.

2. Una condición teórica es la existencia de una descomposición del cálculo en subcálculos más simples de una manera recursiva y uniforme.

En este punto, podemos citar algunos ejemplos para los que la programación dinámica resulta beneficiosa: el cálculo de la función de Fibonacci, el recorrido del camino más largo (o más corto) de un grafo ponderado, o la resolución *Bottom-Up* utilizada en la programación lógica [42, 46, 59, 99].

B.2 El algoritmo asociado

Los algoritmos de programación dinámica tienen todos un elemento común que es el empleo de una tabla donde se almacenan los resultados calculados. A pesar de ello, podemos, distinguir dos grandes familias:

Algoritmos ascendentes, o *Bottom-Up*, parten de los problemas más simples y utilizan las respuestas a éstos para resolver los problemas de orden superior.

Algoritmos descendentes, o *Top-Down*, parten del problema inicial y plantean recursivamente los subproblemas necesarios para su resolución. A pesar de su simplicidad, estos algoritmos muestran en la práctica una eficacia computacional pobre[2].

B.3 El precio de la tabulación

El problema de fondo de todos los algoritmos de programación dinámica es la gestión de la tabla de objetos. La eficacia en cuanto a tiempos de ejecución depende de los tiempos de acceso a la tabla, de ahí el interés en los métodos de indexación de la misma. Esta es especialmente eficaz cuando conocemos *a priori* el tamaño del problema a tratar. Por ejemplo, para calcular $fib(20)$ sabemos que basta con crear una tabla de 21 entradas donde indexar los enteros del 0 al 20.

Por desgracia, el caso anterior no es el que nos ocupa. El número de objetos que necesitamos calcular durante el análisis de un lenguaje no se puede conocer de antemano, pues depende tanto de la gramática que define el lenguaje como de la entrada a analizar en cada caso. Ello nos obliga a manejar la tabla de objetos de manera dinámica.

B.4 El algoritmo de Earley

El algoritmo de Earley se ha convertido en un ejemplo clásico de programación dinámica aplicada al problema del análisis sintáctico. En lugar de considerar una única tabla de objetos, utiliza una tabla por cada símbolo perteneciente a la cadena de entrada a

[2] Más detalles en [97] pp.44-47.

analizar. Al ser las tablas resultantes de menor tamaño, se simplifica su manejo y, por lo tanto, se mejora la eficacia del algoritmo.

En su versión original [26], el algoritmo sólo es capaz de analizar GICs, aunque posteriormente en [57] se describe una adaptación, conocida como *Earley deduction* capaz de tratar con GCDs.

El algoritmo de Earley funciona examinando la cadena de entrada de izquierda a derecha. Por cada símbolo se tienen en cuenta los k símbolos siguientes, llamados *símbolos adelantados*.

Además, por cada posición i de la cadena de entrada se crea un conjunto de *items* S_i llamado *itemset*. Cada item representa la evolución en el proceso de análisis y está compuesto por:

1. Una regla de producción de la gramática, tal que una porción de la cadena de análisis que estamos examinando se deriva de su parte derecha.

2. Un punto dentro de la parte derecha de la regla que indica hasta dónde ha llegado el proceso de análisis.

3. Un puntero, llamado *puntero de retroceso*, que señala la posición donde comienza la porción de la cadena que se deriva de la regla de producción.

4. Una cadena de k símbolos que pueden suceder a la porción instanciada por la regla de producción.

   Para simplificar el algoritmo, se añaden $k+1$ símbolos artificiales, $\dashv$, a la cadena de entrada. Estos símbolos no pertenecen a la gramática.

Denotaremos por I_k^j el item número k perteneciente al itemset S_j:

$$I_k^j \equiv [A \rightarrow \alpha.\beta, i, x]$$

y que puede ser interpretado como sigue:

> "*hemos analizado la cadena de entrada* $X_1 \ldots X_n$ *hasta la posición* j, α *deriva la subcadena* $X_i \ldots X_j$, *nos resta por reconocer la parte comprendida entre* j *y algún* k, *derivada por* β *y por lo tanto* $X_i \ldots X_k$ *derivado por* A *con* x *como sucesor*"

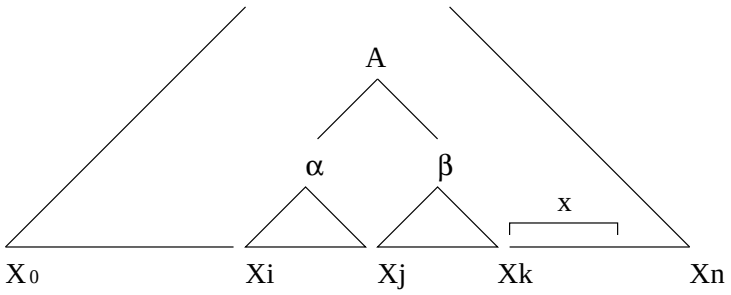


Figura B.1: Representación de un item.

Una interpretación gráfica se muestra en la figura B.1. El algoritmo comienza añadiendo el item

$$I_0^0 \equiv \left[\phi \rightarrow .S \dashv, 0, \dashv^k\right]$$

al itemset S_0, siendo S el axioma de la gramática[3]. A partir de aquí, en general, se aplican las operaciones *completer*, *predictor* y *scanner* hasta que no es posible añadir más items. Si al final del proceso el item $I_k^{n+1} \equiv [\phi \rightarrow S \dashv .\,, S_0, x]$ pertenece al itemset S_{n+1}, la cadena de entrada pertenece al lenguaje generado por la gramática. Por cada itemset S_i las operaciones se definen:

- Operación *predictor*. Es aplicable a todos los items con un símbolo no terminal, A, inmediatamente a la derecha del punto.

  Por cada regla de la forma $A \rightarrow \alpha$, se añade al itemset un nuevo item $I_k^i \equiv [A \rightarrow .\alpha, i, x]$, donde el puntero de retroceso i apunta al itemset en el que es creado, y donde x ha de ser un sucesor permitido de A.

  Esta operación equivale a *predecir* que regla de producción se usará para derivar a partir de A la parte de la cadena de entrada que comienza en la posición i.

- Operación *scanner*. Se aplica cuando es un símbolo terminal a el que sucede al punto en el item y coincide con el símbolo en la posición $i+1$ de la cadena de entrada, X_i.

  Si se cumplen las precondiciones de la operación, se añade un nuevo item al itemset S_{i+1}, con el punto desplazado una posición a la izquierda.

  La operación indica que se ha reconocido el símbolo X_{i+1}.

- Operación *completer*. Se puede aplicar cuando el punto está al final de la regla de producción, $I_k^i \equiv [A \rightarrow \alpha., j, x]$.

  Antes de aplicar el completer, se debe comprobar que los símbolos adelantados, x, coinciden con los siguientes símbolos a examinar en la cadena de entrada, $X_{i+1} \ldots X_{i+k}$.

  Por cada item en el itemset S_j con un símbolo A a la izquierda del punto, se añade un nuevo item con el punto desplazado una posición.

  Con la operación completer terminamos de examinar la parte de la cadena de entrada entre j e i derivada por A.

Si el resultado de aplicar alguna de las operaciones anteriores es un item que ya había sido creado con anterioridad, se ignora igual que si la operación no hubiese producido ningún item. Esto es así por la aplicación directa de los principios de la programación dinámica, en efecto, hemos generado un subproblema, item, que ya ha sido resuelto con anterioridad, y almacenado en la tabla de objetos, itemsets, y por lo tanto no necesitamos volver a resolverlo.

Veamos ahora como funcionan estas operaciones con un ejemplo. Usaremos 0 símbolos adelantados con la gramática de la figura B.2. La cadena de entrada a analizar será: $b\,a\,a\,c\,d\,e \dashv$.

[3]Obsérvese que, implícitamente, se añade una nueva regla a la gramática, $\phi \rightarrow S \dashv$, es decir se *aumentar* la gramática original [34].

$$
\begin{array}{lllll}
0) & \phi & \rightarrow & S \dashv \\
1) & S & \rightarrow & S\, a\, S \\
2) & S & \rightarrow & b \\
3) & S & \rightarrow & c\, d\, E \\
4) & S & \rightarrow a & \\
5) & S & \rightarrow \epsilon & \\
6) & E & \rightarrow e &
\end{array}
$$

Figura B.2: Gramática de ejemplo

Comenzamos el análisis añadiendo el item $I_0^0 \equiv [\phi \rightarrow .\, S \dashv,\, S_0]$ al itemset S_0. Al item I_0^0 se le puede aplicar la operación predictor usando las reglas 1), 2), 3), 4) y 5) produciendo los items:

$$
\begin{array}{l}
I_1^0 \equiv [S \rightarrow .\, S\, a\, S,\, S_0] \\
I_2^0 \equiv [S \rightarrow .\, b,\, S_0] \\
I_3^0 \equiv [S \rightarrow .\, c\, d\, E,\, S_0] \\
I_4^0 \equiv [S \rightarrow .\, a,\, S_0] \\
I_5^0 \equiv [S \rightarrow \epsilon\, .,\, S_0]
\end{array}
$$

Obsérvese que la regla 5) es tratada de manera especial por tratarse de una *regla épsilon*. Sobre I_5^0 es posible aplicar la operación completer, puesto que el puntero de retroceso es S_0, debemos añadir un nuevo item por cada uno perteneciente a S_0 con un punto antes de la S.

$$
\begin{array}{l}
I_6^0 \equiv [\phi \rightarrow S\, .\, \dashv,\, S_0] \\
I_7^0 \equiv [S \rightarrow S\, .\, a\, S,\, S_0]
\end{array}
$$

A continuación, aplicamos la operación scanner a I_2^0, puesto que $X_0 = b$. El item resultante se añade al itemset S_1:

$$I_0^1 \equiv [S \rightarrow b\, .,\, S_0]$$

Si volvemos a aplicar las operaciones a los items pertenecientes a S_0, no obtendremos ningún item nuevo, por lo que pasamos a operar sobre S_1. Aplicando completer sobre I_0^1, completamos I_0^0 e I_1^0:

$$
\begin{array}{l}
I_1^1 \equiv [\phi \rightarrow S\, .\, \dashv,\, S_0] \\
I_2^1 \equiv [\S \rightarrow S\, .\, a\, S,\, S_0]
\end{array}
$$

La única operación que nos resta por aplicar en S_1 es el scanner sobre I_2^1:

$$I_0^2 \equiv [S \rightarrow S\, a\, .\, S,\, S_0]$$

El proceso se repite para los restantes itemsets, hasta terminar el cálculo de S_7. Los itemsets quedan como sigue:

$S_0,\quad X_0 = b$
$I_0^0 \equiv [\phi \to .\, S \dashv,\ S_0]$
$I_1^0 \equiv [S \to .\, S\, a\, S,\ S_0]$
$I_2^0 \equiv [S \to .\, b,\ S_0]$
$I_3^0 \equiv [S \to .\, c\, d\, E,\ S_0]$
$I_4^0 \equiv [S \to .\, a,\ S_0]$
$I_5^0 \equiv [S \to \epsilon .,\ S_0]$
$I_6^0 \equiv [\phi \to S .\, \dashv,\ S_0]$
$I_7^0 \equiv [S \to S .\, a\, S,\ S_0]$

$S_1,\quad X_1 = a$
$I_0^1 \equiv [S \to b .,\ S_0]$
$I_1^1 \equiv [\phi \to S .\, \dashv,\ S_0]$
$I_2^1 \equiv [\S \to S .\, a\, S,\ S_0]$

$S_2,\quad X_2 = a$
$I_0^2 \equiv [S \to S\, a .\, S,\ S_0]$
$I_1^2 \equiv [S \to .\, S\, a\, S,\ S_2]$
$I_2^2 \equiv [S \to .\, b,\ S_2]$
$I_3^2 \equiv [S \to .\, c\, d\, E,\ S_2]$
$I_4^2 \equiv [S \to .\, a,\ S_2]$
$I_5^2 \equiv [S \to \epsilon .,\ S_2]$
$I_6^2 \equiv [S \to S\, a\, S .,\ S_0]$
$I_7^2 \equiv [S \to S .\, a\, S,\ S_2]$
$I_8^2 \equiv [\phi \to S .\, \dashv,\ S_0]$
$I_9^2 \equiv [S \to S .\, a\, S,\ S_0]$

