UNIVERSIDADE DA CORUÑA

TESIS DOCTORAL

Análisis Eficaz de Gramáticas de Cláusulas Definidas

Autor: David Cabrero Souto
Director: Manuel Vilares Ferro

UNIVERSIDADE DA CORUÑA

Departamento de Computación

TESIS DOCTORAL

Análisis Eficaz de Gramáticas de Cláusulas Definidas

Autor: David Cabrero Souto
Director: Manuel Vilares Ferro

Junio de 2002

Resumen

"Análisis Eficaz de Gramáticas de Cláusulas Definidas"

Dentro del análisis sintáctico, la utilización de formalismos gramaticales valuados es, hoy por hoy, punto inconturnable en el desarrollo de estrategias de análisis sintáctico en entornos de procesamiento del lenguaje natural y en programación lógica, este último como representante del paradigma declarativo. El presente trabajo se centra en el estudio y desarrollo de técnicas de análisis sintáctico dirigidas, en última instancia, al tratamiento de sistemas basados en el análisis de formalismos gramaticales valuados donde, típicamente, el concepto de categoría gramatical se extiende a un dominio posiblemente infinito. En estas circunstancias los métodos clásicos de análisis sintáctico basados en la consideración de dominios finitos no son aplicables, al no garantizar la correcta terminación del proceso de cálculo. Referido al campo de las gramáticas lógicas, esta situación fuerza la necesidad del desarrollo e implementación de técnicas de análisis capaces de asegurar la completud de la resolución en el caso de presencia de símbolos funcionales.

Igualmente cobra especial relevancia la consideración de las técnicas de programación dinámica aplicadas al análisis sintáctico. Este hecho es debido a la compactación de las representaciones, que convierte este paradigma en una técnica eficiente para el tratamiento de cálculos con un alto grado de redundancia, relacionados con fenómenos tales como el no determinismo, habituales en formalismos gramaticales complejos.

Hasta el momento, las técnicas aplicadas se han basado fundamentalmente en el concepto subyacente en la técnica de restricción. Esta permite de forma simple y eficaz limitar el proceso de evaluación a aquellos nodos del bosque de prueba en los que la ausencia de bucles a nivel de la unificación está garantizada. La técnica no permite, sin embargo, una detección automatizada de los nodos conflictivos ni su representación.

Nuestro acercamiento prevé la consideración tanto del problema de la detección de ciclos a nivel de la unificación de argumentos, como su representación explícita en los casos en los que ello sea posible[1].

Nuestro punto de partida será el estudio de las propiedades estáticas de evaluación de los formalismos gramaticales considerados y su adecuación a técnicas de evaluación dinámica, las mejor adaptadas al problema por garantizar una compartición óptima de cálculos y estructuras.

Para ello estableceremos un marco descriptivo común sobre el cual desarrollar y comparar diversas estrategias de análisis sintáctico. Este marco también nos permitirá extender de manera intuitiva las técnicas incorporadas al análisis de lenguajes a otros formalismos gramaticales valuados.

[1] el problema es, en general, no decidible [105].

Agradecimientos

La realización del trabajo de tesis doctoral que aquí se presenta no estaría completo sin expresar mi más profundo agradecimiento al Dr. M. Vilares, cuya labor va más alla de lo que concerciente a la simple investigación. Tampoco quiero dejar de agradecer la aportación del Dr. M.A. Alonso. También agradezco al Dr. V.M. Gulías que me recordase contínuamente el "*kick-off*".

Ya que nos encontramos en la sección de agradecimientos, quisiera recordar a otros miembros del grupo *CoLe*: el Dr. J. Graña, Víctor, Fran y Jesús. Además de los mencionados, otras personas han influido positivamente en este trabajo, ellos ya saben quién son.

Por último agradeceré a Rufino su insistencia, a Josefina, Dolores y Víctor su interés. Y a Laura que me quiera aunque no sepa decirlo.

Contactar con el autor

Cualquier persona interesada en contactar con el autor puede hacerlo en la dirección que se indica a continuación. Los comentarios y sugerencias acerca de la memoria, el trabajo que refleja o los aspectos relacionados son bienvenidos.

David Cabrero Souto
Departamento de Computación
Facultad de Informática
Campus de Elviña s/n
15071 A Coruña (España)

Si se prefiere el correo electrónico, la dirección es la siguiente:

cabrero@udc.es

Índice

Índice de figuras

CAPÍTULO 1

INTRODUCCIÓN

El análisis sintáctico ha sido y continúa siendo investigado activamente. Como botón de muestra podemos fijar nuestra atención en el proyecto *Babylon*[1] de la DARPA, Defense Advanced Research Projects Agency, organización encargada de la investigación y desarrollo para el departamento de defensa (DoD) de los EE.UU. Los objetivos de dicho proyecto se encaminan a la instrumentación de interfaces de usuario para la traducción bidireccional, multi-idioma, del habla en combate y otros entornos de campo. Este proyecto debe reemplazar el sistema existente de traducción unidireccional, solucionando sus problemas, entre los que se señala el análisis sintáctico. La base tecnológica incluye la investigación en mejores técnicas de análisis sintáctico, conocimiento semántico, motores morfológicos, ... Por último, uno de los hitos técnicos planificados en el proyecto es el desarrollo de un *shallow parser*, término que podríamos hacer corresponder en castellano con *analizador sintáctico superficial*.

Dentro del análisis sintáctico, la utilización de formalismos gramaticales valuados es, hoy por hoy, una referencia indiscutible en lo que se refiere a los entornos de procesamiento del lenguaje natural, y en programación lógica, este último representante del paradigma declarativo. La presente memoria aborda el estudio y desarrollo de técnicas de análisis sintáctico dirigidas, precisamente, al tratamiento de sistemas basados en el análisis de formalismos gramaticales valuados.

1.1 Sobre el análisis sintáctico

Habitualmente, la estructura sintáctica de un lenguaje se describe mediante una gramática, a partir de la cual es posible determinar los constituyentes de una frase dada. La obtención de dichos constituyentes es lo que se conoce como el *problema del análisis sintáctico*, y los algoritmos encargados de realizar dicha tarea, como *analizadores sintácticos*. A medida que crece la complejidad del lenguaje tratado, crece la complejidad de los formalismo gramaticales que poseen el suficiente poder descriptivo como para expresar su estructura. Típicamente, los lenguajes de programación se diseñan a partir de una gramática a la cual se deben adaptar los programas. Estas gramáticas se construyen eliminando cualquier posible ambigüedad, facilitando de esta manera el desarrollo de los analizadores asociados. En cambio, las lenguas naturales evolucionan de forma continua, siendo las gramáticas las que deben adaptarse a ellas. En éstas, la ambigüedad es inevitable tanto a nivel morfológico, como sintáctico, y semántico.

[1] http://www.darpa.mil/ipto/research/babylon/goals.html

Esta complejidad se ve reflejada a su vez a nivel de los analizadores correspondientes, dando lugar a problemas de eficiencia y completud.

En esta memoria se trata el desarrollo de analizadores sintácticos, tanto en lo que respecta a su ejecución eficiente, como a la extensión del dominio de los formalismos gramaticales a los que resultan aplicables. Para ello se han adoptado las siguientes decisiones de diseño:

- Uso de máquinas abstractas o *autómatas* en lugar de la propia gramática. En efecto, un autómata no es más que un dispositivo matemático que permite un tratamiento más eficiente del proceso de análisis.
- Uso de programación dinámica. La presencia de ambigüedades conlleva la repetición de cálculos, y la recursión, problemas de completud operacional. Las técnicas de programación dinámica abordan estos inconvenientes mediante la compartición de cálculos.
- Análisis estático. Con objeto de reducir el espacio de búsqueda, se realiza un análisis previo de la gramática. Los resultados de este análisis permiten mejorar la complejidad tanto temporal como espacial de los algoritmos.
- Indexación del proceso de análisis. La sincronización del proceso mediante la indexación reduce el espacio de búsqueda.
- Compartición de estructuras. La compartición no sólo mejora la complejidad espacial, sino que además permite la implementación eficiente de operaciones básicas como la *unificación*.
- En el caso del análisis del lenguaje natural, uso de un *lexicón* separado. De esta forma se permite el empleo de técnicas mejor adaptadas a su casuística particular, que difiere de la presente en el análisis sintáctico.

Junto con las decisiones anteriormente mencionadas, y relacionadas con éstas, se han tenido en cuenta los siguientes aspectos:

Formalismos gramaticales. Resulta necesario delimitar la potencia expresiva del formalismo gramatical empleado de forma que se establezca un equilibrio adecuado entre la clase de lenguajes cuya estructura puede representar y la complejidad del proceso de análisis.

En este sentido, la programación lógica mantiene un fuerte vínculo con el análisis sintáctico. La primera versión de PROLOG se desarrolló para resolver problemas deductivos en un sistema para la comunicación hombre-máquina en francés, escrito por Colmerauer *et al.* [37]. Poco después, Colmerauer [38] introdujo el concepto de *gramática lógica*, donde la operación básica es la unificación. Dentro de esta clase de gramáticas, quizás debido a esta evolución histórica y a que, en su definición original, tienen el poder de las máquinas de Turing con respecto al formalismo descrito, las *gramáticas de cláusulas definidas* [104] (GCDs) son el representante más popular. En la presente memoria las técnicas de análisis sintáctico se aplicarán en última instancia al análisis de GCDs.

Análisis sintáctico como deducción. La expresión *análisis sintáctico como deducción*, acuñada por Pereira [105], hace referencia a la representación del conocimiento gramatical mediante un conjunto de axiomas, y el proceso de análisis como un proceso deductivo sobre ellos. Esta idea a dado lugar a diversas estrategias [131, 133] que constituyen un marco descriptivo común sobre el cual desarrollar, comparar o demostrar propiedades de diversos analizadores.

Programación dinámica. La programación dinámica es una técnica conocida para la resolución de problemas de optimización [23]. El tipo de problemas a los que se aplica son aquellos en los que es posible su descomposición en otros más simples, de tal manera que la solución final se obtiene como una combinación de las soluciones de dichos subproblemas. Para que esta aproximación resulte beneficiosa, las soluciones intermedias se almacenan en una tabla y se reutilizan sin necesidad de volver a calcularlas. Así, la condición de aplicabilidad de la programación dinámica es la descomposición en subproblemas redundantes [125], en caso contrario el coste asociado a la gestión de la tabla reduce la eficiencia del sistema resultante.

Ejemplos típicos en los cuales esta técnica resulta beneficiosa son el cálculo de la función de Fibonacci, el recorrido del camino más largo de un grafo ponderado o la resolución ascendente en programación lógica [149, 142, 181]. Por otra parte, un contraejemplo conocido es la búsqueda dicotómica, en la cual no es posible reutilizar ninguna solución intermedia. Sin embargo, no se conoce ninguna condición necesaria y suficiente para establecer la aplicabilidad de la programación dinámica a un problema dado, aunque la práctica parece exigir la concurrencia de las siguientes características:

- El problema inicial se puede descomponer en otros más simples, generalmente a partir de una definición recursiva del problema.
- Gran parte de los subproblemas son redundantes, permitiendo la reutilización de soluciones.
- El coste de resolver un subproblema es mayor que el coste de almacenamiento y búsqueda en la tabla de soluciones.

Una vez establecida la redundancia de los subproblemas a tratar, la cuestión de fondo es la eficiencia en la gestión de la tabla de objetos, de ahí el interés en los métodos de indexación. Estos son especialmente eficaces cuando conocemos *a priori* el tamaño del problema a tratar. Por desgracia, este no es el caso en el problema que nos ocupa, lo que nos obliga a manejar la tabla de objetos de manera dinámica.

En su aplicación al análisis sintáctico, es más frecuente el término *tabulación* respecto a las técnicas de programación dinámica, y *algoritmo tabular* respecto a los algoritmos resultantes de su aplicación. Una de las primeras referencias la constituye el algoritmo de Earley [47], aunque han sido aplicadas a analizadores de diversa naturaleza [84, 81, 9, 10, 13, 14, 179, 132].

1.2 Contextualización del trabajo

El trabajo que nos ocupa comienza con el disfrute de una beca de *Introducción a la Investigación* del departamento de Computación de la Universidad de A Coruña, en el seno de lo que más tarde sería el grupo de investigación CoLe[2]. Como continuación de este período se realiza el trabajo de Tesis de Licenciatura "*Integración de Herramientas para el Análisis Automático de Lenguaje Natural*" [28]. Esta tesina dedica una parte importante del esfuerzo al análisis sintáctico, sirviendo de base al desarrollo del presente trabajo de *tesis doctoral*.

En lo que respecta a los resultados obtenidos, éstos se incorporan a los sistemas *GA*LENA y ERIAL, desarrollados bajo la colaboración directa del grupo CoLe, el grupo Atoll del INRIA[3] en su unidad de Roquencourt, el *Grupo de Sintaxis del Español* de la Universidad de Santiago de Compostela, y el Centro Ramón Piñeiro para la Investigación en Humanidades, de la Xunta de Galicia. El desarrollo de estos sistemas es posible, en parte, gracias a diversos proyectos de financiación oficial, tanto regional (XUGA10501A93, XUGA20403B95, XUGA10505B96, XUGA20402B97, PGIDT99XI10502B, PGIDT01PXI10506PN), como nacional (TIC2000-0370-C02-01) e internacional (HF96-36, HF97-223, 1FD97-0047-C04-02, HP2001-0044).

En este contexto, el proyecto presentado viene a ampliar, de manera significativa, el dominio de aplicación de los formalismos considerados. Ello facilitará en gran medida el diseño de herramientas informáticas dedicadas a los procesos de recuperación automática de información. De forma adicional permitirá la implementación de mecanismos automáticos de recuperación y corrección sintáctica en el procesamiento de bases de datos textuales.

1.3 Estructura de la memoria

La estructura lógica de la presente memoria consta de varias partes. Dicha estructura aborda de manera incremental los conceptos y técnicas expuestos, de forma que cada parte incluye los resultados de las anteriores. A continuación esbozamos las líneas generales de cada uno de los capítulos:

En una primera parte encontramos la presente introducción y el capítulo 2. En este capítulo se realiza un recorrido por las técnicas clásicas y otras más actuales relacionadas con el análisis sintáctico, centrándose en las más representativas. El objeto es ofrecer una visión general del estado del arte en el campo que nos ocupa.

En una segunda parte se trata el problema del análisis de *gramáticas independientes del contexto* (GICs). El objetivo es introducir y desarrollar técnicas como los *sistemas de deducción gramatical* (SDGs), *autómatas de pila* (APs) y sus *esquemas de compilación*, y *tabulación*. Estas técnicas se emplearán en el resto de la memoria. En el capítulo 3 se introduce el problema del análisis sintáctico de GICs. Se ha optado por estas gramáticas debido a su amplia difusión y conocimiento, de esta forma se establecen los conceptos y terminología empleados en el resto de la memoria. A continuación, el capítulo 4 presenta los sistemas de deducción gramatical como marco descriptivo

[2]Por *Co*mpiladores y *Le*nguajes.

[3]Institut National de Recherche en Informatique et Automatique.

común bajo el cual se formulan diversos algoritmos de análisis, mostrando la relación entre ellos, obteniendo unos a partir de los anteriores. Para completar esta parte, en el capítulo 5 se desarrolla la aplicación de los *autómatas de pila* a la construcción de analizadores sintácticos. Asimismo, se introduce la aplicación de las técnicas de tabulación a los autómatas obtenidos para la mejora de su eficiencia. Finalmente se describe el sistema ICE, de generación de analizadores sintácticos para GICs sin restricciones.

La tercera parte trata el problema del análisis de GCDs. El poder descriptivo de las GICs se muestra insuficiente para algunas aplicaciones como pueda ser el *procesamiento del lenguaje natural*. Como alternativa se presentan formalismos más complejos, entre ellos, las *gramáticas lógicas*. En concreto, optaremos por las GCDs como formalismo gramatical de referencia, ya desde el capítulo 6. Este formalismo tienen un mayor poder descriptivo, pero a cambio se incrementa la complejidad de los analizadores asociados. Se tratará la descripción de las GCDs como una generalización de las GICs, la adaptación de los SDGs presentados para dichas gramáticas, la generalización de los APs para la implementación de analizadores, y los mecanismos de tabulación asociados. Finalmente se trata la evolución del sistema ICE para su tratamiento. En el siguiente capítulo, capítulo 7, en primer lugar se tratan las estructuras de datos empleadas para una implementación eficiente de la operación de unificación, que constituye la base del formalismo de GCDs. En segundo lugar se abordará el tratamiento de *estructuras cíclicas*, con la consiguiente ampliación del dominio de terminación de los analizadores propuestos.

La cuarta parte incluye aquellos aspectos relacionados con la aplicación de las técnicas de análisis sintáctico al problema del procesamiento del lenguaje natural, en la que resultan especialmente importantes los resultados de capítulos anteriores dirigidos a la mejora de la eficiencia, tratamiento de formalismos gramaticales complejos, o la ampliación del dominio de terminación. En relación a la adecuación de los analizadores presentados anteriormente al procesamiento del lenguajes natural, el capítulo 8 trata tres aspectos importantes: la integración con otras herramientas habituales como los *etiquetadores*, el tratamiento de palabras desconocidas y la obtención de análisis sintácticos parciales.

En una última parte, el capítulo 9 está dedicado a la presentación de los resultados obtenidos tras la realización de diversos experimentos sobre las técnicas y algoritmos de análisis sintáctico tratadas en los capítulos anteriores. Por último, en el capítulo 10 se exponen las conclusiones sobre el trabajo realizado y se trazan las posibles líneas a seguir para su mejora y ampliación.

1.4 Publicación de resultados

La realización de la tesis doctoral tratada en la presente memoria conlleva la publicación de diversos trabajos en forma de artículos de revista, capítulos de libro y ponencias en congresos. Las referencias a estos trabajos se recopilan en el apéndice B. Si bien no existe una correspondencia directa entre cada una de las publicaciones y los capítulos de esta memoria, sí que es posible establecer la siguiente clasificación:

- Evolución del algoritmo de análisis empleado en el sistema ICE [15, 16, 11, 12].

- Generalización del sistema ICE al caso de las GCDs [169, 168, 166, 172].
- Tratamiento de términos cíclicos en GCDs [170, 171, 176, 167, 155, 154, 156, 157].
- Adaptación de algoritmos al análisis parcial [29, 161].
- Comparación de rendimiento de diversos algoritmos [174, 175, 162].

CAPÍTULO 2

ANTECEDENTES

En este capítulo realizaremos un recorrido por las técnicas relacionadas con el análisis sintáctico, centrándose en las más representativas. Consideraremos los antecedentes y las técnicas actuales tanto si están directamente relacionadas con las expuestas en subsiguientes capítulos, como si no. El objeto es ofrecer una visión general del estado del arte en el campo que nos ocupa.

2.1 Análisis independiente del contexto

El análisis sintáctico de GICs es el más veterano de los aspectos que trataremos, y existen múltiples referencias al respecto [3, 62, 59]. Las primeras técnicas se dirigen al tratamiento de gramáticas deterministas. Al obviar la existencia de ambigüedades, los algoritmos de análisis se simplifican considerablemente. Dentro de estos algoritmos podemos señalar los *ascendentes* o *descendentes* puros [3], o el método de *precedencia simple* [185], si bien este último, desde un punto de vista práctico, ha sido abandonado en favor de las técnicas LR [73], dando lugar a los algoritmos de tipo LR(k) [3], SLR(k) [43] y LALR(k) [44]. Como principal ventaja, estos algoritmos presentan una complejidad temporal lineal, $\mathcal{O}(n)$, sobre la longitud n de la cadena de entrada [3]. Por contra, como ya hemos indicado, sólo son aplicables a gramáticas deterministas por lo que queda descartado el análisis de lenguajes generados por GICs ambiguas. Finalmente, otra solución, paralela a los analizadores de tipo LR, es el *análisis guiado por la esquina izquierda*, LC[1] [118, 22, 135, 149]. Como principal diferencia con los algoritmos basados en autómatas LR, la tabla del autómata se obtiene tras las compilación de la componente predictiva del algoritmo de Earley.

En un segundo grupo podemos abordar los analizadores capaces de manejar gramáticas no deterministas. En una primera aproximación encontramos los analizadores basados en el mecanismo de *retroceso* [3, 77] para el tratamiento de ambigüedades. A pesar de que ésta es una técnica sencilla de implementar, presenta una complejidad temporal exponencial [3]. Como alternativa encontramos los algoritmos de análisis que aplican la programación dinámica [23], entre los que destacamos los algoritmos CYK [187, 69] y Earley [48, 47]. El algoritmo CYK original, desarrollado simultáneamente por Coke [60], Younger [187] y Kasami [69], presenta una complejidad temporal cúbica, $\mathcal{O}(n^3)$, y aunque únicamente es válido para gramáticas en *forma*

[1] En inglés, *left-corner parsing*.

normal de Chomsky, posteriormente se han propuesto versiones que eliminan esta restricción [49, 34]. Por su parte el algoritmo de Earley no impone restricciones sobre la forma de la gramática. Al igual que el algoritmo CYK su complejidad es, en general, $\mathcal{O}(n^3)$, sin embargo, a diferencia de éste, para gramáticas no ambiguas la complejidad es $\mathcal{O}(n^2)$, y para aquellas en el que el conjunto de items que genera está acotado, $\mathcal{O}(n)$.

La siguiente aproximación la constituye el empleo de APs. En [146, 145, 147] Tomita presenta una evolución de los analizadores LR para el caso de las gramáticas no deterministas, habitualmente conocido como *análisis LR generalizado*. Este algoritmo aplica los conceptos de la programación dinámica mediante la construcción de un grafo de pilas en el cual se comparten las partes comunes. El algoritmo original presenta problemas a la hora de tratar gramáticas cíclicas o con recursiones ocultas por la izquierda. Además, la complejidad temporal es $\mathcal{O}(n^{p+1})$, donde p es la longitud máxima de la parte derecha de las producciones, lo cual implica que para conseguir una complejidad $\mathcal{O}(n^3)$, es necesario que la gramática esté en forma normal de Chomsky. Se han propuesto varias mejoras del algoritmo original. Rekers [109] lo modifica para superar las limitaciones sobre la recursividad y ciclicidad de las gramáticas, pero manteniendo la complejidad temporal. Otros autores [97] optan por modificar la construcción del autómata LR, aunque en este caso el tratamiento de gramáticas cíclicas es más complejo, prefiréndose evitarlo. Otras aproximaciones similares a la de Tomita son los sistemas SDF [61] y GLR [109]. Por último, análogamente a los analizadores LR generalizados, también existen versiones generalizadas de los algoritmos LC [89, 123, 86, 96].

Por su parte, Lang en [84] propone el uso de *autómatas de pila no deterministas* como técnica para la implementación de analizadores independientes del contexto paralelos. La familia de analizadores obtenida es capaz de simular el comportamiento de cualquier AP que analiza una GIC, con complejidad cúbica en el peor de los casos. Para mejorar la eficiencia, se aplican técnicas de tabulación a los autómatas de pila [164]. Dichas técnicas se basan en el trabajo de De la Clergerie [179, 181] para su aplicación a los compiladores de programas de cláusulas definidas.

2.1.1 Otras aproximaciones

Otra aproximación al problema del no determinismo consiste en la aplicación de modelos estadísticos de los lenguajes a analizar. En una primera propuesta tenemos las *gramáticas independientes del contexto estocásticas* [56, 140, 121, 78], donde se añade una probabilidad a las producciones de la gramática. Los analizadores son modificados de manera que únicamente se sigue la alternativa más probable. Otra aproximación, directamente relacionada, es aplicar el modelo estadístico sobre el control finito en el caso de los autómatas LR [33], de forma que se eliminan los conflictos mediante la elección de la alternativa más probable. Destacaremos dos inconvenientes en las soluciones basadas en modelos estadísticos. En primer lugar es necesario estimar los parámetros del modelo para cada lenguaje en particular. Este requerimiento implica la necesidad de disponer de un conjunto de entrenamiento compuesto por pares cadena de entrada y análisis correcto. La obtención de dichos conjuntos es una tarea costosa en tiempo y recursos, y sujeta a errores. Los modelos estimados no son 100% correctos.

A mayores de las GICs, los modelos estocásticos también han sido aplicados a otros formalismos gramaticales, p.e. *gramáticas de adjunción de árboles estocásticas* [110,

124] o *gramáticas basadas en unificación estocásticas* [27].

2.2 Análisis de gramáticas de cláusulas definidas

El término *gramática lógica* se origina con los trabajos de Colmerauer [38]. Posteriormente Pereira y Warren presentan, dentro de las gramáticas lógicas, las GCDs [104] como una alternativa a las *redes de transición aumentadas*. Debido a su estrecha relación, el analizador propuesto se implementa directamente sobre el mecanismo de resolución de PROLOG, estableciendo una correspondencia directa entre la gramática y el programa de cláusulas definidas que conforma el analizador. En [104] encontramos el siguiente ejemplo para la gramática:

```
sentence → noun_phrase, verb_phrase.
noun_phrase → determiner, noun, rel_clause.
noun_phrase → name.
verb_phrase → trans_verb, noun_phrase.
verb_phrase → intrans_verb.
rel_clause → [that], verb_phrase.
rel_clause → [ ].
determiner → [every].
determiner → [a].
noun → [man].
noun → [woman].
name → [john].
name → [mary].
trans_verb → [loves].
intrans_verb → [lives].
```

Dicha gramática se traduce en el siguiente programa de cláusulas definidas:

```
sentence(S0,S) :- noun_phrase(S0,S1), verb_phrase(S1,S).
noun_phrase(S0,S) :- determiner(S0,S1), noun(S1,S2), rel_clause(S2,S).
noun_phrase(S0,S) :- name(S0,S).
verb_phrase(S0,S) :- trans_verb(S0,S1), noun_phrase(S1,S).
verb_phrase(S0,S) :- intrans_verb(S0,S).
rel_clause(S0,S) :- connects(S0,that,S1), verb_phrase(S1,S).
rel_clause(S,S).
determiner(S0,S) :- connects(S0,every,S).
determiner(S0,S) :- connects(S0,a,S).
noun(S0,S) :- connects(S0,man,S).
noun(S0,S) :- connects(S0,woman,S).
name(S0,S) :- connects(S0,john,S).
name(S0,S) :- connects(S0,mary,S).
trans_verb(S0,S) :- connects(S0,loves,S).
intrans_verb(S0,S) :- connects(S0,lives,S).
```

La mayoría de intérpretes PROLOG, realizan esta traducción de forma automática. Los problemas de este tipo de analizadores son los propios del mecanismo de resolución de PROLOG: falta de eficiencia reflejada en una complejidad temporal exponencial, al igual que ocurría con los analizadores descendentes con retroceso para GICs, y no-terminación, por la presencia de gramáticas cíclicas o recursivas. Este último problema se ve amplificado al extender el concepto de categoría gramatical a un dominio posiblemente infinito.

Con respecto a las mejoras sobre el análisis de GCDs, podemos señalar parte de aquellas aplicadas sobre los propios intérpretes PROLOG, como el uso de las técnicas de programación dinámica [142, 18, 81, 181]. Otras mejoras se basan en considerar las GCDs como un caso general de las GICs y adaptar los analizadores existentes para éstas últimas. En [105], Pereira y Warren presentan una generalización del algoritmo de Earley conocida como *deducción de Earley*[2]. En este sentido, Tomita también presenta en [145] una evolución del algoritmo LR generalizado para *gramáticas léxico-funcionales* adaptable al caso de las GCDs. Como propuestas similares señalaremos [49] para el caso del algoritmo de Earley o [33] para una versión del algoritmo de Tomita.

Relacionado con las propuestas anteriores, el sistema *SLR Inference* de Rosenblueth y Peralta [116, 117], en lugar del procedimiento de prueba de PROLOG emplea un sistema de inferencia basado en los analizadores sintácticos de la familia LR. Si bien este sistema de inferencia es apropiado para cualquier tipo de programa lógico con *modos fijos*[3], los propios autores indican que los mejores resultados se obtienen cuando se aplica al análisis sintáctico de GCDs. Como principal inconveniente destacaremos que la gramática debe estar en *forma encadenada*, es decir, las cláusulas deben tener uno de los siguientes formatos:

$$a_1(t, t'). \qquad ,var(t') \subseteq var(t)$$
$$a_0(X_0, X_n) \rightarrow b_1(X_0, X_1), b_2(X_1, X_2), \ldots, b_n(X_{n-1}, X_n).$$

Y el símbolo inicial debe ser de la forma: $s(x, Z)$, donde x es un término instanciado. Aunque los autores presentan una transformación análoga a la introducción de *símbolos de copia* [183] en las *gramáticas de atributos*[4] para la evaluación de atributos heredados durante un análisis sintáctico ascendente, esta transformación se basa en el modo de los argumentos de los predicados y, consecuentemente, no parece viable su realización automática. A partir de la forma encadenada, se ignoran los argumentos de los predicados, obteniendo el *esqueleto independiente del contexto* de la gramática sobre la cual se construye en autómata de tipo LR. Por último, incorporando una operación de unificación al intérprete del autómata, obtenemos el sistema de inferencia final.

Una alternativa similar es el sistema AID [99]. En este caso, el entorno se basa en la construcción de un autómata SLR(1) a partir del esqueleto independiente del contexto de la gramática. Tanto en AID como en SLR Inference, el tratamiento del no determinismo en los autómatas correspondientes se realiza mediante el mecanismo de

[2] En la terminología original, *parsing deduction* o *Earley deduction*.

[3] Del inglés, *fixed modes*.

[4] Para más detalles sobre la relación entre las gramáticas de atributos y la programación lógica, ver [42].

retroceso de PROLOG, con la consiguiente penalización sobre la complejidad temporal. De la misma forma, también se han propuesto extensiones de los algoritmos LC al caso de las GCDs [90, 96].

2.3 Otros aspectos

A continuación analizamos otros aspectos no tratados en las secciones anteriores y que hemos considerado importantes en el desarrollo de analizadores sintácticos.

2.3.1 Análisis sintáctico como deducción

En [83] Lang plantea la necesidad de establecer un marco formal común para la definición de analizadores sintácticos. Al mismo tiempo, existe un gran interés por conocer cómo derivar un algoritmo de análisis a partir de otro [97, 91, 132, 11, 8]. Tanto Sikkel [132] como Shieber et al. [131] presentan dos marcos para la descripción a alto nivel de algoritmos de análisis. Ambas propuestas son similares, si bien en [131] se hace mayor hincapié en la metáfora del análisis sintáctico como un proceso deductivo. El uso de estos marcos descriptivos aporta varias ventajas como pueden ser: facilitar el estudio de las relaciones existentes entre diferentes algoritmos, facilitar el prototipado y desarrollo de otros nuevos, o posibilitar la comparación de analizadores en términos de eficiencia.

2.3.2 Programación dinámica y tabulación

La programación dinámica [23] se ha convertido en una técnica estándar en el tratamiento de problemas con un alto grado de redundancia. Un ejemplo típico es la función de Fibonacci:

```
fib(n+2) = fib(n+1) + fib(n)
fib(2) = fib(1) = 1
```

La definición recursiva que acabamos de presentar implica la invocación repetida de la función para valores intermedios de n:

n	5	4	3	2	1
nº llamadas	1	1	2	3	5

En lugar de repetir dichos cálculos, mantendremos una tabla de objetos con las soluciones intermedias. Esta técnica ha dado lugar a numerosos trabajos sobre su aplicación a la programación lógica [141, 142, 18, 82, 81, 178, 179, 180, 181, 108] y al análisis sintáctico [60, 187, 69, 47, 85, 83, 146, 147, 164, 21, 172, 8, 46, 94], entre otros.

Otros autores no aplican la programación dinámica en análisis sintáctico mediante la construcción de una tabla de objetos, sino mediante un grafo etiquetado. Los nodos representan los símbolos de la cadena de entrada, y los arcos constituyen los resultados intermedios de la misma forma que las entradas de la tabla. El algoritmo procede

combinando arcos para dar lugar a otros nuevos. Los analizadores resultantes se suelen denominar *analizadores sintácticos por grafo*[5] [70, 143, 139]. Estas mismas ideas están presentes en el analizador del formalismo gramatical PATR [129]. El principal problema, tal y como apuntan los propios autores, es la estrategia de control del algoritmo. Falcone et al. [50] proponen el uso de una *agenda* capaz de instrumentar diferentes estrategias y Wirén [184], una versión modular de PATR independiente de la estrategia de control.

2.3.3 Bosque de análisis

Un aspecto de crucial importancia en el análisis sintáctico no determinista de GICs es la construcción del *bosque de análisis* como resultado. Por motivos de eficiencia y mejora de la complejidad espacial es deseable emplear algún tipo de estructura que permita compartir las partes comunes [25] del bosque. Dicha estructura puede ser un grafo Y/O, lo que nos permite resolver el problema de la compartición de la cola en una lista de nodos hijos [165], y su extensión a otros formalismos gramaticales [82, 81]. Asimismo, otro aspecto tratado en [165] es la relación entre la compartición de estructuras en el bosque de análisis y la compartición de cálculos durante el análisis debidos a la aplicación de técnicas de tabulación.

2.4 Análisis robusto

Podemos considerar que un sistema informático es *robusto* si es capaz de recuperarse ante condiciones imprevistas. Aplicada al análisis sintáctico, esta característica es muy importante en algunos sistemas, especialmente en el procesamiento del lenguaje natural, donde es necesario obtener un análisis incluso cuando las cadenas de entrada no pertenecen al lenguaje generado por la gramática. En este sentido podemos distinguir entre cadenas que realmente no pertenecen al lenguaje y cadenas que no son reconocidas por la gramática. En el primer caso, el error está en la propia cadena de entrada y en el segundo en la gramática.

A la hora de robustecer los analizadores sintáctico encontramos varias aproximaciones. Algunos autores optan por modificar un algoritmo de análisis existente [130, 145, 85], emplear modelos estadísticos [58], o desarrollar analizadores centrados a este tipo de situación [1, 2]. Otra aproximación posible consiste en añadir al analizador la capacidad de corregir los errores de la cadena de entrada [152, 153].

[5] En terminología anglosajona, *chart parsers*.

CAPÍTULO 3

GENERALIDADES DEL ANÁLISIS SINTÁCTICO

El objeto de este capítulo es introducir conceptos y definiciones que serán empleados con profusión en el resto de la memoria, así como algunos algoritmos de análisis cuyo conocimiento previo se ha considerado esencial para la comprensión posterior de los contenidos expuestos.

3.1 Conceptos previos

En esta sección se desarrollan los conceptos y definiciones a los que se recurre constantemente en el estudio del análisis sintáctico. A mayores, la exposición de los mismos resulta interesante puesto que establece la notación que se usará en los siguientes capítulos.

3.1.1 Lenguajes y gramáticas

El concepto de análisis sintáctico no se entiende si no es ligado a los de lenguaje y gramática. Para llegar a ellos, necesitamos en primer lugar la noción de *alfabeto*. Consideraremos que un alfabeto es un conjunto de *símbolos* y una *cadena* sobre un alfabeto, cualquier secuencia de cero o más símbolos del alfabeto. Por ejemplo, cualquier número binario es una cadena sobre el alfabeto $\{0, 1\}$. En el caso de que la cadena contenga cero símbolos, la representaremos como ϵ y nos referiremos a ella como *cadena vacía*. De manera informal definiremos un *lenguaje* sobre un alfabeto Σ como un conjunto de cadenas sobre Σ.

Esta definición es suficientemente general como para englobar cualquier tipo de lenguaje que deseemos tratar. En particular destacaremos el *lenguaje natural*, término con el que nos referiremos a los lenguajes de comunicación humana, que destacan por su complejidad tanto de definición como de tratamiento computacional. Nótese, también, que cuando nos referimos a este tipo de lenguaje con frecuencia preferiremos los términos *letra* o *carácter* en lugar de símbolo, y *frase* o *palabra* en lugar de cadena.

Una vez introducido el concepto, la siguiente cuestión que se plantea de forma lógica es cómo se representa un lenguaje. Siguiendo con el ejemplo anterior, podemos decir que el lenguaje de los números binarios es cualquier cadena sobre el alfabeto $\{0, 1\}$, ó $L_b = \{X^+ | X \in \{0, 1\}\}$. Por otro lado, si el lenguaje que queremos representar

consta de un número limitado, y convenientemente reducido, de cadenas, nos bastaría simplemente con enumerarlas todas. Sin embargo, ¿qué ocurre con los lenguajes de programación o los lenguajes naturales, cuyo número de cadenas es posiblemente infinito? En este caso es conveniente describir el lenguaje mediante algún mecanismo que lo genere. Este formalismo descriptivo ha de tener, además, un tamaño finito. Los generadores de lenguajes son las *gramáticas*.

DEFINICIÓN 3.1 (gramática)
Una gramática *es una 4-tupla* $\mathcal{G} = (N, \Sigma, P, S)$, *donde:*

- Σ *es el* alfabeto *finito de la gramática o conjunto finito de* símbolos terminales, *o* palabras, *o* categorías léxicas
- N *es un conjunto finito de* símbolos no terminales, *o* variables, *o* categorías sintácticas, $N \cap \Sigma = \emptyset$
- P *es un subconjunto finito de* $(N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$ *a cuyos elementos denominaremos* producciones, reglas, *o* reglas de producción
- $S \in N$ *es el* símbolo inicial, *o* axioma *de la gramática*

En lo sucesivo, para unificar criterios y facilitar la lectura, utilizaremos las siguientes convenciones en la representación de los elementos de una gramática:

- $V = N \cup \Sigma$, *es el conjunto total de símbolos.*
- $a, b, c, \ldots \in \Sigma$, *son símbolos terminales.*
- $A, B, C, \ldots \in N$, *símbolos no terminales.*
- $X, Y, Z, \ldots \in V$, *símbolos arbitrarios, terminales y no terminales.*
- $u, v, w, \ldots \in \Sigma^*$, *cadenas de terminales.*
- $u_{1..n}$, *una cadena* $u \in \Sigma^*$ *de longitud* n.
- $u_{i:j}$, *la subcadena de* $u_{1..n} \in \Sigma^*$ *que va del carácter en la posición* i *al carácter en la posición* j.
- u_i, *el carácter de* $u \in \Sigma^*$ *en la posición* i.
- $\alpha, \beta, \gamma, \ldots \in V^*$, *cadenas arbitrarias de símbolos terminales y no terminales.*
- ε, *la cadena vacía.*

El símbolo inicial, S, *es un símbolo distinguido de entre el conjunto de símbolos no terminales. Frecuentemente se prefiere representar las producciones* $(\alpha, \beta) \in P$ *como* $\alpha \longrightarrow \beta \in P$. *Señalaremos como caso particular las* producciones-ε *en las que la parte derecha está vacía,* $\alpha \longrightarrow \varepsilon$. □

Grosso modo, las cadenas del lenguaje se construyen partiendo del símbolo inicial, siendo las producciones las encargadas de describir cómo. Empleando las reglas de producción de la gramática, podemos generar distintas secuencias de símbolos terminales y no terminales a partir del símbolo inicial. Llamaremos *formas sentenciales* a estas secuencias.

DEFINICIÓN 3.2 (forma sentencial)
Sea $\mathcal{G} = (N, \Sigma, P, S)$ *una gramática, entonces:*

1. S *es una* forma sentencial.
2. *Si* $\alpha\beta\gamma$ *es una* forma sentencial *y* $\beta \to \delta \in P$, *entonces* $\alpha\delta\gamma$ *también es una* forma sentencial. □

Resulta conveniente considerar una clase especial de forma sentencial, aquella en la que únicamente existen símbolos terminales. Este tipo de formas gramaticales pertenecen al conjunto de cadenas definidas sobre el alfabeto de la gramática.

DEFINICIÓN 3.3 (frase)
Dada una gramática $\mathcal{G} = (N, \Sigma, P, S)$, *denominaremos* frase *generada por una gramática a cualquier forma sentencial que únicamente contenga símbolos terminales.* □

A partir de la definición de forma sentencial podemos extraer el significado intuitivo de las producciones. Dada una cadena de símbolos y una producción, podemos reemplazar o reescribir el símbolo o símbolos que encajan con la parte izquierda de la producción por la parte derecha de la misma.

EJEMPLO 3.1
Consideremos la gramática definida por las producciones:

$$\begin{array}{ll} (1) & S \to 0\,A\,1 \\ (2) & 0\,A \to 0\,0\,A\,1 \\ (3) & A \to \varepsilon \end{array}$$

Partiendo del axioma S, *usaremos las producciones para realizar las siguientes reescrituras:*

Formal sentencial	*Regla aplicada*
S	
$0A1$	*(1)*
$00A11$	*(2)*
$000A111$	*(2)*
000111	*(3)*

■

A continuación definiremos la aplicación de producciones en $\mathcal{G}$ descrita en el ejemplo anterior como una relación $\underset{\mathcal{G}}{\Rightarrow}$ sobre $(N \cup \Sigma)^*$, aunque siempre que no exista confusión posible, usaremos la notación $\Rightarrow$.

DEFINICIÓN 3.4 (derivación directa)
Sea $\mathcal{G} = (N, \Sigma, P, S)$ *una gramática, definimos una* derivación directa *o* derivación en un solo paso, $\underset{\mathcal{G}}{\Rightarrow}$, *como sigue: si* $\alpha\beta\gamma \in (N \cup \Sigma)^*$ *y* $\beta \to \delta \in P$, *entonces* $\alpha\beta\gamma \underset{\mathcal{G}}{\Rightarrow} \alpha\delta\gamma$. □

El concepto de derivación directa se extiende de forma natural mediante los cierres transitivo y reflexivo.

DEFINICIÓN 3.5 (derivación indirecta)
Sea $\mathcal{G} = (N, \Sigma, P, S)$ *una gramática, diremos que* $\alpha\beta\gamma$ deriva indirectamente $\alpha\delta\gamma$ *si y sólo si:*

- $\beta \underset{\mathcal{G}}{\Rightarrow} \delta_1 \underset{\mathcal{G}}{\Rightarrow} \delta_2 \ldots \underset{\mathcal{G}}{\Rightarrow} \delta_n \Rightarrow \delta$, *que notaremos* $\alpha\beta\gamma \overset{+}{\underset{\mathcal{G}}{\Rightarrow}} \alpha\delta\gamma$, *o bien*
- $\beta = \delta$ *ó* $\alpha\beta\gamma \overset{+}{\underset{\mathcal{G}}{\Rightarrow}} \alpha\delta\gamma$, *que notaremos* $\alpha\beta\gamma \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \alpha\delta\gamma$

En caso de conocer el número exacto, k, de derivaciones directas, también se usará la notación $\alpha\beta\gamma \overset{k}{\underset{\mathcal{G}}{\Rightarrow}} \alpha\delta\gamma$, □

Nos referiremos a una secuencia de k derivaciones en un solo paso, $\alpha_0 \Rightarrow \cdots \Rightarrow \alpha_k$, como una *derivación de longitud k* o *derivación en k pasos*. Nótese que las formas sentenciales son aquellas que podemos derivar a partir del símbolo inicial de la gramática, y que el conjunto de todas las frases generadas por una gramática forma a su vez un lenguaje sobre el alfabeto de la gramática.

DEFINICIÓN 3.6 (lenguaje generado por una gramática)
Sea $\mathcal{G} = (N, \Sigma, P, S)$ *una gramática, el* lenguaje generado por la gramática, *es el conjunto* $L(\mathcal{G}) = \left\{ w | w \in \Sigma^*,\ S \overset{*}{\Rightarrow} w \right\}$. □

En general, cuanto más complejas sean las producciones, más complejo podrá ser el lenguaje generado. Precisamente, en un intento de clasificar los lenguajes en función de las gramáticas que los generan, Chomsky [36] propuso una jerarquía estructurada en cuatro clases.