$S_3,\quad X_3 = a$
$I_0^3 \equiv [S \to a .,\ S_2]$
$I_1^3 \equiv [S \to S\, a .\, S,\ S_2]$
$I_2^3 \equiv [S \to S\, a .\, S,\ S_0]$
$I_3^3 \equiv [S \to S\, a\, S .,\ S_0]$
$I_4^3 \equiv [S \to S .\, a\, S,\ S_2]$
$I_5^3 \equiv [S \to .\, S\, a\, S,\ S_3]$
$I_6^3 \equiv [S \to .\, b,\ S_3]$
$I_7^3 \equiv [S \to .\, c\, d\, E,\ S_3]$
$I_8^3 \equiv [S \to .\, a,\ S_3]$
$I_9^3 \equiv [S \to \epsilon .,\ S_3]$
$I_{10}^3 \equiv [\phi \to S .\, \dashv,\ S_0]$
$I_{11}^3 \equiv [S \to S .\, a\, S,\ S_0]$
$I_{12}^3 \equiv [S \to S\, a\, S .,\ S_2]$
$I_{13}^3 \equiv [S \to S .\, a\, S,\ S_3]$

$S_4,\quad X_4 = c$
$I_0^4 \equiv [S \to S\, a .\, S,\ S_2]$
$I_1^4 \equiv [S \to a .,\ S_3]$
$I_2^4 \equiv [S \to S\, a .\, S,\ S_0]$
$I_3^4 \equiv [S \to S\, a .\, S,\ S_3]$
$I_4^4 \equiv [S \to .\, S\, a\, S,\ S_4]$
$I_5^4 \equiv [S \to .\, b,\ S_4]$
$I_6^4 \equiv [S \to .\, c\, d\, E,\ S_4]$
$I_7^4 \equiv [S \to .\, a, S_4]$
$I_8^4 \equiv [S \to \epsilon .,\ S_4]$
$I_9^4 \equiv [S \to S\, a\, S .,\ S_2]$
$I_{10}^4 \equiv [S \to S\, a\, S .,\ S_0]$
$I_{11}^4 \equiv [S \to S\, a\, S .,\ S_3]$
$I_{12}^4 \equiv [S \to S .\, a\, S,\ S_3]$
$I_{13}^4 \equiv [S \to S .\, a\, S,\ S_2]$
$I_{14}^4 \equiv [\phi \to S .\, \dashv,\ S_0]$
$I_{15}^4 \equiv [S \to S .\, a\, S,\ S_0]$

$S_5,\ X_5 = d$
$I_0^5 \equiv [S \to c .\, d\, E,\ S_3]$

$S_6,\ X_6 = e$
$I_0^6 \equiv [S \to c\, d .\, E,\ S_3]$
$I_1^6 \equiv [E \to .\, e,\ S_6]$

$S_7,\ X_7 = \vdash$
$I_0^7 \equiv [E \to e .,\ S_6]$
$I_1^7 \equiv [S \to c\, d\, E .,\ S_3]$
$I_2^7 \equiv [S \to S\, a\, S .,\ S_2]$
$I_3^7 \equiv [S \to S\, a\, S .,\ S_0]$
$I_4^7 \equiv [S \to S .\, a\, S,\ S_3]$
$I_5^7 \equiv [S \to S .\, a\, S,\ S_2]$
$I_6^7 \equiv [\phi \to S .\, \dashv,\ S_0]$
$I_7^7 \equiv [S \to S .\, a\, S,\ S_0]$

S_8
$I_0^8 \equiv [\phi \to S \dashv .,\ S_0]$

Puesto que el item $I_0^7 \equiv [\phi \to S \dashv .,\ S_0]$ está en S_7, la cadena de entrada es reconocida por la gramática.

La técnica usada para la generación de los itemsets y los símbolos adelantados proviene del trabajo realizado por Knuth [43] en las gramáticas LR(k) [26]. Intuitivamente, se puede pensar en el proceso de construcción de los itemsets como el equivalente a la construcción de forma dinámica y adaptada de un autómata LR(k), en la que sólo se consideran la parte del autómata que afecta al análisis de una cadena de entrada concreta.

B.4.1 El problema del análisis sintáctico

El algoritmo tal y como ha sido descrito funciona como un *reconocedor*, es decir, dada una cadena de entrada, es capaz de decidir si pertenece o no a la gramática. Si además deseamos obtener al final de análisis el conjunto de todas las derivaciones que dan lugar a la cadena de entrada, tendremos que extender el concepto de item y adaptar las operaciones a realizar sobre ellos, [92].

Con este fin, se añade a cada item una lista formada por una regla de producción y una lista de referencias a los items a partir de los cuales reconstruir la derivación de la parte derecha de la regla. Así, en el ejemplo anterior, tendríamos los items:

$$I_0^5 \equiv [S \to c\,d\,.\,E,\ S_3,\ (c\ d)]$$
$$I_0^6 \equiv [E \to e\,.,\ S_5,\ (6)]$$
$$I_1^6 \equiv [S \to c\,d\,E\,.,\ S_3,\ (3(I_0^5\ I_0^6))]$$

y reconstruiríamos la derivación de I_1^6:

$$\begin{array}{llll} I_0^5 & () & S \to & c\,d\,.\,E \\ I_1^5 & (3) & E \to & e \end{array}$$

$$\begin{array}{llll} I_1^6 & (6) & S \to & \begin{array}{ll} c\,d & E \\ I_0^5 & I_0^6 \\ c\,d & e \end{array} \end{array}$$

lo que equivale a considerar los elementos de cálculo como estructuras de representación sintáctica. A la hora de aplicar las operaciones se ha de tener en cuenta:

- Operación *predictor*. No representa el reconocimiento de ninguna parte de la cadena de entrada, por lo tanto se le añade la lista vacía, ().

  La excepción se da cuando predecimos una regla ϵ. En este caso la salida es el número con que hemos identificado dicha regla.

- Operación *scanner*. La estructura de salida debe representar el reconocimiento del terminal. En general, se ha de añadir al final de la estructura de salida del item sobre el que se aplica la operación scanner.

  A modo de ejemplo, consideremos la gramática:

  $$\begin{array}{llll} 0) & \phi & \to & S \dashv \\ 1) & S & \to & a\,S\,b \\ 2) & S & \to & c \end{array}$$

  Para la cadena de entrada $acb \dashv$ se generan entre otros los items:

  $$I_1^0 \equiv [S \to .\,a\,S\,b,\ S_0,\ ()]$$

  y, tras una operación scanner,

  $$I_0^1 \equiv [S \to\ a\,.\,S\,b,\ S_0,\ (a)]$$

  Si al item generado se le puede aplicar una operación completer, la lista será la regla de producción que se reduce, y hay que tener en cuenta el siguiente punto.

- Operación *completer*. Si la operación completer es resultado de una operación scanner, la salida del item que se completa es el número de regla, más la referencia al item que generó el completer más el terminal.

  La salida de los items creados a partir de un completer se forma añadiendo una referencia al item sobre el que se aplicó el completer. Si estos a su vez dan lugar a otra operación completer, entonces la salida será el número de la regla de

producción, más la lista formada por las referencias a los items que generaron las dos operaciones completer.

Volviendo al ejemplo anterior:

$$I_2^1 \equiv [S \rightarrow .\, c,\ S_1,\ ()]$$

y, después de un completer sobre I_2^1:

$$I_0^2 \equiv [S \rightarrow c\,.,\ S_1,\ (2)]$$

$$I_1^2 \equiv \left[S \rightarrow a\,S\,.\,b,\ S_0,\ (a\ I_0^2)\right]$$

Como última consideración, puede ocurrir que creemos un nuevo item que sólo difiera en la estructura de salida con otro ya existente. En tal caso, se juntan ambas estructuras y sólo se conserva un item, igual que haríamos si ambos fuesen idénticos. Esta situación se presenta cuando existe una ambigüedad en la gramática y obtenemos varias derivaciones diferentes de la misma subcadena de entrada. Por lo tanto la estructura de salida representa un *nodo O*.

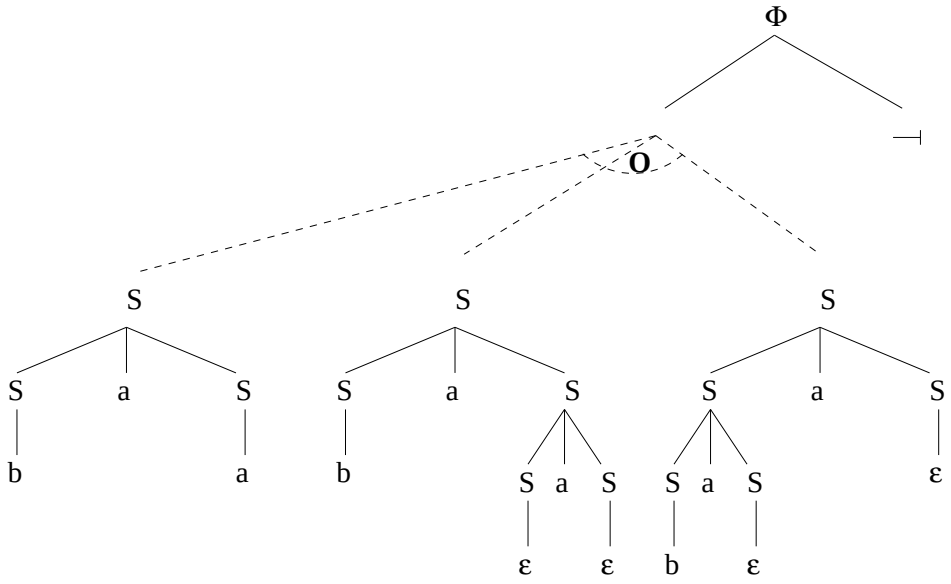


Figura B.3: Arboles sintácticos posibles para *baa* ⊣.

A continuación, se ofrece un ejemplo en el que ocurren todos los casos expuestos. La gramática vuelve a ser la de la figura B.2 y la cadena de entrada *baa* ⊣. El resultado del análisis es:

$S_0, \quad X_0 = b$
$I_0^0 \equiv [\phi \to . \, S \dashv, \, S_0, \, ()]$
$I_1^0 \equiv [S \to . \, S \, a \, S, \, S_0, \, ()]$
$I_2^0 \equiv [S \to . \, b, \, S_0, \, ()]$
$I_3^0 \equiv [S \to . \, c \, d \, E, \, S_0, \, ()]$
$I_4^0 \equiv [S \to . \, a, \, S_0, \, ()]$
$I_5^0 \equiv [S \to \epsilon \, ., \, S_0, \, (4)]$
$I_6^0 \equiv [\phi \to S \, . \dashv, \, S_0, \, (I_4^0)]$
$I_7^0 \equiv [S \to S \, . \, a \, S, \, S_0, \, (I_4^0)]$

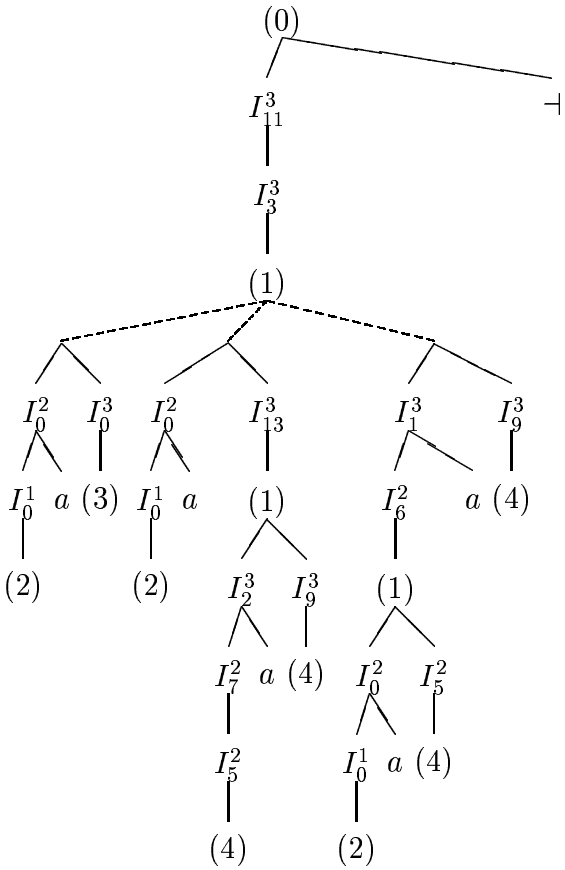
$S_1, \quad X_1 = a$
$I_0^1 \equiv [S \to b \, ., \, S_0, \, (2)]$
$I_1^1 \equiv [\phi \to S \, . \dashv, \, S_0, \, (I_0^1)]$
$I_2^1 \equiv [\S \to S \, . \, a \, S, \, S_0, \, (I_0^1)]$

$S_2, \quad X_2 = a$
$I_0^2 \equiv [S \to S \, a \, . \, S, \, S_0, \, (I_0^1 \; a)]$
$I_1^2 \equiv [S \to . \, S \, a \, S, \, S_2, \, ()]$
$I_2^2 \equiv [S \to . \, b, \, S_2, \, ()]$
$I_3^2 \equiv [S \to . \, c \, d \, E, \, S_2, \, ()]$
$I_4^2 \equiv [S \to . \, a, \, S_2, \, ()]$
$I_5^2 \equiv [S \to \epsilon \, ., \, S_2, \, (4)]$
$I_6^2 \equiv [S \to S \, a \, S \, ., \, S_0, \, (1(I_0^2 \; I_5^2)]$
$I_7^2 \equiv [S \to S \, . \, a \, S, \, S_2, \, (I_5^2)]$
$I_8^2 \equiv [\phi \to S \, . \dashv, \, S_0, \, (I_6^2)]$
$I_9^2 \equiv [S \to S \, . \, a \, S, \, S_0, \, (I_6^2)]$

$S_3, \quad X_3 = \dashv$
$I_0^3 \equiv [S \to a \, ., \, S_2, \, (3)]$
$I_1^3 \equiv [S \to S \, a \, . \, S, \, S_0, \, (I_6^2 \; a)]$
$I_2^3 \equiv [S \to S \, a \, . \, S, \, S_2, \, (I_7^2 \; a)]$
$I_3^3 \equiv [S \to S \, a \, S \, ., \, S_0, \, (1(I_0^2 \; I_0^3)(I_0^2 \; I_{13}^3)(I_1^3 \; I_9^3))]$
$I_4^3 \equiv [S \to S \, . \, a \, S, \, S_2, \, (I_0^3)]$
$I_5^3 \equiv [S \to \; . \, S \, a \, S, \, S_3, \, ()]$
$I_6^3 \equiv [S \to . \, b, \, S_3, \, ()]$
$I_7^3 \equiv [S \to . \, c \, d \, E, \, S_3, \, ()]$
$I_8^3 \equiv [S \to . \, a, S_3, \, ()]$
$I_9^3 \equiv [S \to \epsilon \, ., \, S_3, \, (4)]$
$I_{10}^3 \equiv [S \to S \, . \, a \, S, \, S_3, \, (I_9^3)]$
$I_{11}^3 \equiv [\phi \to S \, . \dashv, \, S_0, \, (I_3^3)]$
$I_{12}^3 \equiv [S \to S \, . \, a \, S, \, S_0, \, (I_3^3)]$
$I_{13}^3 \equiv [S \to S \, a \, S \, ., \, S_2, \, (1(I_2^3 \; I_9^3))]$

S_4
$I_0^4 \equiv [\phi \to S \dashv \, ., \, S_0, \, (0(I_{11}^3 \dashv))]$

Si reconstruimos la derivación a partir del último item I_0^4 obtenemos la figura que se muestra a continuación, que es equivalente a la figura B.3.


(0)
I^3_{11}
⊣
I^3_3
(1)
I^2_0 I^3_0 I^2_0 I^3_{13} I^3_1 I^3_9
I^1_0 a (3) I^1_0 a (1) I^2_6 a (4)
(2) (2) I^3_2 I^3_9 (1)
I^2_7 a (4) I^2_0 I^2_5
I^2_5 I^1_0 a (4)
(4) (2)

Apéndice C

Distribución y acceso a los recursos lingüísticos.

Como ya se explicó en la introducción, estamos tratando con una tarea inherentemente multidisciplinar. En nuestro caso particular, incluye la participación de diferentes grupos de investigación geográficamente distantes. Esto nos lleva a la necesidad de aprovechar la tecnología actualmente disponible para posibilitar el acceso remoto a las herramientas creadas por parte de los diferentes grupos involucrados.

Este apéndice introduce la manera en que se ha llevado a cabo dicha distribución en el caso del analizador léxico [10, 22], y que en un futuro se aplicará a los analizadores sintáctico y semántico. Se ha de tener en cuenta que el trabajo del lingüista no termina una vez que se han definido las distintas categorías en que se clasifican las palabras junto con los fenómenos flexivos que atañen a cada una. Es necesario, además, enumerar que palabras pertenecen a cada grupo, especificando otra información como puede ser el *lema* y la raíz (o raíces) a partir del cuál se construyen las formas inflexionadas.

A partir de ahora nos referiremos a dicha información como el *diccionario* o la *Base léxica*. Este es el recurso cuyo acceso resulta más crítico, puesto que, si bien su mantenimiento no conlleva una complejidad excesiva comparado con la especificación de las categorías y grupos, si que requiere una gran cantidad de trabajo mecánico. Es en este punto donde surge la necesidad inmediata de crear un método de acceso a los recursos descritos fácil de usar y disponible para un gran número de personas, desde diferentes situaciones geográficas.

C.1 Objetivos

Durante las distintas fases del proceso de creación y puesta a punto del analizador léxico se han completando una serie de objetivos con respecto a la distribución y el acceso a la base léxica, que a continuación pasamos a enumerar, explicándose en la sección siguiente las diferentes etapas de desarrollo:

- Consecución de una interfaz gráfica amigable.
- Construcción modular del sistema.

- Acceso al sistema de manera descentralizada.
- Facilidad para realizar modificaciones de forma transparente al usuario.
- Ejecución remota de la interfaz en un estilo cliente-servidor.
- Acceso al sistema a través del WORLD WIDE WEB.

C.2 Etapas de desarrollo

La primera etapa se centra, única y exclusivamente, en el desarrollo, implementación y prueba del analizador léxico, obviando la parte tocante al interfaz de usuario. En esta etapa toda la información referente a las palabras se almacenaba en ficheros ASCII y era accedida, o bien directamente, o bien a través de una serie de *scripts* que se lanzaban desde la línea de comandos del sistema operativo y permitían realizar las operaciones de inserción, borrado, modificación y consulta.

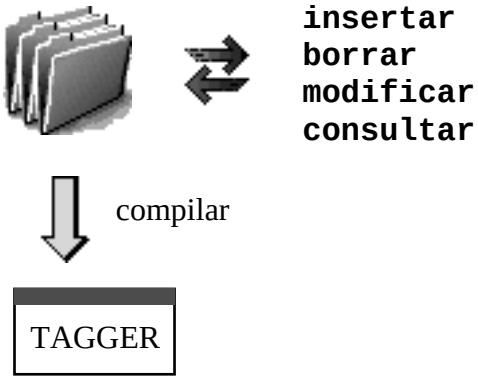


Figura C.1: Interfaz en la primera etapa.

En este momento todos los recursos residían en una misma máquina, sin que existiese ninguna distribución de los mismos, y la única posibilidad de acceso remoto era a través de los protocolos tipo `telnet`, no resultando en absoluto amigable, ni fácil de usar.

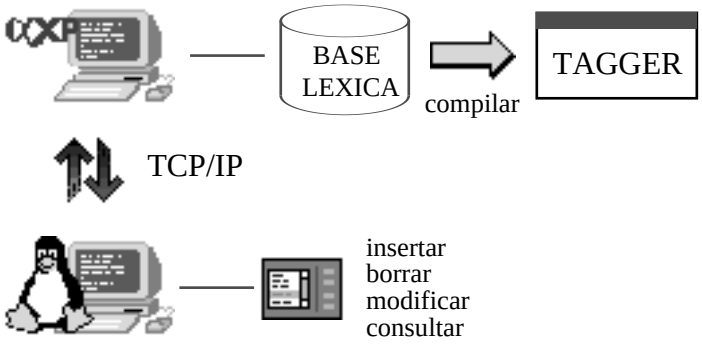


Figura C.2: Ejemplo de configuración en la segunda etapa.

En una segunda etapa, se comenzó por mejorar el método de acceso a la base léxica, creando para ello una interfaz gráfica de usuario, tal y como se muestra en la figura C.3. Llegados a este punto, surge un problema adicional cuando el tamaño de la base léxica crece hasta superar un límite por encima del cuál no resulta práctico su almacenamiento mediante ficheros ASCCI. Esta situación nos lleva de forma natural a la necesidad de implantar una base de datos relacional. Esta base de datos se configura además como un *servidor SQL* de manera que la base léxica puede ser accedida de forma remota.

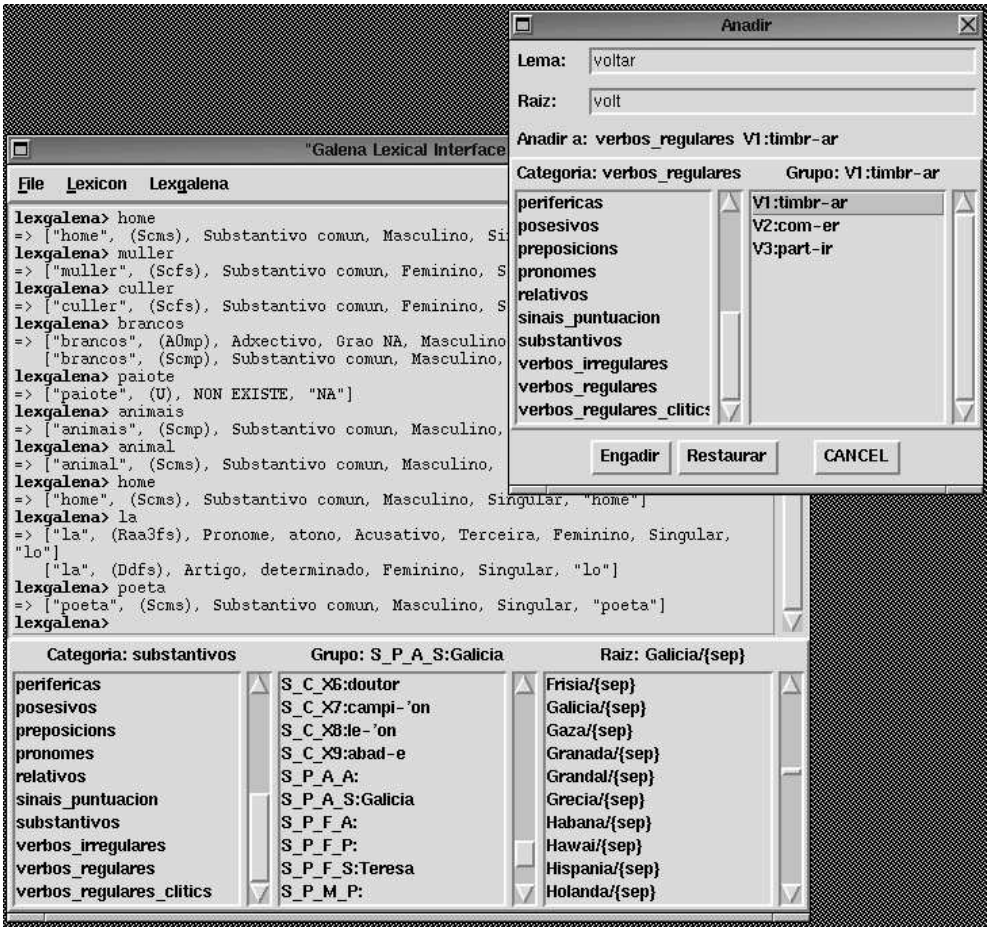


Figura C.3: Ejemplo de la interfaz gráfica de usuario para el caso del gallego.

En este momento, la interfaz gráfica de usuario se podía ejecutar de manera remota en cualquier máquina, pero ésta debía implementar un protocolo adecuado para poder comunicarse con el *servidor SQL*. A lo que había que añadir el hecho de que es necesario adaptar el interfaz gráfico a cada tipo de sistema operativo.

Todo esto nos lleva, en una tercera etapa, a instalar un *sevidor web* [22] que proporciona acceso a la base léxica. El servidor es el encargado de realizar las consultas directamente a la base de datos, mientras que la interfaz gráfica de usuario, recodificada usando *java*, lo hace a través del *www*[1] usando el protocolo *http*. De esta forma se cubren varios objetivos adicionales. Por una parte, el usuario sólo necesita disponer

[1] WORLD WIDE WEB

de un navegador www compatible java. Partimos, además, de la premisa de que un elevado tanto por ciento de usuarios ya conoce el uso de algún navegador. Los cambios de la interfaz se realizan directamente en el servidor *www*, de forma transparente al usuario, sin que sea necesario adaptarla a cada sistema operativo particular.