Jerarquía de Chomsky

Como hemos dicho, a finales de los 50, Chomsky [36] propone una jerarquía en la que se clasifican las gramáticas formales y sus lenguajes asociados. Cada nivel de la jerarquía incluye los lenguajes del nivel anterior. Como se describe a continuación, la diferencia estriba en la forma de sus producciones:

- *Gramáticas regulares*. En este caso, las producciones obligatoriamente son de la forma: $A \to x$ ó $A \to xB$. Este tipo de producciones nos asegura que las formas sentenciales contendrán a lo sumo un único símbolo no terminal. Los lenguajes que pueden ser generados por este tipo de gramáticas se denominan *lenguajes regulares*.

- *Gramáticas independientes del contexto* (GICs). Las producciones forzosamente tienen un único símbolo, no terminal, en la parte izquierda: $A \rightarrow \beta$. De esta forma, a la hora de realizar un paso de derivación directo, es posible decidir qué símbolo no terminal queremos reescribir independientemente del contexto que lo rodea. Los lenguajes que pueden ser generados por este tipo de gramáticas se denominan *lenguajes independientes del contexto* (LICs).

- *Gramáticas dependientes del contexto.* La parte izquierda de las producciones pueden contener cualquier combinación de símbolos terminales y no terminales, siempre y cuando sea de longitud menor o igual que la parte derecha. De esta forma aseguramos que al aplicar una derivación sobre una forma sentencial obtendremos otra forma sentencial de igual o mayor longitud, nunca menor. Las producciones siguen el patrón $\alpha \rightarrow \beta$, $|\alpha| \leq |\beta|$, siendo $|\alpha|$ la longitud de α, esto es, el número de símbolos en α. Los lenguajes que pueden ser generados por este tipo de gramáticas se denominan *lenguajes sensibles al contexto.*

- *Gramáticas con estructura de frase.* No existe ninguna restricción sobre las producciones. Los lenguajes que pueden ser generados por este tipo de gramáticas se denominan *lenguajes recursivamente enumerables.*

3.1.2 Gramáticas independientes del contexto

Debido a su menor complejidad, los lenguajes regulares e independientes del contexto han sido ampliamente estudiados y empleados en la práctica, de ahí el interés en el estudio y formalización de los mismos.

EJEMPLO 3.2
Introduciremos una GIC que describe el lenguaje de las expresiones aritméticas, expresado por las reglas:

$$P = \left\{ \begin{array}{l} S \rightarrow S + S \\ S \rightarrow S * S \\ S \rightarrow (S) \\ S \rightarrow a \end{array} \right\}$$

Con objeto de simplificar la exposición, se han sustituido los números por el terminal "a". La dos primeras producciones indican que una expresión aritmética puede escribirse como una suma de dos expresiones aritméticas, o, respectivamente, como una multiplicación. La tercera producción representa la reescritura como una expresión entre paréntesis y la última como un número. A continuación mostramos para esta gramática la derivación $S \overset{5}{\Rightarrow} 2 * 3 + 4$, *descompuesta en 5 derivaciones directas:*

$$S \Rightarrow S + S \Rightarrow S * S + S \Rightarrow 2 * S + S \Rightarrow 2 * 3 + S \Rightarrow 2 * 3 + 4$$ ■

Es frecuente que las gramáticas presenten varias producciones con la misma parte izquierda $A \rightarrow \alpha_1, \ldots A \rightarrow \alpha_n$, a las que nos referiremos como *alternativas* de A, y opcionalmente representaremos como $A \rightarrow \alpha_1 | \ldots | \alpha_n$

EJEMPLO 3.3
Opcionalmente podemos representar la gramática del ejemplo 3.2 como:

$$S \to S + S \mid S * S \mid (S) \mid a$$

■

En lo que respecta a las derivaciones resulta conveniente, tal y como veremos, realizar las mismas siguiendo siempre un mismo criterio direccional, para lo que definimos las derivaciones por la derecha, o por la izquierda.

DEFINICIÓN 3.7 (derivación directa por la derecha (resp. por la izquierda))
Dada una gramática $\mathcal{G} = (N, \Sigma, P, S)$ y una derivación directa $\alpha A \beta \Rightarrow \alpha\gamma\beta$, diremos que se trata de una derivación directa por la derecha, *(resp.* derivación directa por la izquierda*) si y sólo si:*

- $A \to \gamma \in P$
- $\beta \in \Sigma^*$ *(resp.* $\alpha \in \Sigma^*$*)*

Y emplearemos la notación $\Rightarrow_{rm}$ (resp. $\Rightarrow_{lm}$). De las dos, escogeremos como derivación canónica *la derivación por la derecha, de forma que siempre que no se indique lo contrario, consideraremos que una derivación es canónica.* □

De forma análoga a como extendíamos los derivaciones directas a derivaciones indirectas, haremos lo mismo para las derivaciones indirectas por la derecha en 1 ó más pasos, $\overset{+}{\Rightarrow}_{rm}$; en 0 ó más pasos, $\overset{*}{\Rightarrow}_{rm}$; y en k pasos $\overset{k}{\Rightarrow}_{rm}$, repitiendo el proceso para las derivaciones indirectas por la izquierda, $\overset{+}{\Rightarrow}_{lm}$, $\overset{*}{\Rightarrow}_{lm}$, $\overset{k}{\Rightarrow}_{lm}$.

EJEMPLO 3.4
Continuando con la gramática del ejemplo 3.2 realizamos dos derivaciones, por la derecha y por la izquierda:

$$S \Rightarrow_{lm} (S) \Rightarrow_{lm} (S + S) \Rightarrow_{lm} (a + S) \Rightarrow_{lm} (a + a)$$
$$S \Rightarrow_{rm} (S) \Rightarrow_{rm} (S + S) \Rightarrow_{rm} (S + a) \Rightarrow_{rm} (a + a)$$

■

Tal y como muestra el ejemplo anterior, para una misma forma sentencial es probable que la derivación por la derecha sea diferente de la derivación por la izquierda, aún involucrando a las mismas producciones. De ahí la necesidad de considerar el concepto de derivación canónica, asegurando una correcta definición del concepto de ambigüedad gramatical.

DEFINICIÓN 3.8 (gramática ambigua)
Diremos que una gramática $\mathcal{G} = (N, \Sigma, P, S)$ es una gramática ambigua *si y sólo si $\exists x \in L(\mathcal{G})$ tal que hay al menos dos derivaciones canónicas $S \overset{*}{\Rightarrow} x$ distintas.* □

Esta noción de ambigüedad se extiende de forma natural a los lenguajes.

DEFINICIÓN 3.9 (lenguaje ambiguo)
Diremos que un lenguaje L no es ambiguo *si y sólo si existe una gramática $\mathcal{G}$ no ambigua tal que $L(\mathcal{G}) = L$. En caso contrario diremos que L es un* lenguaje ambiguo. □

Por definición, el hecho de que un lenguaje sea ambiguo implica que todas las gramáticas que lo generan lo han de ser. Sin embargo, lo contrario no es cierto. Es posible escribir gramáticas ambiguas para generar lenguajes no ambiguos.

EJEMPLO 3.5
*La gramática de las expresiones aritméticas del ejemplo 3.2 es ambigua puesto que la cadena "2 + 3 * 4" tiene dos derivaciones canónicas distintas:*

$$S \Rightarrow S+S \Rightarrow S+S*S \Rightarrow S+S*4 \Rightarrow S+3*4 \Rightarrow 2+3*4$$
$$S \Rightarrow S*S \Rightarrow S*4 \Rightarrow S+S*4 \Rightarrow S+3*4 \Rightarrow 2+3*4$$

Sin embargo, el lenguaje de las expresiones aritméticas no es ambiguo puesto que es posible definir una gramática no ambigua que lo genere. Dicha gramática puede ser la dada por las reglas siguientes:

$$\begin{array}{ll} E \to E+T & E \to T \\ T \to T*F & T \to F \\ F \to (E) & F \to a \end{array}$$

*En esta gramática, por ejemplo, la única derivación canónica posible para obtener la cadena "2 + 3 * 4" es:*

$$E \Rightarrow E+T \Rightarrow E+T*F \Rightarrow E+T*4 \Rightarrow E+F*4 \Rightarrow$$
$$\Rightarrow E+3*4 \Rightarrow T+3*4 \Rightarrow F+3*4 \Rightarrow 2+3*4$$ ■

Hasta el momento hemos representado las derivaciones, tal y como han sido definidas, como una secuencia de derivaciones directas. Alternativamente, podemos optar por compactar estas secuencias en *árboles de derivación*.

DEFINICIÓN 3.10 (árbol de derivación)
Dada una GIC $\mathcal{G} = (N, \Sigma, P, S)$, *un* árbol de derivación *es un árbol* $\mathcal{D}$ *ordenado y etiquetado si y sólo si:*

1. *Si* X *es una etiqueta de un nodo, entonces* $X \in \Sigma \cup N$.
2. *Si* $\mathcal{D}$ *es un nodo etiquetado con un símbolo no terminal* A, *y sus hijos son, ordenados de izquierda a derecha,* $X_1 \cdots X_n$, *entonces* $A \to X_1 \cdots X_n$ *es una producción de la gramática.* □

Tal y como indica la definición, los nodos de los árboles de derivación están etiquetados con los símbolos de la gramática, y los subárboles compuestos por un nodo y sus descendientes inmediatos ordenados de izquierda a derecha forman una producción de la gramática. El tipo de derivación aplicada determina un recorrido determinado del árbol. Así una derivación por la derecha (resp. por la izquierda) se corresponden con un recorrido de derecha a izquierda (resp. de izquierda a derecha).

EJEMPLO 3.6
Las derivaciones por la derecha e izquierda del ejemplo 3.4 quedan representadas por el árbol de la figura 3.1. ■

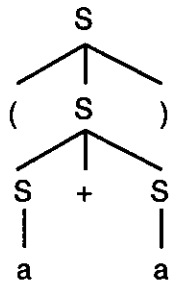


Figura 3.1: Ejemplo de árbol de derivación

Puesto que el recorrido de derecha a izquierda (o de izquierda a derecha) de un árbol es único, cada derivación canónica se corresponderá con un único árbol de derivación, y viceversa. Así, en presencia de gramáticas ambiguas es posible construir árboles de derivación distintos cuyas fronteras sean idénticas.

EJEMPLO 3.7
La figura 3.2 muestra dos árboles de derivación con idéntica frontera. Dichos árboles se corresponden con las derivaciones canónicas del ejemplo 3.5. ■

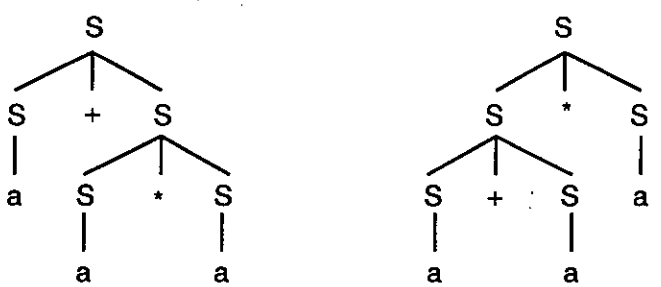


Figura 3.2: Árboles de derivación y ambigüedad

En el caso en que representemos varios árboles de derivación, nos referiremos al conjunto como *bosque de derivación*. Frecuentemente los árboles implicandos tendrán subárboles comunes, por lo que resulta conveniente compartir dichas partes representando el bosque como un grafo Y/O al que denominaremos *bosque Y/O*.

EJEMPLO 3.8
La figura 3.3 muestra un bosque Y/O para las derivaciones:

$$S \Rightarrow NF \Rightarrow Nf \Rightarrow ABf \Rightarrow Abf \Rightarrow aabf$$

$$S \Rightarrow NF \Rightarrow Nf \Rightarrow CDf \Rightarrow Cabf \Rightarrow aabf$$ ■

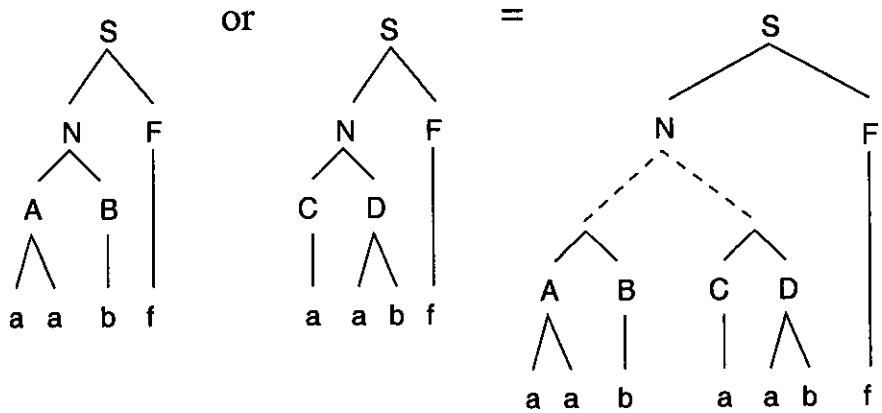


Figura 3.3: Bosque Y/O de análisis

Del hecho de que un árbol de derivación represente una derivación sobre la gramática se sigue inmediatamente que, para cualquier árbol cuya raíz sea el símbolo inicial, su frontera constituye una forma sentencial de la gramática.

DEFINICIÓN 3.11 (árbol de análisis)
Dada una GIC $\mathcal{G} = (N, \Sigma, P, S)$ *diremos que un árbol de derivación es un* árbol de análisis *si y sólo si:*

- *La etiqueta de la raíz es* S.
- *La frontera del árbol es* $w \in L(\mathcal{G})$. □

Una vez llegados a este punto necesitaremos un mecanismo, viable desde el punto de vista computacional, que establezca la gramaticalidad de una oración, es decir, que reconozca si la oración o cadena de entrada pertenece al lenguaje generado por la gramática.

3.1.3 El problema del análisis sintáctico

El problema del análisis sintáctico consiste en diseñar un algoritmo tal que dadas una gramática y una frase de entrada cualesquiera, compruebe si dicha frase pertenece al lenguaje generado por la gramática y, en caso afirmativo, genere una representación sintáctica apropiada. Estos algoritmos se conocen como *analizadores sintácticos*. El análisis sintáctico incluye el problema del reconocimiento sintáctico que hace referencia al desarrollo de *reconocedores sintácticos*. A diferencia de los analizadores, éstos se limitan a determinar si la frase de entrada pertenece o no al lenguaje generado por la gramática, sin generación de estructuras sintácticas de salida.

Una vez definido cuál es el cometido de los analizadores y reconocedores sintácticos, resta por concretar cuál es el resultado que debe proporcionar. En lo que respecta a los reconocedores, es suficiente con una respuesta del tipo la frase de entrada pertenece o la frase de entrada no pertenece al lenguaje generado por la gramática. Sin

embargo, decíamos que un analizador debe, a mayores, proporcionar una representación sintáctica apropiada. Esta representación debe indicar la secuencia de reglas que debemos aplicar para construir una derivación de la frase de entrada a partir del símbolo inicial de la gramática. Como hemos comentado, una forma habitual de realizar esta representación es mediante un árbol de análisis.

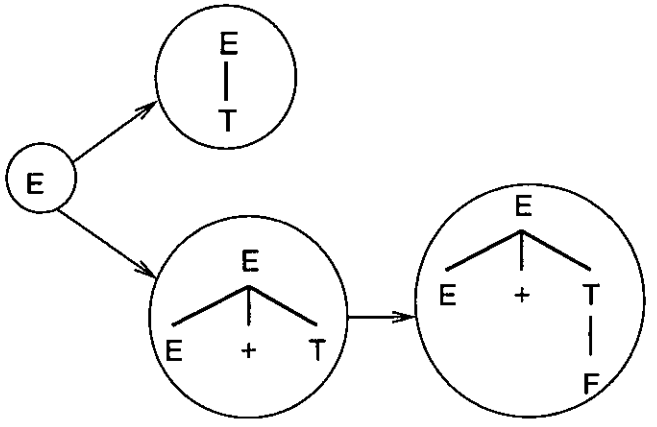


Figura 3.4: Árboles de derivación en espacio de estados

El problema del análisis sintáctico descrito es asimilable al problema de la búsqueda en un espacio del estados [98]. En este caso, los estados intermedios se corresponden con distintos árboles de derivación, mientras que los estados objetivo serán los árboles de análisis. La figura 3.4 muestra parte del espacio de estados para la versión de la gramática de las expresiones aritméticas dada en el ejemplo 3.5 El proceso de búsqueda es no determinista debido a:

- En la frontera de cada árbol de derivación pueden existir varios símbolos no terminales cuyos nodos hemos de expandir. Este problema es análogo al que nos llevó a definir las derivaciones por la derecha e izquierda, y las derivaciones canónicas. De la misma forma los diferentes algoritmos de análisis definen un orden sistemático para la expansión de los nodos en un árbol de derivación.

  A este respecto podemos señalar dos modos básicos de funcionamiento, *ascendente* y *descendente*. El primero aplica de forma sistemática el concepto de derivación por la derecha, construyendo el árbol de análisis desde la raíz hacia las hojas, mientras que el segundo emplea únicamente las derivaciones por la izquierda, actuando en sentido contrario. Como veremos más adelante, existen algoritmos descendentes puros y algoritmos ascendentes puros, aunque en la práctica es frecuente recurrir a estrategias mixtas que combinan ambos métodos de análisis.

- La existencia de alternativas para un símbolo no terminal A, inclusive en gramáticas no ambiguas. Cualquiera de las alternativas se puede emplear en la expansión de los nodos etiquetados A, dando lugar a derivaciones distintas. En el caso

de las gramáticas no ambiguas, a lo sumo sólo una de estas ramas de derivación dará lugar a un árbol de análisis.

Algunas de las aproximaciones más populares descritas para enfrentarse a este problema son la búsqueda en anchura explorando todas las alternativas simultáneamente, y la búsqueda en profundidad, escogiendo siempre una única alternativa. La búsqueda en anchura presenta la ventaja de dar siempre con la solución, si esta existe, pero en su contra está la explosión combinatoria del número de estados a visitar lo que la hace poco atractiva desde el punto de vista computacional. Por su parte la búsqueda en profundidad tiene a su favor un consumo reducido de recursos computacionales, en contra está el hecho de que seleccionar la alternativa incorrecta provoca que no se encuentre la solución, lo que a su vez provoca que en la práctica se combine con un mecanismo de retroceso, dando lugar a los problemas que veremos a continuación. La figura 3.5 esquematiza ambos métodos de búsqueda. En la práctica se optará por soluciones que combinen ambas aproximaciones.

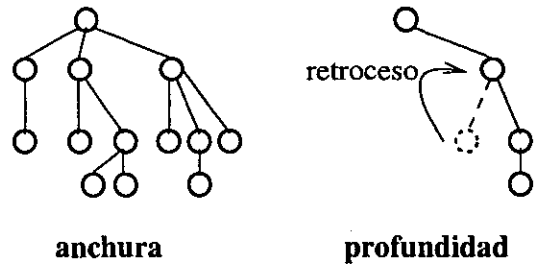


Figura 3.5: Búsqueda en anchura vs. en profundidad

A mayores del problema del no determinismo, se presenta el problema de la ambigüedad de las gramáticas y lenguajes. Dicha ambigüedad implica que pueden existir análisis diferentes para una misma cadena de entrada, lo cual se traduce en diferentes árboles de análisis tal y como veíamos en el ejemplo 3.7. Así pues, el desafío planteado es encontrar todos los análisis posibles y representarlos.

Un tercer problema relacionado con la ambigüedad de las gramáticas y el no determinismo del proceso de análisis es la *recursividad* de las gramáticas.

DEFINICIÓN 3.12 (recursividad)
Sea A un símbolo no terminal en una GIC $\mathcal{G} = (N, \Sigma, P, S)$, diremos que es recursivo si y sólo si $A \stackrel{+}{\Rightarrow} \alpha A \beta$. En particular:

- *Si $\alpha = \varepsilon$, entonces diremos que A es* recursivo por la izquierda.
- *Si $\beta = \varepsilon$, entonces diremos que A es* recursivo por la derecha.

Diremos que una gramática $\mathcal{G} = (N, \Sigma, P, S)$ con al menos un no terminal recursivo por la izquierda (resp. derecha) es recursiva por la izquierda (resp. derecha). □

Las gramáticas recursivas pueden provocar la no-terminación del proceso de análisis. Veamos, por ejemplo, la siguiente situación: el nodo a expandir está etiquetado con A, empleamos una regla del tipo $A \rightarrow A\alpha$, que genera una recursividad por la izquierda, y sistemáticamente empleamos el recorrido en preorden para seleccionar el siguiente nodo a expandir. El resultado es que de nuevo el nodo a expandir está etiquetado con A y se repite el proceso. La siguiente derivación ilustra esta idea:

$$\alpha A\gamma \Rightarrow \alpha A\beta\gamma \Rightarrow \alpha AA\beta\gamma \Rightarrow \ldots$$

DEFINICIÓN 3.13 (ciclo)
Dada una gramática recursiva $\mathcal{G} = (N, \Sigma, P, S)$, diremos que tiene ciclos *si existe alguna derivación $A \stackrel{+}{\Rightarrow} A$. Llamaremos a las derivaciones $A \stackrel{+}{\Rightarrow} A$* ciclos *o* derivaciones cíclicas □

En el caso particular de las gramáticas con ciclos, a los problemas de no-terminación se suma el problema de que para una cadena de entrada pueden existir infinitas derivaciones distintas.

EJEMPLO 3.9
Tomaremos como ejemplo de gramática con ciclos la siguiente gramática de los paréntesis bien balanceados:

$$\begin{aligned} S &\rightarrow SS \\ S &\rightarrow (S) \\ S &\rightarrow \epsilon \end{aligned}$$

Encadenando la primera y, a continuación, la última de las producciones podemos construir una derivación de longitud infinita:

$$S \Rightarrow SS \Rightarrow S \Rightarrow SS \Rightarrow S \ldots$$ ■

Como vemos en el ejemplo, al hablar de infinitas derivaciones distintas, también estamos hablando de derivaciones canónicas, lo que significa infinitos árboles de análisis distintos para una misma cadena de entrada. En la derivación de una frase, todo ciclo implica una ambigüedad en la gramática, siendo una alternativa continuar el ciclo y la/s otra/s avanzar en el proceso de análisis. Al igual que hicimos al tratar las ambigüedades, preferiremos una representación compactada en un único bosque Y/O. Nótese que a diferencia del caso anterior, ahora en el nodo O una de las alternativas volverá sobre alguno de sus ancestros, en cuyo caso indicaremos el sentido del recorrido del bosque. Nótese también que en este caso no estamos tratando un árbol en sentido estricto, sino un grafo cíclico.

EJEMPLO 3.10
La figura 3.6 muestra el bosque de análisis Y/O con ciclos correspondiente a la derivación del ejemplo 3.9. ■

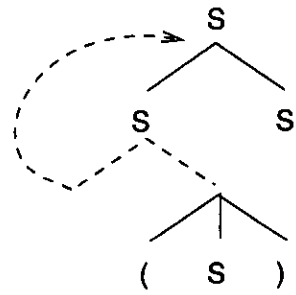


Figura 3.6: Bosque Y/O con ciclos

Formas normalizadas

Para poder realizar su labor, algunos algoritmos de análisis necesitan que la gramática sobre la que trabajan tenga una forma especial, fundamentalmente con el objeto de mejorar las prestaciones computacionales y facilitar la demostración de propiedades [187]. En concreto, nos fijaremos en la *formal normal de Chomsky*.

DEFINICIÓN 3.14 (forma normal de Chomsky)
Diremos que una GIC $\mathcal{G} = (N, \Sigma, P, S)$ *está en* forma normal de Chomsky *si y sólo si toda producción es de la forma:*

- $A \rightarrow BC$, *ó*
- $A \rightarrow a$ □

Una de las propiedades más interesantes de la forma normal de Chomsky es que para cualquier GIC existe una gramática en forma normal de Chomsky que genera el mismo lenguaje.

Teorema 3.1
Sea $\mathcal{G}$ *una GIC arbitraria, entonces* $\exists \mathcal{G}'$ *GIC en forma normal de Chomsky, tal que* $L(\mathcal{G}) = L(\mathcal{G}')$.

Demostración:
El lector interesado puede encontrala en [3]. □

A continuación ilustramos el resultado anterior mediante un ejemplo.

EJEMPLO 3.11
Volviendo a la gramática del ejemplo 3.2, podemos transformarla en otra gramática, que genera el mismo lenguaje, y que está en forma normal de Chomsky. El conjunto de reglas vendría dado por:

$$\begin{array}{llll} S \rightarrow S\,S_1 & S \rightarrow S\,S_3 & S \rightarrow S_5\,S_6 & S \rightarrow a \\ S_1 \rightarrow S_2\,S & S_3 \rightarrow S_4\,S & S_5 \rightarrow (& \\ S_2 \rightarrow + & S_4 \rightarrow * & S_6 \rightarrow S\,S_7 & \\ & & S_7 \rightarrow) & \end{array}$$

■

Por otra parte, para facilitar la realización de los analizadores se suelen considerar *gramáticas aumentadas*.

DEFINICIÓN 3.15 (gramática aumentada)
Dada una gramática $\mathcal{G} = (N, \Sigma, P, S)$, *decimos que* $\mathcal{G}_a = (N_a, \Sigma_a, P_a, S_a)$ *es su correspondiente* gramática aumentada *si verifica las condiciones siguientes:*

- $N_a = N \cup S'$, $S' \notin N$
- $\Sigma_a = \Sigma$
- $P_a = P \cup \{S' \rightarrow S\}$
- $S_a = S'$ □

Una gramática aumentada es básicamente la gramática original a la que se ha añadido un nuevo símbolo inicial S' y una producción inicial $S' \rightarrow S$. De esta forma se facilita la detección del final del proceso de análisis, ya que éste se corresponde, sin duda, con el reconocimiento de S', una categoría sintáctica que aparece una única vez en el conjunto de reglas, y en una sola localización. Habitualmente, también se considera un nuevo símbolo, \$, que se sitúa al final de la cadena de entrada para marcar su finalización.

DEFINICIÓN 3.16 (símbolo de fin de cadena)
Sea $\mathcal{G} = (N, \Sigma, P, S)$ *una GIC, llamaremos* símbolo de fin de cadena *a un símbolo* \$, $\$ \notin \Sigma \cup N$ *que situaremos al final de cualquier cadena de entrada.* □

El cometido del símbolo de fin de cadena es esencialmente facilitar la detección del final de la misma.

3.2 Algoritmos básicos de análisis sintáctico

Esta sección está dedicada a la descripción de algoritmos de análisis sintáctico cuyo conocimiento se ha considerado esencial bien sea en razón a su simplicidad, o bien porque el paso del tiempo les ha otorgado la categoría de clásicos. En cualquier caso, a estos motivos hemos de añadir que dichos algoritmos se emplearán en capítulos posteriores para ilustrar las ideas, conceptos y técnicas descritas.

3.2.1 Una clasificación simple

Podemos clasificar los algoritmos de análisis en diversos grupos atendiendo a diversos criterios. Señalaremos dos de estas clasificaciones establecidas en función de características sobre las que trabajaremos más adelante. La primera clasificación atiende a la estrategia de tratamiento del no determinismo:

Basados en retroceso. En este caso, el no determinismo se simula mediante un mecanismo de retroceso [3, 98, 99, 77]. Una vez llegados a un punto en que es necesario explorar varias alternativas, se escoge únicamente una, continuando el

análisis. Si en algún momento no es posible llegar a la solución, se retrocede hasta el último punto de no determinismo y se escoge una alternativa diferente, continuando el proceso. Desde el punto de vista computacional el mecanismo de retroceso permite evitar el mantenimiento de varios análisis simultáneos, por lo que la parte correspondiente a las alternativas exploradas con anterioridad pueden ser eliminadas. Por contra, esta estrategia presenta una serie de inconvenientes entre los que destacamos:

- La repetición de cálculos. Los cálculos realizados tras la elección de una alternativa, sobre la que posteriormente se realiza un retroceso, se pierden. Como consecuencia, en caso de que sean necesarios en la siguiente alternativa, se han de calcular de nuevo.
- Realización de cálculos innecesarios. La elección incorrecta de alternativas provoca la realización de cálculos que no contribuyen a la solución.
- Dificultad para localizar todas las soluciones existentes. Cuando existen varios análisis diferentes debidos a la ambigüedad de la gramática, puede ser deseable calcularlos todos. En este caso se hace necesario forzar el retroceso por cada solución encontrada con lo que se agravan los dos problemas anteriores.
- Dificultad para tratar gramáticas cíclicas. El análisis de cierto tipo de estructuras cíclicas involucra el análisis simultáneo de varias alternativas, aspecto éste contrario al mecanismo de retroceso.

Basados en programación dinámica. Mediante técnicas de programación dinámica [23, 179, 181, 48, 187, 69] se almacenan los cálculos intermedios de manera que no sea necesario repetirlos. Esta característica permite compartir cálculos incluso entre los diferentes análisis derivados del no determinismo, solucionando parte de los problemas expuestos en el grupo de algoritmos previo.

En una segunda clasificación consideraremos la dependencia de los algoritmos en la estructura gramatical durante el proceso de análisis sintáctico:

Guiados por la gramática. La elección de alternativas se realiza examinando directamente las reglas de la gramática.

Guiados por control finito. Como característica innovadora respecto a los algoritmos anteriores, existe una fase de pre-procesamiento previa a cualquier análisis. Durante ésta, se examina la gramática construyendo un mecanismo de control. Dicho mecanismo se encargará de la elección de alternativas, evitando en la medida de lo posible la exploración de aquellas que resulten infructuosas para el proceso de análisis.

En una última clasificación, emplearemos como criterio la orientación en el proceso de reconocimiento y construcción sintáctica. Aquí podemos considerar tres orientaciones básicas [3, 59]: ascendente, descendente o estrategias de tipo mixto.

Señalar, además, que existen algoritmos que no se corresponden de forma estricta con algunas de las clases enumeradas.

3.2.2 Un algoritmo descendente interpretado con retroceso

Estos analizadores reciben su nombre por la forma en que construyen el árbol de análisis, desde la raíz hacia las hojas. Para acortar la exposición nos referiremos al nodo que debemos expandir en cada paso como el *nodo activo*, y al símbolo que lo etiqueta como el símbolo a analizar. La idea es que el prefijo de símbolos terminales de la frontera del árbol de análisis coincida en todo momento con algún prefijo de la cadena de entrada. Esto se consigue escogiendo siempre como siguiente nodo activo el nodo hoja más a la izquierda. Para llevar cuenta del prefijo de la cadena de análisis que ya ha sido reconocido usaremos un *puntero de entrada*.

Informalmente[1], el algoritmo descendente procede de la siguiente manera. Partimos de un árbol de derivación que únicamente contiene un nodo activo, etiquetado con el metasímbolo de la gramática y situamos el puntero de entrada al comienzo de la cadena a analizar. A continuación se aplican de forma iterativa los siguientes pasos:

1. Si el nodo activo está etiquetado con el no terminal A, escogemos la primera alternativa $A \rightarrow X_1..X_k$. Añadimos k nodos descendientes directos de A y los etiquetamos $X_1, X_2, \ldots, X_k$. Si $k > 0$, entonces X_1 pasa a ser el nodo activo. Si $k = 0$, el nodo inmediatamente a la derecha de A pasa a ser el nodo activo.

2. Si el nodo activo está etiquetado con el terminal a, entonces se compara con el símbolo indicado por el puntero de entrada. Si coinciden, el nodo activo pasa a ser el nodo inmediatamente a la derecha del actual y se avanza el puntero de entrada. En caso contrario, se deshacen los pasos anteriores, retrocediendo hasta llegar a un punto en que habíamos seleccionado una alternativa para la expansión de un nodo etiquetado A. Seleccionamos otra alternativa diferente y continuamos el proceso. Si no existen más alternativas continuamos el retroceso.

El análisis termina cuando se da alguna de las siguientes condiciones:

1. El puntero alcanza el final de la cadena de entrada y no hay más nodos activos. En este caso la cadena de entrada ha sido reconocida como perteneciente al lenguaje generado por la gramática.

2. Llegamos a un punto en que el retroceso no es posible. En este caso, la cadena de entrada no pertenece al lenguaje generado por la gramática.

EJEMPLO 3.12

En este ejemplo usaremos el algoritmo descrito, junto con la gramática del ejemplo 3.2 para analizar la cadena de entrada "$a+a*a$". Para acortar la exposición, la selección de alternativas se realizará sin seguir ningún criterio preestablecido. Los diferentes pasos se muestran en las figuras 3.7, 3.8, y 3.9.

La figura 3.7 muestra los árboles de derivación correspondientes a los primeros pasos del algoritmo. Los nodos activos en cada paso están marcados con un círculo, y debajo de cada árbol se muestra la parte de la cadena de entrada situada a la derecha del puntero de entrada.

[1] El algoritmo descendente ha sido tratado con profusión en la literatura, se puede encontrar una descripción formal del mismo en [3] ó [59], entre otros.

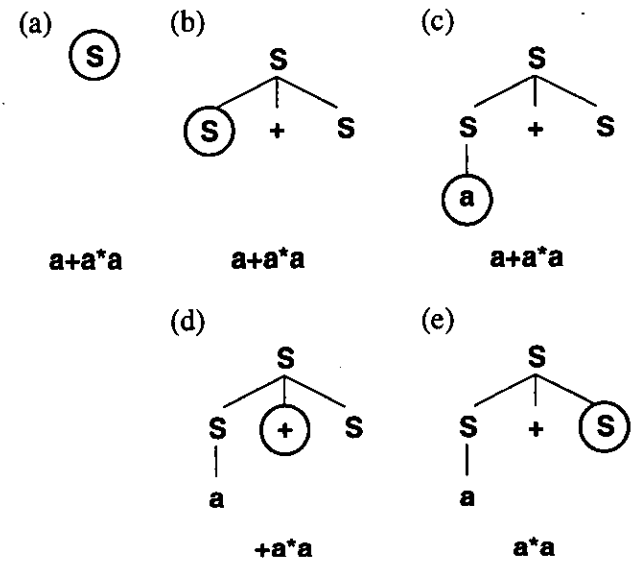


Figura 3.7: Árboles de derivación (parte inicial)

La figura 3.8 muestra los árboles construidos después de predecir una alternativa incorrecta en el paso (e). Como consecuencia, una vez llegados al paso (h) es necesario realizar un retroceso de nuevo al paso (e) y seleccionar otra alternativa. Los árboles que se producen después del retroceso se muestran en la figura 3.9. ■

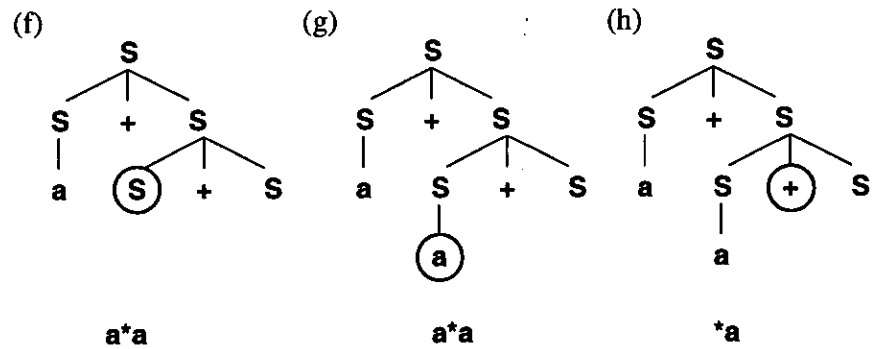


Figura 3.8: Árboles de derivación (alternativa incorrecta)

Uno de los mayores inconvenientes del método descendente es la no-terminación del proceso de análisis en presencia de gramáticas con recursividad por la izquierda[2]. Si retomamos la gramática del ejemplo anterior, vemos durante el análisis que tras expandir un nodo S mediante la primera producción $S \rightarrow S + S$, el nodo activo vuelve

[2]Para más detalles ver [3], Volumen I, pag. 285-286.

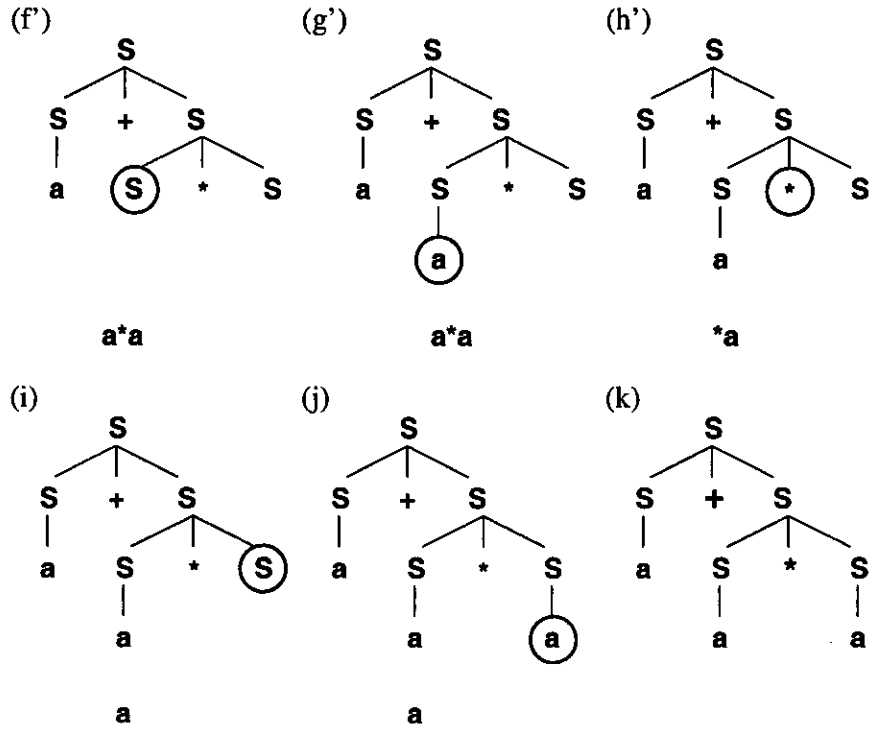


Figura 3.9: Árboles de derivación (resultado final)

a estar etiquetado con S, con lo cual podríamos repetir el paso anterior infinitas veces. Por este motivo el algoritmo de análisis descendente descrito se emplea únicamente con gramáticas sin recursividad por la izquierda.

EJEMPLO 3.13

De nuevo analizaremos la cadena de entrada "$a + a * a$" mediante el algoritmo descrito. En esta ocasión emplearemos una versión de la gramática de las expresiones aritméticas sin recursividad por la izquierda, en la que hemos numerado las alternativas:

$$
\begin{array}{ll}
(E_1) & E \rightarrow T + E \\
(E_2) & E \rightarrow T \\
(T_1) & T \rightarrow F * T \\
(T_2) & T \rightarrow F \\
(F_1) & F \rightarrow a
\end{array}
$$

En la figura 3.10 se desarrolla la secuencia de derivaciones que se sigue de la aplicación del algoritmo. Cada nivel de sangrado representa la elección de una alternativa.■

En el ejemplo anterior se observa la ineficiencia del mecanismo de retroceso aplicado al algoritmo descendente, debido a la repetición de cálculos y a la falta de un

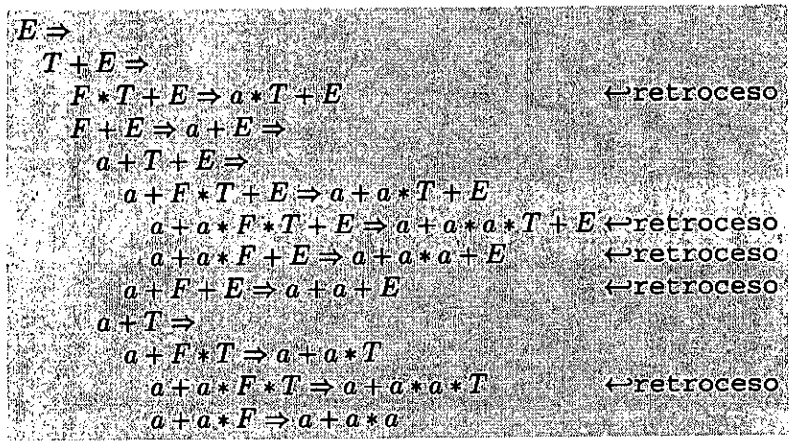


Figura 3.10: Análisis descendente de $a + a * a$

elemento que guíe el proceso de análisis evitando la elección de alternativas que no forman parte de la solución.

3.2.3 Un algoritmo ascendente interpretado

El algoritmo de análisis sintáctico ascendente se describe como el opuesto al descendente: se construye el árbol de análisis desde las hojas hacia la raíz. Describiremos el algoritmo como un analizador *desplazamiento-reducción*. De manera informal, si el siguiente símbolo a analizar es consistente con la entrada, se introduce en una pila y se *desplaza* el puntero de entrada. Si los símbolos en la cima de la pila coinciden con la parte derecha de una producción, se realiza una *reducción* reemplazando dichos símbolos por el no terminal de la parte izquierda. En este tipo de analizador, existen dos formas de no determinismo:

- Reducción-reducción. En algún momento es posible realizar la reducción con dos o más producciones distintas. Supongamos, por ejemplo, la gramática:

$$S \rightarrow AB$$
$$A \rightarrow ab$$
$$B \rightarrow aba$$

  Si la pila contiene "aba" es posible reducir la segunda y la tercera producción.

- Desplazamiento-reducción. Siempre que sea posible una reducción, y no nos encontremos al final de la cadena de entrada, y sea igualmente posible un desplazamiento.

El análisis termina cuando no es posible realizar más operaciones de desplazamiento o reducción. Si la pila contiene únicamente el símbolo inicial de la gramática,

el análisis tuvo éxito. Nótese que por la forma de proceder del algoritmo, en los pasos intermedios se irán construyendo varios árboles de derivación simultáneos formando un bosque de análisis. Si el análisis tiene éxito, los árboles de derivación se unirán en uno solo, formando un bosque Y/O.

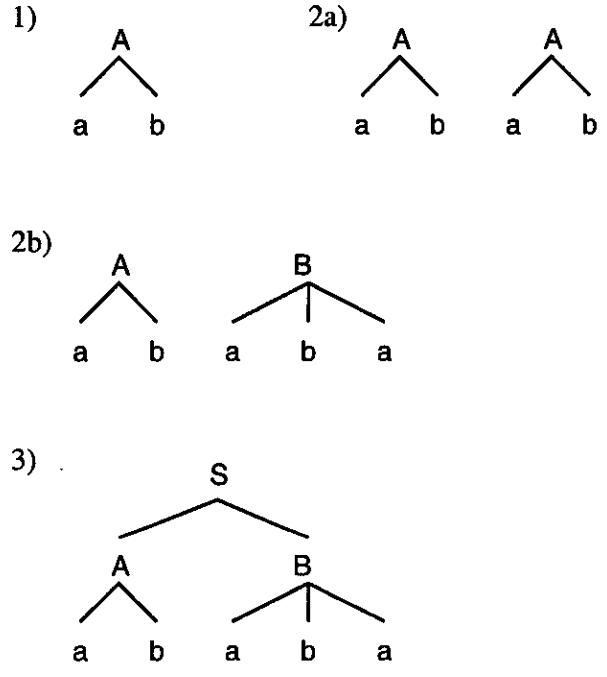


Figura 3.11: Árboles de derivación para un algoritmo ascendente

EJEMPLO 3.14
A continuación realizaremos un análisis empleando de nuevo la gramática:

$$S \rightarrow AB$$
$$A \rightarrow ab$$
$$B \rightarrow aba$$

En caso de existir un no determinismo de tipo desplazamiento-reducción, elegiremos como primera alternativa la reducción. La cadena de entrada será "ababa". Representaremos la pila de análisis como $(x\ y\ z)$, donde z es el elemento en la cima.