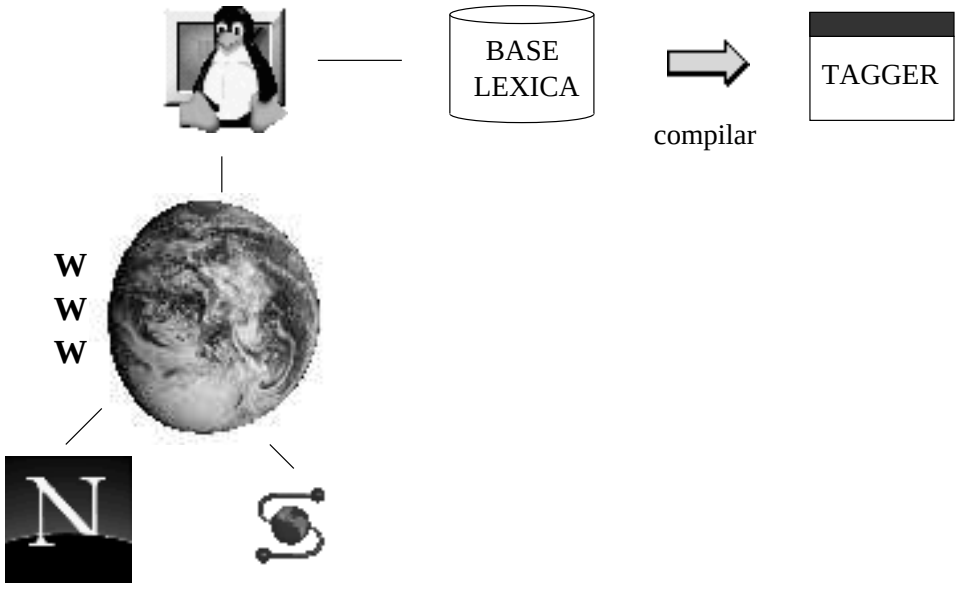


Figura C.4: Ejemplo de configuración en la tercera etapa.

C.3 Accesibilidad de la información

Hasta ahora hemos justificado la accesibilidad únicamente por la facilidad de uso de la interfaz gráfica de usuario. Sin embargo existe otro aspecto a tener en cuenta, su funcionalidad; no sólo ha de ser fácil de usar, sino que ha de permitir realizar aquellas operaciones que potencien la capacidad del usuario para llevar a cabo su trabajo.

A continuación se hará una breve introducción a las facilidades que ofrece la interfaz para el mantenimiento de la base léxica. Se puede clasificar en tres categorías principales:

- Consultar, modificar o borrar.
- Insertar nuevas palabras con un paradigma de inflexión más o menos regular.
- Insertar verbos irregulares.

En la figura C.5 se muestra en ejemplo de consulta de la base léxica. En este caso la consulta se puede expresar como:

> "Muestrame todos los nombres comunes de género masculino y número plural."

Figura C.5: Ejemplo de consulta de la base léxica.

Cualquier palabra obtenida a partir de una consulta puede ser modificada o eliminada de la base léxica. La principal ventaja es la posibilidad de restringir la consulta en base a los valores que toman las categorías, los grupos, el lema, las raíces, ...; en resumen, en base a información con determinada interpretación morfológica.

Las capacidades para añadir verbos irregulares se ejemplifican en la figura C.7. En el ejemplo expuesto se introduce el verbo `hacer`, que pertenece al grupo *VI18* en la clasificación desarrollada para el caso del castellano. Como se puede ver, es uno de los verbos más irregulares, con 7 raíces.

En este apartado, cabe resaltar la posibilidad de comprobar que los datos introducidos son correctos, ya que la interfaz es capaz de mostrar todas las formas flexionadas para cada verbo en particular en base al grupo en que se ha añadido. Más a más, también ofrece al usuario un verbo modelo de cada uno de los grupos existentes.

En la figura C.6 se muestra como esta última facilidad descrita también está presente

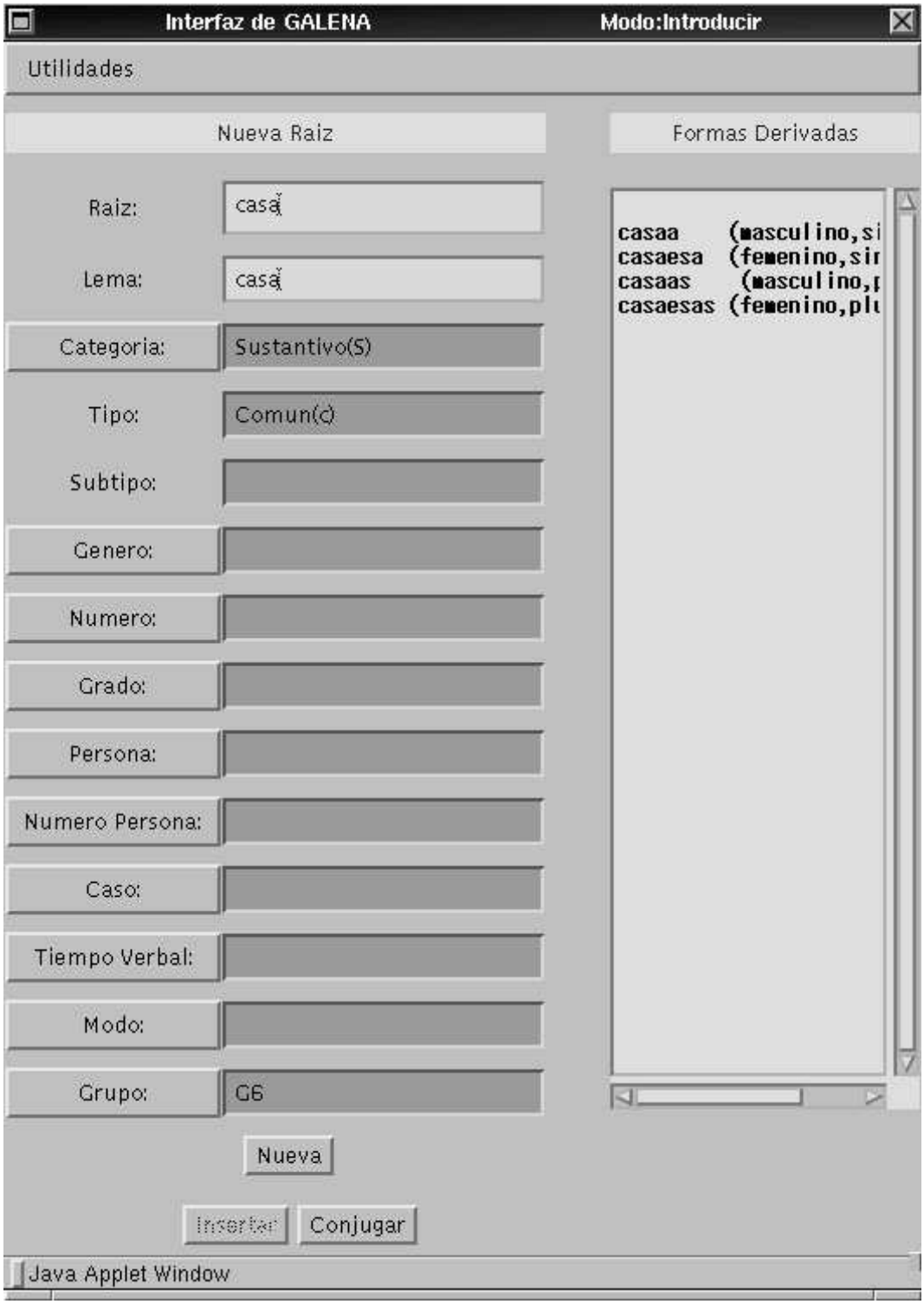


Figura C.6: Ejemplo de un intento de insertar una palabra en un grupo erróneo.

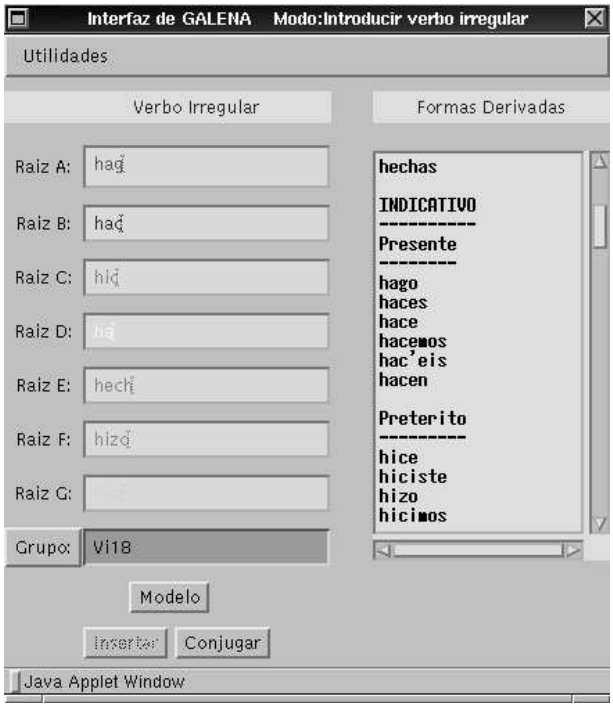


Figura C.7: Ejemplo de conjugación de un verbo irregular.

en el caso de que queramos añadir palabras con un paradigma más o menos regular. En el caso mostrado, el usuario intenta introducir un nuevo nombre común en el grupo equivocado, por suerte la lista de todas sus formas inflexionadas pone de relieve que el grupo seleccionado no es el correcto.

Apéndice D

Gramática del *GATE* para la estructura sintagmática del inglés.

El sistema completo se puede obtener directamente a través de Internet en la dirección: `http://www.dcs.shef.ac.uk/research/groups/nlp/gate/`. La gramática se divide en tres partes:

- Las cláusulas de la primera parte describen la formación de las estructuras más complejas, a nivel de oración.
- En la segunda parte se tratan las estructuras sintagmáticas.
- Por último, la tercera parte enlaza con el análisis morfológico.

D.1 Sentence grammar

```
s(A,B,C,D,E,F)--> np(G,H,I,J,K,L), vp(M,H,I,N,O,P) .

s(A,B,C,D,E,F)--> np(G,H,I,J,K,L), vp(M,H,I,N,O,P), period(Q) .

s(A,B,C,D,E,F)--> prepp(G,H,I,J,K,L), comma(M), np(N,O,P,Q,R,S),
     vp(T,O,P,U,V,W), period(X) .

s(A,B,C,D,E,F)--> np(G,H,I,J,K,L), advp(M,N,O,P,Q,R), vp(S,H,I,T,U,V),
     period(W) .

s(A,B,C,D,E,F)--> cc(G), np(H,I,J,K,L,M), vp(N,I,J,O,P,Q), period(R) .

s(A,B,C,D,E,F)--> advp(G,H,I,J,K,L), comma(M), np(N,O,P,Q,R,S),
     vp(T,O,P,U,V,W), period(X) .

s(A,B,C,D,E,F)--> s(G,H,I,J,K,L), comma(M), cc(N), s(O,P,Q,R,S,T),
     period(U) .
```

```
s(A,B,C,D,E,F)--> np(G,H,I,J,K,L), advp(M,N,O,P,Q,R), vp(S,H,I,T,U,V) .

s(A,B,C,D,E,F)--> s(G,H,I,J,K,L), comma(M), np(N,O,P,Q,R,S),
     vp(T,O,P,U,V,W), period(X) .

s(A,B,C,D,E,F)--> sbar(G,H,I,J,K,L), comma(M), np(N,O,P,Q,R,S),
     vp(T,O,P,U,V,W), period(X) .

s(A,B,C,D,E,F)--> np(G,H,I,J,K,L), vp(M,H,I,N,O,P), period(Q), sym(R) .

sbar(A,B,C,D,E,F)--> in(that), s(G,H,I,J,K,L) .

sbar(A,B,C,D,E,F)--> in(because), s(G,H,I,J,K,L) .

sbar(A,B,C,D,E,F)--> in(while), s(G,H,I,J,K,L) .

sbar(A,B,C,D,E,F)--> in(though), s(G,H,I,J,K,L) .

sbar(A,B,C,D,E,F)--> in(as), s(G,H,I,J,K,L) .

sbar(A,B,C,D,E,F)--> in(whether), s(G,H,I,J,K,L) .

sbar(A,B,C,D,E,F)--> in(although), s(G,H,I,J,K,L) .

sbar(A,B,C,D,E,F)--> whnp(G,H,I,J,K,L), vp(M,N,O,P,Q,R) .

np(A,B,C,D,E,F)--> np(G,B,C,H,I,J), prepp(K,L,M,N,O,P) .

np(A,B,C,D,E,F)--> dt(G), n(H,B,C) .

np(A,B,C,D,E,F)--> prp(G) .

np(A,B,C,D,E,F)--> n(G,B,C) .

np(A,B,C,D,E,F)--> names_np(G,H,I,J,K,L) .

np(A,B,C,D,E,F)--> dt(G), jj(H), n(I,B,C) .

np(A,B,C,D,E,F)--> jj(G), n(H,B,C) .

np(A,B,C,D,E,F)--> qual(G,H,I,J,K,L), n(M,B,C) .

np(A,B,C,D,E,F)--> np(G,H,I,J,K,L), cc(M), np(N,O,P,Q,R,S) .

np(A,B,C,D,E,F)--> dt(G), qual(H,I,J,K,L,M), n(N,B,C) .
```

```
np(A,B,C,D,E,F)--> cd_np(G,H,C,I,J,K), n(L,B,C) .

np(A,B,C,D,E,F)--> cd_np(G,H,I,J,K,L) .

np(A,B,C,D,E,F)--> np(G,B,C,H,I,J), sbar(K,L,M,N,O,P) .

np(A,B,C,D,E,F)--> poss(G,H,I,J,K,L) .

np(A,B,C,D,E,F)--> poss(G,H,I,J,K,L), n(M,B,C) .

np(A,B,sing,C,D,E)--> np(F,G,H,I,J,K), n(L,B,sing) .

np(A,B,C,D,E,F)--> np(G,H,I,J,K,L), comma(M),
     np(N,O,P,Q,R,S), comma(T) .

np(A,B,C,D,E,F)--> dt(G), names_np(H,I,J,K,L,M) .

np(A,B,C,D,E,F)--> np(G,B,C,H,I,J), comma(K), sbar(L,M,N,O,P,Q) .

np(A,B,C,D,E,F)--> basic_np(G,B,C,H,I,J) .

vp(A,B,C,D,E,F)--> to(G), vp(H,I,J,K,L,M) .

vp(A,B,C,D,E,F)--> vpcore(G,B,C,D,H,I), np(J,K,L,M,N,O) .

vp(A,B,C,D,E,F)--> vpcore(G,B,C,D,H,I), prepp(J,K,L,M,N,O) .

vp(A,B,C,D,E,F)--> vpcore(G,B,C,D,H,I), s(J,K,L,M,N,O) .

vp(A,B,C,D,E,F)--> vpcore(G,B,C,D,H,I) .

vp(A,B,C,D,E,F)--> vpcore(G,B,C,D,H,I), np(J,K,L,M,N,O),
     prepp(P,Q,R,S,T,U) .

vp(A,B,C,D,E,F)--> vp(G,H,I,J,K,L), cc(M), vp(N,O,P,Q,R,S) .

vp(A,B,C,D,E,F)--> vpcore(G,B,C,D,H,I), sbar(J,K,L,M,N,O) .

vp(A,B,C,D,E,F)--> vpcore(G,B,C,D,H,I), adjp(J,K,L,M,N,O) .

vp(A,B,C,D,E,F)--> vpcore(G,B,C,D,H,I), prepp(J,K,L,M,N,O),
     prepp(P,Q,R,S,T,U) .

prepp(A,B,C,D,E,F)--> in(G), np(H,I,J,K,L,M) .
```

```
prepp(A,B,C,D,E,F)--> to(G), np(H,I,J,K,L,M) .

prepp(A,B,C,D,E,F)--> in(G), vp(H,I,J,K,L,M) .

adjp(A,B,C,D,E,F)--> jj(G) .

adjp(A,B,C,D,E,F)--> rb(G), jj(H) .

adjp(A,B,C,D,E,F)--> jj(G), prepp(H,I,J,K,L,M) .

adjp(A,B,C,D,E,F)--> cd_np(G,H,I,J,K,L), n(M,N,O) .

adjp(A,B,C,D,E,F)--> jj(G), vp(H,I,J,K,L,M) .

adjp(A,B,C,D,E,F)--> rb(G), v(H,I,J,K) .

adjp(A,B,C,D,E,F)--> jj(G), cc(H), jj(I) .

adjp(A,B,C,D,E,F)--> adjp(G,H,I,J,K,L), prepp(M,N,O,P,Q,R) .

adjp(A,B,C,D,E,F)--> rbr(G), jj(H) .

advp(A,B,C,D,E,F)--> rb(G) .

advp(A,B,C,D,E,F)--> rb(G), rb(H) .

whnp(A,B,C,D,E,F)--> wdt(G) .

whnp(A,B,C,D,E,F)--> wp(G) .

qual(A,B,C,D,E,F)--> n(G,H,I) .

qual(A,B,C,D,E,F)--> names_np(G,H,I,J,K,L) .

poss(A,B,C,D,E,F)--> pps(G) .

poss(A,B,C,D,E,F)--> np(G,H,I,J,K,L), pos(M) .

av(A,B,C,D,E,F)--> v(A,B,C,D) .

av(A,B,C,D,E,F)--> advp(G,H,I,J,K,L), v(A,B,C,D) .

vpcore(A,B,C,past,D,E)--> nonmodal_vpcore(F,B,C,past,D,E) .
```

```
vpcore(A,B,C,present,D,E)--> nonmodal_vpcore(F,B,C,present,D,E) .

vpcore(A,B,C,past,D,E)--> modal_vpcore(F,B,C,past,D,E) .

vpcore(A,B,C,present,D,E)--> modal_vpcore(F,B,C,present,D,E) .

nonmodal_vpcore(A,B,C,D,E,F)--> nonmodal_vpcore1(G,B,C,D,E,F) .

nonmodal_vpcore(A,B,C,D,E,F)--> nonmodal_vpcore2(G,B,C,D,E,F) .

nonmodal_vpcore1(A,B,C,D,E,F)--> vpcore1(G,B,C,D,E,F) .

nonmodal_vpcore1(A,B,C,D,E,F)--> vpcore2(G,B,C,D,E,F) .

nonmodal_vpcore1(A,B,C,D,E,F)--> vpcore3(G,B,C,D,E,F) .

nonmodal_vpcore1(A,B,C,D,E,F)--> vpcore4(G,B,C,D,E,F) .

nonmodal_vpcore2(A,B,C,D,E,F)--> vpcore5(G,B,C,D,E,F) .

modal_vpcore(A,B,C,D,E,F)--> md(G), nonmodal_vpcore1(H,B,C,none,E,F) .

modal_vpcore(A,B,C,D,E,F)--> md(G), advp(H,I,J,K,L,M),
     nonmodal_vpcore1(N,B,C,none,E,F) .

vpcore1(A,3,sing,present,simple,active)--> v(B,C,D,E) .

vpcore1(A,B,C,present,simple,active)--> v(D,E,F,G) .

vpcore1(A,B,C,past,simple,active)--> v(D,E,F,G) .

vpcore1(A,B,C,D,simple,passive)--> v(be,B,C,D), av(E,F,G,H,I,J) .

vpcore1(A,B,C,none,simple,active)--> v(D,E,F,G) .

vpcore2(A,B,C,D,prog,active)--> v(be,B,C,D), av(E,F,G,H,I,J) .

vpcore2(A,B,C,D,prog,passive)--> v(be,B,C,D), av(be,E,F,G,H,I),
     av(J,K,L,M,N,O) .

vpcore3(A,B,C,D,perf,active)--> v(have,B,C,D), av(E,F,G,H,I,J) .

vpcore3(A,B,C,D,perf,passive)--> v(have,B,C,D), av(be,E,F,G,H,I),
     av(J,K,L,M,N,O) .
```

```
vpcore4(A,B,C,D,perfprog,active)--> v(have,B,C,D), av(be,E,F,G,H,I),
     av(J,K,L,M,N,O) .

vpcore4(A,B,C,D,perfprog,passive)--> v(have,B,C,D), av(be,E,F,G,H,I),
     av(be,J,K,L,M,N), av(O,P,Q,R,S,T) .

vpcore5(A,B,C,D,simple,active)--> v(do,B,C,D), av(E,F,G,H,I,J) .

np(A,B,C,D,E,F)--> poss(G,H,I,J,K,L), np(M,B,C,N,O,P) .

n(A,B,C)--> list_np(A) .

np(A,B,C,D,E,F)--> dt(G), np(H,I,J,K,L,M), n(N,B,C) .

np(A,B,C,D,E,F)--> np(G,H,I,J,K,L), sym(M), np(N,O,P,Q,R,S), sym(T) .

v(compound([A,_,B]),C,D,E)--> v(A,C,D,E), rp(B) .

v(compound([step,_,down]),A,B,C)--> v(step,A,B,C), rb(down) .

v(compound([take,_,over]),A,B,C)--> v(take,A,B,C), in(over) .
```