En primer lugar desplazamos el primer símbolo (a), tras lo cual sólo es posible realizar otro desplazamiento, quedando la pila $(a\ b)$. En este momento existe un no determinismo en virtud del cual son posibles dos acciones: realizar un desplazamiento del siguiente símbolo de entrada, o bien realizar una reducción empleando la regla $A \rightarrow ab$. Como primera alternativa, elegimos la reducción, dando lugar a la pila (A) el árbol de derivación resultante se muestra en la parte 1) de la figura 3.11. De nuevo realizamos dos desplazamientos, quedando la pila $(A\ a\ b)$. En este punto encontramos un nuevo conflicto reducción-desplazamiento y volvemos a seleccionar como primera

alternativa reducir $A \rightarrow ab$, tras lo cual obtenemos la pila $(A\,A)$, la parte 2a) de la figura 3.11 muestra el bosque de derivación. A continuación sólo podemos realizar un desplazamiento, $(A\,A\,a)$, tras lo cual no es posible continuar, por lo que realizamos un retroceso, hasta $(A\,a\,b)$. En esta ocasión elegimos realizar un desplazamiento, $(A\,a\,b\,a)$, tras el cual podemos reducir $B \rightarrow aba$ dando lugar a $(A\,B)$, parte 2b) de la figura 3.11. Por último, reducimos $S \rightarrow AB$, obteniendo la pila (S) y el árbol de análisis de la parte 3) de la figura 3.11. ■

Recordemos que en los analizadores descendentes la presencia de gramáticas con recursividad por la izquierda provocaba la creación de árboles de derivación infinitos y por tanto la no-terminación del proceso. Análogamente, en los analizadores ascendentes la presencia de ciclos en la gramática produce el mismo efecto de no-terminación. De la misma forma, las producciones-ε dan lugar a un número arbitrario de reducciones en cualquier momento del análisis. Para simplificar el desarrollo del algoritmo, se omite su tratamiento. Al igual que ocurre con los algoritmos descendentes, los ascendentes han sido tratados con profusión en la literatura. Se puede encontrar una descripción formal de los mismos en [3] ó [59], entre otros.

EJEMPLO 3.15

A continuación analizaremos la cadena de entrada "$a + a * a$", al igual que hicimos en el ejemplo 3.13 para ilustrar el algoritmo descendente. Si empleamos la misma versión de la gramática en que habíamos eliminado la recursividad por la izquierda, enseguida vemos que el algoritmo ascendente tiene un rendimiento muy pobre, ya que la modificación que hemos realizado sobre la gramática provoca que ante un punto de no determinismo se escoja siempre la alternativa errónea. La figura 3.12 muestra las configuraciones iniciales en este caso.

$a + a * a \Leftarrow F + a * a \Leftarrow$
$T + a * a \Leftarrow$
$E + a * a \Leftarrow E + F * a \Leftarrow$
$E + T * a \Leftarrow$
$E + E * a \Leftarrow E + E * F \Leftarrow$
$E + E * T \Leftarrow E + E * E$ ⇐retroceso

Figura 3.12: Comienzo del análisis ascendente de "$a + a * a$"

Para evitar este problema, recurrimos a la versión anterior de la gramática con recursividad por la izquierda. A diferencia del algoritmo descendente, el algoritmo ascendente sí acepta este tipo de gramáticas:

(1) $E \rightarrow E + T$
(2) $E \rightarrow T$
(3) $T \rightarrow T * F$
(4) $T \rightarrow F$
(5) $F \rightarrow a$

En la figura 3.13 se desarrolla la secuencia de derivaciones que produce el algoritmo. ■

No obstante los ejemplos expuestos involucran gramáticas y cadenas de entrada pequeñas, al igual que ocurría en el análisis descendente, se observan los problemas derivados del uso del retroceso, principalmente la repetición de cálculos. En el ejemplo expuesto se observa dicha repetición entre los diferentes subanálisis que son fruto de un posterior retroceso, en concreto la derivación:

$$a \Leftarrow F \Leftarrow T \Leftarrow E$$

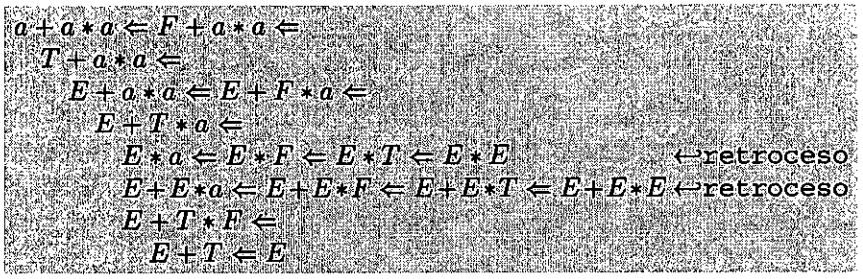


Figura 3.13: Análisis ascendente de $a + a * a$

A continuación se presentan otros algoritmos donde no se emplea el retroceso, a cambio se exploran todas la alternativas y se almacenan los resultados intermedios de forma que es posible su reutilización posterior.

3.2.4 El algoritmo de Cocke-Younger-Kasami

El algoritmo CYK es un algoritmo interpretado, ascendente, en programación dinámica y que toma su nombre de los tres autores que trabajaron inicialmente en él: Cocke, Younger y Kasami [69, 187, 60]. El algoritmo original opera sobre GICs en forma normal de Chomsky. Suponiendo que la cadena a analizar es $w = w_1, w_2, \ldots, w_n$, la idea general del algoritmo es construir una matriz $T = (T_{i,j})$ donde hemos de incluir en cada celda $T_{i,j}$ el símbolo A si y sólo si se cumple la siguiente condición:

$$A \overset{+}{\Rightarrow} w_i \ldots w_{i+j-1}$$

En función de los dos tipos de producciones que podemos encontrar en una gramática en forma normal de Chomsky, consideraremos dos casos distintos:

- Por cada regla $A \to w_i \in P$, aplicando la definición de derivación, es inmediato que $A \overset{*}{\Rightarrow} w_i$, y por lo tanto debemos añadir A a la celda $T_{i,1}$.
- Por cada regla $A \to BC \in P$, si tenemos que $B \in T_{i,k}$ y $C \in T_{i+k,j}$, entonces $B \overset{*}{\Rightarrow} w_i \ldots w_{i+k-1}$ y $C \overset{*}{\Rightarrow} w_{i+k} \ldots w_{i+k+j-1}$. Aplicando la definición de derivación deducimos que $A \overset{*}{\Rightarrow} BC \overset{*}{\Rightarrow} w_i \ldots w_{i+k+j-1}$, y por lo tanto, debemos añadir A a la celda $T_{i,j+k}$.

El algoritmo CYK procede rellenando las celdas de la tabla según los casos expuestos hasta que no es posible añadir ningún símbolo nuevo. Si al finalizar el análisis, el símbolo inicial de la gramática, S, se encuentra en la celda $T_{1,n}$, $S \stackrel{*}{\Rightarrow} w_{1..n}$ y por tanto la cadena de entrada pertenece al lenguaje generado por la gramática.

ALGORITMO 3.1

Entrada: Una GIC $\mathcal{G} = (N, \Sigma, P, S)$ en forma normal de Chomsky y una cadena de entrada $w = w_1 w_2 \cdots w_n \in \Sigma^*$

Salida: La tabla de análisis T del tal forma que $A \in T_{i,j}$ si y sólo si $A \stackrel{+}{\Rightarrow} w_{i..i+j-1}$

Pasos:

1. Inicialización:
   $T_{i,1} = \{A | A \rightarrow w_i \in P\} \forall i, 1 \leq i \leq n$
2. Para cada i desde 1 hasta n
   (a) Para cada j desde 1 hasta $n - i + 1$

$$T_{i,j} = \left\{ \begin{array}{l|l} A & \exists k,\ 1 \leq k < j,\ A \rightarrow BC \in P, \\ & B \in T_{i,k},\ C \in T_{i+k,j-k} \end{array} \right\}$$

□

El algoritmo descrito no es un algoritmo de análisis propiamente dicho, sino un reconocedor puesto que se limita a establecer $w \in L(\mathcal{G}) \iff S \in T_{1,n}$. Para considerarlo como analizador debemos obtener el árbol de análisis a partir de la tabla de análisis, para lo cual emplearemos el siguiente algoritmo:

ALGORITMO 3.2

Entrada: Una GIC $\mathcal{G} = (N, \Sigma, P, S)$ en forma normal de Chomsky con las producciones numeradas de 1 a p, una tabla de análisis construida con el algoritmo 3.1 en la que $S \in T_{1,n}$, y una cadena de entrada $w = w_1 w_2 \cdots w_n \in \Sigma^*$

Salida: Un árbol de análisis de w

Pasos: Calculamos $\mathtt{gen}(1, n, S)$.

$\mathtt{gen}(i, j, A)$ se define recursivamente como:

1. Si $j = 1$ y $A \rightarrow w_i$ es la producción m-ésima, emitir la producción número m.
2. Si $j > 1$, y k es el entero más pequeño tal que $1 \leq k < j$, $B \in T_{i,k}$, $C \in T_{i+k,j-k}$, y $A \rightarrow BC \in P$ es la producción m-ésima (si existen varias alternativas, elegir una). Entonces emitir la producción número m y calcular $\mathtt{gen}(i, k, B)$ seguido de $\mathtt{gen}(i + k, j - k, C)$.

□

A continuación se ilustra el funcionamiento del algoritmo CYK mediante un ejemplo de análisis.

EJEMPLO 3.16

Volvemos una vez más a la gramática de las expresiones aritméticas, cuya forma normal de Chomsky se muestra en el ejemplo 3.11. Nótese que respecto a los ejemplos de los algoritmos anteriores, en esta versión no se han eliminado ni las recursividades, ni las ambigüedades de la gramática. Igual que en los ejemplos anteriores, analizaremos la cadena de entrada "$a + a * a$". Tras el primer paso de inicialización, nos encontramos que:

$$S \in T_{1,1},\ S_2 \in T_{2,1},\ S \in T_{3,1},\ S_4 \in T_{4,1},\ S \in T_{5,1}$$

A continuación podemos proceder a rellenar las siguientes casillas de la tabla:

1. $S_2 \in T_{2,1}, S \in T_{3,1}, S_1 \to S_2 S \in P \Rightarrow S_1 \in T_{2,2}$
2. $S_4 \in T_{4,1}, S \in T_{5,1}, S_3 \to S_4 S \in P \Rightarrow S_3 \in T_{4,2}$
3. $S \in T_{1,1}, S_1 \in T_{2,2}, S \to S S_1 \in P \Rightarrow S \in T_{1,3}$
4. $S \in T_{3,1}, S_3 \in T_{4,2}, S \to S S_3 \in P \Rightarrow S \in T_{3,3}$
5. $S_2 \in T_{2,1}, S \in T_{3,3}, S_1 \to S_2 S \in P \Rightarrow S_1 \in T_{2,4}$
6. $S \in T_{1,1}, S_1 \in T_{2,4}, S \to S S_1 \in P \Rightarrow S \in T_{1,5}$
7. $S \in T_{1,3}, S_3 \in T_{4,2}, S \to S S_3 \in P \Rightarrow S \in T_{1,5}$

La tabla de análisis final se muestra en la figura 3.14. Nótese que a pesar de la ambigüedad de la gramática, algunos cálculos como $S_3 \in T_{4,2}$ se realizan una sola vez.

5	S				
4		S_1			
3	S		S		
2		S_1		S_3	
1	S	S_2	S	S_4	S
	1	2	3	4	5
	a	+	a	*	a

Figura 3.14: Tabla de análisis CYK

Puesto que $S \in T_{1,5}$ la cadena de entrada pertenece a $L(\mathcal{G})$. De la tabla de análisis se pueden extraer los dos árboles de análisis de la figura 3.15. ■

Como hemos visto, el algoritmo CYK, frente a los algoritmos descendente y ascendente con retroceso, evita la repetición de cálculos e incluso permite la compartición de los mismos cuando existen diversos árboles de análisis. En su contra señalaremos el hecho de que es necesario transformar la gramática objeto de análisis para que ésta se presente en forma normal de Chomsky, aunque podríamos presentar una extensión del algoritmo [49, 34] que permite el uso de gramáticas arbitrarias.

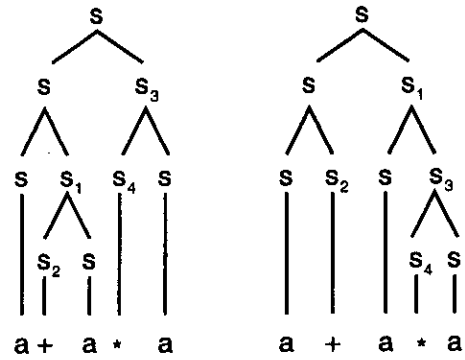


Figura 3.15: Árboles de análisis para $a + a * a$

3.2.5 El algoritmo de Earley

Este algoritmo recibe el nombre de su creador, Earley [48, 47], y analiza cadenas de entrada para GICs arbitrarias mediante una estrategia, interpretada, mixta en programación dinámica. Al contrario que los algoritmos anteriores, es capaz de operar con GICs recursivas sin ninguna restricción sobre la forma de sus producciones. El algoritmo de Earley se basa en la creación y almacenamiento de *items*, que representan estados intermedios del proceso de análisis.

DEFINICIÓN 3.17 (ítem Earley)
Sean $w = w_1 w_2 \cdots w_n$ una cadena de entrada, $\mathcal{G} = (N, \Sigma, P, S)$ una GIC, llamaremos ítem a un objeto de la forma $[A \to X_1 X_2 \ldots \bullet X_k \ldots X_m, i]$, $A \to X_1 X_2 \cdots X_m$, $0 \leq k \leq m$ y $0 \leq i \leq n$, y $'\bullet' \notin \Sigma \cup N$ es un meta-símbolo que no pertenece a la gramática. □

Cada ítem representa un estado intermedio del proceso de análisis. Para facilitar el desarrollo del algoritmo, se prefiere el uso de gramáticas aumentadas. La idea central es construir $n+1$ conjuntos de items I_j, uno por cada elemento de la cadena de entrada, $1 \leq j \leq n$, más el conjunto inicial, de tal forma que:

$$[A \to \alpha \bullet \beta, i] \in I_j \text{ si y sólo si } \exists \gamma, \delta \; S \stackrel{*}{\Rightarrow} \gamma A \delta, \gamma \stackrel{*}{\Rightarrow} w_1 \ldots w_i, \text{ y } \alpha \stackrel{*}{\Rightarrow} w_{i+1} \ldots w_j$$

Por lo tanto, $[S' \to S\bullet, 0] \in I_n$ si y sólo si $S' \Rightarrow S \stackrel{*}{\Rightarrow} w_1 \ldots w_n$, es decir, $w \in L(\mathcal{G})$.

Intuitivamente, $[A \to \alpha \bullet \beta, i] \in I_j$ si $\alpha \stackrel{*}{\Rightarrow} w_{i+1} \ldots w_j$, esto es, los símbolos a la izquierda del punto, α, constituyen la parte de la producción que ya hemos analizado, mientras que a la derecha nos quedan los siguientes símbolos a analizar. El algoritmo de Earley comienza intentando analizar el símbolo inicial de la gramática, consecuentemente el ítem de partida será $[S' \to \bullet S, 0]$. A continuación, se sigue una estrategia diferente a la adoptada en los algoritmos vistos con anterioridad, en los que se seguía una única estrategia descendente o ascendente, según el caso. Por su parte, el algoritmo de Earley combina una fase descendente o predictiva con una fase ascendente.

- En la parte predictiva, por cada ítem cuyo siguiente símbolo a analizar sea un no terminal $[A \to \alpha \bullet B\beta, i] \in I_j$, se predicen todas las producciones, $B \to \gamma$, cuya parte izquierda encaja dando lugar a los items $[B \to \bullet\gamma, j] \in I_j$.

- En la parte ascendente, una vez completado el análisis dado por una producción, $[A \to \gamma\bullet, i] \in I_j$, se actualizan aquellos items $[B \to \alpha \bullet A\beta, k] \in I_i$ cuyo siguiente símbolo a analizar encaja con la parte izquierda de la producción recién completada, dando lugar a los items $[B \to \alpha A \bullet \beta, k] \in I_j$.

Por último, es necesaria una operación más en la que progrese el proceso de análisis mediante el reconocimiento de los terminales de la cadena de entrada. Esta operación se realizará cuando el siguiente símbolo a analizar sea un terminal, $[A \to \alpha \bullet a\beta, j] \in I_i$, teniendo en cuenta que $\alpha \stackrel{*}{\Rightarrow} w_{j+1} \dots w_i$, dicho terminal tiene que coincidir con el siguiente símbolo de la cadena de entrada $a = w_i$. A continuación se presenta el desarrollo formal del algoritmo.

ALGORITMO 3.3

Entrada: La cadena de entrada $w = w_1 \dots w_n \in \Sigma^*$, y la GIC aumentada de $\mathcal{G} = (N, \Sigma, P, S)$

Salida: La lista de análisis $I_0, I_1, \dots, I_n$

Pasos:

1. Inicialización:
   $I_0 = \{[S' \to S, 0]\}$
2. Repetir para todo I_j, $0 \leq j \leq n$
   (a) Repetir hasta que no se puedan añadir items nuevos a I_j
      i. Completar.
      $\forall [A \to \gamma\bullet, i] \in I_j, [B \to \alpha \bullet A\beta, k] \in I_i$: $[B \to \alpha A \bullet \beta, k] \in I_j$
      ii. Predicción.
      $\forall [A \to \alpha \bullet B\beta, i] \in I_j, B \to \gamma \in P$: $[B \to \bullet\gamma, j] \in I_j$
   (b) Reconocimiento.
   $\forall [B \to \alpha \bullet a\beta, i] \in I_j$ tal que $a = w_{j+1}$: $[B \to \alpha a \bullet \beta, i] \in I_{j+1}$ □

De nuevo, el método descrito no es un algoritmo de análisis propiamente dicho, sino un reconocedor puesto que se limita a establecer $w \in L(\mathcal{G}) \iff [S' \to S\bullet, 0] \in I_n$. Para considerarlo como analizador debemos obtener el árbol de análisis a partir de la lista de análisis, para lo cual emplearemos el siguiente algoritmo:

ALGORITMO 3.4

Entrada: Una GIC $\mathcal{G} = (N, \Sigma, P, S)$ con las producciones numeradas de 1 a p, una cadena de entrada $w = w_1 w_2 \cdots w_n \in \Sigma^*$ y la lista de análisis construída con el algoritmo 3.3

Salida: Un árbol de análisis de w

Pasos: Calculamos $\mathrm{R}([S' \to S\bullet, 0], n)$.

$\mathrm{R}([A \to \beta\bullet, i], j)$ se define recursivamente como:

1. Añadir la producción $A \to \beta$ al árbol de análisis.
2. Si $\beta = X_1 \ldots X_m$, $k = m$ y $l = j$
3. Repetir hasta $k = 0$:
   (a) Si $X_k \in \Sigma$, $k = k - 1$, $l = l - 1$
   (b) Si $X_k \in N$, buscar un ítem $[X_k \to \gamma\bullet, r] \in I_l$ tal que $[A \to X_1 \ldots X_{k-1} \bullet X_k \ldots X_m, i] \in I_r$, y calcular $\mathrm{R}([X_k \to \gamma\bullet, r], l)$. $k = k - 1$, $l = r$ □

A continuación se muestra un ejemplo en el que emplearemos el algoritmo de Earley para realizar un análisis sintáctico.

EJEMPLO 3.17
En este ejemplo emplearemos la primera versión de la gramática de las expresiones aritméticas, del ejemplo 3.2. La cadena de entrada será una vez más $w = a + a * a$. *La figura 3.16 muestra los conjuntos de items generados durante el análisis. Al finalizar,* $[S' \to S\bullet, 0] \in I_5$, *por lo tanto la cadena de entrada pertenece al lenguaje.* ■

3.2.6 Los algoritmos LR

Los algoritmos comentados anteriormente se encuadran dentro de la categoría de guiados por la gramática, en cada paso del algoritmo se examina la gramática para determinar la siguiente acción a realizar. Los algoritmos de análisis sintáctico LR[3], introducidos por Knuth en [73], en cambio emplean un control finito en lugar de la propia gramática. Al igual que en el algoritmo ascendente antes comentado, son posibles las acciones de desplazamiento y reducción. A mayores se consideran las acciones especiales: aceptar y error. En cada iteración de un algoritmo de análisis LR(k) para determinar la siguiente acción a realizar se consultan el control finito y los k siguientes símbolos a analizar, referidos como *símbolos adelantados*[4], *símbolos de preanálisis* o *de anticipación*. El uso del control finito mejora la eficacia y la eficiencia del algoritmo en dos aspectos:

- Las estructuras de datos asociadas al control finito se acceden más eficientemente que la representación directa de las gramáticas.
- El control finito se construye de tal forma que, ante la presencia de situaciones de no determinismo, seleccione el menor número de alternativas posible. En el mejor de los casos seleccionará únicamente una, convirtiendo el proceso de análisis en determinista.

[3]Por *Left-to-right scan and Right-most reduction* (reconocimiento de izquierda a derecha y reducción por la derecha).

[4]En terminología anglosajona, *lookahead*

I_0

$[S' \to \bullet S, 0]$
$[S \to \bullet S + S, 0]$
$[S \to \bullet S * S, 0]$
$[S \to \bullet(S), 0]$
$[S \to \bullet a, 0]$

I_1

$[S \to a\bullet, 0]$
$[S' \to S\bullet, 0]$
$[S \to S \bullet + S, 0]$
$[S \to S \bullet * S, 0]$

I_2

$[S \to S + \bullet S, 0]$
$[S \to \bullet S + S, 2]$
$[S \to \bullet S * S, 2]$
$[S \to \bullet(S), 2]$
$[S \to \bullet a, 2]$

I_3

$[S \to a\bullet, 2]$
$[S \to S + S\bullet, 0]$
$[S \to S \bullet + S, 2]$
$[S \to S \bullet * S, 2]$
$[S' \to S\bullet, 0]$
$[S \to S \bullet + S, 0]$
$[S \to S \bullet * S, 0]$

I_4

$[S \to S * \bullet S, 2]$
$[S \to S * \bullet S, 0]$
$[S \to \bullet S + S, 4]$
$[S \to \bullet S * S, 4]$
$[S \to \bullet(S), 4]$
$[S \to \bullet a, 4]$

I_5

$[S \to a\bullet, 4]$
$[S \to S * S\bullet, 2]$
$[S \to S * S\bullet, 0]$
$[S \to S \bullet + S, 4]$
$[S \to S \bullet * S, 4]$
$[S \to S + S \bullet 0]$
$[S \to S \bullet + S, 2]$
$[S \to S \bullet * S, 2]$
$[S' \to S\bullet, 0]$
$[S \to S \bullet + S, 0]$
$[S \to S \bullet * S, 0]$

Figura 3.16: Lista de análisis para $w = a + a * a$

El empleo de dicho control implica la existencia de una fase previa en la cual se construye tras analizar la gramática y que se suele conocer como compilación del autómata LR(k). En la práctica, valores de k mayores que uno producen un control finito cuyo tamaño no es manejable. En adelante supondremos que $k = 1$, salvo que se especifique lo contrario. El control finito se representa mediante un autómata finito determinista. Las transiciones del autómata vienen dadas por el estado actual y un[5] símbolo de la gramática, y están etiquetadas con cuatro tipos de acciones[6]:

desp(n) desplazarse al estado n

ir(n) ir al estado n (*ir_a*)

red(k) reducir empleando la regla k

acep aceptar

[5] Recordar que por defecto suponemos $k = 1$.

[6] En particular, las acciones **red**(k) no se corresponden con ninguna transición, si no que se determinan examinando los ítems de cada estado.

err error

Por otra parte, el algoritmo se define de forma genérica para cualquier control finito por lo que es común referirse a él como *intérprete del autómata*. En su forma básica, implementa un analizador determinista, por lo tanto precisa un control finito capaz de eliminar todas las ambigüedades presentes en la gramática. En general, nos referiremos a dicha clase de gramáticas como *gramáticas LR(k)* [3, 164].

El autómata mediante el cual se representa el control finito no se emplea directamente para analizar la cadena de entrada, sino para establecer las configuraciones que atraviesa el proceso de análisis:

DEFINICIÓN 3.18 (configuración)
Un triple (α, β, γ) *es una configuración del algoritmo LR:*

- α *es una pila de símbolos y estados, cuya cima se sitúa a la derecha.*
- β *es la lista de símbolos terminales que restan por analizar.*
- γ *es la lista que permite construir el árbol de análisis. Para ello se recorre desde la derecha, expandiendo los nodos del árbol en postorden, y teniendo en cuenta el significado de cada uno de sus elementos:*
  - *Un número* j*. Expandimos el nodo con el árbol dado por la regla* j*.*
  - *El símbolo* s*. Pasamos al siguiente nodo.* □

..	'+'	'*'	'a'	$	E	T	F
0			desp(1)		ir(8)	ir(2)	ir(3)
1	red(5)	red(5)		red(5)			
2	desp(4)			red(2)			
3	red(4)	desp(5)		red(4)			
4			desp(1)		ir(6)	ir(2)	ir(3)
5			desp(1)			ir(7)	ir(3)
6				red(1)			
7	red(3)			red(3)			
8				acep			

Tabla 3.1: Tabla de acciones e ir_a para la gramática de ejemplo

Una vez establecidas las configuraciones, el intérprete del autómata opera sobre ellas de la siguiente manera:

ALGORITMO 3.5
Entrada: Un control finito creado a partir de Una GIC $\mathcal{G} = (N, \Sigma, P, S)$, una cadena de entrada $w = w_1 w_2 \cdots w_n \in \Sigma^*$.

Salida: La lista para construir el árbol de análisis si $w \in \mathcal{L}(\mathcal{G})$ o error en caso contrario.

Pasos:

- La configuración inicial es $(0, w, \epsilon)$, y el puntero de entrada $i = 0$.
- Repetir:
  - Para la configuración actual $(\alpha s_j, w_i..w_n, \gamma)$, examinar el control finito para establecer la siguiente acción.
  - Si la acción es:

    desp(s_k) Introducir el siguiente símbolo a analizar, w_i, y s_k en la pila y avanzar el puntero de entrada:$(\alpha s_j w_i s_k, w_{i+1}..w_n, s\gamma)$.

    red(k) Eliminar de la pila tanto símbolos y estados como símbolos tenga la parte derecha de la regla k. Usar el estado de la cima de la pila y el símbolo de la parte izquierda (X) de la regla k para obtener la acción ir(s_l). Apilar X y s_l, y añadir k al comienzo de la lista de construcción del árbol.

    acep Terminar, la cadena ha sido analizada con éxito.

    err Terminar con error. □

A continuación se muestra un ejemplo del funcionamiento del algoritmo. Puesto que, tal y como hemos indicado, es independiente de la construcción del control finito, en el desarrollo del ejemplo se presupone que dicha fase ya se ha llevado a cabo.

EJEMPLO 3.18
En este ejemplo se muestra la lista de configuraciones por las que atraviesa el intérprete LR en el análisis de la cadena de entrada "$a + a * a$" con la gramática del ejemplo 3.13 aumentada. Un posible control finito para dicha gramática se muestra en la tabla 3.1, donde las acciones de error se representan mediante un espacio en blanco. Las configuraciones resultantes se pueden observar en la figura 3.17, donde también se indica la acción aplicada a cada configuración.

En el ejemplo se observa como, al contrario que ocurría en el algoritmo descendente (ver ejemplo 3.15), para la versión de la gramática empleada el control finito es capaz de seleccionar la alternativa adecuada, evitando cálculos inútiles. A continuación, resta por definir cómo se construye dicho control finito, para lo cual emplearemos el concepto de *ítem LR(k)*.

DEFINICIÓN 3.19 (ítem LR(k))
Sea $\mathcal{G} = (N, \Sigma, P, S)$ una GIC, llamaremos ítem LR(k) a una expresión de la forma $[A \to \alpha \bullet \beta, w]$, donde $A \to \alpha\beta \in P$, $w \in \Sigma^k$, y '$\bullet$' $\notin (\Sigma \cup N)$. □

El significado intuitivo de los items es el mismo que en el algoritmo de Earley. A mayores se han añadido k terminales que representan los símbolos de preanálisis válidos para ese ítem, indicando que constituirá un estado fiable del proceso de análisis únicamente si los siguientes k símbolos a analizar coinciden con sus símbolos de preanálisis. Para calcular dichos símbolos se recurre a la noción de conjunto PRIMERO$_k$.

Pila	Entrada	Árbol	Acción
0	a+a*a$	ε	desp(1)
$0a1$	+a*a$	s	red(5)
$0F3$	+a*a$	s5	red(4)
$0T2$	+a*a$	s54	desp(4)
$0T2+4$	a*a$	s54s	desp(1)
$0T2+4a1$	*a$	s54ss	red(5)
$0T2+4F3$	*a$	s54ss5	desp(5)
$0T2+4F3*5$	a$	s54ss5s	desp(1)
$0T2+4F3*5a1$	$	s54ss5ss	red(5)
$0T2+4F3*5F3$	$	s54ss5ss5	red(4)
$0T2+4F3*5T7$	$	s54ss5ss54	red(3)
$0T2+4T2$	$	s54ss5ss543	red(2)
$0T2+4E6$	$	s54ss5ss5432	red(1)
$0E8$	$	s54ss5ss54321	acep

Figura 3.17: Análisis de "$a + a * a$"

DEFINICIÓN 3.20 (PRIMERO)
Sea $\mathcal{G} = (N, \Sigma, P, S)$ *una GIC, y* $\alpha \in (N \cup \Sigma)^*$ *definiremos el conjunto* PRIMERO$_k$ *como:*

$$\text{PRIMERO}_k^{\mathcal{G}}(\alpha) = \left\{ w | w \in \Sigma^*, |w| < k \text{ y } \alpha \overset{*}{\Rightarrow} w, \text{ ó } |w| = k \text{ y } \alpha \overset{*}{\Rightarrow} wx \right\} \quad \square$$

Intuitivamente, el conjunto PRIMERO$_k(\alpha)$ recoge todos aquellos símbolos que pueden ocurrir inmediatamente en las k primeras posiciones de cualquier forma sentencial derivada a partir de α.

A partir de un ítem determinado, los estados del control finito se construyen como conjuntos de items. Para facilitar el proceso, se considerarán siempre gramáticas aumentadas y el ítem inicial será $[S' \to \bullet S, \$^k]$. Este ítem se añade al primer estado creado, que por convención se denominará estado 0, o estado inicial. El ítem inicial constituye la *base* o *núcleo*[7], del estado 0. El resto del estado, y de los estados que se generen posteriormente, se construye mediante la operación de *cierre*[8].

DEFINICIÓN 3.21 (cierre)
Sea $\mathcal{G} = (N, \Sigma, P, S)$ *una GIC y* $kernel_i^{\mathcal{G}}$ *el núcleo del estado* $S_i^{\mathcal{G}}$, *definimos el* conjunto cierre *como:*

$$cierre_i^{\mathcal{G}} = \left\{ [A \to \bullet\alpha, w], \textit{tal que} \begin{cases} A \to \alpha \in P \\ \exists [B \to \beta \bullet A\gamma, x] \in kernel_i^{\mathcal{G}} \cup cierre_i^{\mathcal{G}} \\ w \in \text{PRIMERO}_k^{\mathcal{G}}(\gamma x) \end{cases} \right\}$$

$\square$

[7] En terminología anglosajona, *kernel*.
[8] En terminología anglosajona, *closure*.

Finalmente tenemos que $S_i^{\mathcal{G}} = \text{kernel}_i^{\mathcal{G}} \cup \text{cierre}_i^{\mathcal{G}}$. Intuitivamente, la operación de cierre constituye la contrapartida de la operación de predicción del algoritmo de Earley [47]. Desde otro punto de vista, en un estado se resume su núcleo y todas las predicciones que se pueden realizar a partir de los items de dicho núcleo.

DEFINICIÓN 3.22 (estado LR(k))
Sea $\mathcal{G} = (N, \Sigma, P, S)$ una GIC, definimos un estado LR *como el conjunto:*

$$S_i^{\mathcal{G}} = kernel_i^{\mathcal{G}} \cup cierre_i^{\mathcal{G}}$$ □

Mediante el cierre podemos completar el cálculo del estado inicial (S_0). El resto de estados se calculan de forma iterativa como *sucesores* de un estado anterior.

DEFINICIÓN 3.23 (X-sucesor)
Sea $\mathcal{G} = (N, \Sigma, P, S)$ una GIC y $S_i^{\mathcal{G}}$ un estado LR(k), definiremos el X-sucesor *de $S_i^{\mathcal{G}}$ como un estado $S_j^{\mathcal{G}}$ que verifica:*

$$kernel_j^{\mathcal{G}} = \{[A \to \alpha X \bullet \beta, w] \mid [A \to \alpha \bullet X\beta, w] \in S_i^{\mathcal{G}}\}$$ □

Intuitivamente el X-sucesor de $S_i^{\mathcal{G}}$ recoge en su núcleo los items resultantes de aplicar una operación de desplazamiento o reducción del algoritmo de Earley implicando el símbolo X. En lo sucesivo, para no sobrecargar la notación, evitaremos la referencia explícita a la gramática en los nombres de estados, siempre que no exista posibilidad de confusión.

Una vez construídos los estados del autómata que representa el control finito, resta por determinar las transiciones del mismo. Dichas transiciones definen las acciones que debe efectuar el intérprete. En este sentido, las relaciones S_j es un X-sucesor de S_i se interpretarán como transiciones de S_i a S_j, etiquetadas por el símbolo X. Así cuando X sea un símbolo terminal, la transición implicará una operación de desplazamiento desp(j), mientras que en el caso de que sea una variable, la operación será ir(j). Por último, aquellos estados en que el punto se encuentra al final de las reglas en un ítem, representan el reconocimiento de la parte derecha de dichas reglas, y consecuentemente es posible una acción de reducción.

Antes de continuar con una descripción más detallada del cálculo de transiciones, es necesario remarcar una situación especialmente problemática: la presencia de producciones-ε. Considérese como ejemplo un estado que contenga los siguientes items:

$$[A \to \alpha \bullet B, w]$$
$$[B \to \varepsilon \bullet, w]$$

En este caso no es posible discernir si la acción a aplicar es:

- Reducción de $B \to \varepsilon$, o bien
- Transición a un B-sucesor.

Sin embargo, no existe tal conflicto puesto que la aplicación de la segunda acción es posterior a la aplicación de la primera. Para resolver este problema introducimos una nueva definición del conjunto PRIMERO como PRIMERO *sin épsilon*: NOεPRIMERO.

DEFINICIÓN 3.24 (NOεPRIMERO)
Sea $\mathcal{G} = (N, \Sigma, P, S)$ una GIC, definimos el conjunto NOεPRIMERO$_k$ *como:*

$$\text{NO}\varepsilon\text{PRIMERO}_k(\alpha) = \left\{ w \mid \begin{array}{l} w \in \Sigma^*, \alpha \overset{*}{\Rightarrow}_{rm} \beta \Rightarrow_{rm} wx, \\ si\ |w| < k,\ x = \varepsilon, \\ y\ \forall A, \beta \neq Awx \end{array} \right\}$$ □

Intuitivamente NOεPRIMERO elimina todos aquellos elementos en cuya derivación por la derecha, la última derivación directa se realiza mediante una producción-ε. Llegados a este punto estamos en disposición de completar la definición del control finito.

DEFINICIÓN 3.25 (control finito LR(k))
Sea $\mathcal{G} = (N, \Sigma, P, S)$ una GIC, definimos el control finito *LR(k) asociado como una 4-tupla $(S, S_0, ir\_a, acción)$, donde:*

- *S es el conjunto de estados LR(k)*
- *S_0 es el estado inicial*
- ir_a *es una función* $S \times V \to S \cup \{err\}$
- acción *es una función* $S \times \Sigma^k \to \{desp(i), acep, err\} \cup \{red(p)\}$

Las funciones ir_a *y* acción *se definen como:*

$$ir\_a(S_i^{\mathcal{G}}, X) = S_j^{\mathcal{G}},\ si\ S_j^{\mathcal{G}}\ es\ un\ X\text{-}sucesor\ de\ S_i^{\mathcal{G}}$$

$$acción(S_i^{\mathcal{G}}, w) = \begin{cases} desp(j), & si\ [A \to \alpha \bullet \beta, x] \in S_i^{\mathcal{G}}, \\ & \quad w \in \text{NO}\varepsilon\text{PRIMERO}_k^{\mathcal{G}}(\beta x), \\ & \quad w = ay,\ j = ir\_a(S_i^{\mathcal{G}}), a) \\ red(p), & si\ [A \to \alpha \bullet, w] \in S_i^{\mathcal{G}} \\ acep, & si\ [S' \to S \bullet, w] \in S_i^{\mathcal{G}} \\ err, & en\ otro\ caso \end{cases}$$ □

La definición anterior no hace más que establecer formalmente las ideas presentadas de forma intuitiva sobre la construcción del control finito. Sobre ella destacaremos el hecho de que para un estado y símbolos de preanálisis determinados es posible que existan varias acciones posibles, en este caso diremos que existe un conflicto y el control finito no es determinista y consecuentemente tampoco lo será el análisis LR(k) correspondiente. En esta situación es posible emplear un mecanismo de retroceso, tal y como hicimos con los algoritmos descendente y ascendente, o aplicar técnicas de programación dinámica, tal y como se describirá en la sección sobre los analizadores LR generalizados.

Como ya indicábamos anteriormente, en los algoritmos LR(k) a medida que aumentamos k, el número de estados del control finito puede llegar a crecer de forma exponencial, lo que se conoce como el fenómeno de *división de estados*[9], sin que aumente notablemente la eficacia de dicho control finito. A este fenómeno hemos de

[9] En terminología anglosajona, *state splitting*.

añadir el hecho de que la tabla del autómata resultante deberá contener entradas por cada combinación de k terminales de la gramática. Como consecuencia, en aplicaciones prácticas el valor de k se limita a 1. A pesar de ello el tamaño del control finito resultante aún puede ser inadecuado. Para solucionar este problema podemos considerar las variantes SLR(k) [43] y LALR(k) [44]. En estos algoritmos, respecto al LR(k) [3, 8, 96], únicamente se modifica la forma de construir el control finito.

SLR(k)

Los analizadores sintácticos SLR(k) fueron introducidos por DeRemer [43]. Para construir el control finito se calcula el conjunto de estados LR(0), que también se conoce como *máquina de estados finita característica*[10]. Este conjunto es más reducido que en el caso del LR(1). Por contra su capacidad para determinizar el análisis es menor, lo que se refleja en un número de conflictos más elevado. Para mejorar este último aspecto, se calculan los conjuntos PRIMERO y SIGUIENTE directamente sobre la gramática.

El conjunto PRIMERO ya ha sido definido con anterioridad, por su parte el conjunto SIGUIENTE(A) contiene los símbolos terminales que pueden ocurrir inmediatamente a la derecha de A en cualquier forma sentencial.

DEFINICIÓN 3.26 (SIGUIENTE)
Sea $\mathcal{G} = (N, \Sigma, P, S)$ una GIC, y $\alpha \in (N \cup \Sigma)^*$ definiremos el conjunto SIGUIENTE$_k$ *como:*

$$\text{SIGUIENTE}_k(\beta) = \left\{ w | S \stackrel{*}{\Rightarrow} \alpha\beta\gamma,\ w \in \text{PRIMERO}_k(\gamma) \right\} \qquad \square$$

De la anterior definición se sigue que, para que una reducción conduzca a la derivación de una forma sentencial, es necesario que los símbolos de preanálisis se encuentren en el conjunto SIGUIENTE(A), siendo A la parte izquierda de la producción reducida. Por tanto eliminaremos del control finito todas aquellas acciones de reducción que no cumplan la condición:

$$\text{si } [A \to \alpha\bullet, w] \in S_i, \text{ entonces } w \in \text{SIGUIENTE}_k(A)$$

EJEMPLO 3.19
Siguiendo con la gramática del ejemplo 3.13, la figura 3.18 y la tabla 3.2 muestran el autómata LR(0) y las tablas acción *e* ir_a *asociadas, en el que existen dos conflictos en los estados S_2 y S_3, respectivamente:*

$$\begin{aligned} &\textit{acción}(S_2, '+') = \{\textit{desp}(4), \textit{red}(E \to T)\} \\ &\textit{acción}(S_3, '*') = \{\textit{desp}(5), \textit{red}(T \to F)\} \end{aligned}$$

Sin embargo, tras consultar el conjunto SIGUIENTE*:*

$$'+' \notin \text{SIGUIENTE}_1(E),\ '*' \notin \text{SIGUIENTE}_1(T)$$

Por lo tanto dichos conflictos se eliminan en el autómata SLR(1) resultante. ■

[10] *Characteristic finite state machine* en la terminología original.

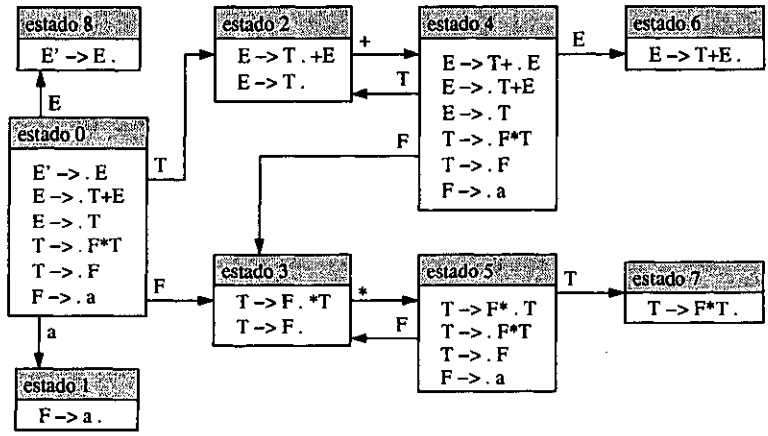


Figura 3.18: Máquina de estados finita característica para la gramática de ejemplo

	'+'	'*'	'a'	$	E	T	F
0			desp(1)		ir(8)	ir(2)	ir(3)
1	red(5)	red(5)	red(5)	red(5)			
2	dep(4),red(2)	red(2)	red(2)	red(2)			
3	red(4)	desp(5),red(4)	red(4)	red(4)			
4			desp(1)		ir(6)	ir(2)	ir(3)
5			desp(1)			ir(7)	ir(3)
6	red(1)	red(1)	red(1)	red(1)			
7	red(3)	red(3)	red(3)	red(3)			
8				acep			

Tabla 3.2: Tablas acción e ir_a para el autómata de la figura 3.18

LALR(k)

A medio camino entre la eficacia de los analizadores sintácticos LR(k) y el tamaño reducido de los SLR(1), se encuentran los LALR(k) [44]. La idea detrás de los autómatas LALR(k) consiste en, partiendo de un autómata LR(k), reducir el número de estados, a cambio de una pequeña pérdida en su capacidad de determinización del análisis. Para conseguir esta reducción, simplemente se fusionan aquellos estados cuyos items resultan idénticos una vez eliminados sus símbolos de preanálisis.

Tal y como se ha dicho anteriormente, la ventaja de los autómatas LALR(k) frente a los LR(k) es su tamaño menor, mientras que su eficacia es similar. Frente a los SLR(k), la comparación es en sentido contrario, su eficacia es mayor, mientras que el número de estados es el mismo. En la práctica, los analizadores LALR(1) gozan de gran popularidad debido a que son capaces de analizar de forma determinista la mayoría de lenguajes de programación.

Una forma de construir un autómata LALR(k) es mediante la aplicación directa de la definición establecida con anterioridad. A pesar de su simplicidad, esta alternativa

nos obliga a construir en primer lugar su homólogo LR(k), lo cual no es deseable, debido a su tamaño. Para evitar este problema, es posible emplear una alternativa análoga a la descrita para los autómatas SLR(k). En primer lugar se construye la máquina de estados finita característica, LR(0), y a continuación se calculan los símbolos de preanálisis sobre los items de los estados [44]. Otros métodos para el cálculo eficaz del conjunto de estados LALR(k) se pueden consultar en la literatura [79, 101, 24, 64].

LR generalizado

Tal y como se ha indicado anteriormente ante la presencia de conflictos en el control finito, el proceso de análisis se vuelve no determinista. Nos referiremos al análisis LR no determinista como *análisis LR generalizado.*

El método más directo para la obtención de analizadores LR generalizados es mediante la aplicación de la técnica de retroceso [99], cuyas debilidades se han resaltado anteriormente. Una forma más eficiente de análisis LR generalizado es la propuesta por Tomita en [146], mediante la aplicación de técnicas de programación dinámica. La propuesta de Tomita rehace, restringiendo el resultado a autómatas LR(0), la propuesta de Lang [84] aplicable a cualquier tipo de esquema de análisis sintáctico basado en un autómata de pila. En concreto, la propuesta de Tomita se basa en la realización de varios análisis simultáneos, compartiendo entre ellos computaciones. Para conseguir esta compartición, las pilas se representan mediante un grafo cuyos nodos son secciones de las pilas de análisis.

El análisis LR generalizado se desarrollará y aplicará con detalle en los siguientes capítulos.

CAPÍTULO 4

ABSTRACCIÓN DEL ANÁLISIS SINTÁCTICO COMO DEDUCCIÓN

En este capítulo, comenzaremos a tratar el problema del análisis sintáctico. Para facilitar la exposición, ésta se realizará de forma incremental, desarrollando en primer lugar reconocedores sintácticos, extendiéndolos posteriormente para cubrir todas las funciones de un analizador.

Por lo general, a medida que aumenta el poder descriptivo de la gramática, también aumenta la complejidad de los posibles algoritmos de análisis sintáctico asociados. Como consecuencia, el diseño de éstos ha sido y continúa siendo un tema de gran interés y complejidad, en el que se investiga activamente. Fruto de los diversos trabajos llevados a cabo, se han desarrollado multitud de algoritmos, algunos de los cuales son ampliamente conocidos debido a su sencillez o a su calidad.

4.1 Introducción

Habitualmente, los trabajos sobre algoritmos de análisis sintáctico establecen un formalismo sobre el cual se especifican las operaciones que los conforman. Este tipo de formalismos presentan como ventaja más inmediata su carácter explícitamente operativo. Por contra, las características particulares de cada algoritmo han provocado que prácticamente todos empleen un formalismo propio, diferente de los demás. Como ejemplo de esta heterogeneidad nos referimos a los algoritmos presentados en el capítulo 3, algoritmo de Earley [47], y algoritmos LR(k) [73]. Ambos están estrechamente relacionados, aunque la mayoría de autores no tratan esta afinidad a partir de la descripción de los mismos.

Como solución a los problemas de disparidad en la formulación de los algoritmos, se propone el uso de *sistemas de deducción gramatical* (SDGs) [131, 132]. Los SDGs no se centran en describir el método de operación, sino en establecer un formalismo descriptivo de alto nivel que resulte independiente del tipo de gramática a tratar y de cualquier realización concreta del algoritmo. De esta forma, se establece un marco descriptivo común que nos aporta, entre otras, las siguientes ventajas:

- Posibilidad de establecer comparaciones entre las características esenciales de diferentes algoritmos de análisis.
- Simplificación de las pruebas de corrección de los algoritmos.

- Simplificación del cálculo de complejidad de los algoritmos.
- Derivación de algoritmos correctos a partir de otros que se sabe que son correctos.

La idea de partida es conseguir el mayor grado de abstracción posible de las cuestiones de implementación de los algoritmos de análisis: datos, control y estructuras de comunicación. Para ello, consideramos el análisis sintáctico como un proceso deductivo. Adoptaremos una lógica cuya semántica establece afirmaciones gramaticales, es decir, afirmaciones sobre los elementos que constituyen las oraciones, sus propiedades gramaticales, y el orden que se establece entre ellos. Nuestro objetivo final será aplicar un proceso deductivo sobre los elementos de dicha lógica, capaz de demostrar si una oración pertenece o no al lenguaje generado por una gramática y, al mismo tiempo, obtener una representación del proceso de análisis. Esta visión del análisis sintáctico como deducción supone dos ventajas importantes:

1. Las lógicas existentes pueden usarse [131, 31] como base para la definición de nuevos formalismos gramaticales con propiedades computacionales y de representación conocidas y que resulten beneficiosas para realizar el análisis sintáctico.

2. Para cada lógica de afirmaciones gramaticales que empleemos en la descripción de nuestros algoritmos de análisis, existirán procedimientos alternativos de búsqueda de demostraciones. La separación de estos elementos nos habilita para la investigación, desde un punto de vista aplicativo, de un amplio rango de algoritmos de análisis sintáctico, combinando distintas lógicas y procedimientos de demostración.

Una introducción informal

Como ya adelantábamos anteriormente, haremos uso de algoritmos de análisis conocidos para ejemplificar la descripción de los SDGs. En concreto, nos remitiremos a algoritmos de análisis de GICs presentados en el capítulo 3.

La idea general es basar la descripción de los analizadores sintácticos en un proceso iterativo de creación de *items* o *fórmulas*. Cada uno de éstos representa un estado intermedio en el proceso de análisis, que comienza con un conjunto de items deducidos de la oración de entrada y un conjunto de items conocidos *a priori*. A partir de los mismos, se calculan nuevos items hasta llegar a un conjunto final. De la interpretación de estos items se deduce la gramaticalidad de la oración.

Ilustraremos estas ideas con la descripción informal del algoritmo CYK. Este algoritmo opera sobre una GIC $\mathcal{G} = (N, \Sigma, P, S)$ en forma normal de Chomsky, y una cadena de entrada $w = w_1, w_2, \ldots, w_n$. Recordemos que el algoritmo construye una matriz $T = (T_{ij})$ donde hemos de incluir en cada celda $T_{i,j}$ el símbolo A si se cumple la condición: $A \stackrel{*}{\Rightarrow} w_i \ldots w_{i+j-1}$. Si al finalizar el análisis, el meta-símbolo de la gramática, S, se encuentra en la celda $T_{1,n}$, la cadena de entrada pertenece al lenguaje generado por la gramática.

Abstrayéndonos de la estructura de datos empleada, en este caso una matriz, el algoritmo CYK construye el siguiente conjunto de items, donde el estado intermedio del proceso de análisis que representa cada ítem[1] se corresponde con la acción de añadir el símbolo A a la celda $T_{i,j}$:

$$\left\{[A,i,j] \mid A \overset{*}{\Rightarrow} w_i \ldots w_{i+j-1}\right\}$$

Por lo tanto, antes de comenzar el análisis, establecemos las hipótesis respecto a la cadena de entrada representadas por los siguientes items:

$$\{[a,i,1] \mid a = w_i \wedge 1 \leq i \leq n\}$$

Los pasos a aplicar en el algoritmo CYK se pueden describir con las reglas:

- Si tenemos un ítem $[w_i,i,1]$ y una producción $A \rightarrow w_i \in P$, deduciremos el ítem $[A,i,1]$.
- Si tenemos los items $[B,i,k],[C,i+k,j-k]$ y una producción $A \rightarrow BC \in P$, deduciremos el ítem $[A,i,j]$.

Por último, si el ítem $[S,1,n]$ se encuentra en el conjunto de items calculados, entonces la cadena de entrada pertenece a la gramática. De manera informal, hemos descrito los componentes básicos de un sistema de deducción para el caso del esquema CYK:

- El conjunto de items $\mathcal{I}$: $\left\{[A,i,j] \mid A \overset{*}{\Rightarrow} w_i \ldots w_{i+j-1}\right\}$
- El conjunto de hipótesis $\mathcal{H}$: $\{[a,i,1] \mid a = w_i \wedge 1 \leq i \leq n\}$
- El conjunto de pasos deductivos $\mathcal{D}$:
  $\{[w_i,i,1] \times [A,i,1] | A \rightarrow w_i \in P\} \cup$
  $\{[B,i,k],[C,i+k,j-k] \times [A,i,j] | A \rightarrow BC \in P\}$.
- Y por último, el conjunto de items finales $\mathcal{F}$: $\{[S,1,n]\}$

A continuación realizaremos una exposición más detallada de los sistemas de deducción, ilustrando sus posibilidades mediante ejemplos.

4.2 Esquemas de análisis y sistemas de deducción

A continuación presentamos de manera formal las ideas expuestas. A este respecto, existen dos referencias fundamentales: los *esquemas de análisis sintáctico*[2], presentados por Sikkel en [132], y de manera más breve en [134], y por otro lado, los SDGs, propuestos por Shieber *et al.* [131].

Mientras que Sikkel enfoca el desarrollo de los esquemas de análisis a la comparación y verificación de la corrección de los algoritmos de análisis [133], Shieber *et al.* emplean los esquemas de deducción gramatical como base para el desarrollo rápido de prototipos. No obstante esta diferencia de planteamiento, ambos formalismos resultan muy próximos.

[1] No confundir con los items LR(k).

[2] *Parsing schemata* en la terminología original.

4.2.1 Esquema de análisis

Como paso previo a la definición de los esquemas de análisis de forma genérica, se introducirá la definición de los *sistemas de análisis sintáctico* aplicables al caso de una gramática y una cadena de entrada concretas.

DEFINICIÓN 4.1 (sistema de análisis sintáctico)
Un sistema de análisis sintáctico $\mathbb{P}$ *para una GIC* $\mathcal{G}$ *y una cadena de entrada* $w_1 \dots w_n$ *es un triple* $\mathbb{P} = (\mathcal{I}, \mathcal{H}, \mathcal{D})$, *donde:*

- $\mathcal{I}$ *es el* conjunto de items *o* dominio *de* $\mathbb{P}$.
- $\mathcal{H}$ *es un conjunto inicial de items denominados* hipótesis.
- $\mathcal{D} \subseteq (\mathcal{H} \cup \mathcal{I})_{\mathcal{P}_{fin}} \times \mathcal{I}$ *es un conjunto de* pasos deductivos.

$\mathcal{P}_{fin}$ *es el* conjunto de potencias *restringido a los conjuntos finitos* □

Intuitivamente, los items de $\mathcal{I}$ representan resultados intermedios en el proceso de análisis, y su forma y significado dependen del algoritmo descrito. Habitualmente los items de $\mathcal{H}$ representan la cadena que va a ser analizada, y, por lo tanto, $\mathcal{H} \cap \mathcal{I} = \emptyset$. Adicionalmente, será necesario un conjunto $\mathcal{F} \subseteq \mathcal{I}$ de items finales que indican el reconocimiento de la cadena de entrada.

Por su parte, los pasos deductivos permiten derivar nuevos items a partir de items ya existentes. Preferentemente los representaremos como $\{\eta_1, \dots, \eta_k\} \vdash \xi$, donde $\{\eta_1, \dots, \eta_k\}$ serán los *antecedentes*, y ξ el *consecuente*. Cada uno de estos pasos establece que si todos los antecedentes existen, entonces el analizador deberá deducir el consecuente ξ. Los pasos deductivos se suponen cuantificados universalmente para todas las variables que ocurren en ellos, a no ser que se explicite lo contrario.

EJEMPLO 4.1
Siguiendo con la descripción del algoritmo CYK iniciada en la sección 4.1, y suponiendo la cadena de entrada "bc" y la gramática:

$$S \to BC \quad B \to b \quad C \to c$$

El conjunto de hipótesis es $\mathcal{H} = \{[b, 1, 1], [c, 2, 1]\}$. *Mientras que los pasos deductivos son:*

$$[b, i, 1] \vdash [B, i, 1]$$
$$[c, i, 1] \vdash [C, i, 1]$$
$$\{[B, i, k][C, i + k, j - k]\} \vdash [A, i, k]$$

Y el conjunto de items finales $\mathcal{F} = \{[S, 1, 2]\}$. ■

Intuitivamente, el significado de los sistemas de análisis sintáctico es el siguiente: partiendo de las hipótesis, aplicamos los pasos deductivos de forma iterativa para producir nuevos items. Si alguno de ellos pertenece al conjunto de items finales, también pertenece la cadena de entrada al lenguaje generado por la gramática.

DEFINICIÓN 4.2 (relación de inferencia $\vdash$)
Sea $\mathbb{P}$ un sistema de análisis sintáctico, definiremos una relación de inferencia, $\vdash$, *sobre $\mathcal{D}$ como $\eta_1, \ldots, \eta_k \vdash \xi$, si y sólo si verifica:*

$$\{\eta_1, \ldots, \eta_k\} \vdash \xi \Rightarrow \{\eta_1, \ldots, \eta_k, \zeta\} \vdash \xi, \forall \zeta.$$

Para ello, definimos $\vdash$ como el cierre de $\mathcal{D}$ sobre la adición de antecedentes:

$$Y \vdash \xi \text{ si } (Y', \xi) \in \mathcal{D} \wedge Y' \subseteq Y \qquad \square$$

Intuitivamente, la relación de inferencia que hemos establecido nos dice que podemos deducir un ítem ξ a partir de un conjunto de items Y, si existe un paso deductivo (Y', ξ) tal que $Y' \subseteq Y$, cuyo consecuente es ξ y cuyo conjunto de antecedentes está incluido en Y.

EJEMPLO 4.2
Empleando el sistema de análisis sintáctico del ejemplo 4.1, podemos establecer, entre otras, las siguientes relaciones de inferencia:

$$\begin{array}{l} \{[b,1,1]\} \vdash [B,1,1] \\ \{[b,1,1],[c,2,1]\} \vdash [B,1,1] \\ \{[b,1,1],[C,2,1]\} \vdash [B,1,1] \\ \{[b,1,1],[c,2,1],[C,1,2]\} \vdash [B,1,1] \\ \{[b,1,1],[c,2,1],\ldots\} \vdash [C,2,1] \end{array} \qquad \blacksquare$$

Una vez definida una relación de inferencia que nos permite deducir nuevos items a partir de un conjunto de items existentes, el siguiente paso es definir la forma de enlazar en una secuencia la aplicación de varias relaciones de inferencia.

DEFINICIÓN 4.3 (secuencia de deducción)
Sea $\mathbb{P} = \langle \mathcal{I}, \mathcal{H}, \mathcal{D} \rangle$ un sistema de análisis sintáctico. Denotamos $\mathcal{I}^+$ el conjunto no vacío de secuencias finitas $\xi_1, \ldots, \xi_j$ donde $\xi_i \in \mathcal{I}$ y $1 \leq i \leq j$. Una secuencia de deducción *es un par*

$$(Y; \xi_1, \ldots, \xi_j) \in (\mathcal{H} \cup \mathcal{I})^{\mathcal{P}_{fin}} \times \mathcal{I}^+, \text{tal que } Y \cup \{\xi_1, \ldots, \xi_{i-1}\} \vdash \xi_i, \forall 1 \leq i \leq j$$

Como forma alternativa, notaremos una secuencia de deducción $(Y; \xi_1, \ldots, \xi_j)$ como $Y \vdash \xi_1 \vdash \ldots \vdash \xi_j$. Al mismo tiempo, definimos el cierre transitivo de la relación de inferencia como $Y \vdash^* \xi$ si y sólo si $\xi \in Y$, o bien, $Y \vdash \ldots \vdash \xi$. $\square$

EJEMPLO 4.3
Continuando con el ejemplo 4.2, emplearemos las relaciones de inferencia listadas para construir la siguiente secuencia de deducción:

$$\{[b,1,1],[c,2,1]\} \vdash [B,1,1] \vdash [C,2,1] \vdash [S,1,2] \qquad \blacksquare$$

Hasta el momento hemos definido los sistemas de análisis sintáctico, y la forma de operar con ellos. No obstante, tal y como indicábamos al principio, cada sistema se especifica para una gramática y cadena de entrada concretas. Para conseguir que estos sistemas resulten útiles en la práctica, hemos de extender su definición para que sean independientes de la gramática y de la cadena de entrada.

DEFINICIÓN 4.4 (sistema de análisis sintáctico no instanciado)
Dada una GIC $\mathcal{G} = (N, \Sigma, P, S)$, *denominaremos* sistema de análisis sintáctico no instanciado *a un triple* $\langle \mathcal{I}, \mathcal{H}, \mathcal{D} \rangle$, *si verifica que:*

- $\mathcal{H}$ *es una función que asigna a una cadena de entrada* $w_1 \dots w_n$ *un conjunto de hipótesis* $\mathcal{H}(w_1 \dots w_n)$
- $\langle \mathcal{I}, \mathcal{H}(w_1 \dots w_n), \mathcal{D} \rangle$ *es un sistema de análisis* □

De esta forma, sustituimos el conjunto de hipótesis de un sistema de análisis sintáctico, específico de cada cadena de entrada, por una función que transforme una entrada arbitraria en dicho conjunto de hipótesis.

EJEMPLO 4.4
Típicamente, la función de asignación de hipótesis para el algoritmo CYK es:

$$\mathcal{H}(w_1 \dots w_n) = \{[a, i, 1] \mid a = w_i \wedge 1 \leq i \leq n\}$$ ■

Una vez generalizada a cualquier cadena de entrada la definición de sistema de análisis, resta hacer lo propio, generalizando la definición a cualquier gramática.

DEFINICIÓN 4.5 (esquema de análisis sintáctico)
Sean G *una subclase de gramáticas independientes del contexto y* $(w_1 \dots w_n)$ *una cadena de entrada arbitraria, un* esquema de análisis sintáctico *es una función* P, *tal que* $P(\mathcal{G})(w_1 \dots w_n) = \langle \mathcal{I}, \mathcal{H}, \mathcal{D} \rangle$ *es un sistema de análisis no instanciado,* $\forall \mathcal{G} \in G$.□

En base a esta última definición simplemente hemos de describir los esquemas de análisis como una función que nos precise para cada gramática y cadena de entrada concretas, el sistema de análisis sintáctico correspondiente.

EJEMPLO 4.5
Llegados a este punto, podemos completar los ejemplos anteriores estableciendo el esquema de análisis sintáctico para el algoritmo CYK.

$$\mathcal{I}_{CYK} = \left\{ [A, i, j] \mid A \overset{*}{\Rightarrow} w_i \dots w_{i+j-1} \right\}$$
$$\mathcal{H}_{CYK}(w_1 \dots w_n) = \{[a, i, 1] \mid a = w_i \wedge 1 \leq i \leq n\}$$
$$\mathcal{D}_{CYK} = \left\{ \begin{array}{l} \{[a, i, 1]\} \vdash [A, i, 1], \forall A \to a \in P \\ \{[B, i, k][C, i + k, j - k]\} \vdash [A, i, j], \forall A \to BC \in P \end{array} \right\}$$ ■

Después de definir los esquemas de análisis sintáctico e ilustrarlos mediante la definición del esquema CYK, pasamos a hacer lo propio con los SDGs, observando el paralelismo entre ambos.

4.2.2 Sistema de deducción gramatical

Una vez introducida la visión de Sikkel [134], pasamos a describir los sistemas de deducción gramatical presentados por Shieber *et al.* [131].

DEFINICIÓN 4.6 (sistema de deducción gramatical)
Un sistema de deducción gramatical *(SDG) es un cuádruple* (I, A, G, R), *donde:*

- *I es un conjunto de* fórmulas, *a las que también llamaremos items.*
- *A es un conjunto de* axiomas.
- *G es un conjunto de* fórmulas objetivo.
- *R es un conjunto de* reglas de inferencia *sobre I, de la forma*

$$\frac{A_1 \quad \cdots \quad A_k}{B} \ \langle \textit{condiciones colaterales sobre } A_1, \ldots, A_k, B \rangle \qquad \square$$

Los resultados intermedios se representan mediante fórmulas de un lenguaje formal previamente establecido. Normalmente se tratará de fórmulas lógicas de primer orden, en este caso usaremos los conceptos de instancias de fórmulas y de reglas de inferencia con su significado habitual. Las reglas de inferencia nos permitirán deducir nuevas fórmulas a partir de las existentes. Para su aplicación efectiva es imprescindible que se cumplan sus condiciones colaterales.

Los axiomas son ciertos de por sí y no necesitan ser probados. Consecuentemente podemos considerar que existe una regla de inferencia implícita por cada axioma B:

$$\frac{}{B}$$

Las fórmulas objetivo representan la prueba de que la cadena de entrada pertenece a la gramática.

EJEMPLO 4.6 (sistema de deducción CYK)
Retomando el ejemplo del analizador CYK para una GIC $G = (N, \Sigma, P, S)$ y una cadena de entrada $w_{1..n}$, lo describiremos mediante un SDG (I, A, G, R) donde:

- *Al igual que los items del ejemplo 4.5, estas fórmulas representan los estados intermedios en la construcción de la matriz (T_{ij}) del algoritmo CYK:*

$$I = \{[A, i, j] \mid 1 \leq i, j \leq n,\ i + j - 1 \leq n\}$$

- *El conjunto de axiomas estará compuesto por aquellas fórmulas que representan los pasos intermedios en la fase de inicialización del algoritmo:*

$$A = \{[A, i, 1] \mid A \to w_i \in P\}$$

- *Mientras que el conjunto de objetivos tendrá un único elemento, cuyo cálculo establece la condición de pertenencia de la cadena de entrada al lenguaje generado por la gramática:*

$$G = \{[S, 1, n]\}$$

- *Por último, el conjunto de reglas de inferencia nos permite derivar nuevas fórmula a partir de otras ya existentes:*

$$R = \left\{ \frac{[B,i,k][C,i+k,j-k]}{[A,i,j]} \ \langle A \rightarrow BC \in P \rangle \right\}$$ ■

De forma análoga a los esquemas de análisis sintáctico, el proceso de análisis parte de una serie de fórmulas o items (axiomas) y aplica las reglas de inferencia disponibles para deducir nuevas fórmulas.

DEFINICIÓN 4.7 (derivación de una fórmula)
Sea (I, A, G, R) un SDG, una derivación de una fórmula *B a partir de $A_1, \ldots, A_m$ es una secuencia de fórmulas $S_1, \ldots, S_n$ tal que:*

- $S_n = B$
- $\forall i, 1 \leq i < n$
  - $S_i = A_j$, $1 \leq j \leq m$, *o bien*
  - S_i *es una instancia de un axioma, o bien*
  - $\frac{S_{i_1}, \ldots, S_{i_k}}{S_i}$, *con $i_1, \ldots, i_k < i$ es una instancia de una regla de inferencia.*

Si existe una derivación como la descrita, diremos que B es un consecuente *de $A_1, \ldots, A_m$ y lo notaremos $A_1, \ldots, A_m \vdash B$.* □

Intuitivamente, una derivación de este tipo denota una secuencia de aplicación de reglas de inferencia a un conjunto de fórmulas $A_1, \ldots, A_m$ para obtener otra fórmula B. En relación a los esquemas de análisis sintáctico, constituye la contrapartida a las secuencias de derivación.

Existe un caso particular de derivación, que se da cuando algunas reglas de inferencia tienen como antecedente el conjunto vacío. En este caso, es posible derivar su consecuente a partir del conjunto vacío de fórmulas.

DEFINICIÓN 4.8 (fórmula derivable)
Dado un sistema de deducción gramatical (I, A, G, R), diremos que una fórmula B es derivable, *y lo escribiremos $\vdash B$, si B es un consecuente del conjunto vacío.* □

En la definición dada de derivación se echa en falta poder indicar de forma explícita el orden en que se aplican las reglas de inferencia para obtener nuevas fórmulas, que unidas a las fórmulas anteriores nos permitirán aplicar nuevas reglas para, a su vez, obtener nuevas fórmulas. En este caso, estamos hablando de una *secuencia de derivación*.

DEFINICIÓN 4.9 (secuencia de derivación)
Dado un SDG (I, A, G, R), diremos que $A_1, \ldots, A_m \vdash B_1 \vdash B_2 \vdash \ldots \vdash B_n$ es una secuencia de derivación *si y sólo si:*

$$A_1, \ldots, A_m, B_1, \ldots B_{i-1} \vdash B_i, \ \forall i, 1 \leq i \leq n$$ □

Empleando las definiciones anteriores, podemos decir que una cadena de entrada pertenece al lenguaje generado por la gramática si y sólo si existe una secuencia de derivación que nos lleve desde los axiomas hasta alguna de las fórmulas objetivo.

EJEMPLO 4.7
Dado el SDG del ejemplo 4.6, la cadena de entrada "bbbc"y la gramática:

$$S \rightarrow BC \quad B \rightarrow BB \quad B \rightarrow b \quad C \rightarrow c$$

Tenemos que el conjunto de axiomas es:

$$A = \{[B,1,1],[B,2,1],[B,3,1],[C,4,1]\}$$

y podemos establecer las siguientes derivaciones:

$$[B,2,1],[B,3,1] \vdash [B,2,2]$$
$$[B,1,1],[B,2,1],[B,3,1],[C,4,1] \vdash [S,1,4]$$
$$[B,3,1],[C,4,1] \vdash [S,3,2]$$
$$[B,1,1],[B,2,1] \vdash [B,1,2]$$
$$[B,2,1],[B,3,1] \vdash [B,2,2]$$
$$[B,1,1],[B,2,2] \vdash [B,1,3]$$
$$[B,1,1],[B,2,1],[B,1,2],[B,3,1],[B,1,3][C,4,1] \vdash [S,1,4]$$
$$\ldots$$

Como vemos, es posible construir una secuencia de derivación desde los axiomas hasta la fórmula objetivo:

$$[B,1,1],[B,2,1],[B,3,1],[C,4,1] \vdash [B,1,2] \vdash [B,1,3] \vdash [S,1,4]$$
$$[B,1,1],[B,2,1],[B,3,1],[C,4,1] \vdash [B,2,2] \vdash [B,1,3] \vdash [S,1,4]$$ ■

Una vez introducidos los sistemas de deducción gramatical, en el siguiente apartado resumimos las diferencias con los esquemas de análisis sintáctico.

Comparación con esquemas de análisis sintáctico

Como ya habíamos adelantado, si olvidamos el planteamiento de sus autores, los esquemas de análisis sintáctico y los sistemas de deducción gramatical presentan una analogía casi completa. Sus principales diferencias consisten en la notación y la terminología empleadas en su definición. La tabla 4.1 muestra la correlación entre sus componentes, y para ilustrar esta idea, la figura 4.1 recoge la descripción del algoritmo CYK empleando ambos formalismos.

A pesar de la similitud de ambos formalismos, en aras de la homogeneidad de la exposición, resulta conveniente usar únicamente uno de ellos en lo que resta de la documentación. En este caso, nos decantamos por los SDG, si bien, preferiremos el término ítem en lugar de fórmula.

Esquema de análisis sintáctico	Sistema de deducción gramatical
ítems	fórmulas
pasos deductivos	reglas de inferencia
hipótesis	axiomas
ítems finales	fórmulas objetivo

Tabla 4.1: Relación entre esquemas de análisis y sistemas de deducción

Esquema de análisis sintáctico

$$\mathcal{I} = \{[A,i,j] \mid 1 \leq i,j \leq n,\ i+j-1 \leq n\}$$
$$\mathcal{H}(w_1 \ldots w_n) = \{[A,i,1] \mid A \to w_i\}$$
$$\mathcal{F} \subseteq \mathcal{I} = \{[S,1,n]\}$$
$$\mathcal{D} = \left\{ \frac{[B,i,k][C,i+k,j-k]}{[A,i,j]} \mid A \to BC \in P \right\}$$

Sistema de deducción gramatical

$$\text{Ítems} = \{[A,i,j] \mid 1 \leq i,j \leq n,\ i+j-1 \leq n\}$$
$$\text{Axiomas} = \{[A,i,1] \mid A \to w_i\}$$
$$\text{Objetivos} = \{[S,1,n]\}$$
$$\text{Reglas de inferencia} = \left\{ \frac{[B,i,k][C,i+k,j-k]}{[A,i,k]} \ (A \to BC \in P) \right\}$$

Figura 4.1: Dos descripciones del algoritmo CYK

4.3 Corrección

Siguiendo la exposición de Sikkel [134] definiremos la corrección de un SDG en función de la corrección de sus items o fórmulas. Comenzaremos con la definición de ítem mixto.

DEFINICIÓN 4.10 (ítem mixto)
Dado un SDG (I, A, G, R), diremos que un ítem B es un ítem mixto *si y sólo si tiene al menos dos interpretaciones B' y B'', tales que $B' \in G$ y $B'' \notin G$.* □

De manera intuitiva, recordemos que un ítem representa una afirmación sobre un estado del proceso de análisis de la cadena de entrada. En este sentido, un ítem mixto es aquel que representa a la vez un estado no final y un estado final del análisis. Este tipo de items mixtos son muy artificiales, por lo que en la práctica no son habituales.

EJEMPLO 4.8
Siguiendo el ejemplo del algoritmo CYK, consideremos una cadena de entrada de longitud n, siendo S el meta-símbolo de la gramática, y $A \to SC$ y $A \to BC$ dos reglas de producción. Podríamos adecuar la forma de los items a esta gramática en concreto. Para ello compartiríamos la información del análisis de ambas reglas en un único

ítem de la forma $[(S, B), i, j]$. *Como efecto colateral no deseado aparecerían items mixtos como* $[(S, B), 0, n]$. *Obsérvese que dado este ítem, no podemos determinar si la cadena de entrada pertenece o no al lenguaje definido por la gramática, puesto que no sabemos si el símbolo reducido realmente es* S *ó* B. ■

Para simplificar la exposición, consideraremos que en el conjunto de items de un SDG no existen *items mixtos*.

DEFINICIÓN 4.11 (semiregularidad)
Un SDG es semiregular *si no contiene items mixtos y por cada análisis posible de una cadena de entrada contiene un ítem que lo representa.* □

La condición de semiregularidad nos asegura que el conjunto de items construidos es válido para el algoritmo de análisis que deseamos describir. Por un lado establece de forma explícita la no existencia de items mixtos y, por otro, nos asegura que realmente podemos representar cualquier estado intermedio mediante un ítem.

EJEMPLO 4.9
En el ejemplo 4.6, que venimos empleando, el SDG para el algoritmo de análisis sintáctico CYK cumple las condiciones de semiregularidad. No existen items mixtos puesto que se establece una separación clara entre los items $[A, i, j]$ *que representan estados intermedios, y el ítem objetivo* $[S, 1, n]$, *que únicamente puede representar el reconocimiento de la cadena de entrada. Además, cada estado intermedio del análisis añadirá un nuevo símbolo no-terminal a una celda de la tabla, lo cual siempre podrá ser representado mediante un ítem de la forma* $[A, i, j]$. ■

Si la semiregularidad nos asegura que para cualquier estado del proceso de análisis tendremos un ítem que lo representa, también nos interesa que cualquier ítem represente un estado de dicho proceso.

DEFINICIÓN 4.12 (items fiables)
Dado un SDG (I, A, G, R), *diremos que un ítem es* fiable *si representa un estado en el análisis de alguna cadena de entrada, y denotaremos por* $\mathcal{C} \subseteq G$ *el conjunto de items objetivo fiables.* □

La propiedad de fiabilidad se extiende de forma directa a los SDGs.

DEFINICIÓN 4.13 (SDG fiable)
Un SDG (I, A, G, R) *semiregular es* fiable *si y sólo si* $\forall B \in G$, B *es fiable.* □

La condición de fiabilidad de un SDG nos asegura que cada vez que obtengamos un ítem objetivo, éste representa realmente un análisis de la cadena de entrada. Esta condición es necesaria para que nuestro SDG resulte útil, pero también debemos asegurar que si una cadena pertenece al lenguaje generado por la gramática, entonces es posible derivar un ítem objetivo correspondiente a su análisis.

DEFINICIÓN 4.14 (items válidos, SDG completo y SDG correcto)
Dado un SDG, diremos que un ítem es válido *si es derivable a partir de los axiomas.*
Un SDG (I, A, G, R) *semiregular es* completo *si y sólo si* $\forall B \in \mathcal{C}$, B *es válido.*
Un SDG semiregular es correcto *si es fiable y completo.* □

En la práctica la forma habitual de demostrar la corrección de un SDG es mediante el establecimiento y prueba de una invariante sobre la interpretación de los items.

EJEMPLO 4.10
Volviendo una vez más al SDG del ejemplo 4.6, estableceremos el significado de los items de la siguiente forma:

$$[\![[A,i,j]]\!] = \begin{cases} \text{verdadero} & \textit{si } A \overset{+}{\Rightarrow} w_i \ldots w_{i+j-1} \\ \text{falso} & \textit{en otro caso} \end{cases}$$

En el caso de los axiomas, esta condición es trivial por su propia definición:

$$A = \{[A,i,1] \mid A \to w_i\}$$

Por lo tanto la existencia de un axioma $[A,i,1]$ *implica que* $A \overset{+}{\Rightarrow} w_i$. *Aplicando la definición de derivación es inmediato que todos los axiomas son fiables.*

A continuación, suponiendo que partimos de dos items fiables $[B,i,k]$ *y* $[C,i+k,j-k]$, *aplicamos la regla de inferencia del esquema CYK para obtener* $[A,i,j]$*:*

$$\frac{[B,i,k][C,i+k,j-k]}{[A,i,k]} \ \langle A \to BC \in P \rangle$$

Puesto que los antecedentes son fiables, sabemos que $B \overset{+}{\Rightarrow} w_i..w_{i+k-1}$ *y* $C \overset{+}{\Rightarrow} w_{i+k}..w_{i+j-1}$. *La aplicación de la regla de inferencia anterior exige el cumplimiento de la condición* $A \to BC \in P$, *de lo cual se sigue que*

$$A \Rightarrow BC \overset{+}{\Rightarrow} w_i..w_{i+k-1}C \overset{+}{\Rightarrow} w_i..w_{i+k-1}w_{i+k}..w_{i+j-1} = w_i..w_{i+j-1}$$

Por lo tanto, el nuevo ítem es fiable. Consecuentemente cualquier ítem derivado a partir de los axiomas, incluidos los items objetivo, es fiable, y el SDG es fiable.

Resta por comprobar que del SDG es completo, es decir que todo ítem objetivo fiable es válido. Tenemos que comprobar, pues, que para cualquier cadena generada por la gramática $S \overset{*}{\Rightarrow} w_1..w_n$, *el correspondiente ítem objetivo* $[S,1,n]$ *es válido, o lo que es lo mismo, se puede derivar a partir de los axiomas. Para ello demostraremos de forma general que para toda derivación* $A \overset{*}{\Rightarrow} w_i..w_{i+j-1}$, $[A,i,j]$ *es un ítem válido. Procederemos por inducción en la longitud de* $A \overset{*}{\Rightarrow} w_i..w_{i+j-1}$*:*

1. *Caso base,* $A \Rightarrow w_i..w_{i+j-1}$ *en un solo paso. Puesto que las producciones en una gramática en forma normal de Chomsky tienen que ser de la forma* $A \to a$ *ó* $A \to BC$, *necesariamente* $j = 1$ *y* $A \to w_i \in P$. *En consecuencia, existe un axioma* $[A,i,1]$ *que por definición es fiable y válido.*

2. *Caso general. Suponemos que para toda derivación* $A \overset{*}{\Rightarrow} w_i..w_{i+j-1}$ *de longitud menor o igual que* l, $[A,i,j]$ *es válido. A continuación consideramos una derivación de longitud* $l+1$*:*

$$A \overset{l+1}{\Rightarrow} w_i..w_{i+j-1}$$

Debido a que la derivación es de longitud mayor que 1 ($l + 1 > 1$) y a la forma de las producciones, necesariamente:

$$A \Rightarrow BC \overset{l}{\Rightarrow} w_i..w_{i+j-1}$$

Por lo tanto, $B \overset{l_1}{\Rightarrow} w_i..w_{i+j_1-1}$ y $C \overset{l_2}{\Rightarrow} w_{i+j_1}..w_{i+j-1}$, con $l_1 + l_2 = l$. Aplicando la hipótesis de inducción sobre l_1 y l_2, obtenemos que los items $[B, i, j_1]$ y $[C, i + j_1, j - j_1]$ son válidos. Utilizando estos items como consecuentes de la regla de inferencia obtenemos un nuevo ítem válido $[A, i, j]$.

Una vez demostrado en general que para toda derivación $A \overset{*}{\Rightarrow} w_i..w_{i+j-1}$, $[A, i, j]$ es un ítem válido, para el caso particular $S \overset{*}{\Rightarrow} w_1..w_n$, el ítem objetivo $[S, 1, n]$ es válido y por tanto, el SDG es completo. Puesto que, como hemos visto, el SDG es fiable y completo, también es correcto. ■

En la siguiente sección aplicaremos la definición de corrección expuesta a la descripción de algoritmos de análisis bien conocidos.

4.4 SDGs básicos de análisis de GICs

En esta sección desarrollaremos los conceptos expuestos basándonos en dos algoritmos de análisis de GICs bien conocidos, y que destacan por su simplicidad. Para aligerar la exposición, nos centraremos únicamente en el problema del reconocimiento, posponiendo la generación de estructuras sintácticas de análisis hasta un apartado posterior.

4.4.1 Algoritmo descendente

Para una GIC $\mathcal{G} = (N, \Sigma, P, S)$ y una cadena de entrada $w = w_1..w_n$ construimos items de la forma $[\bullet\beta, j]$ con $0 \leq j \leq n$. La parte izquierda de la figura 4.4 muestra una representación gráfica. Tales items afirman que la subcadena de entrada $w_1..w_j$ seguida de la cadena de símbolos β es derivable a partir del meta-símbolo de la gramática S, esto es, $S \overset{*}{\Rightarrow} w_1 \cdots w_j\beta$. Dicho de otra forma, hemos reconocido hasta la posición j de la cadena de entrada y nos resta por analizar la secuencia de símbolos β. El punto $\bullet$ no es más que una referencia a la posición j.

Como corresponde a un análisis descendente, partiremos del símbolo inicial de la gramática, lo cual expresaremos mediante el axioma $[\bullet S, 0]$. El ítem objetivo $[\bullet, n]$ se corresponde con la situación de reconocimiento de la cadena de entrada hasta la posición final n, sin que reste ningún símbolo por analizar. Por último nos quedan por definir las reglas de inferencia. La primera de ellas es la regla de *reconocimiento*. El antecedente $[\bullet w_{j+1}\beta, j]$ de la regla establece que el siguiente símbolo a reconocer w_{j+1} debe ser igual al siguiente símbolo de la entrada. Por su parte, el consecuente $[\bullet\beta, j + 1]$ es un nuevo ítem donde hemos avanzado una posición en el reconocimiento de la cadena de entrada. La segunda regla es la de *predicción*. Ésta predice el uso de la producción $B \rightarrow \gamma$ para reescribir el siguiente símbolo a analizar, B. La figura 4.2 muestra el SDG completo.

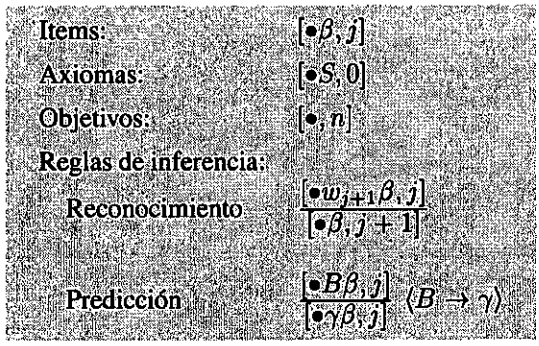


Figura 4.2: SDG del algoritmo de análisis descendente

EJEMPLO 4.11
*La figura 4.3 muestra los items que es necesario derivar para llegar al ítem objetivo, dados la gramática del ejemplo 3.2 y la cadena de entrada w = "a+a*a". El último ítem derivado es un ítem objetivo, que indica que la cadena de entrada es reconocida por la gramática.*

It_1	$[\bullet S, 0]$	axioma
It_2	$[\bullet S + S, 0]$	*predicción*$(It_1, S \to S + S)$
It_3	$[\bullet a + S, 0]$	*predicción*$(It_2, S \to a)$
It_4	$[\bullet + S, 1]$	*reconocimiento*$(It_3, ' a')$
It_5	$[\bullet S, 2]$	*reconocimiento*$(It_4, ' +')$
It_6	$[\bullet S * S, 2]$	*predicción*$(It_5, S \to S * S)$
It_7	$[\bullet a * S, 2]$	*predicción*$(It_6, S \to a)$
It_8	$[\bullet * S, 3]$	*reconocimiento*$(It_7, ' a')$
It_9	$[\bullet S, 4]$	*reconocimiento*$(It_8, ' *')$
It_{10}	$[\bullet a, 4]$	*predicción*$(It_9, S \to a)$
It_{11}	$[\bullet, 5]$	*reconocimiento*$(It_{10}, ' a')$

Figura 4.3: Ejemplo de derivación para un SDG descendente

Obsérvese, que dada la ambigüedad de la gramática, podríamos haber llegado al mismo resultado construyendo una derivación diferente que corresponde a:

$$[\bullet S, 0] \vdash [\bullet S * S, 0] \vdash [\bullet S + S * S, 0] \vdash [\bullet a + S * S, 0] \vdash [\bullet + S * S, 1] \vdash$$
$$[\bullet S * S, 2] \vdash [\bullet a * S, 2] \vdash [\bullet * S, 3] \vdash [\bullet S, 4] \vdash [\bullet a, 4] \vdash [\bullet, 5]$$

Por otra parte, dada la naturaleza predictiva de los algoritmos descendentes, es posible construir derivaciones que no produzcan items objetivos:

$$[\bullet S, 0] \vdash [\bullet S * S, 0] \vdash [\bullet S * S * S, 0] \vdash [\bullet a * S * S, 0] \vdash [\bullet * S * S, 1]$$

y en el peor de los casos, podemos construir derivaciones de longitud infinita, provocando que el proceso de análisis no termine:

$$[\bullet S, 0] \vdash [\bullet S * S, 0] \vdash [\bullet S * S * S, 0] \vdash [\bullet S * S * S, 0] \vdash [\bullet S * S * S * S, 0] \vdash \ldots$$

■

Corrección

En lo que respecta a la corrección del sistema, decíamos que los items $[\bullet\beta, j]$ representan que existe una derivación de S hasta $w_{1..j}\beta$. Más formalmente podemos expresarlo como:

$$[\![[\bullet\beta, j]]\!] = \begin{cases} \textit{verdadero} & \text{si } S \overset{*}{\Rightarrow} w_1 \cdots w_j\beta \\ \textit{falso} & \text{en otro caso} \end{cases}$$

Una vez establecido el significado de los items, podemos explorar la corrección del sistema de deducción. Para ello debemos demostrar que el sistema es fiable y completo.

Proposición 4.1 (el SDG es fiable)
Dados el SDG del algoritmo de análisis descendente de la figura 4.2 y una GIC, dicho SDG es fiable.