D.2 Entity grammar

```
basic_np(A,B,C,D,E,F)--> list_np(G) .

basic_np(A,B,C,D,E,F)--> organ_names_np(G,H,I,J,K,L), list_np(M) .

basic_np(A,B,C,D,E,F)--> organ_np(G,H,I,J,K,L) .

organ_punc_np(A,B,C,D,E,F)--> pn(G,H,I), period(J), pn(K,L,M),
     period(N), organ_names_np(O,P,Q,R,S,T), cdg_np(U,V,W,X,Y,Z) .

organ_punc_np(A,B,C,D,E,F)--> pn(G,H,I), period(J), pn(K,L,M),
     period(N), pn(O,P,Q), period(R), cdg_np(S,T,U,V,W,X) .

organ_np(A,B,C,D,E,F)--> person_period_np(G,H,I,J,K,L),
     cdg_np(M,N,O,P,Q,R) .

organ_np(A,B,C,D,E,F)--> organ_names_np(G,H,I,J,K,L), cd(M),
     cdg_np(N,O,P,Q,R,S) .

organ_np(A,B,C,D,E,F)--> organ_names_np(G,H,I,J,K,L),
     cdg_np(M,N,O,P,Q,R) .
```

```
organ_np(A,B,C,D,E,F)--> list_np(G), tagged_location_np(H,I,J,K,L,M) .

organ_np(A,B,C,D,E,F)--> tagged_location_np(G,H,I,J,K,L), list_np(M) .

organ_np(A,B,C,D,E,F)--> organ_punc_np(G,H,I,J,K,L),
     cdg_np(M,N,O,P,Q,R) .

organ_np(A,B,C,D,E,F)--> organ_punc_np(G,H,I,J,K,L) .

organ_np(A,B,C,D,E,F)--> names_np2(G,H,I,J,K,L), cc(M),
     organ_names_np(N,O,P,Q,R,S), cdg_np(T,U,V,W,X,Y) .

organ_np(A,B,C,D,E,F)--> names_np2(G,H,I,J,K,L), cc(M),
     organ_names_np(N,O,P,Q,R,S) .

organ_np(A,B,C,D,E,F)--> organ_names_np(G,H,I,J,K,L), cc(M),
     organ_names_np(N,O,P,Q,R,S), cdg_np(T,U,V,W,X,Y) .

organ_np(A,B,C,D,E,F)--> organ_names_np(G,H,I,J,K,L), cc(M),
     organ_names_np(N,O,P,Q,R,S) .

organ_np(A,B,C,D,E,F)--> names_np2(G,H,I,J,K,L), cdg_np(M,N,O,P,Q,R) .

organ_np(A,B,C,D,E,F)--> names_np2(G,H,I,J,K,L), cc(M), pn(N,O,P),
     in(Q), tagged_location_np(R,S,T,U,V,W) .

organ_np(A,B,C,D,E,F)--> organ_names_np(G,H,I,J,K,L), in(M),
     organ_names_np(N,O,P,Q,R,S), cdg_np(T,U,V,W,X,Y) .

organ_np(A,B,C,D,E,F)--> organ_names_np2(G,H,I,J,K,L), in(M),
     tagged_location_np(N,O,P,Q,R,S) .

organ_np(A,B,C,D,E,F)--> list_np(G), pn(H,I,J) .

organ_np(A,B,C,D,E,F)--> names_np(G,H,I,J,K,L), list_np(M) .

organ_np(A,B,C,D,E,F)--> list_np(G) .

organ_np(A,B,C,D,E,F)--> organ_names_np(G,H,I,J,K,L), list_np(M) .

organ_np(A,B,C,D,E,F)--> list_np(G), organ_names_np(H,I,J,K,L,M),
     list_np(N) .

organ_np(A,B,C,D,E,F)--> names_np(G,H,I,J,K,L), list_np(M) .
```

```
organ_np(A,B,C,D,E,F)--> list_np(G), names_np(H,I,J,K,L,M),
     list_np(N) .

organ_np(A,B,C,D,E,F)--> organ_names_np(G,H,I,J,K,L), list_np(M),
     in(N), tagged_location_np(O,P,Q,R,S,T) .

organ_np(A,B,C,D,E,F)--> list_np(G), in(H),
     organ_names_np(I,J,K,L,M,N) .

organ_np(A,B,C,D,E,F)--> organ_names_np(G,H,I,J,K,L), list_np(M),
     tagged_location_np(N,O,P,Q,R,S) .

organ_np(A,B,C,D,E,F)--> names_np2(G,H,I,J,K,L), cc(M), pn(N,O,P),
     list_np(Q) .

organ_np(A,B,C,D,E,F)--> organ_names_np(G,H,I,J,K,L), list_np(M) .

organ_np(A,B,C,D,E,F)--> list_np(G), names_np(H,I,J,K,L,M) .

organ_np(A,B,C,D,E,F)--> list_np(G), tagged_location_np(H,I,J,K,L,M) .

organ_np(A,B,C,D,E,F)--> organ_names_np(G,H,I,J,K,L), list_np(M) .

organ_np(A,B,C,D,E,F)--> list_np(G), names_np(H,I,J,K,L,M) .

organ_np(A,B,C,D,E,F)--> list_np(G), tagged_location_np(H,I,J,K,L,M) .

organ_np(A,B,C,D,E,F)--> organ_names_np(G,H,I,J,K,L), in(M),
     organ_names_np(N,O,P,Q,R,S), list_np(T) .

organ_np(A,B,C,D,E,F)--> organ_names_np(G,H,I,J,K,L), cc(M),
     organ_names_np(N,O,P,Q,R,S), list_np(T) .

organ_np(A,B,C,D,E,F)--> organ_names_np(G,H,I,J,K,L), list_np(M),
     in(N), jj(O), organ_names_np(P,Q,R,S,T,U) .

organ_np(A,B,C,D,E,F)--> organ_names_np(G,H,I,J,K,L), list_np(M),
     in(N), organ_names_np(O,P,Q,R,S,T) .

organ_np(A,B,C,D,E,F)--> organ_names_np(G,H,I,J,K,L), list_np(M),
     in(N), organ_names_np(O,P,Q,R,S,T) .

organ_np(A,B,C,D,E,F)--> list_np(G), in(H), jj(I),
     organ_names_np(J,K,L,M,N,O) .
```

```
organ_np(A,B,C,D,E,F)--> list_np(G), in(H),
     organ_names_np(I,J,K,L,M,N) .

organ_np(A,B,C,D,E,F)--> person_np(G,H,I,J,K,L), list_np(M) .

organ_np(A,B,C,D,E,F)--> organ_names_np(G,H,I,J,K,L), list_np(M) .

cdg_np(A,B,C,D,E,F)--> cc(G), list_np(H), period(I) .

cdg_np(A,B,C,D,E,F)--> cc(G), list_np(H), period(I) .

cdg_np(A,B,C,D,E,F)--> cc(G), list_np(H) .

cdg_np(A,B,C,D,E,F)--> cc(G), list_np(H) .

cdg_np(A,B,C,D,E,F)--> list_np(G), period(H) .

cdg_np(A,B,C,D,E,F)--> n(G,H,I), list_np(J) .

cdg_np(A,B,C,D,E,F)--> v(G,H,I,J), list_np(K) .

cdg_np(A,B,C,D,E,F)--> list_np(G) .

names_np(A,B,C,D,E,F)--> pn(G,H,I), names_np(J,K,L,M,N,O) .

names_np(A,B,C,D,E,F)--> pn(G,H,I), sym(J), pn(K,L,M) .

names_np(A,B,C,D,E,F)--> pn(G,H,I) .

names_np2(A,B,C,D,E,F)--> pn(G,H,I), comma(J), names_np2(K,L,M,N,O,P) .

names_np2(A,B,C,D,E,F)--> list_np(G), comma(H),
     names_np2(I,J,K,L,M,N) .

names_np2(A,B,C,D,E,F)--> pn(G,H,I), comma(J), pn(K,L,M) .

names_np2(A,B,C,D,E,F)--> list_np(G), comma(H), pn(I,J,K) .

names_np2(A,B,C,D,E,F)--> pn(G,H,I), comma(J), list_np(K) .

names_np2(A,B,C,D,E,F)--> list_np(G), comma(H), list_np(I) .

names_np3(A,B,C,D,E,F)--> pn(G,H,I), cc(J), pn(K,L,M) .

organ_names_np(A,B,C,D,E,F)--> organ_names_np2(G,H,I,J,K,L),
```

```
     organ_names_np(M,N,O,P,Q,R) .

organ_names_np(A,B,C,D,E,F)--> tagged_location_np(G,H,I,J,K,L),
     organ_names_np(M,N,O,P,Q,R) .

organ_names_np(A,B,C,D,E,F)--> list_np(G),
     organ_names_np(H,I,J,K,L,M) .

organ_names_np(A,B,C,D,E,F)--> jj(G), organ_names_np(H,I,J,K,L,M) .

organ_names_np(A,B,C,D,E,F)--> tagged_location_np(G,H,I,J,K,L) .

organ_names_np(A,B,C,D,E,F)--> organ_names_np2(G,H,I,J,K,L) .

organ_names_np(A,B,C,D,E,F)--> list_np(G) .

organ_names_np(A,B,C,D,E,F)--> names_np3(G,H,I,J,K,L) .

organ_names_np2(A,B,C,D,E,F)--> names_np(G,H,I,J,K,L),
     organ_names_np2(M,N,O,P,Q,R) .

organ_names_np2(A,B,C,D,E,F)--> list_np(G),
     organ_names_np2(H,I,J,K,L,M) .

organ_names_np2(A,B,C,D,E,F)--> names_np(G,H,I,J,K,L) .

organ_names_np2(A,B,C,D,E,F)--> list_np(G) .

organ_names_np2(A,B,C,D,E,F)--> list_np(G) .

organ_names_np2(A,B,C,D,E,F)--> list_np(G) .

organ_names_np2(A,B,C,D,E,F)--> list_np(G) .

organ_punc_np(A,B,C,D,E,F)--> organ_names_np(G,H,I,J,K,L), sym(M),
     tagged_location_np(N,O,P,Q,R,S), sym(T) .

organ_punc_np(A,B,C,D,E,F)--> names_np(G,H,I,J,K,L), sym(M),
     names_np(N,O,P,Q,R,S), sym(T), names_np(U,V,W,X,Y,Z) .

organ_punc_np(A,B,C,D,E,F)--> list_np(G), sym(H),
     tagged_location_np(I,J,K,L,M,N) .

organ_punc_np(A,B,C,D,E,F)--> names_np(G,H,I,J,K,L), sym(M),
     tagged_location_np(N,O,P,Q,R,S) .
```

```
organ_punc_np(A,B,C,D,E,F)--> tagged_location_np(G,H,I,J,K,L),
     sym(M), names_np(N,O,P,Q,R,S) .

basic_np(A,B,C,D,E,F)--> person_np(G,H,I,J,K,L) .

basic_np(A,B,C,D,E,F)--> title_np(G,H,I,J,K,L) .

person_np(A,B,C,D,E,F)--> pn(G,H,I), list_np(J) .

person_np(A,B,C,D,E,F)--> person_period_np(G,H,I,J,K,L),
     de_np(M,N,O,P,Q,R), names_np(S,T,U,V,W,X) .

person_np(A,B,C,D,E,F)--> person_special_np(G,H,I,J,K,L),
     de_np(M,N,O,P,Q,R), names_np(S,T,U,V,W,X) .

person_np(A,B,C,D,E,F)--> person_period_np(G,H,I,J,K,L) .

person_np(A,B,C,D,E,F)--> person_special_np(G,H,I,J,K,L) .

person_np(A,B,C,D,E,F)--> list_np(G), names_np(H,I,J,K,L,M),
     person_ending(N,O,P,Q,R,S) .

person_np(A,B,C,D,E,F)--> list_np(G), person_ending(H,I,J,K,L,M) .

person_np(A,B,C,D,E,F)--> person_ending_np(G,H,I,J,K,L) .

person_np(A,B,C,D,E,F)--> title_np(G,H,I,J,K,L),
     person_period_np(M,N,O,P,Q,R) .

person_np(A,B,C,D,E,F)--> title_np(G,H,I,J,K,L),
     person_special_np(M,N,O,P,Q,R) .

person_np(A,B,C,D,E,F)--> title_np(G,H,I,J,K,L),
     names_np(M,N,O,P,Q,R) .

person_np(A,B,C,D,E,F)--> title_np(G,H,I,J,K,L),
     person_ending_np(M,N,O,P,Q,R) .

person_np(A,B,C,D,E,F)--> title_np(G,H,I,J,K,L),
     tagged_list_np(M,N,O,P,Q,R) .

tagged_list_np(A,B,C,D,E,F)--> list_np(G) .

person_period_np(A,B,C,D,E,F)--> pn(G,H,I), pn(J,K,L), period(M),
```

```
     pn(N,O,P) .

person_period_np(A,B,C,D,E,F)--> pn(G,H,I), period(J),
     pn(K,L,M), pn(N,O,P) .

person_period_np(A,B,C,D,E,F)--> pn(G,H,I), pn(J,K,L), period(M),
     list_np(N) .

person_period_np(A,B,C,D,E,F)--> list_np(G), pn(H,I,J), period(K),
     pn(L,M,N) .

person_period_np(A,B,C,D,E,F)--> pn(G,H,I), period(J),
     list_np(K), pn(L,M,N) .

person_period_np(A,B,C,D,E,F)--> list_np(G), pn(H,I,J), period(K),
     list_np(L) .

person_period_np(A,B,C,D,E,F)--> pn(G,H,I), period(J), list_np(K) .

person_special_np(A,B,C,D,E,F)--> list_np(G) .

person_special_np(A,B,C,D,E,F)--> list_np(G) .

person_special_np(A,B,C,D,E,F)--> list_np(G), names_np(H,I,J,K,L,M) .

person_special_np(A,B,C,D,E,F)--> list_np(G), list_np(H) .

person_special_np(A,B,C,D,E,F)--> list_np(G), list_np(H) .

person_special_np(A,B,C,D,E,F)--> list_np(G),
     tagged_location_np(H,I,J,K,L,M) .

person_special_np(A,B,C,D,E,F)--> names_np(G,H,I,J,K,L), sym(M),
     pn(N,O,P), sym(Q), names_np(R,S,T,U,V,W) .

person_special_np(A,B,C,D,E,F)--> list_np(G), sym(H), pn(I,J,K),
     sym(L), names_np(M,N,O,P,Q,R) .

person_ending_np(A,B,C,D,E,F)--> names_np(G,H,I,J,K,L),
     person_ending(M,N,O,P,Q,R) .

person_ending_np(A,B,C,D,E,F)--> person_period_np(G,H,I,J,K,L),
     person_ending(M,N,O,P,Q,R) .

person_ending_np(A,B,C,D,E,F)--> person_special_np(G,H,I,J,K,L),
```

```
     person_ending(M,N,O,P,Q,R) .

person_ending(A,B,C,D,E,F)--> pn(G,H,I), period(J) .

person_ending(A,B,C,D,E,F)--> pn(G,H,I), period(J) .

person_ending(A,B,C,D,E,F)--> pn(G,H,I) .

person_ending(A,B,C,D,E,F)--> pn(G,H,I) .

de_np(A,B,C,D,E,F)--> pn(G,H,I) .

de_np(A,B,C,D,E,F)--> pn(G,H,I) .

de_np(A,B,C,D,E,F)--> pn(G,H,I) .

de_np(A,B,C,D,E,F)--> pn(G,H,I) .

title_np(A,B,C,D,E,F)--> list_np(G), in(H), n(I,J,K) .

title_np(A,B,C,D,E,F)--> list_np(G), in(H), jj(I), n(J,K,L) .

title_np(A,B,C,D,E,F)--> list_np(G), in(H), jj(I), cc(J),
     jj(K), n(L,M,N) .

title_np(A,B,C,D,E,F)--> list_np(G) .

basic_np(A,B,C,D,E,F)--> date_np(G,H,I,J,K,L) .

date_np(A,B,C,D,E,F)--> cd(G), n(H,I,J), cd(K), n(L,M,N), cd(O) .

date_np(A,B,C,D,E,F)--> list_np(G), vbd(H), date(I), cd(J) .

date_np(A,B,C,D,E,F)--> date(G), comma(H), date(I), cd(J), comma(K),
     list_np(L) .

date_np(A,B,C,D,E,F)--> date(G), period(H), cd(I), comma(J),
     list_np(K) .

date_np(A,B,C,D,E,F)--> date(G), cd(H), comma(I), list_np(J) .

date_np(A,B,C,D,E,F)--> list_np(G), in(H), list_np(I) .

date_np(A,B,C,D,E,F)--> list_np(G), list_np(H) .
```

```
date_np(A,B,C,D,E,F)--> list_np(G) .

date_np(A,B,C,D,E,F)--> date(G), period(H), list_np(I) .

date_np(A,B,C,D,E,F)--> date(G), comma(H), list_np(I) .

date_np(A,B,C,D,E,F)--> date(G), list_np(H) .

date_np(A,B,C,D,E,F)--> date(G), period(H), cd(I) .

date_np(A,B,C,D,E,F)--> date(G), cd(H) .

date_np(A,B,C,D,E,F)--> date(G) .

basic_np(A,B,C,D,E,F)--> time_np(G,H,I,J,K,L) .

time_np(A,B,C,D,E,F)--> cd(G), list_np(H) .

time_np(A,B,C,D,E,F)--> list_np(G) .

basic_np(A,B,C,D,E,F)--> money_np(G,H,I,J,K,L) .

money_np(A,B,C,D,E,F)--> cd_np(G,H,I,J,K,L), jj(M), jj(N),
     m_unit_np(O,P,Q,R,S,T) .

money_np(A,B,C,D,E,F)--> cd_np(G,H,I,J,K,L), jj(M),
     m_unit_np(N,O,P,Q,R,S) .

money_np(A,B,C,D,E,F)--> cd_np(G,H,I,J,K,L),
     tagged_location_np(M,N,O,P,Q,R), m_unit_np(S,T,U,V,W,X) .

money_np(A,B,C,D,E,F)--> tagged_location_np(G,H,I,J,K,L),
     m_unit_np(M,N,O,P,Q,R), cd_np(S,T,U,V,W,X) .

money_np(A,B,C,D,E,F)--> cd_np(G,H,I,J,K,L), m_unit_np(M,N,O,P,Q,R) .

money_np(A,B,C,D,E,F)--> m_unit_np(G,H,I,J,K,L), cd_np(M,N,O,P,Q,R) .

m_unit_np(A,B,C,D,E,F)--> sym(G) .

m_unit_np(A,B,C,D,E,F)--> sym(G) .

m_unit_np(A,B,C,D,E,F)--> list_np(G) .

m_unit(A,B,C,D,E,F)--> cd(G) .
```

```
m_unit(A,B,C,D,E,F)--> cd(G) .

m_unit(A,B,C,D,E,F)--> cd(G) .

m_unit(A,B,C,D,E,F)--> cd(G) .

m_unit(A,B,C,D,E,F)--> cd(G) .

m_unit(A,B,C,D,E,F)--> cd(G) .

basic_np(A,B,C,D,E,F)--> percent_np(G,H,I,J,K,L) .

percent_np(A,B,C,D,E,F)--> cd_np(G,H,I,J,K,L), n(M,N,O) .

percent_np(A,B,C,D,E,F)--> fraction_np(G,H,I,J,K,L), n(M,N,O) .

percent_np(A,B,C,D,E,F)--> fraction_np(G,H,I,J,K,L),
     n(M,N,O), n(point,P,Q) .

percent_np(A,B,C,D,E,F)--> cd_np(G,H,I,J,K,L), n(M,N,O), n(point,P,Q) .

fraction_np(A,B,C,D,E,F)--> cd(G), cd(H), n(I,J,K), cd(L) .

fraction_np(A,B,C,D,E,F)--> cd(G), n(H,I,J), cd(K) .

cd_np(A,B,C,D,E,F)--> cd_np(G,H,I,J,K,L), m_unit(M,N,O,P,Q,R) .

cd_np(A,B,C,D,E,F)--> cd(G), comma(H), cd(I) .

cd_np(A,B,C,D,E,F)--> cd(G), period(H), cd(I) .

cd_np(A,B,C,D,E,F)--> cd(G) .

basic_np(A,B,C,D,E,F)--> location_np(G,H,I,J,K,L) .

basic_np(A,B,C,D,E,F)--> tagged_location_np(G,H,I,J,K,L) .

location_np(A,B,C,D,E,F)--> pn(G,H,I), in(J), names_np(K,L,M,N,O,P) .

location_np(A,B,C,D,E,F)--> pn(G,H,I), in(J),
     tagged_location_np(K,L,M,N,O,P) .

location_np(A,B,C,D,E,F)--> dt(G), names_np(H,I,J,K,L,M), n(N,O,P) .
```

```
location_np(A,B,C,D,E,F)--> names_np(G,H,I,J,K,L), list_np(M) .

location_np(A,B,C,D,E,F)--> tagged_location_np(G,H,I,J,K,L),
     list_np(M) .

location_np(A,B,C,D,E,F)--> names_np(G,H,I,J,K,L), comma(M),
     tagged_location_np(N,O,P,Q,R,S) .

location_np(A,B,C,D,E,F)--> names_np(G,H,I,J,K,L), comma(M),
     tagged_location_np(N,O,P,Q,R,S) .

location_np(A,B,C,D,E,F)--> tagged_location_np(G,H,I,J,K,L), comma(M),
     tagged_location_np(N,O,P,Q,R,S) .

tagged_location_np(A,B,C,D,E,F)--> list_np(G) .
```