Demostración:

Nótese que tanto $[\bullet w_{j+1}\beta, j]$ *como* $[\bullet\beta, j+1]$ *son ciertos si* $S \overset{*}{\Rightarrow} w_1 \cdots w_j\beta$. *Por lo tanto, podemos concluir que la regla de reconocimiento aplicada a un ítem fiable, produce un ítem fiable.*

Un argumento similar se puede aplicar a la regla de predicción. *Partimos de que* $[\bullet B\beta, j]$ *es fiable, por lo tanto* $S \overset{*}{\Rightarrow} w_1 \cdots w_j B\beta$, *y, teniendo en cuenta que* $B \to \gamma \in P$, *es inmediato que* $S \overset{*}{\Rightarrow} w_1 \cdots w_j B\beta \Rightarrow w_1 \cdots w_j\gamma$, *y por lo tanto el consecuente de la regla,* $[\bullet\gamma\beta, j]$, *es fiable.*

Ya que $S \overset{*}{\Rightarrow} S$ *es trivial, podemos concluir que el axioma* $[\bullet S, 0]$ *es fiable. Puesto que el axioma es fiable y las reglas de inferencia producen items fiables, cualquier ítem válido, es decir, cualquier ítem derivable a partir del axioma, es fiable. Por lo tanto, si el ítem objetivo* $[\bullet, n]$ *es válido, será fiable, es decir,* $S \overset{*}{\Rightarrow} w_1 \cdots w_n$, *la cadena de entrada pertenece a la gramática. Consecuentemente, el SDG es fiable.* □

Para demostrar que el SDG es completo, introduciremos el siguiente resultado intermedio:

Lema 4.1
Dados el SDG del algoritmo de análisis sintáctico descendente de la figura 4.2 y una GIC, si $S \overset{*}{\Rightarrow} w_1 \cdots w_j\beta$ *es una derivación por la izquierda, entonces* $[\bullet\beta, j]$ *es válido.*

Demostración:

Para cada β *y* j *concretos, definimos su rango como la suma de* j *y la longitud de la derivación a la izquierda* $S \overset{*}{\Rightarrow} w_1 \cdots w_j\beta$ *más corta, y procedemos por inducción en éste:*

1. *En el caso base, el rango es cero. Entonces* $j = 0$ *y* $S \overset{0}{\Rightarrow} \beta$, *por lo tanto* $\beta = S$. *Es trivial que* $[\bullet S, 0]$ *es válido, pues es el axioma.*

2. *Para el paso inductivo, tomamos un rango* r *mayor que cero y suponemos que la proposición es cierta para cualquier rango menor que* r. *Se presentan dos casos:*

   Caso 1 $S \Rightarrow w_1 \cdots w_j\beta$ *en un único paso, por lo tanto,* $S \rightarrow w_1 \cdots w_j\beta \in P$. *La aplicación de la regla predicción al axioma del SDG produce el ítem* $[\bullet w_1 \cdots w_j\beta]$ *y tras* j *aplicaciones de la regla de reconocimiento obtenemos* $[\bullet\beta, j]$.

   Caso 2 $S \overset{*}{\Rightarrow} w_1 \cdots w_j\beta$ *en* $n+1$ *pasos. Por las definiciones de GIC y derivación a la izquierda,* $S \overset{*}{\Rightarrow} w_1 \cdots w_{j-k}B\gamma' \Rightarrow w_1 \cdots w_j\gamma\gamma'$, *con* $\beta = \gamma\gamma'$ *y* $B \rightarrow w_{j-k+1} \cdots w_j\gamma \in P$. *Por hipótesis de inducción* $[\bullet B\gamma', j-k]$ *es válido. Por la regla de predicción,* $[\bullet w_{j-k+1} \cdots w_j\gamma\gamma', j-k]$ *es válido y, tras* k *aplicaciones de la regla de reconocimiento, obtenemos* $[\bullet\beta, j]$. □

Proposición 4.2 (el SDG es completo)
Dados el SDG del algoritmo de análisis sintáctico descendente de la figura 4.2 y una GIC, dicho SDG es completo.

Demostración:
Para ver si el sistema es completo, tendremos que comprobar que para toda cadena de entrada reconocida por la gramática, $S \overset{*}{\Rightarrow} w_1 \cdots w_n$, $[\bullet, n]$ *es válido, es decir, para cualquier cadena reconocida por la gramática, podemos derivar el ítem objetivo correspondiente. Por el lema anterior, si* $S \overset{*}{\Rightarrow} w_1 \cdots w_n$, *entonces el ítem* $[\bullet, n]$ *es válido y, consecuentemente, el SDG es completo.* □

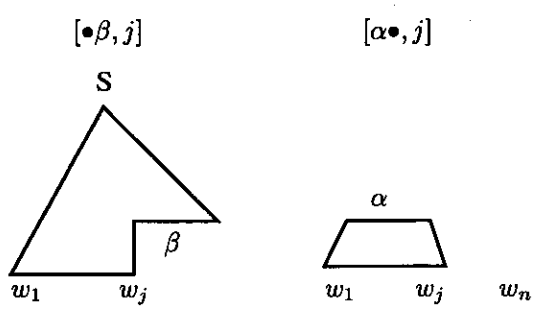


Figura 4.4: Representación gráfica de los items

4.4.2 Algoritmo ascendente

Otro algoritmo de amplia difusión es el algoritmo de análisis sintáctico ascendente cuyo SDG se muestra en la figura 4.5. Los items son de la forma $[\alpha\bullet, j]$ e indican que la

cadena de entrada es derivable a partir de la secuencia de símbolos $\alpha w_{j+1..n}$. La parte derecha de la figura 4.4 muestra una representación gráfica. De nuevo el punto $\bullet$ es una referencia a la posición j en la cadena de entrada. Más formalmente, el significado de los items es:

$$[[\alpha\bullet, j]] = \begin{cases} \textit{verdadero} & \text{si } \alpha w_{j+1} \dots w_n \stackrel{*}{\Rightarrow} w_1 \cdots w_n \\ \textit{falso} & \text{en otro caso} \end{cases}$$

El axioma $[\bullet, 0]$ representa la situación inicial en que no existe ningún progreso en el análisis. Si la cadena de entrada es reconocida por la gramática, $S \stackrel{*}{\Rightarrow} w$, hemos de ser capaces de derivar el ítem objetivo $[\bullet S, n]$. Para completar el SDG necesitamos dos reglas de deducción. La primera, desplazamiento, análoga a la regla de reconocimiento del algoritmo descendente, partiendo del antecedente $[\alpha\bullet, j]$ añade el siguiente símbolo de entrada w_{j+1} a la secuencia de símbolos reconocidos, avanzando el punto una posición en la cadena de entrada, y generando el consecuente $[\alpha w_{j+1}\bullet, j+1]$. La siguiente regla, de reducción, se aplica cuando los últimos k símbolos reconocidos coinciden con la parte derecha de una producción $B \to \gamma$. Estos símbolos son sustituidos en el ítem antecedente $[\alpha\gamma\bullet, j]$ por la parte izquierda de la producción en el consecuente $[\alpha B\bullet, j]$. Puesto que no hemos reconocido ningún símbolo nuevo de la cadena de entrada, no avanzamos el punto.

Retomando el ejemplo anterior, mostraremos los items que es necesario derivar para llegar al ítem objetivo, dados la gramática del ejemplo 3.2 y el análisis de la cadena de entrada $w = a + a * a$.

1	$[\bullet, 0]$	axioma
2	$[a\bullet, 1]$	*desplazamiento*(1)
3	$[S\bullet, 1]$	*reducción*(2)
4	$[S + \bullet, 2]$	*desplazamiento*(3)
5	$[S + a\bullet, 3]$	*desplazamiento*(4)
6	$[S + S\bullet, 3]$	*reducción*(5)
7	$[S\bullet, 3]$	*reducción*(6)
8	$[S * \bullet, 4]$	*desplazamiento*(7)
9	$[S * a\bullet, 5]$	*desplazamiento*(8)
10	$[S * S\bullet, 5]$	*reducción*(9)
10	$[S\bullet, 5]$	*reducción*(10)

Podemos concluir que la cadena de entrada es reconocida por la gramática, pues hemos sido capaces de derivar el ítem objetivo.

Tal y como hicimos para el algoritmo descendente, debemos demostrar que el sistema es fiable y completo, para asegurar su corrección.

Proposición 4.3 (el SDG es fiable)
Dados el SDG del algoritmo de análisis descendente de la figura 4.5 y una GIC, dicho SDG es fiable.

Demostración:

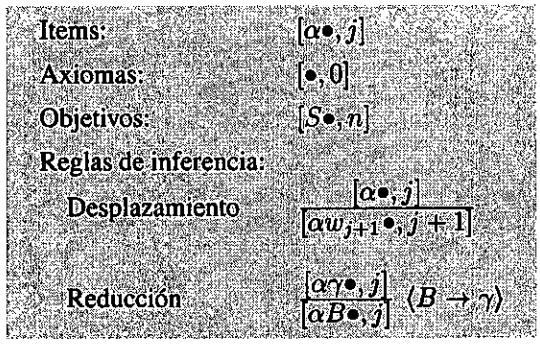


Figura 4.5: SDG del algoritmo de análisis ascendente

Tanto $[\alpha\bullet, j]$ *como* $[\alpha w_{j+1}\bullet, j+1]$ *son ciertos si* $\alpha w_{j+1} \dots w_n \stackrel{*}{\Rightarrow} w$. *Consecuentemente la regla de desplazamiento, aplicada a un ítem fiable, deriva otro ítem fiable.*

Por otro lado, si el ítem antecedente, $[\alpha\gamma, j]$, *de la regla de reducción es fiable, entonces* $\alpha\gamma w_{j+1} \dots w_n \stackrel{*}{\Rightarrow} w$. *Si se cumple la condición* $B \to \gamma$ *podemos deducir que* $\alpha B w_{j+1} \dots w_n \stackrel{*}{\Rightarrow} w$ *y, por lo tanto, el ítem consecuente derivado es fiable.*

El axioma $[\bullet, 0]$ *es fiable, puesto que de forma trivial,* $w_1 \dots w_n \stackrel{*}{\Rightarrow} w_1 \dots w_n$. *Teniendo en cuenta que las reglas de deducción son fiables, cualquier ítem derivado a partir del axioma, incluido el ítem objetivo, será fiable. Concluyendo, si el ítem objetivo es válido, será fiable, o lo que es lo mismo, el SDG es fiable.* □

Como paso previo a la demostración de la completud del SDG, probaremos el siguiente resultado:

Lema 4.2
Dados el SDG del algoritmo de análisis descendente de la figura 4.2 y una GIC, si $\alpha w_{j+1} \dots w_n \stackrel{*}{\Rightarrow} w_1 \dots w_n$ *es una derivación a la derecha, entonces* $[\alpha\bullet, j]$ *es un ítem válido.*

Demostración:

Definimos el rango de α *y* j *como la suma de* j *y la longitud de la derivación a la derecha* $\alpha w_{j+1} \dots w_n \stackrel{*}{\Rightarrow} w_1 \dots w_n$ *más corta. Procedemos por inducción en el rango.*

Para el caso base, el rango es 0, $j = 0$ *y* $\alpha = \epsilon$. *Trivialmente,* $[\bullet, 0]$ *es válido por ser el axioma.*

Para el paso inductivo, tomamos un rango r *mayor que cero y suponemos que la proposición es cierta para cualquier rango menor que* r. *Se presentan dos casos:*

Caso 1 $\alpha w_j \dots w_n \Rightarrow w_1 \dots w_n$ *en un único paso, por lo tanto* $\alpha = w_{1..i} B w_{k+1..j}$ *y* $B \to w_{i+1} \dots w_k$. *Aplicando* k *veces la regla de desplazamiento al axioma obtenemos el ítem* $[w_1 \dots w_k\bullet, k]$, *al que aplicaremos la regla de reducción para obtener* $[w_1 \dots w_i B\bullet, k]$. *Tras* $j - k$ *desplazamientos más, obtendremos* $[\alpha\bullet, j]$.

Caso 2 *$\alpha w_j \dots w_n \overset{*}{\Rightarrow} w_1 \dots w_n$ en $n+1$ pasos. Por ser una derivación a la derecha sobre una GIC, $\alpha = \alpha' B$, $B \to w_i \dots w_{j-1}$, $\alpha w_j \dots w_n \Rightarrow \alpha' w_i \dots w_j \dots w_n$ en un paso, y $\alpha' w_i \dots w_n \overset{*}{\Rightarrow} w_1 \dots w_n$ en n pasos. Por hipótesis de inducción $[\alpha' \bullet, i]$ es válido. Aplicando las reglas de reducción y desplazamiento de manera análoga a como hicimos en el caso anterior obtenemos $[\alpha' B, j]$, o lo que es lo mismo $[\alpha, j]$.* □

Proposición 4.4 (el SDG es completo)
Dados el SDG del algoritmo de análisis descendente de la figura 4.2 y una GIC, dicho SDG es completo.

Demostración:
Debemos comprobar que para toda cadena de entrada reconocida por la gramática ($S \overset{*}{\Rightarrow} w_1 \dots w_n$) es posible derivar el ítem objetivo $[S\bullet, n]$. Por el lema anterior, si $S \overset{*}{\Rightarrow} w_1 \cdots w_n$, entonces el ítem $[S\bullet, n]$ es válido y, consecuentemente, el SDG es completo. □

4.5 Transformación

Tanto Sikkel [132, 134], como Alonso [8], desarrollan métodos formales para obtener algoritmos de análisis correctos a partir de otros algoritmos que ya se ha demostrado que son correctos. En nuestro caso, presentaremos estas ideas desde un punto de vista menos formal, basándonos siempre en ejemplos concretos.

4.5.1 Algoritmo de Earley

Frecuentemente el algoritmo de Earley [47] se clasifica como un algoritmo mixto. Como veremos a continuación esta definición está justificada por el hecho de que es posible obtener dicho algoritmo como una combinación natural de los algoritmos descendente y ascendente. Partiendo de las invariantes que establecían los items en ambos algoritmos y combinándolas, tenemos que:

$$\frac{\begin{array}{ll}[\bullet\beta, j] & S \overset{*}{\Rightarrow} w_1 \dots w_j\beta \\ [\alpha\bullet, j] & \alpha w_{j+1} \dots w_n \overset{*}{\Rightarrow} w_1 \dots w_n\end{array}}{\begin{array}{ll}[\alpha \bullet \beta, j] & S \overset{*}{\Rightarrow} w_1 \dots w_j\beta \\ & \alpha w_{j+1} \dots w_n \overset{*}{\Rightarrow} w_1 \dots w_n\end{array}}$$

Pero el algoritmo de Earley, aún añade un refinamiento más. Los items considerados hasta ahora son globales, en el sentido que establecen afirmaciones sobre el análisis de la cadena de entrada al completo. En lugar de esto, emplearemos items con información local, referida al análisis de la parte derecha de una producción $A \to \alpha\beta$, desde la posición i:

$$\begin{array}{ll}[i, A \to \alpha \bullet \beta, j] & S \overset{*}{\Rightarrow} w_1 \dots w_i A\gamma \\ & \alpha w_{j+1} \dots w_n \overset{*}{\Rightarrow} w_{i+1} \dots w_n \\ & A \to \alpha\beta \in P\end{array}$$

La figura 4.6 muestra una representación gráfica de los items. En la parte izquierda, se muestra el resultante de la mezcla directa de los items del algoritmo descendente y ascendente, $[\alpha \bullet \beta, j]$, y en la parte derecha, los items tal y como se definen en el algoritmo de Earley, $[i, A \rightarrow \alpha \bullet \beta, j]$.

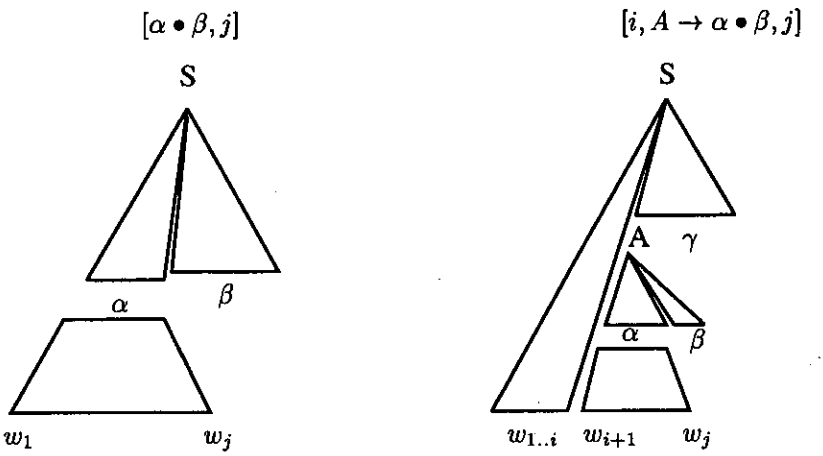


Figura 4.6: Representación gráfica de los items

Una vez definidos los items, hemos de hacer lo propio con los axiomas e items objetivos. Para simplificar la descripción presupondremos gramáticas aumentadas en el caso del algoritmo de Earley.

	Descendente	Ascendente	Earley
Axiomas	$[\bullet S, 0]$	$[\bullet, 0]$	$[0, S' \rightarrow \bullet S, 0]$
Objetivos	$[\bullet, n]$	$[S\bullet, n]$	$[0, S' \rightarrow S\bullet, n]$

Para finalizar el SDG debemos definir los pasos deductivos. La combinación de los algoritmos descendente y ascendente da lugar a tres pasos distintos: *reconocimiento*, *predicción* y *compleción*.

Reconocimiento Al igual que ocurría anteriormente, este paso combina sus pasos homónimos en los algoritmos descendente y ascendente, reconociendo el siguiente símbolo de la cadena de entrada y avanzando una posición en la misma.

$$\frac{[i, A \rightarrow \alpha \bullet w_{j+1}\beta, j]}{[i, A \rightarrow \alpha w_{j+1} \bullet \beta, j+1]}$$

Predicción Se corresponde con el paso predicción del algoritmo descendente, que no existe en el ascendente, y de forma análoga, predice el uso de la producción $B \rightarrow \gamma$ en el análisis del símbolo B, inmediatamente a la derecha del punto, esto es, de la posición j.

$$\frac{[i, A \rightarrow \alpha \bullet B\beta, j]}{[j, B \rightarrow \bullet\gamma, j]} \ \langle B \rightarrow \gamma \rangle$$

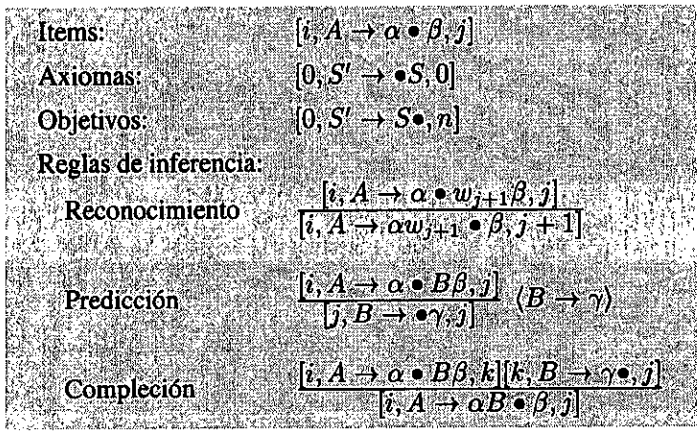


Figura 4.7: SDG del algoritmo de Earley

Compleción Este paso no existe en el algoritmo descendente, y en el ascendente se corresponde con el de reducción. Una vez completado el análisis de la parte derecha de la producción $B \to \gamma$ entre las posiciones k y j, avanzamos en el análisis de su parte derecha, B, en aquellos items en los que estuviese pendiente.

$$\frac{[i, A \to \alpha \bullet B\beta, k]\,[k, B \to \gamma \bullet, j]}{[i, A \to \alpha B \bullet \beta, j]}$$

La figura 4.7 reúne todos los elementos descritos del SDG para el algoritmo de Earley.

4.5.2 Algoritmos LR

Ateniéndonos al orden cronológico, el algoritmo de Earley [48] puede ser considerado una versión interpretada de los algoritmos de la familia LR, desarrollados originalmente por Knuth [73]. A continuación mostraremos esta relación en sentido inverso, es decir, obteniendo la descripción de los algoritmos de análisis sintáctico LR como una versión compilada del algoritmo de Earley. Durante esta compilación, se factorizan parte de los pasos de deducción, creando el control finito, para simplificar el proceso de análisis. Desde el punto de vista de los SDGs esta compilación agrupa conjuntos de items equivalentes, representados por estados del control finito. La mejora en la eficacia del algoritmo se produce al aplicar los pasos deductivos directamente sobre los conjuntos de items, en lugar de aplicarlos sobre cada uno de los items pertenecientes a cada conjunto. Para introducir la transformación del algoritmo de Earley en un algoritmo LR(1) de manera más ilustrativa, realizaremos varias transformaciones intermedias.

LR(0) sin control finito

En primer lugar describiremos un algoritmo LR(0) sin control finito. Los items tienen la misma estructura que en el algoritmo de Earley: $[i, A \to \alpha \bullet \beta, j]$. La diferencia

estriba en su significado, se modifica para que representen el análisis de una porción menor de la cadena de entrada. En el caso de Earley, la cadena α de símbolos a la izquierda del punto deriva la subcadena de entrada entre las posiciones i y j, esto es $\alpha \stackrel{*}{\Rightarrow} w_{i,j}$. En el nuevo algoritmo, sólo el último símbolo de α deriva $w_{i,j}$, esto es $X \stackrel{*}{\Rightarrow} w_{i,j}$, con $\alpha = \alpha' X$.

$$[i, A \to \alpha \bullet \beta, j] \quad \begin{array}{l} S \stackrel{*}{\Rightarrow} w_1 \dots w_i A\gamma \\ Xw_{j+1} \dots w_n \stackrel{*}{\Rightarrow} w_{i+1} \dots w_n \\ A \to \alpha\beta \in P, \ \ \alpha = \alpha' X \end{array}$$

En lo que respecta a los axiomas y objetivos, seguiremos usando los mismos que en el algoritmo de Earley:

$$\begin{array}{ll} \text{Axiomas} & [0, S' \to \bullet S, 0] \\ \text{Objetivos} & [0, S' \to S\bullet, n] \end{array}$$

La siguiente diferencia la encontramos en los pasos deductivos, que tenemos que adaptar al nuevo significado de los items, en especial, el paso de *compleción*.

Desplazamiento En primer lugar, cambiamos el nombre de reconocimiento por el de desplazamiento por ser el que se usa habitualmente en la descripción de los algoritmos tipo LR. A diferencia de Earley, el ítem generado indica la subcadena derivada por el símbolo inmediatemente a la izquierda del punto, es decir, $w_{j,j+1}$.

$$\frac{[i, A \to \alpha \bullet w_{j+1}\beta, j]}{[j, A \to \alpha w_{j+1} \bullet \beta, j+1]}$$

Predicción Este paso deductivo no cambia con respecto al que empleábamos en el algoritmo de Earley.

$$\frac{[i, A \to \alpha \bullet B\beta, j]}{[j, B \to \bullet\gamma, j]} \ \langle B \to \gamma \rangle$$

Reducción Al igual que hicimos con el paso de desplazamiento, emplearemos el nombre habitual reconocimiento en lugar del nombre que habíamos usado en Earley: compleción. En este paso deductivo es donde encontramos la principal diferencia. En Earley, el antecedente es un ítem $[i, A \to \alpha \bullet B\beta, k]$, en el cual el siguiente símbolo a analizar, a partir de la posición k, es B, y un único ítem $[k, B \to \gamma\bullet, j]$ que representa el análisis de B entre las posiciones k y j. En nuestro nuevo algoritmo necesitaremos sustituir este último ítem por m items que representen el análisis de cada uno de los símbolos X_i de γ.

$$\frac{\begin{array}{l} [i, A \to \alpha \bullet B\beta, k] \\ [k, B \to \bullet X_1 X_2 \cdots X_m, j_1] \\ \dots \\ [j_m, B \to X_1 X_2 \cdots X_m \bullet, j] \end{array}}{[k, A \to \alpha B \bullet \beta, j]}$$

Nótese que la única forma de producir un ítem $[k, B \rightarrow \bullet X_1 X_2 \cdots X_m, j_1]$ es mediante un paso predicción, y por lo tanto $j_1 = k$.

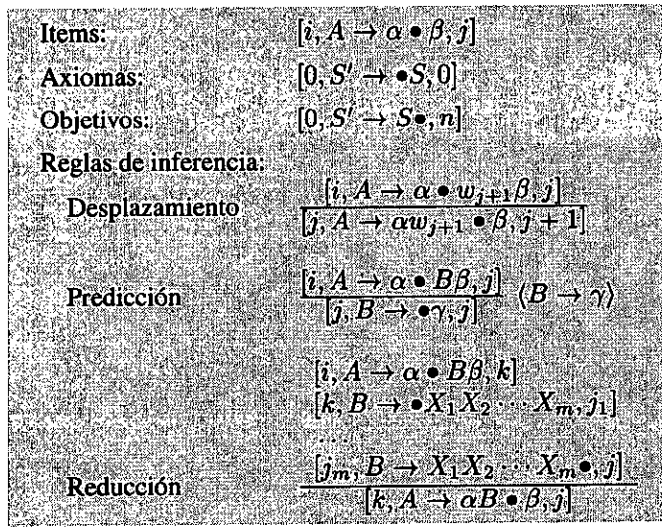


Figura 4.8: SDG del algoritmo LR(0) sin control finito

Para terminar este apartado, la figura 4.8 recapitula el SDG descrito.

LR(0) con control finito

Para mejorar la eficiencia del algoritmo anterior, lo modificaremos introduciendo un control finito. La existencia de dicho control implica dividir el proceso de análisis en dos fases. En la primera, debemos crear las clases de equivalencia de los items. Cada clase estará representada por un estado. El conjunto de estados y las transiciones válidas entre ellos conforman el control finito. Este control será específico para cada gramática. En la segunda fase analizaremos las cadenas de entrada. El proceso de análisis se guiará por el control finito creado durante la fase de compilación, sustituyendo los items por los estados representativos de sus clases de equivalencia.

La construcción de las clases de equivalencia ya ha sido expuesta en el capítulo 3. Examinando la operación de cierre, vemos que realiza los mismos cálculos que el paso deductivo de predicción en la versión del algoritmo sin control finito. Por esta razón dicho paso deductivo no es necesario en la versión con control finito. Dicho control viene definido por el conjunto de estados y las funciones *acción*, *ir_a* y *revela*. Las dos primeras ya fueron tratadas al hablar de la construcción del autómata LR. Por su parte, la función *revela*, es una función de conveniencia que nos indica para un estado dado, desde que estados podemos llegar aplicando alguna de las acciones válidas del autómata, bien sea un desplazamiento o una reducción. Esta función nos servirá para establecer la relación entre los items que constituyen el antecedente del paso deductivo de reducción.

La segunda fase del algoritmo, en la que se realiza el análisis efectivo de la cadena de entrada es común para todas las gramáticas y puede ser descrita mediante un SDG. En primer lugar sustituimos los items por el estado representativo de su clase de equivalencia y recuperamos los índices a la subcadena de entrada analizada: $[i, st, j]$. A continuación hacemos lo propio con los axiomas y objetivos:

$$\begin{array}{ll} \text{Axiomas} & [0, st_0, 0] \\ \text{Objetivos} & [0, st_f, n] \end{array}$$

Para completar el SDG, debemos redefinir los pasos deductivos. Como ya se mencionó anteriormente, el paso predicción se realiza durante el cálculo del cierre, y por lo tanto está implícito en el control finito, y podemos eliminarlo.

Desplazamiento Como ya hicimos con los axiomas y objetivos, sustituimos los items por sus estados representativos. A mayores incluimos en las condiciones colaterales comprobar en el control finito que es posible realizar un desplazamiento al estado st', consecuente, para el estado st, antecedente, y el símbolo w_j en la posición j de la cadena de entrada.

$$\frac{[i, st, j]}{[j, st', j+1]} \ \langle \text{desplazamiento}_{st'} \in \text{acción}(st, w_j) \rangle$$

Reducción Aplicamos el mismo razonamiento para los items y estados. Igualmente hemos de realizar varias comprobaciones. En primer lugar comprobaremos que es posible ir del estado st al st_1 mediante el símbolo B, $st_1 \in \text{ir\_a}(st, B)$, o dicho de otra forma, que existe un paso predicción de una regla del tipo $B \to \gamma$. A continuación comprobaremos que todos los items $[j_i, st_{i+1}, j_{i+1}]$ se pueden generar mediante un desplazamiento o reducción de un ítem anterior $[j_{i-1}, st_i, j_i]$, $st_i \in \text{revela}(st_{i+1})$. Por último, una vez que tenemos un ítem por cada uno de los símbolos de la parte derecha de la regla $B \to \gamma$, comprobaremos que es posible realizar una reducción.

$$\frac{\begin{array}{l}[i, st, k] \\ [k, st_1, j_1] \\ \cdots \\ [j_m, st_{m+1}, j]\end{array}}{[k, st', j]} \left\langle \begin{array}{l} \textit{reducción}_{B \to \gamma} \in \text{acción}(st_{m+1}), \\ st_1 \in \text{ir\_a}(st, B),\ st_i \in \text{revela}(st_{i+1})\ \forall i, \\ m = \text{longitud}(\gamma) \end{array} \right\rangle$$

El SDG completo queda recogido en la figura 4.9.

Símbolos adelantados: LR(k)

La siguiente transformación sobre el algoritmo anterior consiste en el uso de los símbolos adelantados para la construcción del control finito. Cada acción se especifica en función del estado del autómata y de los símbolos compatibles. La acción descrita sólo

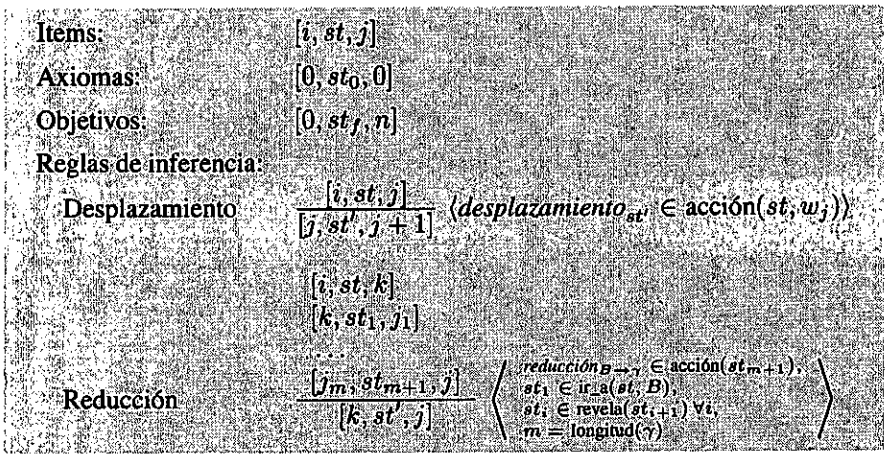


Figura 4.9: SDG del algoritmo LR(0) con control finito

se aplicará si estos símbolos coinciden con los que se encuentren en las siguientes posiciones de la cadena de entrada. Este cambio en el control finito implica modificar los pasos deductivos del SDG a la hora de consultar dicho control, tal y como se muestra en la figura 4.10. A este respecto, recordaremos que las variantes SLR(k) y LALR(k) únicamente difieren del LR(k) en la forma de construir el control finito, por lo que el SDG será el mismo.

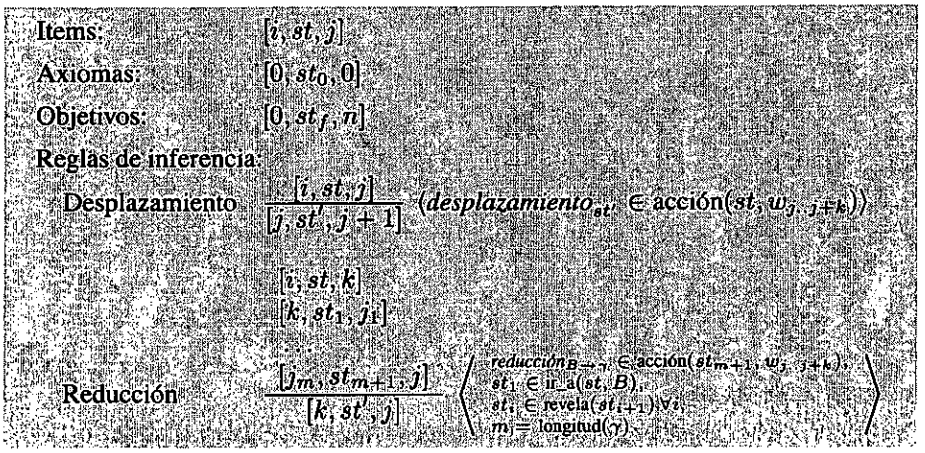


Figura 4.10: SDG del algoritmo LR(k) con control finito

4.6 Arboles de análisis

Hasta el momento, los algoritmos descritos tratan solamente el problema del reconocimiento, limitándose a decidir si una cadena de entrada pertenece al lenguaje generado por una gramática, es decir, si en el proceso de inferencia obtenemos un ítem objetivo. Pero para poder hablar con propiedad de analizadores sintácticos, es necesario construir una estructura que represente el proceso de análisis. Tradicionalmente, en las GICs esta estructura es un árbol de análisis o, en su defecto, una estructura que nos permita construir dicho árbol de análisis.

Para transformar los SDG vistos de algoritmos de reconocimiento en algoritmos de análisis, añadiremos un nuevo componente a los items. En este nuevo componente iremos construyendo la estructura resultante del análisis. La figura 4.11 muestra esta idea sobre el SDG de análisis descendente definido en la figura 4.2. Hemos añadido a los items una lista en la que se acumulan reglas y símbolos terminales a medida que avanza el proceso de análisis. Una vez terminado éste, es posible construir una representación del árbol de análisis, recorriendo la lista y expandiendo los nodos del árbol de derecha a izquierda según se trate de una regla o un terminal.

4.7 Implementación

Una vez desarrollado un SDG es necesario transformarlo en un software que podamos ejecutar. Para ello disponemos principalmente de tres alternativas:

- Crear una implementación *ad-hoc* a partir del SDG
- Emplear un meta-intérprete.
- Traducir el SDG a un formalismo operativo.

A continuación examinaremos cada una de estas alternativas.

4.7.1 Implementación *ad-hoc*

Para esta alternativa, simplemente desarrollaremos un algoritmo siguiendo las especificaciones dadas por el SDG. De nuevo rompiendo el orden cronológico, podemos pensar que en el trabajo de Earley [47] se desarrolla una implementación del SDG 4.7. Como muestra, citaremos algunos puntos del apartado dedicado a la implementación en [47]:

> (1) Por cada variable, mantenemos una lista enlazada de sus alternativas, para su uso en la predicción.
>
> (4) Para el uso de la compleción, también mantendremos, por cada conjunto de estados S_i y variable N, una lista de todos los estados $\langle p, j, f, \alpha \rangle \in S_i$ tales que $C_p(j+1) = N$.

Sobre esta aproximación señalaremos como principal ventaja la posibilidad de obtener implementaciones de gran eficiencia. Por lo que respecta a las desventajas destacaremos:

- Necesidad de un esfuerzo, no intuitivo, de traducción del SDG a la especificación del algoritmo.
- Ausencia de un método formal que pruebe la correción de dicha traducción.

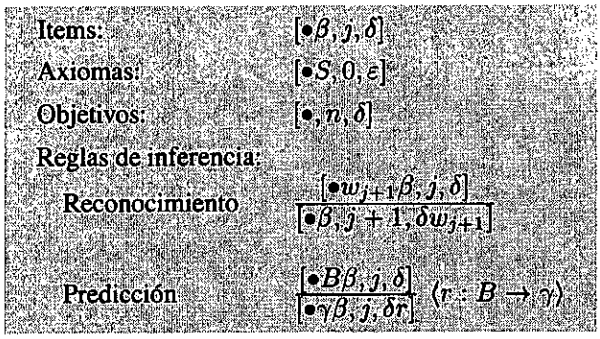


Figura 4.11: SDG del algoritmo de análisis descendente

4.7.2 Meta-interpretación

Junto con los SDGs, en [131] también se presenta un meta-intérprete implementado en PROLOG. Éste consta de tres partes principales: un grafo dónde se almacenan los items generados, una agenda dónde se almacenan los items pendientes de procesar, y un motor de deducción específico que genera nuevos items a partir de los existentes. De esta manera es posible ejecutar un analizador a partir de su definición como sistema de deducción, sin necesidad de llegar a una implementación *ad-hoc*.

El principal cometido de un meta-intérprete es prototipar, es decir, realizar pruebas de concepto sobre los sistemas de análisis a medida que los desarrollamos. De esta manera evitamos el esfuerzo innecesario de realizar implementaciones que se invalidan acto seguido, en cuanto se corrigen los defectos del sistema.

Otra aplicación de la meta-interpretación es la comparación de diferentes sistemas de deducción dentro de un marco común. Precisamente, Díaz y Alonso [46] presentan una comparación entre seis tipos de analizadores sintácticos para gramáticas de adjunción de árboles. La ejecución de los experimentos se instrumentó usando el meta-intérprete definido por Shieber *et al.* en [131]. Gracias a la abstracción de los SDGs, esta comparación resulta más coherente puesto que se realizó sobre parámetros generales como son el número de items generados por el analizador, o el número de adjunciones realizadas, abstrayéndose de detalles específicos de implementación como los tiempos de ejecución o la memoria ocupada. Nótese que los parámetros generales son comunes a todos los analizadores, mientras que en los parámetros más específicos inciden mucho más las implementaciones concretas y no el esquema de análisis en sí mismo.

Como se podría esperar, la principal desventaja de la meta-interpretación es la falta de eficiencia. Si bien sería suficiente para las aplicaciones descritas anteriormente,

no lo es para su puesta en producción, por lo que una vez desarrollado un esquema satisfactorio, deberemos buscar una implementación lo más eficiente posible. Esto no quiere decir que el tema quede cerrado, a continuación veremos y propondremos líneas de trabajo dirigidas a aumentar la eficacia de los meta-intérpretes.

Como ejemplo señalaremos el trabajo de Morawietz [94] cuya propuesta es representar los items del SDG como restricciones y los pasos deductivos como reglas para la manipulación de restricciones. Esta idea se justifica por el hecho de que, en este caso, los sistemas de manejo de restricciones son más eficientes que, por ejemplo, el mecanismo de resolución de PROLOG en el que se apoya el meta-intérprete de [131].

4.7.3 Traducción a un formalismo operativo

La tercera alternativa propuesta para la implementación de los SDGs es su traducción a un formalismo con un carácter más operativo. Como ejemplo se proponen los *autómatas de pila* (APs) [3, 59], y los *autómatas lógicos de pila*[3] (ALPs) desarrollados inicialmente por Lang [82, 81], en primer lugar como un mecanismo operacional para la ejecución de programas Datalog, extendiéndolos posteriormente a programas de cláusulas definidas. La definición y uso de los ALPs se desarrollará en la presente memoria. Esta alternativa se sitúa entre las dos anteriores, buscando paliar sus defectos principales:

- Falta de eficiencia. El formalismo destino de la traducción posee un carácter marcadamente operativo lo que facilita la obtención de un intérprete o compilador eficientes.
- Dificultad de traducción. El formalismo destino conserva un nivel de abstracción adecuado, de forma que se facilita la traducción.

[3] *Logical PushDown Automata* (LPDA) en la terminología original.

CAPÍTULO 5

AUTÓMATAS DE PILA Y TABULACIÓN

En el presente capítulo se desarrolla el empleo de los autómatas de pila para la construcción de analizadores. Siguiendo la línea establecida, la exposición se desarrolla para el caso de los analizadores de GICs, sentando la base sobre la que se desarrollará su extensión y aplicación al análisis de otro tipo de gramáticas más complejas. Asimismo, se introduce la aplicación de las técnicas de tabulación a los autómatas obtenidos para la mejora de sus características. Finalmente se describe el sistema ICE, de generación de analizadores sintácticos para GICs sin restricciones. El trabajo posterior se desarrollará sobre la base de dicho analizador.

5.1 Autómatas de pila

En primer lugar realizaremos una introducción de los APs, fijándonos en su definición habitual [3]. Esta introducción servirá para fijar la notación a emplear en lo sucesivo. Recordemos que el objeto de estos autómatas es su aplicación al desarrollo de analizadores sintácticos para GICs.

DEFINICIÓN 5.1 (autómata de pila)
Un autómata de pila es una 7-tupla $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, \$_0, Q_F)$*, donde:*

- Q *es un conjunto finito de* estados.
- Σ *es un conjunto finito de* símbolos terminales.
- Γ *es un conjunto finito de* símbolos de pila.
- $q_0 \in Q$ *es el* estado inicial.
- $\$_0 \in \Gamma$ *es el* símbolo inicial de la pila.
- $\mathcal{F} \subseteq Q$ *es el conjunto de* estados finales.
- δ *es el conjunto finito de* transiciones *del autómata:*

$$\begin{array}{rccc} \delta: & Q \times \Gamma \times (\Sigma \cup \{\varepsilon\}) & \longrightarrow & Q \times \Gamma^* \\ & (q, Z, a) & \rightsquigarrow & (q', \beta) \end{array}$$

□

Intuitivamente, las transiciones tienen el siguiente significado: el autómata se encuentra en el estado q, el siguiente terminal de la cadena de entrada es a y el símbolo en la cima de la pila es Z. Entonces, aplicando la transición, el autómata pasa al estado q' y sustituye la cima de la pila por β. Las transiciones permiten evolucionar entre los estados intermedios del proceso de análisis. Cada uno de estos estados vendrá representado por una *configuración*. Por tanto, habrá una *configuración inicial* que represente el comienzo del análisis, y que vendrá dada por el estado inicial de autómata, la cadena a analizar y una pila que únicamente contenga el símbolo inicial.

DEFINICIÓN 5.2 (configuración)
Dados un AP $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, \$_0, Q_F)$ *y una cadena de entrada* w *diremos que una* configuración *del autómata en un momento del análisis es un triple* (q, α, x)*, donde* q *es el estado en que se encuentra o* estado actual*,* α *es el contenido de la pila, y* x *es la parte de la cadena de entrada que resta por analizar. La* configuración inicial *viene dada por:* $(q_0, \$_0, w)$.
La aplicación de una transición $(q', \beta) \in \delta(q, Z, a)$ *a una configuración* $(q, \alpha Z, ax)$ *produce una nueva configuración* $(q', \alpha\beta, x)$*, lo cual notaremos como:*

$$(q, \alpha Z, ax) \vdash (q', \alpha\beta, x)$$

Asimismo, notaremos el cierre transitivo de $\vdash$ *como* $\overset{+}{\vdash}$*, y el cierre reflexivo y transitivo como* $\overset{*}{\vdash}$. □

De la misma forma que la configuración inicial marca el comienzo del análisis, habrá una serie de *configuraciones finales* que marquen el final del análisis, aunque en este caso tenemos dos alternativas. En ambos casos la cadena de entrada se ha analizado por completo, y, mientras que en el primero la configuración final es aquella en que el autómata alcanza un estado final, en el segundo será aquella en que la pila se ha vaciado. Estos dos casos dan lugar a dos definiciones del *lenguaje aceptado por el autómata*.

DEFINICIÓN 5.3 (lenguaje aceptado por un autómata)
Dado un AP $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, \$_0, Q_F)$*, el* lenguaje aceptado por estado final *es el conjunto de cadenas:*

$$\mathcal{L}(\mathcal{A}) = \left\{ w \in \Sigma^* \mid (q_0, \$_0, w) \overset{*}{\vdash} (p, \alpha, \epsilon),\ p \in \mathcal{F},\ \alpha \in \Gamma^* \right\}$$

Respectivamente, el lenguaje aceptado por pila vacía *es el conjunto de cadenas:*

$$\mathcal{L}(\overline{\mathcal{A}}) = \left\{ w \in \Sigma^* \mid (q_0, \$_0, w) \overset{*}{\vdash} (p, \epsilon, \epsilon),\ \forall p \in Q \right\}$$ □

Las definiciones dadas de lenguaje aceptado por el autómata son intercambiables, puesto que dado un AP que reconoce un lenguaje por estado final es inmediato construir otro AP que reconoce el mismo lenguaje por pila vacía, y viceversa.