D.3 Terminals

```
list_np(A)--> [list_np(A)] .

n(A,B,C)--> [n(A,B,C)] .

pn(A,B,C)--> [pn(A,B,C)] .

jj(A)--> [jj(A)] .

v(A,B,C,D)--> [v(A,B,C,D)] .

cc(A)--> [cc(A)] .

cd(A)--> [cd(A)] .

comma(A)--> [comma(A)] .

date(A)--> [date(A)] .

dt(A)--> [dt(A)] .

in(A)--> [in(A)] .

md(A)--> [md(A)] .

period(A)--> [period(A)] .

pos(A)--> [pos(A)] .
```

```
pps(A)--> [pps(A)] .

prp(A)--> [prp(A)] .

rb(A)--> [rb(A)] .

rbr(A)--> [rbr(A)] .

rp(A)--> [rp(A)] .

sym(A)--> [sym(A)] .

to(A)--> [to(A)] .

vbd(A)--> [vbd(A)] .

wdt(A)--> [wdt(A)] .

wp(A)--> [wp(A)] .
```

Bibliografía

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers Principles, Techniques, and Tools.* Addison Wesley, 1986.

[2] M.A. Alonso, D. Cabrero, and M. Vilares. A new approach to the construction of generalized LR parsing algorithms. In *Proc. of RANLP'97, Recent Advances in Natural Language Processing*, pages 171–178, Tzigov Chark, Bulgaria, 1997.

[3] M.A. Alonso, D. Cabrero, and M. Vilares. Construction of efficient generalized LR parsers. In *(to be published) Lecture Notes in Artificial Intelligence*, Berlin-Heildeberg-New York, 1998.

[4] M.A. Alonso, E. de la Clergerie, and M. Vilares. Automata-based parsing in dynamic programming for linear indexed grammars. In A.S. Narin'yani, editor, *Proc. of DIALOGUE'97. Int. Conf. on Computational Linguistics and its Applications*, pages 22–27, Moscow, Russia, 1997.

[5] Javier Andrade Garda, Alberto Valderruten Vidal, María Concepción Alvarez Lebredo, and Susana Sotelo Docío. Un supresor de ambigüedades léxicas mediante métodos estadísticos. *Procesamiento del Lenguaje Natural*, 20:1–12, Marzo 1997.

[6] Andrew W. Appel. *Modern Compiler Implementation in C: Basic Techniques.* Cambridge University Press, 1997. An advanced edition is to be published in 1998.

[7] R. Backofen and G. Smolka. A complete and recursive feature theory. *Theoretical Computer Science*, 146(1-2):243–268, Julio 1995.

[8] R.E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, New Jersey, U.S.A., 1957.

[9] J. Bresnan and R. Kaplan. *Lexical-Functional Grammar: A formal System for Grammatical Representation*, chapter 4, pages 173–281. The MIT Press, 1982.

[10] D. Cabrero, M. Vilares, L. Docampo, and S. Sotelo. Web-surfing the lexicon. In *Workshop on distributing and Accessing Lexical Resources*, Granada, España, 1998.

[11] Noam Chomsky. *Lectures on Government and Binding.* D. Reidel, Dordrecht, The Netherlands, 1981.

[12] Noam Chomsky. *Barriers.* Linguistic Inquiry Monograph 13. MIT Press, Cambridge, MA, 1986.

[13] Noam Chomsky. A minimalist program for linguistic theory. In Kenneth Hale and Samuel J. Keyser, editors, *The View from Building 20*, pages 1–52. MIT Press, Cambridge, Mass., 1993.

[14] J. Cohen. A view of the origins and development of prolog. *Communications of the ACM*, 31(1):26–36, 1988.

[15] A. Colmerauer. Opening the Prolog-III universe. *Byte Magazine*, 12(9):177–182, Agosto 1987.

[16] Alain Colmerauer. Les grammaires de metamorphose. Technical report, Groupe d'Intelligence Artificielle, Université de Marseille-Luminy, Noviembre 1975. Translated into English as Colmerauer [15].

[17] Robert Paul Corbett. Static semantics and compiler error recovery. Technical Report CSD-85-251, University of California, Berkeley, Junio 1985.

[18] Miguel de Cervantes Saavedra. *El ingenioso hidalgo Don Quijote de la Mancha.*

[19] Frank DeRemer and Thomas Pennello. Efficient computation of LALR(1) look-ahead sets. *ACM Transactions on Programming Languages and Systems*, 4(4):615–649, Octubre 1982.

[20] Franklin L. DeRemer. Simple LR(k) grammars. *Communications of the ACM*, 14(7):453–460, Julio 1971.

[21] C.F. Derksen. *Manual for de AGFL system.* Deparment of Informatics, University of Nijmegen, Univeristy of Nijmegen, 1995. http://www.cs.kun.nl/agfl/Papers.html#General.

[22] L. Docampo. Explotación remota del sistema galena. Master's thesis, Universidad de Informática, A Coruña, Septiembre 1997.

[23] EAGLES. Formalisms working group final report. Expert advisory group on language engineering standards document, September 1996.

[24] EAGLES. Recommendations on subcategorization. Expert Advisory Group on Language Engineering Standards document EAG-CLWG-SYNLEX/P, August 1996.

[25] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, Febrero 1970.

[26] Jay Earley. *An Efficient Context-Free Parsing Algorithm.* PhD thesis, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, PA, 1968.

[27] Gregor Erbach. *Bottom-Up Earley Deduction for Preference-Driven Natural Language Processing.* PhD thesis, Sthulsatzenhausweg, Saarbrücken, 1995.

[28] G. Gazdar and C. Mellish. *Natural Language Processing in Prolog: an introduction to computational linguistics.* Addison Wesley, 1989.

[29] Gerald Gazdar. Applicability of indexed grammars to natural languages. In U. Reyle and C. Rohrer, editors, *Natural Language Parsing and Linguistic Theories*, pages 69–94. D. Reidel, Dordrecht, 1988.

[30] Jorge Graña Gil, Miguel Angel Alonso Pardo, and Alberto Valderruten Vidal. Análisis léxico no determinista: Etiquetación eficiente del lenguaje natural. Technical Report 16, Departamento de Computación, Facultade de Informática, Universidade da Coruña, Campus de Elviña s/n, 15071 La Coruña, España, 1994.

[31] J. Heering, P.R.H. Hendriks, P. Klint, and J. Rekers. The syntax definition formalism SDF - reference manual. *SIGPLAN Notices*, 24(11):43–75, 1989.

[32] P. Van Hentenrynck. Constraint satisfaction in logic programming. 1989.

[33] Markus Höhfeld and Gert Smolka. Definite relations over constraint languages. LILOG Report 53, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, Germany, Octubre 1988.

[34] J.E. Hopcroft and J.E. Ullman. *Introduction to Automata Theory, Languages and Computation.* Series in Computer Science. Addison-Wesley, 1979.

[35] Joxan Jaffar and Jean-Louis Lassez. Constraing logic programming. In *Conference Record of the Fourteenth Annual ACM Symposium on Principles of Programming Languages*, pages 111–119, Munich, West Germany, Enero 1987. ACM SIGACT/SIGPLAN, ACM.

[36] M. Johnson. *Attribute-value Logic and the Theory of Grammar*, volume 16 of *CSLI Lecture Notes.* CSLI, Stanford, 1988. Distributed by University of Chicago Press.

[37] Aravind K. Joshi. An introduction to tree adjoining grammars. In Alexis Manaster-Ramer, editor, *Mathematics of Language*, pages 87–115. John Benjamins Publishing Co., Amsterdam/Philadelphia, 1987.

[38] R. Kasper and W. C. Rounds. A logical semantics for feature structures. In *Proceedings* 24^{th} *Meeting of the Association for Computational Linguistics*, New York, 1986.