Teorema 5.1
Dado un AP $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, \$_0, Q_F)$*, el lenguaje aceptado por estado final y el lenguaje aceptado por pila vacía son idénticos.*

Demostración:
La demostración formal de este resultado se puede seguir en [59] ó en [62]. □

En lo sucesivo, para unificar criterios y facilitar la lectura, utilizaremos las siguientes convenciones en la representación de los elementos de un AP:

- $a, b, c, \ldots \in \Sigma \cup \{\epsilon\}$ son símbolos terminales.
- $A, B, C, D, E, F, G, \ldots \in \Gamma$ son símbolos de pila.
- $\xi, \xi', \ldots \in \Gamma^*$ son pilas o cadenas arbitrarias de símbolos de pila.

Las configuraciones del AP representan los estados intermedios durante el proceso de análisis. A su vez, el conjunto de estados implementa un control finito sobre las transiciones que se puede aplicar en cada configuración. Sin embargo, este control finito es prescindible [84] ya que el estado puede almacenarse como un elemento más de la pila, o, simplificando más, el estado está implícito en la secuencia de símbolos almacenados en la pila. De esta forma, si eliminamos la obtenemos una representación más simple y homogénea de las configuraciones del AP.

DEFINICIÓN 5.4 (autómata de pila)
Un autómata de pila *es una tupla* $\mathcal{A} = (\Sigma, \Gamma, \Theta, \$_0, \$_f)$*, donde:*

- Σ *es un conjunto finito de* símbolos terminales.
- Γ *es un conjunto finito de* símbolos de pila.
- $\$_0$ *es el* símbolo inicial de la pila.
- $\$_f$ *es el* símbolo final de la pila.
- Θ *es un conjunto de* transiciones

$$\begin{array}{rccc} \delta: & \Gamma^* \times (\Sigma \cup \{\varepsilon\}) & \longrightarrow & \Gamma^* \\ & (\alpha, a) & \rightsquigarrow & (\beta) \end{array}$$

que se clasifican en tres tipos: HORIZONTAL, APILAR, *y* ELIMINAR*:*

HORIZONTAL*:* $C \overset{a}{\longmapsto} F$*. Reemplazamos el elemento* C *de la cima de la pila por* F*, y leemos "*a*" de la cadena de entrada.*

APILAR*:* $C \overset{a}{\longmapsto} CF$*. Apilamos un nuevo elemento* F *en la cima de la pila y leemos "*a*" de la cadena de entrada.*

ELIMINAR*:* $CF \overset{a}{\longmapsto} G$*. Eliminamos los elementos* C *y* F *de la cima de la pila, sustituyéndolos por* G*, y leemos "*a*" de la cadena de entrada.* □

Intuitivamente, las transiciones tienen el siguiente significado: cuando la cima de la pila contiene la secuencia de símbolos α, y el siguiente símbolo de la cadena de entrada es "a", reemplazamos α por β, reconociendo "a". Al igual que en la definición anterior de AP, los estados intermedios del proceso de análisis vendrá determinados por las configuraciones del autómata.

DEFINICIÓN 5.5 (configuración)
Dados un AP $\mathcal{A} = (\Sigma, \Gamma, \Theta, \$_0, \$_f)$ *y una cadena de entrada* w, *definiremos una* configuración *del AP como un par* (ξ, x), *donde* ξ *es la pila y* x *la parte de la cadena de entrada que resta por analizar. La configuración inicial viene dada por* $(\$_0, w)$, *mientras que la configuración final es* $(\$_0\$_f, \varepsilon)$.

Diremos que una configuración (ξ, ax) deriva en un único paso *otra configuración* (ξ', x), *si y sólo si existe una transición que aplicada a* ξ *produce* ξ' *y lee "*a*" de la cadena de entrada:*

$$\begin{array}{ll} (\xi C, ax) \vdash (\xi F, x) & \textit{Transición} \text{ HORIZONTAL} \\ (\xi C, ax) \vdash (\xi CF, x) & \textit{Transición} \text{ APILAR} \\ (\xi CF, ax) \vdash (\xi G, x) & \textit{Transición} \text{ ELIMINAR} \end{array}$$ □

Si $(\xi, aw) \overset{*}{\vdash} (\xi', w)$ *decimos que* (ξ, aw) *deriva* (ξ', w) *en* n *pasos.*

A diferencia de los AP con representación explícita de los estados, no existe reconocimiento por estado final. La configuración final será siempre por pila vacía.

DEFINICIÓN 5.6 (lenguaje aceptado por un autómata)
Dado un AP $\mathcal{A} = (\Sigma, \Gamma, \Theta, \$_0, \$_f)$, *el* lenguaje aceptado *por el autómata es el conjunto de cadenas* $w \in \Sigma^*$:

$$\mathcal{L}(\mathcal{A}) = \left\{ w \mid (\$_0, w) \overset{*}{\vdash} (\$_0\$_f, \epsilon) \right\}.$$ □

A continuación se ilustra la aplicación de los APs para la construcción de analizadores sintácticos mediante un ejemplo. Posteriormente se desarrollarán los métodos necesarios para construir el autómata a partir de una GIC.

EJEMPLO 5.1
En la figura 5.1 se presenta el conjunto de transiciones correspondientes a un AP que acepta el lenguaje $\{a^n b^n \mid n \geq 0\}$.

Con este conjunto de transiciones podemos construir la derivación de la figura 5.2 que acepta la cadena de entrada "$aabb$*".* ■

5.1.1 Esquemas de compilación

A continuación debemos resolver la cuestión planteada en el apartado anterior: cómo obtener el autómata que acepta un lenguaje determinado a partir de la GIC que genera dicho lenguaje. La solución propuesta [84] es usar un *esquema de compilación*. Dicho esquema no es más que un conjunto de reglas que nos dicen como crear el conjunto

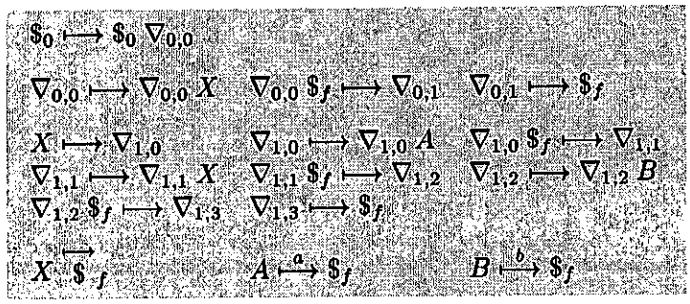


Figura 5.1: Transiciones de un autómata que acepta $a^n b^n, n \geq 0$.

de transiciones del autómata a partir de una GIC siguiendo una estrategia de análisis concreta. Esta estrategia determina que cálculos puede realizar el autómata y de que manera se emplean para encontrar la solución. Los esquemas de compilación no son únicos, ni siquiera para una misma estrategia de análisis.

EJEMPLO 5.2
A continuación se muestra un conjunto de reglas pertenecientes a un esquema de compilación. Por cada uno de los símbolos de las producciones de una GIC $\mathcal{G} = (N, \Sigma, P, S)$ que cumpla las condiciones de la parte derecha de las reglas E1, E2, E3 y E4; hemos de generar las transiciones del autómata indicadas.

$$
\begin{array}{lll}
(E1) & A'_{k,0} \longmapsto \nabla_{k,0} & \forall r_k \\
(E2) & \nabla_{k,i} \longmapsto \nabla_{k,i}\, A'_{k,i+1} & \forall r_k, i \; t.q. \; 0 \leq i < n_k, \; A_{k,i+1} \in N \\
(E3) & \nabla_{k,i}\, A''_{k,i+1} \longmapsto \nabla_{k,i+1} & \forall r_k, i \; t.q. \; 0 \leq i < n_k, \; A_{k,i+1} \in N \\
(E4) & \nabla_{k,i} \overset{a}{\longmapsto} \nabla_{k,i+1} & A_{k,i+1} = a \in \Sigma \\
(E5) & \nabla_{k,n_k} \longmapsto A''_{k,0} & \forall r_k
\end{array}
$$

Donde r_k son las producciones de la gramática, y n_k la longitud de su parte derecha. Por otro lado, $A_{k,i}$ es un símbolo de la producción r_k, teniendo en cuenta que para $i = 0$ es el símbolo de la parte izquierda, y en otro caso el símbolo en la posición i de la parte derecha. Los símbolos $\nabla_{r,i}$ son símbolos de pila, que en esta ocasión representan la realización del análisis de la producción r_k hasta el símbolo en la posición i. Las reglas anteriores aplicadas a la siguiente gramática:

$$
\begin{array}{ll}
r_1: & X \rightarrow A\, B \\
r_2: & A \rightarrow a \\
r_3: & B \rightarrow b
\end{array}
$$

producen el siguiente conjunto de transiciones del autómata:

$$
\begin{array}{ll}
X' \longmapsto \nabla_{1,0} & \nabla_{1,2} \longmapsto X'' \\
A' \longmapsto \nabla_{2,0} & \nabla_{2,1} \longmapsto A'' \\
B' \longmapsto \nabla_{3,0} & \nabla_{3,1} \longmapsto B''
\end{array}
$$

Transición	Configuración	
	$\$_0$	$aabb$
(a)	$\$_0\ \nabla_{0,0}$	$aabb$
(b)	$\$_0\ \nabla_{0,0}\ X$	$aabb$
(e)	$\$_0\ \nabla_{0,0}\ \nabla_{1,0}$	$aabb$
(f)	$\$_0\ \nabla_{0,0}\ \nabla_{1,0}\ A$	$aabb$
(n)	$\$_0\ \nabla_{0,0}\ \nabla_{1,0}\ \$_f$	abb
(g)	$\$_0\ \nabla_{0,0}\ \nabla_{1,1}$	abb
(h)	$\$_0\ \nabla_{0,0}\ \nabla_{1,1}\ X$	abb
(e)	$\$_0\ \nabla_{0,0}\ \nabla_{1,1}\ \nabla_{1,0}$	abb
(f)	$\$_0\ \nabla_{0,0}\ \nabla_{1,1}\ \nabla_{1,0}\ A$	abb
(n)	$\$_0\ \nabla_{0,0}\ \nabla_{1,1}\ \nabla_{1,0}\ \$_f$	bb
(g)	$\$_0\ \nabla_{0,0}\ \nabla_{1,1}\ \nabla_{1,1}$	bb
(h)	$\$_0\ \nabla_{0,0}\ \nabla_{1,1}\ \nabla_{1,1}\ X$	bb
(m)	$\$_0\ \nabla_{0,0}\ \nabla_{1,1}\ \nabla_{1,1}\ \$_f$	bb
(i)	$\$_0\ \nabla_{0,0}\ \nabla_{1,1}\ \nabla_{1,2}$	bb
(j)	$\$_0\ \nabla_{0,0}\ \nabla_{1,1}\ \nabla_{1,2}\ B$	bb
(o)	$\$_0\ \nabla_{0,0}\ \nabla_{1,1}\ \nabla_{1,2}\ \$_f$	b
(k)	$\$_0\ \nabla_{0,0}\ \nabla_{1,1}\ \nabla_{1,3}$	b
(l)	$\$_0\ \nabla_{0,0}\ \nabla_{1,1}\ \$_f$	b
(i)	$\$_0\ \nabla_{0,0}\ \nabla_{1,2}$	b
(j)	$\$_0\ \nabla_{0,0}\ \nabla_{1,2}\ B$	b
(o)	$\$_0\ \nabla_{0,0}\ \nabla_{1,2}\ \$_f$	
(k)	$\$_0\ \nabla_{0,0}\ \nabla_{1,3}$	
(l)	$\$_0\ \nabla_{0,0}\ \$_f$	
(c)	$\$_0\ \nabla_{0,1}$	
(d)	$\$_0\ \$_f$	

Figura 5.2: Secuencia de configuraciones en el análisis de $aabb$

$$\nabla_{1,0} \longmapsto \nabla_{1,0}\, A' \qquad \nabla_{1,0}\, A'' \longmapsto \nabla_{1,1} \qquad \nabla_{2,0} \overset{a}{\longmapsto} \nabla_{2,1}$$
$$\nabla_{1,1} \longmapsto \nabla_{1,1}\, B' \qquad \nabla_{1,1}\, A'' \longmapsto \nabla_{1,2} \qquad \nabla_{3,0} \overset{b}{\longmapsto} \nabla_{3,1} \quad \blacksquare$$

En el ejemplo anterior se observa la proximidad entre el formalismo de los SDGs y los esquemas de compilación que estamos describiendo. Desglosaremos esta similitud en varios puntos:

- Los símbolos de pila $\nabla_{k,i}$ realizan una función similar a los puntos '•' en los items SDG, señalando el análisis de la producción k hasta la posición i.

- Por su parte, las transiciones resultantes de la aplicación de las reglas E1 y E2 realizan los pasos deductivos de predicción, las reglas E3 y E5 se corresponden con el reconocimiento de una variable y la regla E4 el reconocimiento de un terminal de la gramática.

- Las configuraciones del autómata guardan de manera explícita la porción de la cadena de entrada que resta por analizar. De forma equivalente, los items SDG habitualmente incluyen una referencia a la posición de la cadena de entrada donde comienza dicha porción.

Siguiendo con esta línea, a continuación describiremos un esquema de compilación genérico a partir del cual es posible instanciar esquemas de compilación para las estrategias de análisis descendente, ascendente y Earley. Este esquema genérico no es más que una plantilla con dos parámetros relativos a la información usada en sus distintas fases. Estableciendo dichos parámetros obtendremos los esquemas de compilación concretos mencionados.

[INICIA]	comienza el análisis en la configuración inicial.
[LLAMADA]	requiere el análisis de un no terminal de una cláusula.
[RET]	continúa el análisis después de terminar una cláusula.
[SELEC]	selecciona una cláusula.
[PUBLICA]	determina que una cláusula ha sido completamente analizada.
[RECONOC]	reconoce un terminal de la cadena de entrada.

Tabla 5.1: Reglas de compilación en el paradigma llamada/retorno.

Para describir los esquemas de compilación emplearemos el paradigma de llamada/retorno [181]. Tal y como su nombre indica, se divide el análisis en dos fases: llamada y retorno. En la fase de llamada se inicia el reconocimiento de una producción de la gramática de forma predictiva. En la fase de retorno se finaliza el reconocimiento de una producción propagando el resultado del análisis. Ambas fases están gobernadas por reglas de compilación distribuidas en seis categorías: *inicialización*, *llamada*, *retorno*, *selección*, *publicación* y *reconocimiento*, que representaremos, respectivamente, como [INICIA], [LLAMADA], [RET], [SELEC], [PUBLICA] y [RECONOC]. A continuación establecemos el significado de cada una de ellas:

- Las reglas [LLAMADA] y [SELEC] realizan la *fase de llamada* que representa la etapa predictiva del análisis. Se encargan, respectivamente, de iniciar la fase de predicción y de seleccionar una producción de la gramática.
- Las reglas [RET] y [PUBLICA] propagan la información de análisis de forma ascendente. Ambas conforman la *fase de retorno*.
- La regla [RECONOC] avanza el proceso de análisis mediante el reconocimiento de un terminal de la cadena de entrada.

Por cada regla de tipo [LLAMADA], que inicia la fase de llamada para una variable de la gramática, es necesaria una regla [RET], que termina la fase de retorno, y viceversa. La tabla 5.1 recoge un resumen de las reglas descritas.

A continuación definimos un esquema de compilación genérico basado en el paradigma llamada/retorno. Para ello emplearemos la siguiente notación:

- $A_{r,s}$ es una variable que ocupa la posición s en la producción r, teniendo en cuenta que la posición 0 corresponde al símbolo de su parte izquierda y el resto al símbolo número s de la parte derecha.
- r_k es la producción número k, y n_k el número de símbolos de su parte derecha.
- $\nabla_{r,s}$ es un símbolo de pila. Tal y como hemos comentado anteriormente en el ejemplo, en esta ocasión lo usaremos para indicar que se ha realizado el análisis de la producción r hasta la posición s.

$$A_{r,0} \rightarrow \boxed{A_{r,1} \ldots A_{r,s}} A_{r,s+1} \ldots A_{r,n_r}$$

- $\overrightarrow{A_{r,s}}$ es un símbolo de pila que indica la predicción de la información relativa al análisis de $A_{r,s}$.
- $\overleftarrow{A_{r,s}}$ es un símbolo de pila que indica la propagación de la información relativa al análisis de $A_{r,s}$.

Además supondremos, sin perdida de generalidad, gramáticas aumentadas. De esta forma $A_{0,0}$ es el axioma de la gramática y no aparece en la parte izquierda de ninguna producción, excepto r_0. En la notación empleada, a la derecha de cada regla de compilación se indican las condiciones bajo las cuales se puede aplicar a una producción.

DEFINICIÓN 5.7 (esquema de compilación genérico)
Dada una GIC definiremos el esquema de compilación genérico, *parametrizable mediante la especificación de los símbolos $\overrightarrow{A_{r,s}}$ y $\overleftarrow{A_{r,s}}$, como el siguiente conjunto de reglas:*

[INICIA]	$\$_0 \longmapsto \$_0 \nabla_{0,0}$	
[LLAMADA]	$\nabla_{r,s} \longmapsto \nabla_{r,s} \overrightarrow{A_{r,s+1}}$	$\forall r, \forall s, 0 \leq s < n_r$
[SELEC]	$\overrightarrow{A_{r,0}} \longmapsto \nabla_{r,0}$	$\forall r, r \neq 0$
[PUBLICA]	$\nabla_{r,n_r} \longmapsto \overleftarrow{A_{r,0}}$	$\forall r$
[RET]	$\nabla_{r,s} \overleftarrow{A_{r,s+1}} \longmapsto \nabla_{r,s+1}$	$\forall r, \forall s, 0 \leq s < n_r$
[RECONOC]	$\nabla_{r,s} \xmapsto{a} \nabla_{r,s+1}$	$a = A_{r,s+1}$

Donde $\$_0$ y $\overleftarrow{A_{0,1}}$ son los símbolos de pila inicial y final, respectivamente. □

Si lo deseamos, para las producciones de la forma $A_{r,0} \longrightarrow a$, podemos sustituir las reglas [SELEC], [RECONOC] y [PUBLICA] por una única regla:

$$\text{[RECONOC-u]} \quad \overrightarrow{A_{r,0}} \xmapsto{a} \overleftarrow{A_{r,n_r}} \qquad \forall A_{r,0} \longrightarrow a,\ a \in \Sigma \cup \{\epsilon\}$$

Seguidamente, se ilustra el esquema de compilación genérico que acabamos de introducir mediante un ejemplo. Posteriormente instanciaremos este esquema especificando los parámetros definidos.

EJEMPLO 5.3

En el presente ejemplo emplearemos la misma gramática que hemos usado en el desarrollo de los SDGs en el capítulo 4, que viene dada por el siguiente conjunto de producciones:

$$P = \left\{ \begin{array}{l} \Phi \rightarrow S \\ S \rightarrow S + S \\ S \rightarrow S * S \\ S \rightarrow a \end{array} \right\}$$

La figura 5.3 muestra el conjunto de transiciones del autómata resultantes de la aplicación del esquema de compilación genérico. En dichas transiciones los parámetros del esquema, $\overrightarrow{A_{r,s}}$ y $\overleftarrow{A_{r,s}}$, se encuentran sin instanciar. En cualquier caso, el símbolo de fin de pila será $\overleftarrow{\Phi}$, una vez instanciado. ■

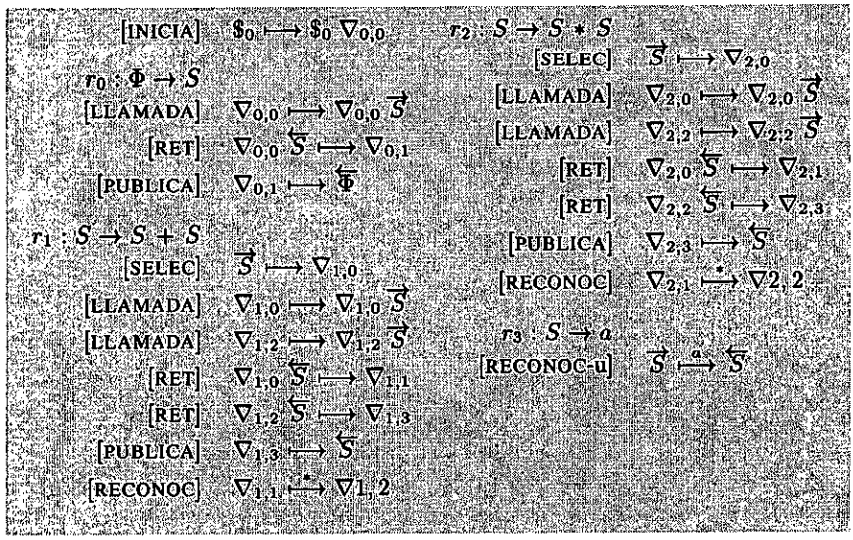


Figura 5.3: Transiciones del autómata con esquema de compilación genérico

Como ya se ha indicado en la definición, podemos adaptar el esquema genérico a diversas estrategias de análisis estableciendo los símbolos $\overrightarrow{A_{r,s}}$ y $\overleftarrow{A_{r,s}}$. Ambos símbolos controlan el inicio, respectivamente, de la fase de llamada y de la fase de retorno. En la tabla 5.2 se pueden ver los valores de estos parámetros, que describimos brevemente, para las estrategias descendente, ascendente y Earley:

Estrategia descendente. Tal y como veíamos en el capítulo 4 al presentar el SDG correspondiente, únicamente existe la fase predictiva. Por lo tanto en la fase de llamada predeciremos el análisis del símbolo correspondiente: $\overrightarrow{A_{r,s+1}} = A_{r,s+1}$, anulando la fase de retorno: $\overleftarrow{A_{r,s+1}} = \$_f$. Debido a que durante la fase de retorno no propagamos información de análisis ($\overleftarrow{A_{r,s+1}} = \$_f$), podemos simplificar el esquema uniendo las reglas [PUBLICA] y [RET] en una sola:

Estrategia	$\overrightarrow{A_{r,s+1}}$	$\overleftarrow{A_{r,s+1}}$
Descendente	$A_{r,s+1}$	$\$_f$
Ascendente	$\$_f$	$A_{r,s+1}$
Earley	$A'_{r,s+1}$	$A''_{r,s+1}$

Tabla 5.2: Parámetros del esquema de compilación genérico.

$$[\text{PUBLICA} + \text{RET}] \quad \nabla_{r',s}\, \nabla_{r,n_r} \longmapsto \nabla_{r',s+1}, \quad 0 \leq s < n_{r'}\ A_{r,0} = A_{r',s+1}$$

El esquema resultante se muestra en la figura 5.4.

[INICIA]	$\$_0 \longmapsto \$_0\, \nabla_{0,0}$	
[LLAMADA]	$\nabla_{r,s} \longmapsto \nabla_{r,s}\, A_{r,s+1}$	$\forall r, \forall s, 0 \leq s < n_r$
[SELEC]	$A_{r,0} \longmapsto \nabla_{r,0}$	$\forall r, r \neq 0$
[PUBLICA+RET]	$\nabla_{r',s}\, \nabla_{r,n_r} \longmapsto \nabla_{r',s+1}$	$\forall r, r', s, 0 \leq s < n_{r'}$
[RECONOC]	$\nabla_{r,s} \xrightarrow{a} \nabla_{r,s+1}$	$\forall a = A_{r,s+1}$
[RECONOC-u]	$A_{r,0} \xrightarrow{a} \$_f$	$A_{r,0} \longrightarrow a,\ a \in \Sigma \cup \{\epsilon\}$

Figura 5.4: Esquema de compilación con estrategia descendente

Estrategia ascendente. Este caso es opuesto al descendente. La fase de llamada queda anulada $\overrightarrow{A_{r,s+1}} = \$_f$, mientras que en la fase de retorno se propaga el símbolo recientemente analizado $\overleftarrow{A_{r,s+1}} = A_{r,s+1}$. Debido a que durante la fase de llamada no propagamos información de análisis ($\overrightarrow{A_{r,s+1}} = \$_f$), podemos simplificar el esquema uniendo las reglas [LLAMADA] y [SELEC] en una sola:

$$[\text{LLAMADA} + \text{SELEC}] \quad \nabla_{r,s} \longmapsto \nabla_{r,s}\nabla_{r',0}, \quad \forall r, r' \neq 0, s, 0 \leq s < n_r$$

El esquema resultante se muestra en la figura 5.5.

Estrategia Earley. Simplemente es la combinación de los dos anteriores. En este caso, representaremos la predicción y la compleción del análisis del símbolo $A_{r,s+1}$ mediante los símbolos $A'_{r,s+1}$ y $A''_{r,s+1}$, respectivamente. El esquema resultante se muestra en la figura 5.6.

5.1.2 Autómata con control finito

Atendiendo a una de las clasificaciones de los algoritmos de análisis dadas en el capítulo 3, los APs vistos hasta el momento de alejan de la clase de analizadores guiados

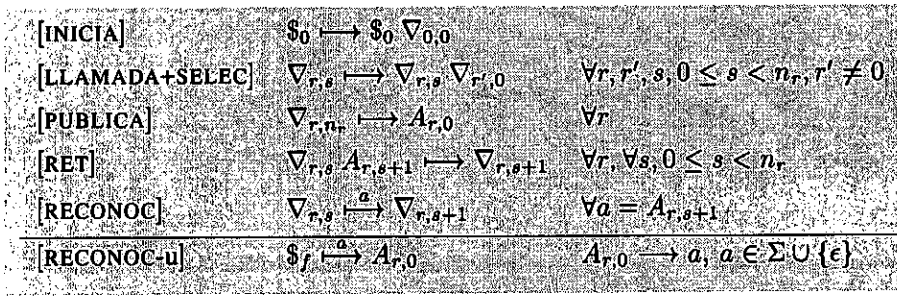


Figura 5.5: Esquema de compilación con estrategia ascendente

por la gramática. En la presente sección avanzaremos en este sentido, extendiendo la definición del autómata para incorporar de forma natural el control finito.

DEFINICIÓN 5.8 (autómata de pila con control finito)
Un autómata de pila con control finito *es un AP donde cada transición está ligada a un conjunto de restricciones, vacío o no, sobre elementos del autómata o de una GIC asociada.* □

Intuitivamente, las restricciones asociadas a una transición se han de verificar antes de la aplicación de la misma. De esta forma buscamos reducir el carácter no determinista del proceso de análisis sintáctico. Estas restricciones puede ser de cualquier tipo, desde funciones heurísticas hasta, como es el caso que se propone, máquinas de estado finito de tipo LR(k).

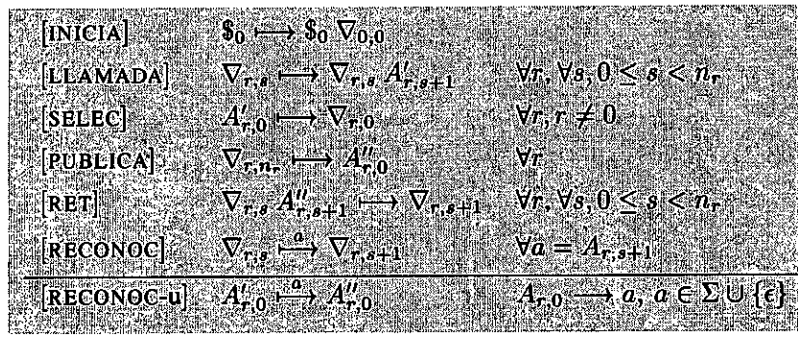


Figura 5.6: Esquema de compilación con estrategia Earley

EJEMPLO 5.4
Para ilustrar el concepto de restricciones sobre las transiciones del autómata, modificaremos el esquema de compilación con estrategia descendente. En este caso, el control finito será el encargado de evitar la selección de aquellas producciones que no conduzcan a una análisis válido. Para ello emplearemos el conjunto PRIMERO$_k(A)$ *que nos permite conocer todos aquellos símbolos terminales que pueden ocurrir en las k primeras posiciones de cualquier forma sentencial derivada a partir de A.*

La figura 5.7 muestra el nuevo esquema de compilación, donde "b" es el siguiente símbolo a analizar en la cadena de entrada. La restricción añadida a las transiciones de tipo [SELEC] *impiden que el análisis prosiga seleccionando producciones que no son capaces de derivar ninguna cadena que comience por el siguiente símbolo de la cadena de entrada.* ■

[INICIA]	$\$_0 \longmapsto \$_0\ \nabla_{0,0}$	
[LLAMADA]	$\nabla_{r,s} \longmapsto \nabla_{r,s}\ A_{r,s+1}$	$\forall r, \forall s, 0 \leq s < n_r$
[SELEC]	$A_{r,0} \longmapsto \nabla_{r,0}\ \{b \in \text{PRIMERO}_1(A_{r,1})\}$	$\forall r, r \neq 0$
[PUBLICA+RET]	$\nabla_{r',s}\ \nabla_{r,n_r} \longmapsto \nabla_{r',s+1}$	$\forall r, r', s, 0 \leq s < n_{r'}$
[RECONOC]	$\nabla_{r,s} \xrightarrow{a} \nabla_{r,s+1}$	$\forall a \doteq A_{r,s+1}$

Figura 5.7: Esquema de compilación con estrategia descendente y control finito

5.2 Tabulación

En el capítulo 3 se introducían los problemas de no determinismo, no-terminación y repetición de cálculos asociados al análisis sintáctico. La solución a estos problemas puede ser de diversos tipos, en concreto, ya se ha presentado el uso de mecanismos de control finito para reducir el grado de no determinismo. En la presente sección se propone el uso de técnicas de tabulación para evitar o reducir el impacto de los problemas señalados.

Las técnicas de tabulación tienen su origen en los trabajos de programación dinámica [23], cuyos principios han sido aplicados al análisis sintáctico [47, 69, 146] y a la programación lógica [142, 18, 81], bajo diversas formas y nombres: *memoization* [142, 119], *chart parsing* [70, 151], *magic sets* [18], ... Todos estos nombres hacen referencia a diferentes formas de implementar las ideas de la programación dinámica. En adelante emplearemos el término tabulación por considerarlo más extendido en el ámbito del análisis sintáctico.

Algunos de los algoritmos de análisis vistos hasta el momento, tienen un carácter inherentemente tabulado, como es el caso de los algoritmos CYK [69, 187] o Earley [47]. No obstante, en la presente sección nos centraremos en la aplicación de las técnicas de tabulación al caso general de los APs.

5.2.1 Aspectos generales de la tabulación

La idea base de la programación dinámica, y, por tanto de la tabulación, dado un problema que puede descomponerse en subproblemas más sencillos, consiste en almacenar por cada uno de estos subproblemas su solución. De esta forma buscamos que no sea necesario resolver dos veces la misma cuestión. El almacenamiento de soluciones,

desde el punto de vista lógico, se puede describir como una tabla[1] cuyos índices se corresponden con los subproblemas y los elementos con las soluciones. Como tal se han de especificar tres elementos:

Granularidad. Es necesario definir el tipo y tamaño de los subproblemas a considerar. Cuanto menores sean, mayor será la tabla resultante, por contra, mayores serán las posibilidades de reutilización de las soluciones almacenadas.

Indexación. Es necesario un mecanismo de indexación que nos permita acceder eficientemente al contenido de la tabla a partir de la descripción de un subproblema, para obtener su solución.

Comprobación de redundancias. Es necesario un mecanismo que evite la inclusión de objetos redundantes en la tabla. Este aspecto es especialmente importante cuando existe una relación de subsumción entre los subproblemas. En este caso no nos interesa incluir en la tabla soluciones a aquellos que son casos particulares de otro más general cuya solución ya se encuentra en la tabla.

Estos tres elementos se han de ajustar de manera que el tamaño de la tabla resultante sea manejable, al mismo tiempo que el coste computacional derivado de su manejo sea menor que el generado por la repetición de cálculos que se pretende evitar. En concreto, el coste del mecanismo de indexación a la hora de recuperar la solución a un subproblema dado tiene que ser menor que el coste de volver a calcular dicha solución.

5.2.2 Tabulación de autómatas de pila

Una forma de representar las soluciones intermedias del análisis es mediante las configuraciones del autómata. En el caso que nos ocupa, necesitaremos por tanto una representación que incluya únicamente resultados intermedios de manera que esta información pueda ser reutilizada.

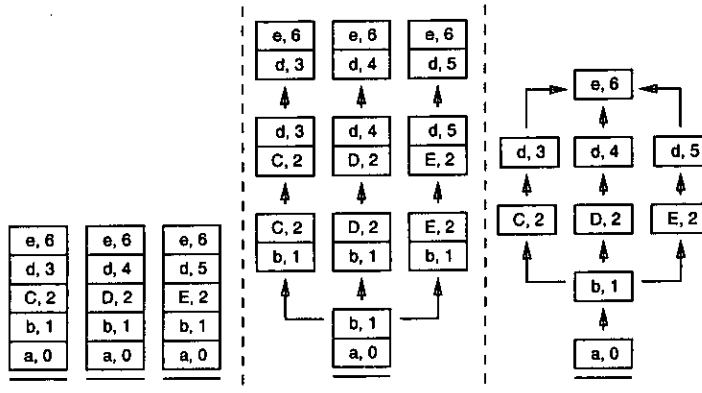


Figura 5.8: Distintas opciones de compartición de pilas

[1] De ahí el nombre de tabulación.

En adelante, nos centraremos en la pila como parte primordial de la configuración del autómata. Para ejemplificar las ideas expuestas recurriremos a la figura 5.8, donde la parte izquierda muestra tres pilas correspondientes a tres configuraciones de tres análisis distintos. Las partes central y derecha muestras esas mismas configuraciones, pero esta vez compartiendo elementos intermedios según la idea descrita. En este ejemplo también se ilustra como, cuanto más pequeños son los elementos intermedios, más posibilidades de compartición existen.

La idea en que se basa el ejemplo expuesto es diferenciar las configuraciones únicamente en función de la parte superior de la pila, permitiendo la compartición del resto. Esta idea se adapta de forma natural a la definición de AP que venimos manejando, puesto que las transiciones operan únicamente sobre los elementos de la parte superior de la pila. Para aplicar esta idea, definiremos una relación de equivalencia que nos permita agrupar todas las configuraciones cuyos n elementos en la cima de la pila coincidan. El representante canónico de cada clase de equivalencia es una imagen compacta de la configuración de la pila asociada. A continuación, será necesario definir una operación que transforme las transiciones del autómata de manera que sean capaces de operar directamente con las clases de equivalencia. Todo esto, se recoge en el concepto de *entorno dinámico* [178]. Mediante el uso de entornos dinámicos representaremos las configuraciones intermedias del autómata como items[2].

DEFINICIÓN 5.9 (entorno dinámico)
Dado un AP $\mathcal{A} = (\Sigma, \Gamma, \Theta, \$_0, \$_f)$, *un* entorno dinámico *es un par* $(\Re, Op)$, *donde:*

- $\Re$ *es una relación de equivalencia sobre las configuraciones del autómata,* Ξ. *Llamaremos* items *a las clases de equivalencia. Notaremos:*
  - *La clase de* Ξ *como* $\overline{\Xi}$.
  - *El conjunto de items como* $It_{A,\Re}$, *o simplemente* It, *cuando no hay ambigüedad.*
- *Op es un operador de la forma:*

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

Notaremos $Op(\tau)(It_0, \ldots, It_n) = It$, *como* $It_0, \ldots, It_n \overline{\vdash\!\!-} It$. □

Intuitivamente, la relación de equivalencia nos permite calcular la representación compacta o ítem para cada configuración del autómata, y $\overline{\vdash\!\!-}$ corresponde a la aplicación de una transición en el entorno dinámico. Los items serán los elementos susceptibles de ser tabulados. Por su parte, es necesario asegurar que los resultados obtenidos realizando una análisis directamente en el entorno dinámico son los mismos que obtendríamos con el autómata original. Para ello se definen las propiedades de *compatibilidad*, *completud* y *corrección* del entorno dinámico.

[2]No confundir con items Earley, o items LR(k), ver capítulo 3.

DEFINICIÓN 5.10 (compatibilidad,completud,corrección)
Dados un AP $\mathcal{A} = (\Sigma, \Gamma, \Theta, \$_0, \$_f)$ *y un entorno dinámico es un par* $(\Re, Op)$*, definimos las siguientes propiedades del entorno dinámico:*

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

$$\forall \tau \in \Theta,\ \Xi \in \mathrm{Dom}(\tau), \begin{cases} \exists I_{1\ldots n},\ t.q.\ (\overline{\Xi}, I_1, \ldots, I_n) \in \mathrm{Dom}(Op(\tau)) \\ Op(\tau)(\overline{\Xi}, I_1, \ldots, I_n) = \tau(\Xi) \end{cases}$$

  donde $\mathrm{Dom}(\tau)$ *representa el dominio para* $\tau \in \Theta$*.*

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

$$\forall \Xi,\ t.q\ \vdash^{*} \Xi \quad \Rightarrow \quad \bar{\vdash}^{*} \overline{\Xi}$$

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

$$\forall I,\ t.q.\ \bar{\vdash}^{*} I \quad \Rightarrow \quad \exists \Xi,\ t.q.\ \vdash^{*} \Xi \ y\ \overline{\Xi} = I \qquad \square$$

El entorno dinámico más sencillo es el que no realiza compactación y cada configuración del autómata está representada por todos sus elementos. Nos referiremos a este entorno como *entorno dinámico estándar* o S^T:

- $\Re$: Para cualquier configuración Ξ, $\overline{\Xi} = \Xi$.
- El operador Op es la identidad.

Intuitivamente, en S^T cada ítem equivale a una configuración del autómata, por lo tanto las propiedades de compatibilidad, completud y corrección son inmediatas. Como contrapartida, al no haber compactación, la compartición y reutilización son prácticamente nulas.

En lo que respecta al resto de entornos dinámicos, y dado que las transiciones de un AP operan, a lo sumo, con los dos elementos en la cima de la pila, desde un punto de vista práctico, únicamente resultan de interés los entornos que denominaremos S^2 y S^1.

Entorno dinámico S^2: Desde un punto de vista histórico fue el primero en ser definido [81]. Para la representación de los items se emplean los dos elementos de la cima de la pila:

$$\langle A', A \rangle = \{\Xi A' A\}$$

Y el operador Op se define como:

- $Op(C \longmapsto F)(\langle A', A\rangle) = \langle A', F\rangle,\ C = A$
- $Op(C \longmapsto CF)(\langle A', A\rangle) = \langle A, F\rangle,\ C = A$
- $Op(CF \longmapsto G)(\langle A', A\rangle, \langle A, E\rangle) = \langle A', G\rangle,\ C = A, F = E$

La principal ventaja del entorno S^2 es la sencillez a la hora de definir el entorno dinámico, y a la hora de demostrar sus propiedades de compatibilidad, completud y corrección [81]. En efecto, siempre podemos suponer una representación canónica del AP en la que las acciones dependen, a lo sumo, de los dos últimos elementos de la pila [62]. En este punto, la verificación de las propiedades requeridas es inmediata. Como desventaja, el grado de compactación, y por lo tanto su capacidad de compartición, es menor que en S^1; que pasamos a describir a continuación.

Entorno dinámico S^1: Este entorno fue propuesto por De la Clergerie [178] para el caso de los programas de cláusulas definidas y, previamente, por Vilares [164] para el caso de las GICs. Los items se construyen a partir del elemento en la cima de la pila, con lo que se alcanza el mayor grado de compactación posible:

$$\langle A\rangle = \{\Xi A\}$$

En contra con lo que ocurría en S^2, la construcción del operador Op es más compleja. Concretamente exige la introducción del concepto de *transición dinámica* para tratar las transiciones de tipo ELIMINAR:

- $Op(C \longmapsto F)(\langle A\rangle) = \langle F\rangle,\ C = A$
- $Op(C \longmapsto CF)(\langle A\rangle) = \langle F\rangle,\ C = A$
- $Op(CF \longmapsto G)(\langle A\rangle) = (C \longmapsto G),\ F = A$

Las transiciones dinámicas son transiciones de tipo HORIZONTAL que se crean de forma dinámica. Su misión es completar el segundo paso de una transición de tipo ELIMINAR $CF \longmapsto G$ sobre el ítem $\langle A\rangle$. El primer paso, eliminar el símbolo F de la cima de la pila, se realiza de forma inmediata. Sin embargo, el segundo, eliminar C de la cima de la pila sustituyéndolo por G, no es posible ya que el ítem no contiene información sobre el resto de la pila. En este sentido se crea una nueva transición que se aplicará a todos los items en el contexto de $\langle A\rangle$. La figura 5.9 muestra esta idea. La transición ELIMINAR$(BA \longmapsto D)(\langle A\rangle)$ genera la transición dinámica $B \longmapsto D$ que se aplicará sobre los dos items $\langle B\rangle$, independientemente del orden en que se creen.

Una vez introducidas las transiciones dinámicas, la principal desventaja frente al entorno dinámico S^2 reside en que la corrección no está garantizada [178, 164]. En concreto, dados los esquemas de compilación vistos con anterioridad, el entorno S^1 es correcto para los esquemas ascendente y Earley, pero no para el esquema descendente [178, 164].

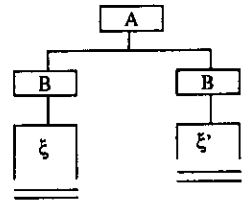


Figura 5.9: Aplicabilidad de las transiciones dinámicas

5.3 ICE

El sistema ICE, por *Incremental Context-free Environment*, es descrito por Vilares en [164]. ICE es un generador de analizadores sintácticos incrementales para GICs, sin restricciones. Ante la presencia de una ambigüedad en el proceso de análisis, procesa cada una de las alternativas de forma concurrente. Diremos que por cada alternativa se lanza una *rama de análisis* diferente. Para reducir el número de ramas de análisis necesario y mejorar la velocidad de tratamiento, se incorpora un control finito a la estrategia de evaluación empleada. Al mismo tiempo se busca la compartición de cálculos entre las distintas ramas aplicando técnicas de tabulación.

5.3.1 Estrategia de evaluación

La base de la estrategia de evaluación en ICE son los autómatas de tipo LR(k)[3], que ya han sido descritos en el capítulo 4 donde mostrábamos el SDG correspondiente. Una propuesta similar es la de Tomita [146], sobre la cual señalaremos su complejidad temporal. Efectivamente, podemos calcular dicha complejidad a partir del número de índices que es necesario examinar en los items antecedente de los pasos deductivos [8]. En el caso que nos ocupa el límite viene dado por el paso deductivo de reducción, donde es necesario examinar un hasta de $p+1$ índices, siendo p la longitud máxima de la parte derecha de las producciones. Por tanto, para una cadena de entrada de longitud n la complejidad temporal del algoritmo es $\mathcal{O}(n^{p+1})$. Se han propuesto varias mejoras sobre el algoritmo de Tomita original, por ejemplo las modificaciones de Rekers [109] solucionan las restricciones relacionadas con las derivaciones cíclicas y la recursión izquierda oculta, aunque manteniendo la complejidad temporal. Otros autores prefieren reducir los límites temporales transformando la gramática [126] o modificando la construcción del autómata LR [97], aunque en este último caso el tratamiento de derivaciones cíclicas se obvia debido a su complejidad.

En el caso de ICE, el objetivo es mantener la complejidad temporal del algoritmo en $\mathcal{O}(n^3)$ para el peor de los casos y $\mathcal{O}(n)$ para las gramáticas LR[4]. Para conseguir esta complejidad dividiremos cada paso deductivo de reducción, que implica m antecedentes, en $m+1$ pasos deductivos con un máximo de 2 antecedentes. De esta forma el número de índices a examinar queda limitado a un máximo de 3, obteniendo

[3]En concreto, las implementaciones actuales emplean autómatas LALR(1).

[4]Es decir, los casos en que el algoritmo se comporta de forma determinista para el tipo de autómata elegido como control finito.

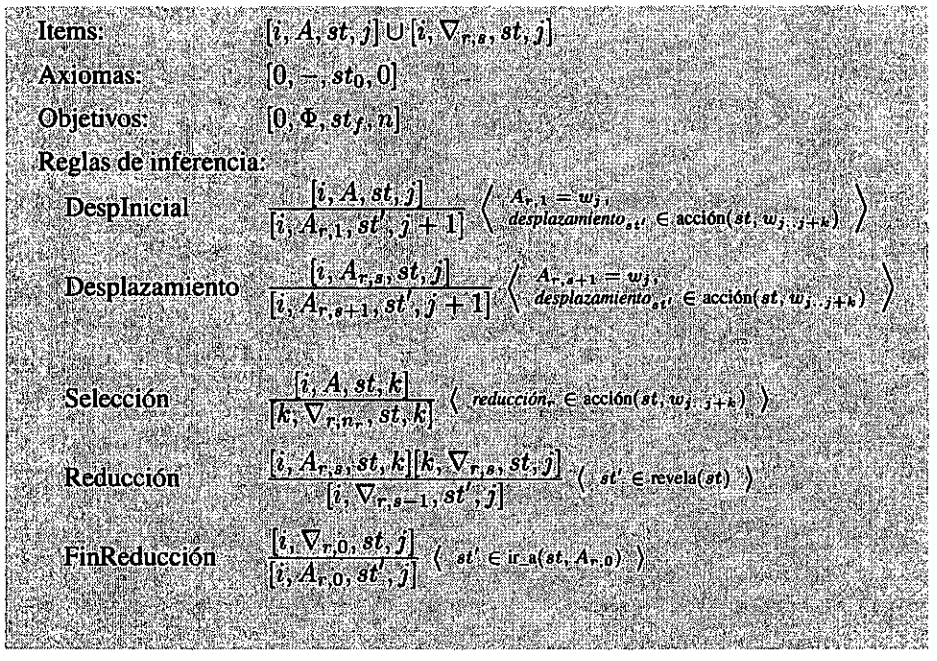


Figura 5.10: SDG del algoritmo LR(k) con complejidad $\mathcal{O}(n^3)$

la complejidad deseada. La división del paso de reducción la realizaremos mediante la binarización implícita de las producciones [83]. Para ello utilizaremos los símbolos $\nabla_{r,i}$ que hemos empleado con anterioridad en la descripción de los APs. Así, la reducción de una producción:

$$A_{r,0} \rightarrow A_{r,1}\, A_{r,2} \ldots A_{r,n_r}$$

se realiza de forma equivalente como la siguiente secuencia de reducciones:

$$\begin{array}{l} A_{r,0} \rightarrow \nabla_{0,0} \\ \nabla_{r,0} \rightarrow A_{r,1}\, \nabla_{r,1} \\ \nabla_{r,1} \rightarrow A_{r,2}\, \nabla_{r,2} \\ \vdots \\ \nabla_{r,n_r-1} \rightarrow A_{r,n_r}\, \nabla_{r,n_r} \\ \nabla_{r,n_r} \rightarrow \varepsilon \end{array}$$

Para aplicar este cambio al SDG original, debemos modificar la forma de los items, añadiendo un nuevo elemento que será, o bien un símbolo de la gramática, o bien un símbolo $\nabla_{r,i}$. Debido a la introducción de estos símbolos necesitamos tratar de forma especial el paso deductivo de desplazamiento cuando el símbolo afectado es el primero en la parte derecha de la producción. Este caso queda recogido en *DespInicial*. Por último, la reducción se divide en tres pasos diferentes: *Selección*, *Reducción* y *FinRe-*

ducción. A continuación se muestra su correspondencia con la binarización implícita de las producciones:

$$
\begin{array}{ll}
A_{r,0} \rightarrow \nabla_{0,0} & \text{Selección} \\
\nabla_{r,0} \rightarrow A_{r,1}\, \nabla_{r,1} & \text{Reducción} \\
\nabla_{r,1} \rightarrow A_{r,2}\, \nabla_{r,2} & \text{Reducción} \\
\vdots & \\
\nabla_{r,n_r-1} \rightarrow A_{r,n_r}\, \nabla_{r,n_r} & \text{Reducción} \\
\nabla_{r,n_r} \rightarrow \epsilon & \text{FinReducción}
\end{array}
$$

La figura 5.10 muestra el SDG resultante. A continuación se muestra mediante un ejemplo el comportamiento de los pasos de deducción con respecto a la pila de un algoritmo LR.

EJEMPLO 5.5

En el presente ejemplo consideraremos el análisis de la porción de la cadena de entrada dada por "abc", mediante la producción $B \rightarrow a\,b\,c$. *La figura 5.11 muestra la evolución de la pila y la relación de cada cambio de configuración con los pasos deductivos del SDG de la figura 5.10. A mayores, cada cambio de configuración está etiquetado por una flecha que indica su correspondencia con uno de los tres tipos de transiciones dadas en la definición del AP.* ■

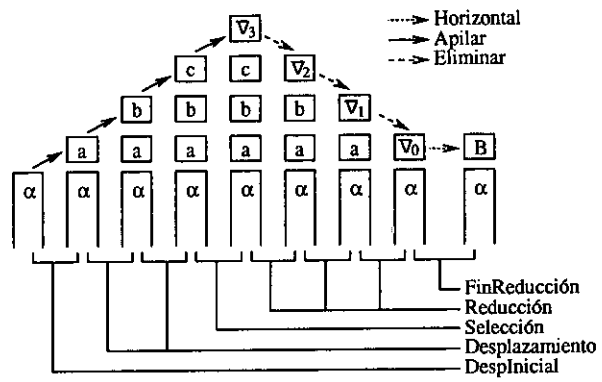


Figura 5.11: Representación gráfica del comportamiento del algoritmo LR

Para la implementación del SDG descrito en ICE se ha optado por el formalismo de APs con control finito. Informalmente, el ejemplo anterior esboza las reglas que componen el esquema de compilación correspondiente, básicamente una regla por cada paso deductivo. El conjunto de reglas resultante es similar al esquema de compilación con estrategia Earley de la figura 5.6. A continuación es necesario añadir a cada regla las restricciones dadas por el autómata LR[5]. Para posibilitar la resolución de dichas restricciones, se incluye el estado del control finito en los elementos de la pila. El esquema de compilación resultante se muestra en la figura 5.12, donde l son los símbolos adelantados en la cadena de entrada.

[5]En la implementación concreta de ICE, se ha optado por un autómata LALR(1).

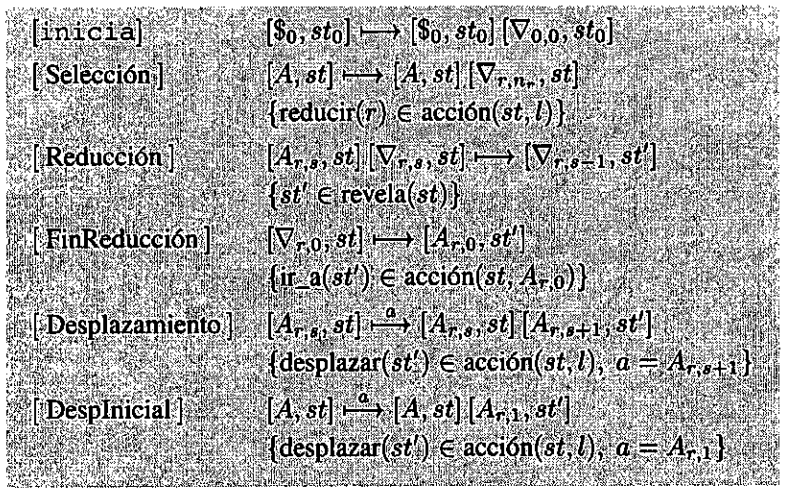


Figura 5.12: Esquema de compilación en ICE

EJEMPLO 5.6

A fin de mostrar el funcionamiento del control finito, emplearemos la gramática $\mathcal{G}_1$ dada por las siguientes producciones:

$$(0)\quad \Phi \to S \qquad (1)\quad S \to Aa \qquad (3)\quad A \to cd$$
$$(2)\quad S \to Bb \qquad (4)\quad B \to cd$$

El lenguaje generado por $\mathcal{G}_1$ es el conjunto $\{cda, cdb\}$. La figura 5.13 muestra el autómata LALR(1) que constituye el control finito para el análisis de la gramática. Sobre dicho control finito se han marcado las transiciones correspondientes al análisis del prefijo "cd" de la cadena de entrada "cda", en la figura 5.14.

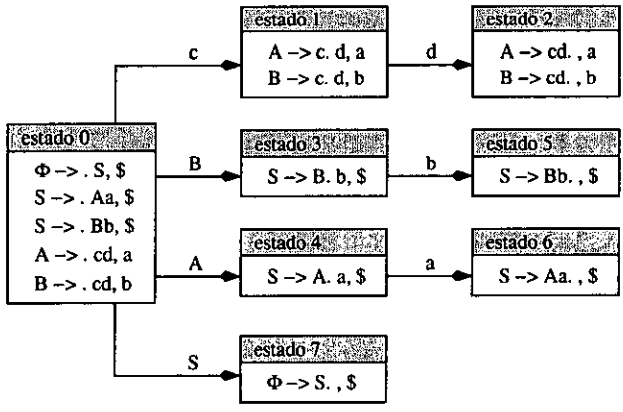


Figura 5.13: Autómata LALR(1) de la gramática $\mathcal{G}_1$

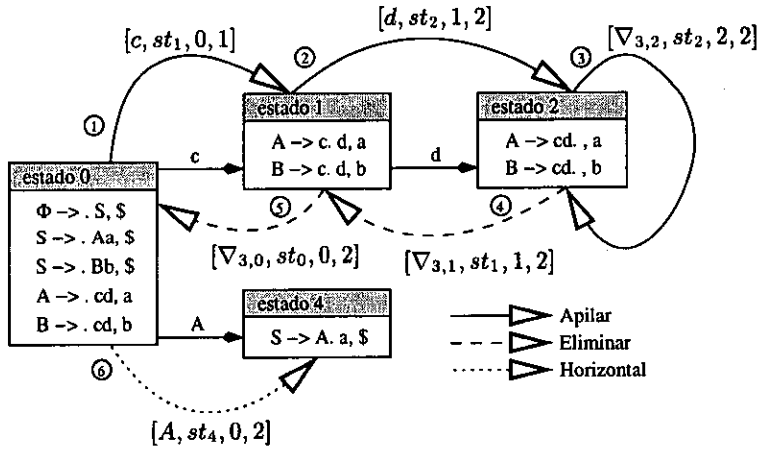


Figura 5.14: Transiciones del autómata ICE sobre el control finito LALR(1)

El análisis de la cadena "cda" comienza en el estado 0. En este estado intentaremos[6] *analizar usando las producciones 1 y 2, y posteriormente las producciones 3 y 4. Dado que el símbolo adelantado es "c", la primera acción es del tipo [DesplInicial], apilando el ítem* $[c, st_1, 0, 1]$ *y cambiando el control finito al estado 1. En este estado seguimos empleando las producciones 3 y 4, pero aún no sabemos cual de las dos será la correcta. La siguiente acción es [Desplazamiento], apilando el ítem* $[d, st_2, 1, 2]$*, cambiando al estado 2. En dicho estado sabemos que tanto la producción 3, como la 4 reconocen la subcadena "cd", pero el símbolo adelantado "a" indica que únicamente la 3 es la correcta. Sin la presencia del control finito, deberíamos continuar con la reducción de ambas producciones para más adelante encontrar que únicamente la número 3 es válida. La reducción de dicha producción completa el análisis del prefijo "cd" y se corresponde con las siguientes acciones:*

1. *[Selección]:* APILAR $[\nabla_{3,2}, st_2, 2, 2]$
2. *[Reducción]:* ELIMINAR $[\nabla_{3,2}, st_2, 2, 2]$ *y* $[d, st_2, 1, 2]$ *por* $[\nabla_{3,1}, st_1, 1, 2]$
3. *[Reducción]:* ELIMINAR $[\nabla_{3,1}, st_1, 1, 2]$ *y* $[c, st_1, 0, 1]$ *por* $[\nabla_{3,0}, st_0, 0, 2]$
4. *[FinReducción]:* HORIZONTAL $[\nabla_{3,0}, st_0, 0, 2]$ *por* $[A, st_4, 0, 2]$

Nótese que el proceso de análisis ha sido determinista en todo momento. En la siguiente tabla, se recogen los conjuntos de items generados para el análisis de la cadena de entrada "cda" completa:

Cjto. items 0	*Cjto. items 1*	*Cjto. items 2*	*Cjto. items 3*
$[-, st_0, 0, 0]$	$[c, st_1, 0, 1]$	$[d, st_2, 1, 2]$	$[a, st_5, 2, 3]$
		$[\nabla_{3,2}, st_2, 2, 2]$	$[\nabla_{1,2}, st_5, 3, 3]$
		$[\nabla_{3,1}, st_1, 1, 2]$	$[\nabla_{1,1}, st_4, 2, 3]$
		$[\nabla_{3,0}, st_0, 0, 2]$	$[\nabla_{1,0}, st_4, 0, 3]$
		$[A, st_4, 0, 2]$	$[S, st_7, 0, 3]$

[6] En el sentido del paso de predicción del algoritmo de Earley [47].

■

Una vez definidos el SDG del algoritmo y el esquema de compilación para el AP con control finito que lo implementa, pasaremos a aplicar sobre él las técnicas de tabulación vistas anteriormente.

5.3.2 Tabulación y sincronización

Para la tabulación del AP usado en ICE, se ha optado por el entorno dinámico S^1, por ser el que posibilita el mayor grado de compartición. Recordemos que en este entorno la clase de equivalencia de las configuraciones del autómata viene dada por el elemento en la cima de la pila:

$$\langle A \rangle = \{A\xi\}$$

Y el operador Op exige el empleo de transiciones dinámicas cuando se aplica a las transiciones de tipo ELIMINAR:

- $Op(C \longmapsto F)(\langle A \rangle) = \langle F \rangle,\ C = A$
- $Op(C \longmapsto CF)(\langle A \rangle) = \langle F \rangle,\ C = A$
- $Op(CF \longmapsto G)(\langle A \rangle) = (C \longmapsto G),\ C = A$

Sobre la transiciones dinámicas destacaremos dos aspectos. En el esquema de compilación de ICE, sólo se producen durante la reducción. Una vez creada una transición dinámica, en principio, por cada nuevo ítem que generemos *a posteriori*, debemos comprobar si la transición dinámica es aplicable.

Para asegurar la corrección y facilitar la implementación del marco dinámico S^1, estableceremos un mecanismo de *sincronización* del análisis. Esta sincronización se realiza al estilo del algoritmo de Earley [47]. El analizador construye, de forma secuencial, un conjunto de items por cada posición de la cadena de entrada. Cada uno de estos conjuntos se completa antes de pasar al siguiente. De forma análoga al algoritmo de Earley [47], añadiremos dos índices a cada ítem:

- Un índice establece la posición en la cadena de entrada que se corresponde con el conjunto de items al que pertenece.
- El otro índice, llamado *puntero hacia atrás*[7], indica el inicio de la porción de la cadena de entrada reconocida por el ítem.

Con los cambios descritos, los items pasan a tener la forma $[A, st, b, i]$, donde A y st siguen siendo, respectivamente, un símbolo y un estado del control finito, b es el puntero hacia atrás, e i el índice al conjunto de items al que pertenece. El esquema de compilación resultante se muestra en la figura 5.15.

La inclusión del mecanismo de sincronización nos permite optimizar el manejo de las transiciones dinámicas en dos sentidos [164]:

[7] *Back-pointer* en la terminología original

$$
\begin{array}{ll}
[\texttt{inicia}] & [\$_0, st_0, 0, 0] \longmapsto [\$_0, st_0, 0, 0]\,[\nabla_{0,0}, st_0, 0, 0] \\
[\text{Selección}] & [A, st, i, j] \longmapsto [A, st, i, j]\,[\nabla_{r,n_r}, st, j, j] \\
 & \{\text{reducir}(r) \in \text{acción}(st, w_{j+1..})\} \\
[\text{Reducción}] & [A_{r,s}, st, i, k]\,[\nabla_{r,s}, st, k, j] \longmapsto [\nabla_{r,s-1}, st', i, j] \\
 & \{st' \in \text{revela}(st)\} \\
[\text{FinReducción}] & [\nabla_{r,0}, st, i, j] \longmapsto [A_{r,0}, st', i, j] \\
 & \{\text{ir\_a}(st') \in \text{acción}(st, A_{r,0})\} \\
[\text{Desplazamiento}] & [A_{r,s}, st, i, j] \xrightarrow{a} [A_{r,s}, st, i, j]\,[A_{r,s+1}, st', i, j+1] \\
 & \{\text{desplazar}(st') \in \text{acción}(st, w_{j+1..}),\ a = A_{r,s+1}\} \\
[\text{DespInicial}] & [A, st, i, j] \xrightarrow{a} [A, st, i, j]\,[A_{r,1}, st', i, j+1] \\
 & \{\text{desplazar}(st') \in \text{acción}(st, w_{j+1..}),\ a = A_{r,1}\}
\end{array}
$$

Figura 5.15: Esquema de compilación en ICE, con sincronización

- Su ámbito de aplicabilidad se restringe al conjunto de items en el que fueron generadas. Gracias a esta característica, una vez completada la construcción de un conjunto de items, es posible ignorar todas las transiciones dinámicas generadas hasta el momento.
- La generación de transiciones dinámicas dentro de un conjunto de items sólo es necesaria bajo determinadas circunstancias: se ha realizado la reducción de una producción-ε y existe una ambigüedad en la aplicación de las transiciones[8]. Informalmente, estas dos condiciones son necesarias para que sea posible generar dos items que compartan el mismo contexto sintáctico, lo cual es, a su vez, condición necesaria para que sea posible aplicar una transición dinámica.

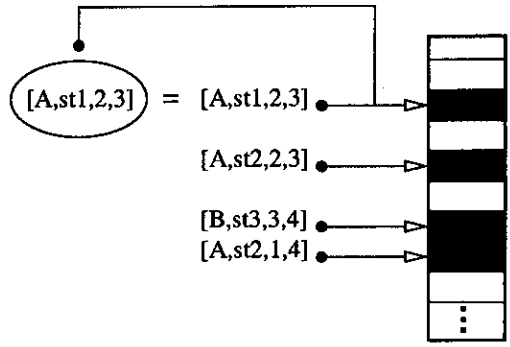


Figura 5.16: Indexación y comprobación de redundancias

[8] En la práctica esto implica la presencia de un no determinismo en el control finito LALR(1).

EJEMPLO 5.7

En el siguiente ejemplo mostraremos la aplicación de la tabulación a la compartición de cálculos en el tratamiento de derivaciones cíclicas. Al mismo tiempo se mostrará el funcionamiento del mecanismo de sincronización a la hora de reducir el número de transiciones dinámicas generadas. Emplearemos la gramática $\mathcal{G}_2$, que genera el lenguaje $\{a\}$. En esta gramática es posible la construcción de derivaciones cíclicas y existen infinitas derivaciones que generan la cadena "a", según el siguiente conjunto de producciones:

$$\begin{array}{llll} (0) & \Phi \to S & (2) & A \to B \\ (1) & S \to A\,a & (3) & B \to A \\ & & (4) & B \to \varepsilon \end{array}$$

Y la figura 5.17 muestra el control finito LALR(1) de la gramática y, sobre él, las transiciones efectuadas durante el análisis de la derivación cíclica que implica las producciones 2, 3 y 4.

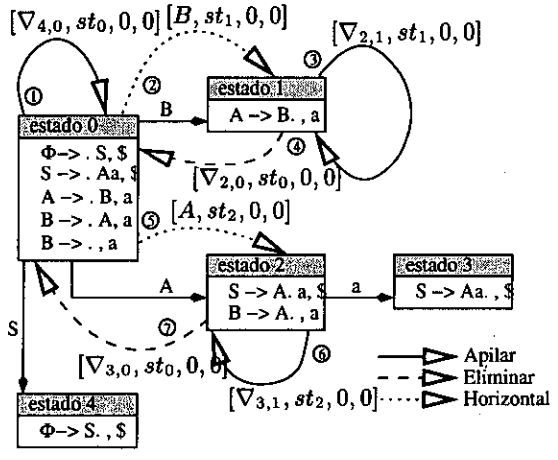


Figura 5.17: Transiciones del autómata ICE sobre el control finito LALR(1) de $\mathcal{G}_2$

El análisis de la cadena de entrada "a" comienza con el ítem inicial $[-, st_0, 0, 0]$, sobre el cual debemos realizar la reducción de la producción 4, que es una producción-ε:

1. *[Selección]:* APILAR $[\nabla_{4,0}, st_0, 0, 0]$
2. *[FinReducción]:* HORIZONTAL $[\nabla_{4,0}, st_0, 0, 0]$ *por* $[B, st_1, 0, 0]$

El control finito está en el estado 1, marcando como siguiente acción la reducción de la producción 2, que comprende la aplicación de las transiciones:

1. *[Selección]:* APILAR $[\nabla_{2,1}, st_1, 0, 0]$
2. *[Reducción]:* ELIMINAR $[\nabla_{2,1}, st_1, 0, 0]$ *y* $[B, st_1, 0, 0]$ *por* $[\nabla_{2,0}, st_0, 0, 0]$

3. [*FinReducción*]: HORIZONTAL $[\nabla_{2,0}, st_0, 0, 0]$ *por* $[A, st_2, 0, 0]$

A continuación, en el estado 2 del control finito, hay dos acciones posibles, reducir la producción 2 o desplazar el símbolo "a". Realizamos en primer lugar la reducción:

1. [*Selección*]: APILAR $[\nabla_{3,1}, st_2, 0, 0]$
2. [*Reducción*]: ELIMINAR $[\nabla_{3,1}, st_2, 0, 0]$ *y* $[A, st_2, 0, 0]$ *por* $[\nabla_{3,0}, st_0, 0, 0]$
3. [*FinReducción*]: HORIZONTAL $[\nabla_{3,0}, st_0, 0, 0]$ *por* $[B, st_1, 0, 0]$

El ítem $[B, st_1, 0, 0]$ *había sido generado anteriormente y, por tanto, no necesitamos volver a calcular los items que derivan de él. Mediante esta compartición de cálculos queda detectada y representada la aplicación cíclica de las producciones 2 y 3.*

Puesto que se han dado las dos condiciones de no determinismo y reducción de producciones-ε, a partir de aquí generaríamos transiciones dinámicas si aplicásemos más transiciones ELIMINAR *dentro del conjunto de items 0.*

Para continuar el análisis, aplicaremos una transición [*Desplazamiento*] *sobre el ítem* $[A, st_2, 0, 0]$, *apilando* $[a, st_3, 0, 1]$. *Este último ítem ya pertenece al conjunto de items 1, para el cual no se ha establecido la necesidad de generar transiciones dinámicas. La siguiente acción, para el estado 3, es la reducción de la producción 1:*

1. [*Selección*]: APILAR $[\nabla_{1,2}, st_3, 1, 1]$
2. [*Reducción*]: ELIMINAR $[\nabla_{1,2}, st_3, 1, 1]$ *y* $[a, st_3, 0, 1]$ *por* $[\nabla_{1,1}, st_2, 0, 1]$
3. [*Reducción*]: ELIMINAR $[\nabla_{1,1}, st_2, 0, 1]$ *y* $[A, st_2, 0, 0]$ *por* $[\nabla_{1,0}, st_0, 0, 1]$
4. [*FinReducción*]: HORIZONTAL $[\nabla_{1,0}, st_0, 0, 1]$ *por* $[S, st_5, 0, 1]$

La siguiente tabla, recoge los conjuntos de items generados para el análisis de la cadena de entrada "a":

Cjto. items 0	*Cjto. items 1*
$[-, st_0, 0, 0]$	$[a, st_3, 0, 1]$
$[\nabla_{4,0}, st_0, 0, 0]$	$[\nabla_{1,2}, st_3, 1, 1]$
$[B, st_1, 0, 0]$	$[\nabla_{1,1}, st_2, 0, 1]$
$[\nabla_{2,1}, st_1, 0, 0]$	$[\nabla_{1,0}, st_0, 0, 1]$
$[\nabla_{2,0}, st_0, 0, 0]$	$[S, st_5, 0, 1]$
$[A, st_2, 0, 0]$	
$[\nabla_{3,1}, st_2, 0, 0]$	
$[\nabla_{3,0}, st_0, 0, 0]$	

■

El ejemplo anterior muestra el funcionamiento del mecanismo de sincronización a la hora de reducir el número de transiciones dinámicas. En el siguiente ejemplo, se muestra el funcionamiento de dichas transiciones.

EJEMPLO 5.8

En este ejemplo emplearemos la gramática, $\mathcal{G}_3$, de Dyck de los paréntesis bien balanceados. Ésta nos sirve para ilustrar la compartición de cálculos en un caso más complejo que en ejemplo anterior, y, al mismo tiempo, el uso de transiciones dinámicas. El conjunto de producciones es el siguiente:

$$(0)\quad \Phi \to S \qquad (1)\quad S \to \epsilon \qquad (2)\quad S \to S\,S \qquad (3)\quad S \to (\,S\,)$$

El control finito LALR(1) viene dado por la siguiente tabla:

	'('	')'	$	S
0	desp(1)/red(1)		red(1)	ir(2)
1	desp(1)/red(1)	red(1)	red(1)	ir(3)
2	desp(1)/red(1)	red(1)	red(1)/acep	ir(4)
3	desp(1)/red(1)	desp(5)/red(1)	red(1)	ir(4)
4	desp(1)/red(1)/red(2)	red(1)/red(2)	red(1)/red(2)	ir(4)
5	red(3)	red(3)	red(3)	

La cadena de entrada será ϵ, la cadena vacía. Como es habitual el análisis comienza con el ítem $[-, st_0, 0, 0]$. A continuación la primera acción es la reducción de la producción 1, tras la cual existe un no determinismo dado por las acciones aceptar y reducir una vez más la producción 1. Una vez aceptada la cadena de entrada, continuamos con la acción de reducción para obtener todos los análisis posibles. Nótese que, llegados a este punto, ya se han dado las dos condiciones que establecen la generación de transiciones dinámicas dentro del conjunto de items actual.

1. [*Selección*]: APILAR $[\nabla_{1,0}, st_0, 0, 0]$
2. [*FinReducción*]: HORIZONTAL $[\nabla_{1,0}, st_0, 0, 0]$ *por* $[S, st_2, 0, 0]$
3. [*Selección*]: APILAR $[\nabla_{1,0}, st_2, 0, 0]$
4. [*FinReducción*]: HORIZONTAL $[\nabla_{1,0}, st_2, 0, 0]$ *por* $[S, st_4, 0, 0]$

En la siguiente acción existe un nuevo no determinismo, reducir la producción 1 o reducir la producción 2. Comenzaremos por la segunda de las acciones:

1. [*Selección*]: APILAR $[\nabla_{2,2}, st_4, 0, 0]$
2. [*Reducción*]: ELIMINAR $[\nabla_{2,2}, st_4, 0, 0]$ *y* $[S, st_4, 0, 0]$ *por* $[\nabla_{2,1}, st_2, 0, 0]$ *que es el resultado de aplicar la primera de las transiciones dinámicas recién generadas*[9]*:*

$$d_1 = [S, st_4, 0, 0] \longmapsto [\nabla_{2,1}, st_2, 0, 0]$$
$$d_2 = [S, st_4, 0, 0] \longmapsto [\nabla_{2,1}, st_4, 0, 0]$$

[9]Nótese que $\{st_2, st_4\} \subseteq \text{revela}(st_4)$.

3. *[Reducción]:* ELIMINAR $[\nabla_{2,1}, st_2, 0, 0]$ *y* $[S, st_2, 0, 0]$ *por* $[\nabla_{2,0}, st_0, 0, 0]$ *que es el resultado de aplicar la transición dinámica recién generada:*

$$d_3 = [S, st_2, 0, 0] \longmapsto [\nabla_{2,0}, st_0, 0, 0]$$

4. *[FinReducción]:* HORIZONTAL $[\nabla_{2,0}, st_0, 0, 0]$ *por* $[S, st_2, 0, 0]$

El último ítem $[S, st_2, 0, 0]$*, ya había sido generado anteriormente, por lo cual no es necesario continuar los cálculos sobre él. Continuamos con la reducción de la producción 1 sobre el ítem* $[S, st_4, 0, 0]$*:*

1. *[Selección]:* APILAR $[\nabla_{1,0}, st_4, 0, 0]$

2. *[FinReducción]:* HORIZONTAL $[\nabla_{1,0}, st_4, 0, 0]$ *por* $[S, st_4, 0, 0]$

El último ítem $[S, st_4, 0, 0]$*, ya había sido generado anteriormente, lo cual lo sitúa en el contexto sintáctico en el que se debe aplicar las transiciones dinámicas* d_1 *y* d_2*, que dan lugar a los items* $[\nabla_{2,1}, st_2, 0, 0]$ *y* $[\nabla_{2,1}, st_4, 0, 0]$*. El primero ya existe en la tabla de items por lo cual continuamos el análisis únicamente sobre el segundo:*

1. *[Reducción]:* ELIMINAR $[\nabla_{2,1}, st_4, 0, 0]$ *y* $[S, st_4, 0, 0]$ *por* $[\nabla_{2,0}, st_2, 0, 0]$

2. *[FinReducción]:* HORIZONTAL $[\nabla_{2,0}, st_2, 0, 0]$ *por* $[S, st_4, 0, 0]$

De nuevo, el último ítem $[S, st_4, 0, 0]$*, ya había sido generado anteriormente, por lo que no es necesario continuar.* ■

En lo que respecta a la comprobación de redundancias dentro de la tabla de items, es suficiente establecer una condición de igualdad entre items. En efecto, dado que estamos tratando GICs sólo consideraremos relaciones de subsumción triviales. El mecanismo de comprobación de redundancias implica que, como paso previo a la introducción de un ítem en la tabla, se compruebe que no existe otro ítem igual[10], en el peor de los casos esta comprobación significa recorrer la tabla entera. Para mejorar la eficiencia de este mecanismo, haremos que se apoye en la función de indexación de la tabla, como se muestra en la figura 5.16. Así para un ítem $[A, st, b, i]$ la función será de la forma $h(A, st, b, i)$. Como aspecto negativo, señalaremos que incluir el estado en la comprobación de redundancias puede evitar la compartición en determinados casos en los que el control finito realiza distinciones entre estados cuyo contexto sintáctico es el mismo.

Para completar el analizador ICE, y poder observar de forma más intuitiva las comparticiones de cálculos, debemos adaptar el formalismo descrito de manera que construya la estructura sintáctica correspondiente al análisis de la cadena de entrada. Dicha estructura será un bosque de análisis, donde se reflejarán las ambigüedades y derivaciones cíclicas, así como la compartición de cálculos.

[10]Es decir, mismo símbolo, mismo estado, mismo puntero hacia atrás y mismo índice.

5.3.3 Bosque de análisis

Para obtener el bosque sintáctico como resultado del análisis, sustituiremos los APs por *traductores de pila* (TPs) como formalismo operacional.

DEFINICIÓN 5.11 (traductor de pila)
Un traductor de pila *es una tupla* $\mathcal{T} = (\Sigma, \Gamma, \Pi, \Theta, \$_0, \$_f)$, *donde* Σ, Γ, $\$_0$, *y* $\$_f$ *tienen el mismo significado que en un AP, y:*

- Π *es un conjunto de* símbolos de salida
- Θ *es un conjunto de* transiciones

$$\begin{array}{rccc} \delta: & \Gamma^* \times (\Sigma \cup \{\varepsilon\}) & \longrightarrow & \Gamma^* \times \Pi^* \\ & (\alpha, a) & \rightsquigarrow & (\beta, u) \end{array}$$

□

Informalmente, respecto a un AP, un TP especifica la salida generada tras la aplicación de una transición. En consecuencia, una *configuración del traductor*, es similar a la configuración de un autómata a la que añadimos un nuevo elemento donde acumulamos la salida generada por cada transición: (ξ, x, u).

$$\begin{array}{ll} Op(C \longmapsto F)(\langle A \rangle) = (F), & \langle F \rangle \to \langle A \rangle \\ Op(C \longmapsto CF)(\langle A \rangle) = (F), & \varepsilon \\ Op(CF \longmapsto G)(\langle A \rangle) = (C \longmapsto G), & G \to C\,\langle A \rangle \end{array}$$

Figura 5.18: Salida para las transiciones en S^1

Recordemos que para la representación de los bosques de análisis emplearemos grafos Y/O. En ese sentido, seguimos la propuesta de construir dicho grafo a partir de una GIC [84, 25] a la que denominaremos *gramática de salida*. Los nodos O serán las variables de la gramática y los nodos Y, las producciones.

EJEMPLO 5.9
La figura 5.19 muestra el grafo Y/O construido a partir de la siguiente gramática:

$$\begin{array}{llll} (1) & A \to BC & (5) & B \to BD \\ (2) & A \to AD & (6) & D \to dB \\ (3) & B \to b & (7) & C \to cB \\ (4) & B \to db & & \end{array}$$

■

En concreto la salida producida por cada transición del traductor de pila será un nodo Y del bosque de análisis. Estos nodos serán producciones de la gramática de salida, cuyos símbolos son referencias a los items generados en el entorno dinámico S^1, tal y como muestra la figura 5.18. Cuando se produce la compartición de cálculos entre items provenientes de distintas ramas de análisis, la salida de dichos items se compacta en un nodo O. Las ventajas de esta aproximación son la siguientes:

- El empleo de un grafo Y/O nos permite representar mediante una estructura finita, un conjunto infinito de árboles de análisis.

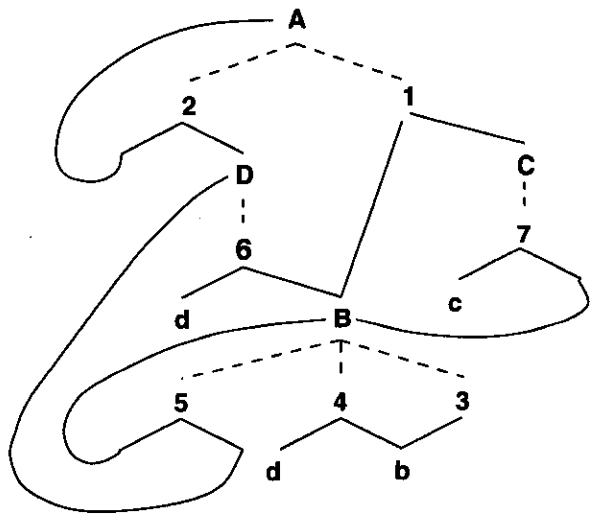


Figura 5.19: Representación de la gramática como grafo Y/O

- Las estructuras de datos sintácticas son las mismas usadas en el proceso de análisis.
  - Esta correspondencia nos permite establecer una relación directa entre la compartición de computaciones y la compartición del estructuras sintácticas. En efecto, el empleo de un grafo Y/O permite representar la compartición de dichas estructuras como nodos O, al mismo tiempo que los nodos Y son referencias a los items del analizador dados por las producciones de la gramática de salida.

[Inicial]	$it_0 \longmapsto it_0\, it_1,$	ε
[Selección]	$it_0 \longmapsto it_0\, it_1,$	ε
[Reducción]	$it_0\, it_1 \longmapsto it_2,$	$it_2 \rightarrow it_0 it_1$
[FinReducción]	$it_0 \longmapsto it_1,$	$it_1 \rightarrow it_0$
[Desplazamiento]	$it_0 \overset{a}{\longmapsto} it_0\, it_1,$	it_1
[DespInicial]	$it_0 \overset{a}{\longmapsto} it_0\, it_1,$	it_1

Figura 5.20: Gramática de salida para el esquema de compilación en ICE

  - Dado que las transiciones del TP dependen del primer o de los dos primeros elementos de la cima de la pila, las producciones de la gramática de salida poseen como máximo dos elementos en su parte derecha. Como consecuencia es posible compartir la cola de la lista de hijos de un subárbol en lugar de la lista completa.

EJEMPLO 5.10

Para este ejemplo emplearemos la gramática $\mathcal{G}_3$ del apartado anterior. El análisis de la cadena de entrada vacía producía el conjunto de items de la tabla de la parte superior de la figura 5.21, donde hemos añadido a cada uno la salida producida según el esquema de la figura 5.20. El bosque de análisis resultante se muestra en la parte inferior izquierda de la figura 5.21, mientras que a su derecha se muestra un representación convencional del bosque de análisis una vez eliminados los items que contienen símbolos $\nabla_{i,j}$. ■

Cjto. items 0	
it_0	$[-, st_0, 0, 0, \varepsilon]$
it_1	$[\nabla_{1,0}, st_0, 0, 0, \varepsilon]$
it_2	$[S, st_2, 0, 0, (it_2 \rightarrow it_1)\ (it_2 \rightarrow it_7)]$
it_3	$[\nabla_{1,0}, st_2, 0, 0, \varepsilon]$
it_4	$[S, st_4, 0, 0, (it_4 \rightarrow it_3)\ (it_4 \rightarrow it_{10})]$
it_5	$[\nabla_{2,2}, st_4, 0, 0, \varepsilon]$
it_6	$[\nabla_{2,1}, st_2, 0, 0, it_6 \rightarrow it_4\, it_5]$
it_7	$[\nabla_{2,0}, st_0, 0, 0, it_7 \rightarrow it_2\, it_6]$
it_8	$[\nabla_{1,0}, st_4, 0, 0, \varepsilon]$
it_9	$[\nabla_{2,1}, st_4, 0, 0, it_9 \rightarrow it_8\, it_5]$
it_{10}	$[\nabla_{2,0}, st_4, 0, 0, it_{10} \rightarrow it_4\, it_9]$
d_1	$[S, st_4, 0, 0] \longmapsto [\nabla_{2,1}, st_2, 0, 0]$
d_2	$[S, st_4, 0, 0] \longmapsto [\nabla_{2,1}, st_4, 0, 0]$
d_3	$[S, st_2, 0, 0] \longmapsto [\nabla_{2,0}, st_0, 0, 0]$

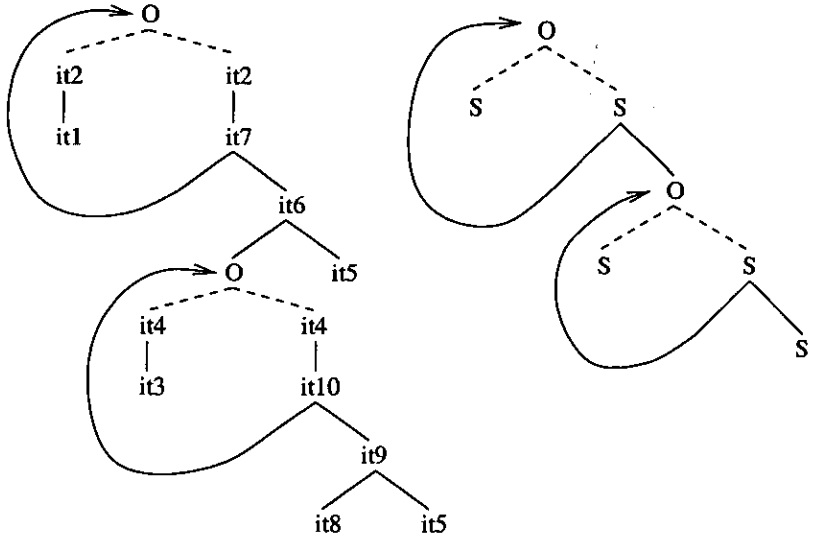


Figura 5.21: Conjunto de items y bosque de análisis de ε con la gramática $\mathcal{G}_3$

CAPÍTULO 6

ANÁLISIS DE GRAMÁTICAS DE CLÁUSULAS DEFINIDAS

Las GICs se han usado y se usan continuamente en la definición de lenguajes formales, como pueden ser los de programación o los lenguajes de marcas[1]. En otros campos, como los lenguajes naturales, por citar un ejemplo más concreto, el poder descriptivo de las GICs se ha mostrado insuficiente y se han propuesto otros formalismos gramaticales para cubrir el hueco descriptivo existente. A cambio se incrementa la complejidad de los analizadores asociados. En el presente capítulo trataremos el formalismo de *gramáticas de cláusulas definidas*, (GCDs).

6.1 Introducción

Como ejemplo de la falta de poder descriptivo de las GICs podemos pensar en la dificultad que entraña representar la concordancia de número y género entre dos categorías sintácticas, digamos, nombre, `N` y determinante, `Det`, dentro de la siguiente producción:

$$\texttt{SN} \rightarrow \texttt{Det N}$$

En este caso, tendríamos que expandir el conjunto de símbolos para contemplar todas las combinaciones posibles: `Det-masc-sing`, `Det-fem-sing`, `Det-masc-plu`, `Det-fem-plu`, ..., `N-masc-sing`, `N-fem-sing`, `N-masc-plu`, `N-fem-plu`, ..., lo cual implicaría la creación de nuevas producciones que contemplasen los nuevos símbolos:

$$\begin{aligned}
&\texttt{SN} \rightarrow \texttt{Det-masc-sing N-masc-sing}\\
&\texttt{SN} \rightarrow \texttt{Det-fem-sing N-fem-sing}\\
&\texttt{SN} \rightarrow \texttt{Det-masc-plu N-masc-plu}\\
&\texttt{SN} \rightarrow \texttt{Det-fem-plu N-fem-plu}\\
&\vdots
\end{aligned}$$

Este primer problema se puede solucionar aumentado el poder descriptivo de las GICs, añadiendo argumentos a los símbolos de la gramática, tal y como se hace en AGFL [45].

[1] Ver por ejemplo, el lenguaje XML: `http://www.w3c.org/XML/`.

Estos argumentos son variables que sólo pueden ser instanciadas con un valor atómico de un dominio finito una única vez, es decir, argumentos con el mismo nombre deben tener el mismo valor. De esta forma se obliga a cumplir la concordancia de género y número:

```
sintagma nominal(número, género):
        determinante(número, género),
        nombre(número, género).
```

Esta extensión de las GICs está próxima a la propuesta en otros lenguajes como `Datalog` [82], y se puede considerar asimilable a una extensión simple, basada en lógica proposicional, del problema expuesto anteriormente. Sin embargo, se muestra insuficiente para representar otras características del lenguaje natural que requieren un dominio infinito para la definición de las categorías gramaticales. Como alternativa se propone una generalización de las GICs basada en la lógica de primer orden.

Sobre la base de las GICs, se generalizan los símbolos de la gramática añadiendo información adicional, a la que nos referiremos como *atributos* del símbolo. Una propiedad importante de la adición de atributos es que los símbolos de la gramática nos permiten representar un conjunto infinito de elementos, extendiendo de este modo su dominio de definición. A continuación se establece una operación que nos permita la manipulación de los símbolos gramaticales con atributos y se adapta convenientemente el mecanismo de derivación de la gramática de forma que tenga en cuenta la información contenida en éstos. La extensión se realiza mediante términos lógicos de primer orden, considerando la unificación como mecanismo de manipulación [113].