[39] Martin Kay. Functional grammar. In *Proceedings of the 5th Annual Meeting of the Berkeley Linguistic Society*, pages 142–158, Berkeley, CA, Febrero 17-19, 1979.

[40] Martin Kay. Functional unification grammar: a formalism for machine translation. *COLING-84*, pages 75–78, 1984.

[41] Martin Kay. Parsing in functional unification grammar. In Lauri Karttunen David R. Dowty and Arnold M. Zwicky, editors, *Natural Language Parsing*, pages 251–278. Cambridge University Press, Cambridge, 1985.

[42] M. Kifer and E. L. Lozinskii. SYGRAF: Implementing logic programs in a database style. *IEEE Transactions on Software Engineering*, 14(7):922–935, Julio 1988.

[43] Donald E. Knuth. On the translation of languages from left to right. *Information and Control*, 8(6):607–639, Diciembre 1965.

[44] K. Koskenniemi. Compilation of automata from morphological two-level rules. In *Proc. of the* 5^{th} *Scandinavian Conference of Computational Linguistics*, pages 143–149, Helsinki, Finland, 1985.

[45] B. Lang. Deterministic techniques for efficient non-deterministic parsers. Technical Report 72, INRIA, Rocquencourt, France, 1974.

[46] B. Lang. Complete evaluation of Horn Clauses, an automata theoretic approach. Technical Report 913, INRIA, Rocquencourt, France, 1988.

[47] B. Lang. Datalog automata. In C. Beeri, J. W. Schmidt, and U. Dayal (eds), editors, *Proc. of the* 3^{rd} *International Conference on Data and Knowledge Bases*, pages 389–404, Jerusalem, Israel, 1988. Morgan Kaufmann Pub.

[48] Suresh Manandhar. An attributive logic of set descriptions and set operations. In *Proceedings of the 32nd. Annual Meeting of the Association for Computational Linguistics*. ACL, 1994.

[49] Suresh Manandhar. Deterministic checking of LP constraints. In *Proceedings of the Seventh Conference of the European Chapter of the Association for Computational Linguistics*, Dublin, Ireland, 1995. Association for Computational Linguistics.

[50] Johannes Matiasek. Conditional constraints in a CLP-based HPSG implementation. In Harald Trost, editor, *KONVENS '94*, pages 230–239, Vienna, 1994.

[51] H. Meijer. The project on extended affix grammars at Nijmegen. *Attribute Grammars and their Applications, SLNC*, 461:130–142, 1990.

[52] Monica Monachini and Nicoletta Calzolari. Synopsis and comparison of morphosyntactic phenomena encoded in lexicons and corpora. A common proposal and applications to European languages. Technical report, Istituto di Linguistica Computazionale, Octubre 1994. Draft version of EU-LRE Project Eagles deliverable EAG-LSG/IR-T4.6/CSG-T3.2.

[53] A. Moreno. *Un modelo computacional basado en la unificación para el análisis y la generación de ls morfología del español.* PhD thesis, Universidad Autónoma de Madrid, España, 1992.

[54] Mark-Jan Nederhof. Linear indexed automata and tabulation of TAG parsing. In *Proc. of First Workshop on Tabulation in Parsing and Deduction (TAPD'98)*, pages 1–9, Paris, France, Abril 1998.

[55] V Paxon. *Flex 2.4.6 (January 1994). Reference Documentation, Full User Documentation & Changes between Releases.* Lawrence Berkeley Laboratory, University of California, Berkeley, California, U.S.A., 1994.

[56] F. C.Ñ. Pereira and D. H. D. Warren. Definite clause grammars for language analysis - A survey of the formalism and a comparison with augmented transition network. *Artificial Intelligence*, pages 231–278, 1980.

[57] F.C.N. Pererira and D.H.D. Warren. Parsing as deduction. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 137–144, Cambridge (Massachusetts), 1983.

[58] Carl Pollard and Ivan Sag. *Head-Driven Phrase Structure Grammar.* University of Chicago Press, Chicago, 1994. Draft distributed at the Third European Summer School in Language, Logic and Information, Saarbrücken, 1991.

[59] R. Ramakrishnan. Magic templates: A spellbinding approach to logic programs. In *Proc. of the 5th International Conference on Logic Programming*, 1988.

[60] Owen Rambow. *Formal and Computational Aspects of Natural Languege Syntax.* PhD thesis, University of Pennsylvania, 1994. Available as IRCS Report 94-08 of the Institute of Research in Cognitive Science, University of Pennsylvania.

[61] J. Rekers. *Parser Generation for Interactive Environments.* PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, 1992.

[62] G. Ritchie. On the generative power of two-level morphological rules. In *European Chapter of the ACL*, pages 51–57, Manchester, 1989.

[63] G. Ritchie, D. Pulman, Stephen, A.W. Black, and G.J Russ ell. *Computational Morphology.* The MIT Press, Cambridge, Massachusetts, U.S.A., 1991.

[64] J. A. Robinson. A machine-oriented logic based on resolution principle. *Journal of the ACM*, 12(1):23–49, Enero 1965.

[65] E. Roche and Y. Schabes. Deterministic part-of-speech tagging with finite-state transducers. *Computational Linguistics*, 21(2):227–253, 1995.

[66] W. C. Rounds and R. Kasper. A complete logical calculus for record structures representing linguistic information. In *Symposium on Logic in Computer Science (LICS '86)*, pages 38–43, Washington, D.C., USA, Junio 1986. IEEE Computer Society Press.

[67] Robert Sedgewick. *Algorithms*, chapter 11 - Priority Queues, pages 145–162. Addison-Wesley Series in Computer Sience. Addison-Wesley Publishing Company, Inc., second edition edition, 1988. privat.

[68] Stuart M. Shieber. *An Introduction to Unification-Based Approaches to Grammar.* Number 4 in CSLI Lecture Notes. Center for the Study of Language and Information, Stanford University, Stanford, CA, 1986.

[69] Stuart M. Shieber. *Constraint-Based Grammar Formalisms: Parsing and Type Inference for Natural and Computer Languages.* MIT Press, Cambridge, Massachusetts, 1992.

[70] Stuart M. Shieber, Hans Uszkoreit, Fernando Pereira, Jane Robinson, and M. Tyson. The formalism and implementation of PATR-II. In Barbara J. Grosz, editor, *Research on interactive acquisition and use of knowledge*, pages 39–79. SRI International, Stanford, California, 1983.

[71] Gert Smolka. Feature-constraint logics for unification grammars. *Journal of Logic Programming*, 12(1-2):51–87, Enero 1992.

[72] L. Sterling and E. Shapiro. *The Art of Prolog.* MIT Press, Cambridge, Mass., 1986.

[73] M. Tomita. *Efficient Parsing for Natural Language. A Fast Algorithm for Practical Systems.* Kluwer Academic Publishers, Norwell, Massachusetts, U.S.A., 1986.

[74] Matsuru Tomita. *Generalized LR Parsing.* Matsuru Tomita, editor, Kluwer, Boston/Dordrecht/London, 1991.

[75] Harald Trost and Johannes Matiasek. Morphology with a null-interface. In *Proceedings of the 15th International Conference on Computational Linguistics*, Kyoto, Japan, 1994.

[76] Hans Uszkoreit. Categorial unification grammars. In *Proceedings of COLING-86*, 1986. Also appears as Center for the Study of Language and Information Report No. CSLI-86-66, Stanford, CA.

[77] K. Vijay-Shanker. *A Study of Tree Adjoining Grammars.* PhD thesis, University of Pennsylvania, Enero 1988. Available as Technical Report MS-CIS-88-03 LINC LAB 95 of the Department of Computer and Information Science, University of Pennsylvania.

[78] M. Vilares. Error repair for the Earley's algorithm. Technical report, CERICS, Sophia-Antipolis, Valbonne, France, 1989.

[79] M. Vilares. An efficient context-free backbone for natural languages analyzers. *Boletín de la SEPLN*, 14:201–214, 1994.

[80] M. Vilares. Efficient sharing in ambiguous parsing. In *Actas del X Congreso de la SEPLN*, Córdoba, España, 1994.

[81] M. Vilares and M.A. Alonso. Towards analyzers based on efficient logical frames. In *Proc. of the II International Conference on Mathematical Linguistics*, pages 97–98, Tarragona, España, 1996.

[82] M. Vilares and M.A. Alonso. An LALR extension of DCGs in dynamic programming. In C. Martín Vide, editor, *Mathematical Linguistics*, volume 2, Amsterdam, The Netherlands, 1997. John Benjamins Publishing Company.

[83] M. Vilares, M.A. Alonso, and D. Cabrero. Autómatas lógicos y lenguaje natural. In *Actas de la V Conf. Iberoamericana de Inteligencia Artificial, ISBN 968-18-5429-2*, pages 341–351, Universidad de las Américas, Puebla, México, 1996.

[84] M. Vilares, M.A. Alonso, and D. Cabrero. Efficient parsing of fixed-mode DCGs. In Geert-Jan M. Kruijff, Glyn V. Morrill, and Richard T. Oehrle, editors, *Proc. of Formal Grammar'97*, pages 199–209, Aix-en-Provence, France, 1997.

[85] M. Vilares, M.A. Alonso, and D. Cabrero. An operational model for parsing fixed-mode DCGs. In *Proc. of LACL'97. Int. Conf on Logical Aspects on Computational Linguistics*, pages 61–64, Nancy, France, 1997.

[86] M. Vilares, M.A. Alonso, J. Graña, and D. Cabrero. GALENA: Tabular DCG parsing for natural languages. In *Proc. First Workshop on Tabulation in Parsing and Deduction (TAP D'98)*, Paris, France, 1998.

[87] M. Vilares, D. Cabrero, and M.A. Alonso. An approach to infinite term traversal in DCGs. In M. Falaschi and M.Ñavarro, editors, *Proc. of AGP'97, Joint Conf. in Declarative Programming*, pages 307–317, Grado, Italy, 1997.

[88] M. Vilares, D. Cabrero, and M.A. Alonso. A comparison for unification-based parsers. In M. Falaschi and M.Ñavarro, editors, *Proc. of AGP'98, Joint Conf. in Declarative Programming*, pages 113–123, A Coruña, España, 1998.

[89] M. Vilares and B. A. Dion. Efficient incremental parsing for context-free languages. In *Proc. of the* 5^{th} *IEEE International Conference on Computer Languages*, pages 241–252, Toulouse, France, 1994.

[90] M. Vilares and J. Graña. Parsing as resolution. In *Proc. of the First Compulog Network Meeting of Parallelism and Implementation Technologies*, Madrid, España, 1993.

[91] M. Vilares, J. Graña, and F. Cacheda. Verification of morphological analyzers. In A.S. Narin'yani, editor, *Proc. of DIALOGUE'96. Int. Conf. on Computational Linguistics and its Applications*, pages 69–72, Moscow, Russia, 1996.

[92] Manuel Vilares Ferro. *Efficient Incremental Parsing for Context-Free Languages.* PhD thesis, University of Nice, France, 1992.

[93] Manuel Vilares Ferro and Miguel Angel Alonso Pardo. Exploring interactive chart parsing. *Procesamiento del Lenguaje Natural*, 17:158–170, Septiembre 1995.

[94] Manuel Vilares Ferro and Miguel Angel Alonso Pardo. A predictive bottom-up evaluator. *Logic Programming Newsletter*, 8(4):9–10, 1995.

[95] Manuel Vilares Ferro, Jorge Graña Gil, and Alberto Pan Bermúdez. Building friendly architectures for tagging. *Procesamiento del Lenguaje Natural*, 19:127–132, 1996.

[96] Manuel Vilares Ferro, Alberto Valderruten Vidal, Jorge Graña Gil, and Miguel Angel Alonso Pardo. Une approche formelle pour la génération d'analyseurs de langages naturels. In P. Blache, editor, *Actes de la Seconde Conférence Annuelle sur le Traitement Automatique du Langage Naturel*, pages 246–255, Marseille, France, Junio 1995.

[97] Eric Villemonte de la Clergerie. *Automates à Piles et Programmation Dynamique. DyALog : Une Application à la Programmation en Logique*. PhD thesis, Université Paris 7, Paris, France, 1993.

[98] Terry Winograd. *Language as a Cognitive Process, Volume 1: Syntax*. Addison Wesley, New York, 1983.

[99] C. Zaniolo and D. Saccà. Rule rewriting methods for efficient implementations of horn logic. In M. Boscarol; L. Carlucci Aiello; G. Levi, editor, *Proceedings of the Workshop on Foundations of Logic and Functional Programming*, volume 306 of *LNCS*, pages 114–142, Berlin, Diciembre 1987. Springer.

Índice de Materias