Términos lógicos y GCDs

Tal y como hemos mencionado uno de los atributos con que generalizaremos las GICs son los términos lógicos de primer orden[2]. Esta extensión nos permitirá definir las GCDs.

DEFINICIÓN 6.1 (término lógico de primer orden)
Dados dos conjuntos finitos denominados funtores *y* variables, *respectivamente* $\mathcal{F}$ *y* $\mathcal{V}$, *un* término lógico *se define inductivamente como:*

- *Una* constante, $c \in \mathcal{F}$
- *Una* variable, $V \in \mathcal{V}$
- *Un* término compuesto *o* función, $f(t_1, t_2, \ldots, t_n)$, *donde* $f \in \mathcal{F}$ *y los* t_i *son términos lógicos que denominaremos* argumentos *de* f.

  Nos referiremos a f *como el* funtor *de la función, y al número de argumentos* n *como su* aridad. *Alternativamente representaremos las funciones indicando su funtor y aridad:* f/n. □

[2]Para una descripción más detallada el lector puede consultar [87, 173], entre otros.

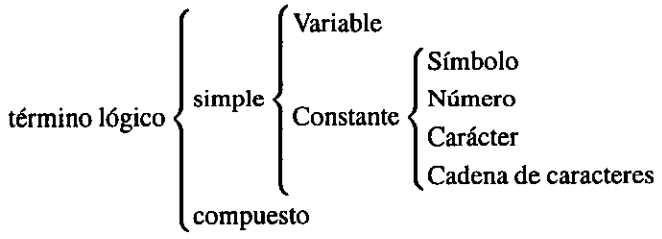
Nótese que una constante puede considerarse una función de aridad 0, es decir, una función sin argumentos. Por convención[3], los nombres de constantes y funtores comenzarán por minúscula y los nombres de variable por mayúscula. Intuitivamente, las constantes representan objetos simples como puedan ser el número o género:

masculino, singular, plural, femenino, neutro, ...

Por su parte las funciones permiten relacionar objetos, agrupándolos en estructuras complejas como fechas, subcategorizaciones, información de concordancia, etc:

fecha(cinco,abril), verbo(transitivo), concordancia(masculino,plural), ...

Finalmente, las variables representan objetos desconocidos *a priori*. En resumen, los términos lógicos se clasifican en la siguiente jerarquía:



Una vez se conoce el valor ligado a una variable, se puede definir mediante el concepto de *sustitución*.

DEFINICIÓN 6.2 (sustitución)
Una sustitución Θ *es una lista de pares (variable,término lógico), que representaremos, indistintamente, como:*

$$\Theta \equiv \{X_1 \leftarrow t_1, \ldots, X_n \leftarrow t_n\}$$
$$\Theta \equiv \{X_1/t_1, \ldots, X_n/t_n\}$$ □

Intuitivamente, una sustitución asocia a cada variable X_i un término t_i. De esta forma, es posible aplicar una sustitución a un término, cambiando cada variable por su valor asociado.

DEFINICIÓN 6.3 (instanciación)
Dados una sustitución Θ y un término lógico t, notaremos la aplicación de Θ a t como $t\Theta$, y diremos que es una instancia *de t.* □

Sobre la operación de instanciación hemos de remarcar que la sustitución de variables se hace de forma simultánea, y por consiguiente:

$$f(X,Y)\{X/Y, Y/8\} = f(Y,8) \quad \text{es cierto}$$
$$f(X,Y)\{X/Y, Y/8\} = f(8,8) \quad \text{es falso}$$

Análogamente, también es posible aplicar una sustitución sobre los términos que aparecen en otra sustitución, es decir, instanciar una sustitución:

[3]Convención heredada de PROLOG.

$$\{X/Y, Y/2\}\,\{X/1, Y/h(Z)\} = \{X/h(Z), Y/2\}$$

En relación a la instanciación, el término a partir del cual instanciamos otro, es más general que éste último, que a su vez puede ser instanciado dando lugar a otro término menos general. Sobre esta idea, establecemos la relación de *subsumción* entre términos.

DEFINICIÓN 6.4 (subsumción)
Dados dos términos lógicos t_1 y t_2 diremos que t_1 subsume *a t_2, (resp. t_2 es* subsumido *por t_1), y lo notaremos $t_1 \preceq t_2$, si y sólo si: $\exists\sigma,\ t_2 = t_1\sigma$* □

Nótese que los términos más generales, contienen menos información que sus instancias:

concordancia(G,N) $\preceq$ concordancia(G,plural) $\preceq$ concordancia(masculino,plural)

A continuación definiremos una operación más para la manipulación de términos lógicos, la *unificación*, que nos permite restringir las sustituciones a aplicar sobre ellos.

DEFINICIÓN 6.5 (unificador y umg)
Dados dos términos lógicos t_1 y t_2 diremos que una sustitución Θ es un unificador *si y sólo si $t_1\Theta = t_2\Theta$.*

Dado un unificador Θ, diremos que es el unificador más general *o* umg[4] *si y sólo si, para cualquier otro unificador Θ', $\exists\sigma,\ \Theta\sigma = \Theta'$.* □

Para cualquier par de términos, si existe un unificador, entonces existe su umg [113]. Intuitivamente el unificador restringe la instanciación de dos términos para que variables con el mismo nombre sean sustituidas por el mismo término en ambos. El umg representa el conjunto mínimo de restricciones a considerar.

EJEMPLO 6.1
Dados los términos lógicos:

$$t_1 = p(A, 3, f(Y))$$
$$t_2 = p(B, C, f(g(Z)))$$

un conjunto de posibles unificadores es el siguiente:

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

donde el umg(t_1, t_2) *es* Θ_3. ■

Desde un punto de vista práctico, el umg nos permite igualar dos términos conservando la mayor generalidad posible. Por ello, es la operación empleada en la definición de derivación en las GCDs. Dichas gramáticas tienen su origen en los trabajos de Colmerauer [38], siendo desarrolladas posteriormente por Pereira y Warren en [104].

[4] En terminología anglosajona, *most general unificator* (mgu).

DEFINICIÓN 6.6 (gramática de cláusulas definidas)
Una GCD es una tupla $\mathcal{D} = (V_T, V_N, \mathcal{V}, \mathcal{F}, \gamma, \overset{\circ}{\mathcal{P}})$ *donde:*

- $\mathcal{V}$ *es un conjunto finito de variables*
- $\mathcal{F}$ *es un conjunto finito de funtores*
- $\mathcal{P} = V_T \cup V_N$ *es un conjunto finito de* símbolos de predicado
- $\overset{\circ}{\mathcal{P}}$ *es el predicado inicial*
- γ *es un conjunto finito de* cláusulas *de la forma:*

$$A_{r,0}(t^1_{r,0}, \ldots, t^{m_0}_{r,0}) \rightarrow A_{r,1}(t^1_{r,1}, \ldots, t^{m_1}_{r,1}) \cdots A_{r,n_r}(t^1_{r,n_r}, \ldots, t^{m_{n_r}}_{r,n_r})$$

donde $A_{r,0} \in V_N$, $A_{r,i} \in \mathcal{P}$, *y* $A_{r,i}(t^1_{r,i}, \ldots, t^{m_i}_{r,i})$ *son* símbolos de la gramática:

  - símbolos terminales *si* $A_{r,i} \in V_T$
  - símbolos no terminales *si* $A_{r,i} \in V_N$
  - $A_{r,0}(t^1_{r,0}, \ldots, t^{m_0}_{r,0})$ *es la* cabeza *de la cláusula* γ_r

y $t^{m_j}_{r,i}$ *son términos lógicos de primer orden.*

Por cada cláusula γ_k *usaremos el vector* $\vec{t_k}$ *para representar las variables que pertenecen a los términos de la cláusula.*

$$\vec{t_k} = \left\{ X \mid X \in \mathcal{V} \text{ y } X \text{ ocurre en } t^{m_j}_{k,i} \right\} \qquad \square$$

Intuitivamente, los símbolos terminales y no terminales se corresponden, respectivamente, con la extensión de los terminales y variables de una GIC, sobre los cuales se añaden los términos lógicos $t^{m_j}_{r,i}$. Nótese que en la definición se han considerado las constantes como funtores de aridad 0 y, por tanto, pertenecen al conjunto $\mathcal{F}$. En lo sucesivo, para unificar criterios y facilitar la lectura, utilizaremos las mismas convenciones que venimos empleando para las GICs, con las siguientes excepciones:

- $X, Y, Z, \ldots \in \mathcal{V}$, son nombres de variables lógicas.
- $A, B, C, \ldots$ son símbolos de la gramática.
- '_' es un nombre de variable lógica especial a la que nos referiremos como *variable anónima*. Intuitivamente es cualquier variable cuyo nombre es irrelevante.
- Al igual que los nombres de funtores, los nombres de predicados comienzan todos por minúscula[5]. Para distinguir los símbolos terminales de los variables, escribiremos los primeros entre comillas.

[5]Una vez más, heredamos esta convención de los sistemas PROLOG.

Una vez añadidos los términos a los símbolos de la gramática, el concepto de derivación cambia de tal manera que un símbolo variable se podrá reescribir mediante una cláusula γ_k, tras aplicar la operación de unificación. El resultado de dicha unificación es una sustitución que se aplica a todos los símbolos de la cadena que se está derivando.

DEFINICIÓN 6.7 (derivación directa)
Dada una GCD $\mathcal{D} = (V_T, V_N, \mathcal{V}, \mathcal{F}, \gamma, \overset{\circ}{\mathcal{P}})$, *definimos una* derivación directa *como:*

$$\alpha B\gamma \Rightarrow (\alpha\delta\gamma)\sigma, \quad \alpha B\gamma \in (V_N \cup V_T)^*,\ A \to \delta \in \mathcal{P},\ \sigma = \text{umg}(A, B) \qquad \square$$

Al igual que en las GICs, el concepto de derivación directa se extiende de forma natural, mediante los cierres transitivo y reflexivo, al de derivación indirecta o derivación en n pasos. A continuación se ilustran las definiciones anteriores mediante un ejemplo.

EJEMPLO 6.2
En este ejemplo emplearemos una GCD dada por el siguiente conjunto de cláusulas:

s → sn(concor(N,G)), sv(N)
s → s, sp
sn(concor(N,G)) → nombre(N,G)
sn(concor(N,G)) → determinante(N,G), nombre(N,G)
sn(concor(N,G)) → sn(concor(N,G)), sp
sp → preposición, sn(concor(N,G))
sv(N) → verbo(transitivo,_,N), sn(_)
nombre(singular,masculino) → "*gorrión*"
nombre(singular,femenino) → "*ventana*"
nombre(singular,masculino) → "*alpiste*"
determinante(singular,masculino) → "*el*"
determinante(singular,femenino) → "*la*"
verbo(transitivo,3ª,singular) → "*come*"
verbo(intransitivo,3ª,singular) → "*canta*"
preposición → "*en*"

A continuación se muestra una derivación de la cadena "el gorrión come alpiste"*:*

s ⇒ sn(concor(N,G)) sv(N) ⇒ determinante(N,G) nombre(N,G) sv(N) ⇒
el nombre(singular,masculino) sv(singular) ⇒ *el gorrión* sv(singular) ⇒
el gorrión verbo(transitivo,_,singular) sn(_) ⇒ *el gorrión come* sn(_) ⇒
el gorrión come nombre(N,G) ⇒ *el gorrión come alpiste*

Nótese que el empleo de la operación de unificación imposibilita la generación de derivaciones como la siguiente:

el nombre(singular,masculino) sv(singular) ⇒ *el ventana sv(singular)* ■

Por la forma en que hemos definido las GCDs como una generalización de las GICs, se sigue que por cada GCD existe una GIC subyacente, que podemos obtener eliminando los términos lógicos que suponen los atributos de los símbolos de predicado. Nos referiremos a esta GIC como *esqueleto independiente del contexto* o esqueleto IC.

DEFINICIÓN 6.8 (esqueleto independiente del contexto)
Dada una GCD $\mathcal{D} = (V_T, V_N, \mathcal{V}, \mathcal{F}, \gamma, \overset{\circ}{\mathcal{P}})$ *definimos su* esqueleto IC *como una GIC* $\mathcal{G} = (N, \Sigma, P, S)$ *donde:*

- $\Sigma = V_T$ *y* $N = V_N$
- $S = A$, $\overset{\circ}{\mathcal{P}} = A(t_1, \ldots, t_n)$
- $A_{r,0} \rightarrow A_{r,1} \cdots A_{r,n_r} \in P$,
  $\forall\, A_{r,0}(t^1_{r,0}, \ldots, t^{m_0}_{r,0}) \rightarrow A_{r,1}(t^1_{r,1}, \ldots, t^{m_1}_{r,1}) \cdots A_{r,n_r}(t^1_{r,n_r}, \ldots, t^{m_{n_r}}_{r,n_r}) \in \gamma$ □

A continuación se ilustra esta definición mediante un ejemplo.

EJEMPLO 6.3
El esqueleto IC de la GCD del ejemplo 6.2 viene dado por el siguiente conjunto de producciones:

$S \rightarrow SN\ SV$	*Nombre* $\rightarrow$ *gorrión*
$S \rightarrow S\ SP$	*Nombre* $\rightarrow$ *ventana*
$SN \rightarrow$ *Nombre*	*Nombre* $\rightarrow$ *alpiste*
$SN \rightarrow$ *Determinante Nombre*	*Determinante* $\rightarrow$ *el*
$SN \rightarrow SN\ SP$	*Determinante* $\rightarrow$ *la*
$SP \rightarrow$ *Preposición* SN	*Verbo* $\rightarrow$ *come*
$SV \rightarrow$ *Verbo* SN	*Verbo* $\rightarrow$ *canta*
	Preposición $\rightarrow$ *en*

■

Siguiendo la evolución en la definición de las GCDs presentaremos los algoritmos de análisis correspondientes como una generalización de los presentados para las GICs.

6.2 Generalización de SDGs

Los SDGs definidos en el capítulo 4 pueden ser generalizados para su aplicabilidad a formalismos con esqueleto IC y unificación de símbolos gramaticales [131]. En general, los pasos deductivos son remplazados por otros donde se comprueban las restricciones impuestas por el nuevo formalismo gramatical. Por ejemplo, el paso de predicción de Earley:

$$\frac{[A \rightarrow \alpha \bullet B\beta, i, j]}{[B \rightarrow \bullet\gamma, j, j]}$$

es sustituido por:

$$\frac{[A \rightarrow \alpha \bullet B\beta, i, j]}{[B' \rightarrow \bullet\gamma, j, j]}\ \langle B \doteq B' \rangle$$

donde $\doteq$ representa la restricción impuesta por la operación de unificación del formalismo elegido. En concreto, en el caso de las GCDs esta restricción viene dada por el cálculo del umg:

$$\frac{[A \to \alpha \bullet B'\beta, i, j]}{[(B \to \bullet\gamma)\sigma, j, j]} \ \langle \sigma = \mathrm{umg}(B, B') \rangle$$

Aplicando estos cambios a los SDG vistos para los algoritmos de análisis sintáctico descendente, ascendente y Earley, obtenemos los pasos deductivos de la figura 6.1.

	Descendente	Ascendente	Earley
Reconocimiento	$\frac{[\bullet a\beta, j]}{[\bullet\beta\sigma, j+1]}$ $\langle(1)\rangle$	$\frac{[\alpha\bullet, j]}{[\alpha w_{j+1}\bullet, j+1]}$	$\frac{[A \to \alpha \bullet a\beta, i, j]}{[(A \to \alpha w_{j+1} \bullet \beta)\sigma, i, j+1]}$ $\langle(1)\rangle$
Predicción	$\frac{[\bullet B'\beta, j]}{[\bullet(\delta\beta)\sigma, j]}$ $\langle(2)\rangle$		$\frac{[A \to \alpha \bullet B'\beta, i, j]}{[(B \to \bullet\delta)\sigma, j, j]}$ $\langle(2)\rangle$
Compleción		$\frac{[\alpha\delta'\bullet, j]}{[\alpha B\sigma\bullet, j]}$ $\langle(3)\rangle$	$\frac{[A \to \alpha \bullet B\beta, i, k][B \to \delta\bullet, k, j]}{[(A \to \alpha B\sigma \bullet \beta)\sigma, i, j]}$ $\langle(4)\rangle$

$\langle(1)\ \sigma = \mathrm{umg}(a, w_{j+1})$
$\langle(2)\ \sigma = \mathrm{umg}(B, B'),\ B \to \delta \in \gamma$
$\langle(3)\ \sigma = \mathrm{umg}(\delta, \delta'),\ B \to \delta \in \gamma$
$\langle(4)\ \sigma = \mathrm{umg}(B, B')$

Figura 6.1: Pasos deductivos de los SDGs para una GCD $\mathcal{D} = (V_T, V_N, \mathcal{V}, \mathcal{F}, \gamma, \overset{\circ}{\mathcal{P}})$

En lo que respecta a las pruebas de fiabilidad y completud de los esquemas presentados, éstas se obtienen directamente a partir de las presentadas para el caso de las GICs aplicando la nueva definición de derivación de la gramática. Desafortunadamente, no es posible garantizar la propiedad de terminación del análisis. Como contraejemplo proponemos la siguiente gramática extraída de [131]:

$$\begin{aligned} s &\to r(0, N) \\ r(X, N) &\to r(s(X), N), b \\ r(N, N) &\to a \end{aligned}$$

Esta gramática genera el lenguaje $\{ab^n\}$, instanciando la variable N al valor de n representado como los sucesores de 0: $s(0), s(s(0)), \ldots$ Con el esquema ascendente no tendremos problemas para analizar las cadenas de entrada, sin embargo con los esquemas descendente y Earley [47] el paso de predicción generará una secuencia infinita de items. Para el caso de Earley, esta secuencia sería:

$$\begin{aligned} &[s \to \bullet r(0, N), 0, 0] \\ &[r(0, N) \to \bullet r(s(0), N)\ b, 0, 0] \\ &[r(s(0), N) \to \bullet r(s(s(0)), N)\ b, 0, 0] \\ &[r(s(s(0)), N) \to \bullet r(s(s(s(0))), N)\ b, 0, 0] \\ &\ldots \end{aligned}$$

Sobre el problema de no-terminación descrito existen varias soluciones propuestas, de las que señalaremos las siguientes:

- Una primera solución consiste en realizar el análisis en dos fases, o en diferido[6]. En una primera fase se analiza la cadena de entrada empleando el esqueleto IC de la gramática, obteniendo como resultado el bosque de análisis [96]. Sobre dicho bosque, en una segunda fase, se decoran los nodos con sus atributos correspondientes, añadiendo nuevas alternativas cuando sea necesario y eliminando alternativas cuando la operación de unificación no tenga éxito.

- Otra solución es el uso de *restrictores positivos* [127, 128] o *restrictores negativos* [148]. El empleo de restrictores establece un número finito de clases de equivalencia en las que se ordenan el conjunto infinito de símbolos gramaticales. Empleando los representantes de estas clases en lugar de los propios símbolos, se relaja el paso de predicción evitando la generación de secuencias infinitas de items.

  En general, la elección del restrictor depende de cada gramática particular, limitando la profundidad de los atributos en los símbolos de la gramática. Para el ejemplo anterior podríamos elegir un restrictor como $r(s(s(s(N))), X)$, de forma que cualquier símbolo que sea subsumido por éste será ignorado.

- En un apartado posterior del presente capítulo presentaremos una solución general válida para cualquier gramática, que será la adoptada por ICE.

A continuación procedemos a transformar el SDG de Earley en el SDG para algoritmos LR(k), tal y como hicimos en el capítulo 4. En primer lugar obtenemos el esquema de análisis LR(0) sin control finito cuyos pasos deductivos se muestran en la figura 6.2.

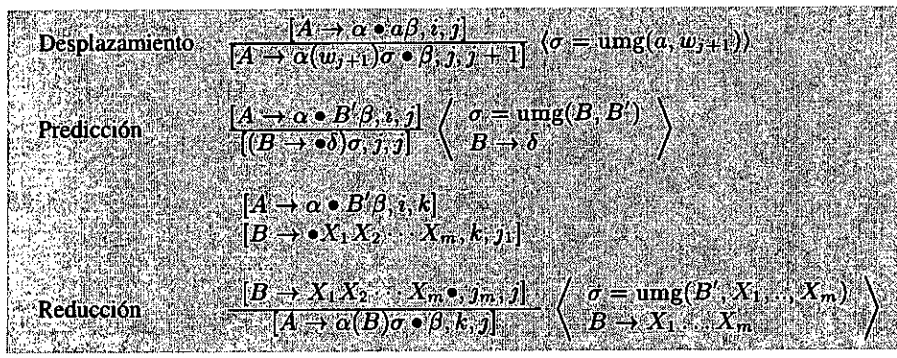


Figura 6.2: SDG del algoritmo LR(0) sin control finito para GCDs

En los esquemas de análisis Earley y LR(0) sin control finito existe un problema de no-terminación, originado por el paso deductivo de predicción, tal y como hemos ilustrado en el caso de Earley. En los algoritmos LR(k) este paso de predicción se traslada a la función de cierre durante la construcción del control finito. A continuación mostramos la extensión de la función de cierre dada en el capítulo 3:

[6] En terminología anglosajona, *off-line parsing*.

DEFINICIÓN 6.9 (cierre)
Sea $\mathcal{D} = (V_T, V_N, \mathcal{V}, \mathcal{F}, \gamma, \overset{\circ}{\mathcal{P}})$ *una GCD y* $kernel_i^{\mathcal{D}}$ *el núcleo del estado* $S_i^{\mathcal{D}}$*, definimos el conjunto cierre como:*

$$cierre_i^{\mathcal{D}} = \left\{ [(A \to \bullet\alpha)\sigma, w], \begin{cases} A \to \alpha \in \gamma \\ \exists [B \to \beta \bullet A'\delta, x] \in kernel_i^{\mathcal{D}} \cup cierre_i^{\mathcal{D}} \\ \sigma = \text{umg}(A, A') \\ w \in \text{PRIMERO}_k^{\mathcal{D}}(\delta x) \end{cases} \right\} \quad \square$$

Además del problema de no-terminación que afecta a su construcción, el control finito resultante presenta otros problemas:

- El número de entradas en la tabla del control finito con respecto a las GICs crece ostensiblemente. En efecto, cada símbolo que instanciamos durante el cálculo del cierre de un estado, es susceptible de convertirse en una nueva entrada de la tabla. Otro efecto de la instanciación de los símbolos es el aumento del número de estados, de forma análoga al fenómeno de división de estados que se produce al aumentar la longitud de los símbolos adelantados.

  EJEMPLO 6.4
  La figura 6.3 muestra dos estados LR, n *y* $n+1$*, del control finito para la gramática del ejemplo 6.2. Estos estados son sucesores del estado* 0 *mediante los símbolos* determinante(sing,masc) y determinante(sing,fem). ■

- La tabla del control finito tiene que estar indexada por estados y símbolos gramaticales. En el caso de las GCDs el número de símbolos gramaticales es potencialmente infinito y el acceso a la tabla se realiza mediante la relación de subsumción. Al contrario de lo que ocurría con las GICs, esta indexación no es trivial y no nos garantiza un acceso en tiempo constante [180, 150, 103, 26].

estado n
sn(concor(sing,masc)) → determinante(sing,masc) • nombre(sing,masc) nombre(sing,masc) → • "gorrión" nombre(sing,masc) → • "alpiste"

estado $n+1$
sn(concor(sing,fem)) → determinante(sing,fem) • nombre(sing,fem) nombre(sing,fem) → • "ventana"

Figura 6.3: Estados LR para la GCD del ejemplo 6.2

Como posibles soluciones a los problemas planteados para el análisis LR(k) de las GCDs destacaremos las siguientes:

- Uso de restrictores [127]. Sólo soluciona el problema de la no-terminación en la construcción del control finito.

- Modificar la función de cierre de forma que se siga comprobando la unificación de los símbolos, pero sin llegar a realizar la instanciación de los nuevos items [95]:

DEFINICIÓN 6.10 (cierre)
Sea $\mathcal{D} = (V_T, V_N, \mathcal{V}, \mathcal{F}, \gamma, \overset{\circ}{\mathcal{P}})$ *una GCD y* $kernel_i^{\mathcal{D}}$ *el núcleo del estado* $S_i^{\mathcal{D}}$, *definimos el* conjunto cierre *como:*

$$cierre_i^{\mathcal{D}} = \left\{ [A \rightarrow \bullet\alpha, w], \begin{cases} A \rightarrow \alpha \in \gamma \\ \exists [B \rightarrow \beta \bullet A'\delta, x] \in kernel_i^{\mathcal{D}} \cup cierre_i^{\mathcal{D}} \\ \sigma = \mathrm{umg}(A, A') \\ w \in \textsc{primero}_k^{\mathcal{D}}(\delta x) \end{cases} \right\}$$

□

De esta forma se evita el problema de la no-terminación, se reduce el número de símbolos que indexan la tabla y el número de estados del control finito.

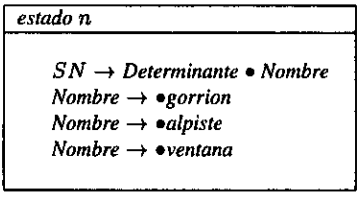
EJEMPLO 6.5
Si calculamos el control finito de la gramática del ejemplo 6.2 con la operación de cierre sin instanciación, los dos estados LR de la figura 6.3 se fusionan en uno solo:

estado n
sn(concor(N,G)) → determinante(N,G) • nombre(N,G) *nombre(sing,masc) → • "gorrión"* *nombre(sing,masc) → • "alpiste"* *nombre(sing,fem) → • "ventana"*

Por contra sigue siendo necesario una operación de subsumción a la hora de indexar la tabla del control finito.

- Construir el control finito a partir del esqueleto IC. Únicamente el intérprete del autómata tendrá en cuenta las operaciones de unificación e instanciación. De esta forma tenemos un mecanismo de indexación de la tabla del autómata trivial y se reduce el número de estados del control finito. Por contra, al igual que ocurre al reducir la longitud de los símbolos adelantados, se pierde parte de la capacidad de determinización del control finito.

EJEMPLO 6.6
Si calculamos el control finito sobre el esqueleto IC de la gramática del ejemplo 6.2, los dos estados LR de la figura 6.3 se fusionan en uno solo:



Una vez presentada la adaptación de los SDGs al formalismo de las GCDs, en la siguiente sección haremos lo propio con el formalismo operacional usado como base en el sistema ICE, es decir, trataremos la generalización de los APs para el caso de las GCDs.

6.3 Autómatas lógicos de pila

El modelo de *autómata lógico de pila* (ALP) fue introducido por Lang en [81]. En esencia es un AP que almacena átomos lógicos y sustituciones, generalizando las transiciones con la aplicación de la operación de unificación.

DEFINICIÓN 6.11 (autómata lógico de pila)
Un autómata lógico de pila *es una 6-upla* $\mathcal{A} = (\mathcal{X}, \mathcal{F}, \Sigma, \Delta, \$, \$_f, \Theta)$, *donde:*

- $\mathcal{X}$ *es un conjunto finito de* variables.
- $\mathcal{F}$ *es un conjunto finito de* símbolos de función.
- Σ *es un conjunto finito de* símbolos terminales.
- Δ *es un conjunto finito de* símbolos de predicado *en el alfabeto de la pila.*
- $\$$ *es el* predicado inicial *de la pila.*
- $\$_f$ *es el* predicado final *de la pila.*
- Θ *es un conjunto de* transiciones *que se clasifican en tres tipos:*

  HORIZONTAL: $B \overset{a}{\longmapsto} C \quad \{A\}$
  APILAR: $B \overset{a}{\longmapsto} BC \quad \{A\}$
  ELIMINAR: $BD \overset{a}{\longmapsto} C \quad \{A\}$

  donde B, C *y* D *son literales en el álgebra de términos* $T_\Delta[\mathcal{F} \cup \mathcal{X}]$, $a \in T_\Sigma[\mathcal{F} \cup \mathcal{X}]$, *y* $\{A\}$ *es un conjunto de restricciones a satisfacer antes de la aplicación de la transición.*

En adelante, representaremos los elementos de la pila como $A.\sigma$, *donde* σ *es una sustitución. En cuanto a las reglas que rigen la aplicación de las transiciones, éstas son:*

- HORIZONTAL: $B \overset{a}{\longmapsto} C(\xi\, E.\rho) = \xi\, C\sigma.\rho\sigma$, *si y sólo si existe una sustitución* σ *tal que* $\sigma = \mathrm{umg}(B, E)$.

  Si el elemento E *de la cima unifica con* B, *lo reemplazamos en la pila por* C *tras aplicar el unificador resultante, y leemos* w_i *de la cadena de entrada, si y sólo si* $w_i\sigma = a\sigma$.

- APILAR: $B \overset{a}{\longmapsto} BC(\xi\, E.\rho) = \xi\, B.\rho\, C\sigma.\sigma$, *si y sólo si existe una sustitución* σ *tal que* $\sigma = \mathrm{umg}(B, E)$.

  Si el elemento E *de la cima unifica con* B, *apilamos un nuevo elemento* C *tras aplicar el unificador resultante, y leemos* w_i *de la cadena de entrada, si y sólo si* $w_i\sigma = a\sigma$.

- ELIMINAR: $BD \overset{a}{\longmapsto} C(\xi\, E'.\rho'\, E.\rho) = \xi\, C\sigma.\rho'\rho\sigma$, *si y sólo si existe una sustitución* σ *tal que* $\sigma = \mathrm{umg}(\langle E, E'\rho\rangle, \langle D, B\rangle)$.

  Eliminamos los elementos E *y* E' *de la cima de la pila, si unifican con* DB, *sustituyéndolos por* C *tras aplicar el unificador resultante, y leemos* w_i *de la cadena de entrada, si y sólo si* $w_i\sigma = a\sigma$. □

Intuitivamente, Σ y Δ se corresponden con los símbolos terminales y símbolos de pila, respectivamente, de un AP. Por su parte, $\mathcal{X}$ y $\mathcal{F}$ son las variables y funtores a partir de los cuales se construyen los términos lógicos con los que se extienden los símbolos anteriores. Esta extensión es análoga a la extensión de GIC a GCD vista anteriormente.

Las transiciones del ALP, como ya se ha dicho, respecto a un AP, han sido generalizadas considerando la unificación como mecanismo de transferencia de información. Sobre ellas destacaremos el hecho de que la unificación de los elementos de transición y la cima de la pila produce una sustitución que se aplica sobre la pila de forma *perezosa*. Si se llevase a cabo de forma inmediata sería necesaria su aplicación sobre cada uno de los elementos de la pila:

$$B \overset{a}{\longmapsto} C(\xi\, E) = (\xi\, E)\sigma, \quad \sigma = \mathrm{umg}(B, E)$$

Por el contrario, la sustitución se almacena en la pila y su aplicación se pospone hasta que las siguientes transiciones acceden a los siguientes elementos de la misma. La demostración de la corrección de esta aproximación se puede encontrar en [81]. De esta manera se mantiene la condición existente en los APs, según la cual las transiciones sólo afectan, a lo sumo, a los dos elementos superiores de la pila del autómata. Esto nos permitirá aplicar las mismas técnicas de tabulación.

Las definiciones de configuración y lenguaje aceptado por el autómata son análogas a las presentadas para el caso de los APs.

EJEMPLO 6.7

En la figura 6.4 se muestra el conjunto de transiciones de un ALP que reconoce el lenguaje generado por el siguiente fragmento de la GCD del ejemplo 6.2:

s → sn(C)
sn(c(N,G)) → determinante(N,G), nombre(N,G)
nombre(sing,masc) → "*gorrion*"
determinante(sing,masc) → "*el*"

Y el análisis de la cadena de entrada "el gorrión" *produce la secuencia de configuraciones que se muestra a continuación:*

\$,	el gorrión
\$ $\nabla_{0,0}(C)$,	el gorrión
\$ $\nabla_{0,0}(C)$ sn(C),	el gorrión
\$ $\nabla_{0,0}(C)$ $\nabla_{1,0}(N,G)$. $\{C/\text{c}(N,G)\}$,	el gorrión
\$ $\nabla_{0,0}(C)$ $\nabla_{1,0}(N,G)$. $\{C/\text{c}(N,G)\}$ determinante(N,G),	el gorrión
\$ $\nabla_{0,0}(C)$ $\nabla_{1,0}(N,G)$. $\{C/\text{c}(N,G)\}$ $\nabla_{3,1}()$. $\{C/\text{c}(sing,masc)\}$,	gorrión
\$ $\nabla_{0,0}(C)$ $\nabla_{1,1}$(sing,masc). $\{C/\text{c(sing,masc)}\}$,	gorrión
\$ $\nabla_{0,0}(C)$ $\nabla_{1,1}$(sing,masc). $\{C/\text{c(sing,masc)}\}$ nombre(sing, masc). $\{N/\text{sing}, G/\text{masc}\}$,	gorrión
\$ $\nabla_{0,0}(C)$ $\nabla_{1,1}$(sing, masc). $\{C/\text{c(sing,masc)}\}$ $\nabla_{2,1}()$. $\{N/\text{sing}, G/\text{masc}\}$,	
\$ $\nabla_{0,0}(C)$ $\nabla_{1,2}$(sing, masc). $\{C/\text{c(sing,masc)}\}$,	
\$ $\nabla_{0,1}$(c(sing,masc)). $\{C/\text{c(sing,masc)}\}$,	

■

$\$ \mapsto \$\ \nabla_{0,0}(C)$
$\nabla_{0,0}(C) \mapsto \nabla_{0,0}(C)$ sn(C)
$\nabla_{1,0}(N,G) \mapsto \nabla_{1,0}(N,G)$ determinante(N,G)
$\nabla_{1,1}(N,G) \mapsto \nabla_{1,1}(N,G)$ nombre(N,G)
sn(c(N,G)) $\mapsto \nabla_{1,0}(N,G)$
$\nabla_{0,0}(C)\ \nabla_{1,2}(N,G) \mapsto \nabla_{0,1}(C)$
$\nabla_{1,0}(N,G)\ \nabla_{3,1}() \mapsto \nabla_{1,1}(N,G)$
$\nabla_{1,1}(N,G)\ \nabla_{2,1}() \mapsto \nabla_{1,2}(N,G)$
nombre(sing,masc) $\overset{el}{\mapsto} \nabla_{2,1}()$
determinante(sing,masc) $\overset{gorrión}{\mapsto} \nabla_{3,1}()$

Figura 6.4: Transiciones del ALP

Como se observa en el ejemplo anterior, seguimos utilizando los símbolos $\nabla_{k,i}$, que habíamos introducido en el análisis de GICs para representar la pasos intermedios en el reconocimiento de las producciones de la gramática. En el caso de las GCDs, además, extendemos estos símbolos con el vector $\vec{t_k}$ de variables[7] que ocurren en la cláusula correspondiente: $\nabla_{k,i}(\vec{t_k})$.

De forma análoga al caso de los APs definiremos una serie de esquemas de compilación que nos permitirán obtener las transiciones de un ALP que reconoce una GCD dada. Para ello presupondremos una gramática aumentada donde hemos añadido una cláusula inicial del estilo

$$\mathcal{P}' \rightarrow \overset{\circ}{\mathcal{P}}$$

DEFINICIÓN 6.12 (esquema de compilación genérico para GCDs)
Dada una GCD definiremos el esquema de compilación genérico, *parametrizable mediante la especificación de los símbolos* $\overrightarrow{A_{r,s}}$ *y* $\overleftarrow{A_{r,s}}$, *como el siguiente conjunto de reglas:*

[7]Nótese que este conjunto se puede simplificar eliminando variables que no serán usadas [178].

[INICIA]	$\$_0 \longmapsto \$_0\ \nabla_{0,0}(\vec{t_0})$	
[LLAMADA]	$\nabla_{r,s}(\vec{t_r}) \longmapsto \nabla_{r,s}(\vec{t_r})\ \overrightarrow{A_{r,s+1}}$	$\forall r, \forall s, 0 \leq s < n_r$
[SELEC]	$\overrightarrow{A_{r,0}} \longmapsto \nabla_{r,0}(\vec{t_r})$	$\forall r, r \neq 0$
[PUBLICA]	$\nabla_{r,n_r}(\vec{t_r}) \longmapsto \overleftarrow{A_{r,0}}$	$\forall r$
[RET]	$\nabla_{r,s}(\vec{t_r})\ \overleftarrow{A_{r,s+1}} \longmapsto \nabla_{r,s+1}(\vec{t_r})$	$\forall r, \forall s, 0 \leq s < n_r$
[RECONOC]	$\nabla_{r,s}(\vec{t_r}) \overset{a}{\longmapsto} \nabla_{r,s+1}(\vec{t_r})$	$a = A_{r,s+1}$

Donde $\$_0$ *y* $\overleftarrow{A_{0,1}}$ *son los símbolos de pila inicial y final, respectivamente.* □

Intuitivamente, el significado de las reglas de compilación es el mismo que en el caso de las GICs, teniendo en cuenta la diferencia entre las transiciones de un ALP y de un AP. Los esquemas de compilación concretos se obtienen de forma análoga, instanciando los parámetros $\overrightarrow{A_{r,s}}$ y $\overleftarrow{A_{r,s}}$.

A continuación, y siguiendo el orden de exposición de los APs, trataremos la tabulación de los ALPs.

6.4 Tabulación

Tal y como mencionábamos anteriormente, la aplicación de las transiciones de un ALP afecta, a lo sumo, a los dos elementos de la cima de la pila. Esta condición nos permite aplicar el mismo concepto de entorno dinámico visto en el capítulo 5 para los APs. En concreto, definiremos los entornos S^2 [81] y S^1 [178, 164], que comparten las mismas características que sus homólogos.

- **Entorno dinámico S^2.** Para la representación de los items se emplean los dos elementos de la cima de la pila:

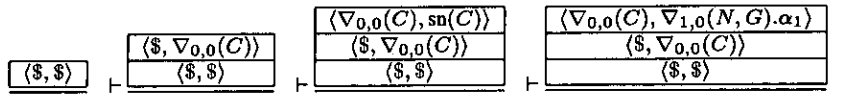
$$\langle A', A\rangle = \{\xi\ B'.\rho' B.\rho \mid B' = A',\ B\rho = A\}$$

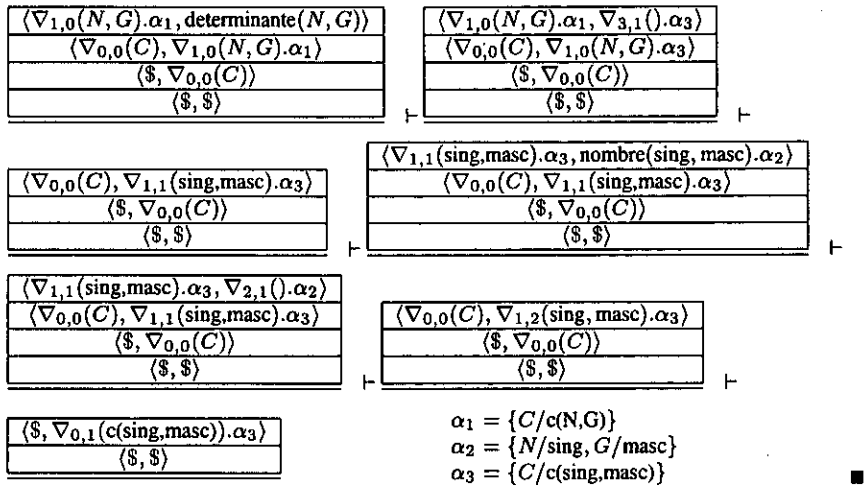
Y el operador Op, que traduce las transiciones de S^T en S^2, se define como:

- $Op(B \longmapsto C)(\langle A', A\rangle) = \langle A'\sigma, C\sigma\rangle,\ \sigma = \mathrm{umg}(A, B)$
- $Op(C \longmapsto CF)(\langle A', A\rangle) = \langle A\sigma, F\sigma\rangle,\ \sigma = \mathrm{umg}(C, A)$
- $Op(CF \longmapsto G)(\langle E', E\rangle, \langle A', A\rangle) = \langle E'\sigma', G\sigma\sigma'\rangle,$
  $\sigma = \mathrm{umg}(A'A, CF),\ \sigma' = \mathrm{umg}(A', E)$

EJEMPLO 6.8

A continuación se muestra la secuencia de configuraciones de la pila en S^2 para el análisis del ejemplo 6.7.





- **Entorno dinámico S^1.** Los items se construyen a partir del elemento en la cima de la pila, con lo que se alcanza el mayor grado de compactación posible:

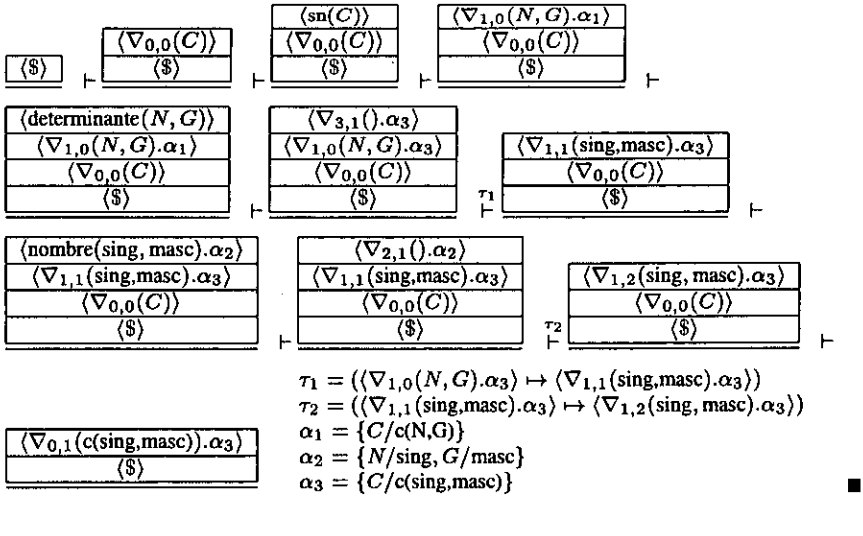
$$\langle A \rangle = \{\xi A.\rho\}$$

Al igual que en el caso de los APs, la construcción del operador Op es más compleja, y exige el uso de transiciones dinámicas:

- $Op(B \longmapsto C)(\langle A \rangle) = \langle C\sigma \rangle,\ \sigma = \text{umg}(B, A)$
- $Op(C \longmapsto CF)(\langle A \rangle) = \langle F\sigma \rangle,\ \sigma = \text{umg}(C, A)$
- $Op(CF \longmapsto G)(\langle A \rangle) = (C\sigma \longmapsto G\sigma),\ \sigma = \text{umg}(F, A)$

EJEMPLO 6.9

A continuación se muestra la secuencia de configuraciones de la pila en S^1 para el análisis del ejemplo 6.7.



Una vez definidos los ALPs y las técnicas de tabulación aplicables sobre ellos, pasamos a describir su aplicación en el sistema ICE.

6.5 ICE

Aunque fue desarrollado originalmente para las GICs [164, 15, 16, 11, 12], posteriormente se amplió al análisis de GCDs [169, 168, 166, 172]. Esta extensión se ha realizado siguiendo la línea descrita en el presente capítulo.

6.5.1 Control estático

La propuesta de control estático en ICE trata de evitar conflictos de evaluación entre términos del ALP mediante la extensión del concepto de símbolo adelantado, ampliamente utilizado en la determinización de las GICs, a las GCDs. En concreto, este control sigue basándose en un autómata de tipo LR(k). En un apartado anterior ya adelantábamos los problemas de no-terminación y de explosión en el número de estados generados cuando se construye el autómata directamente a partir de la GCD, por lo que la propuesta en ICE es calcular el control finito a partir del esqueleto IC de la gramática. La figura 6.5 muestra el SDG resultante tras añadir el control estático y realizar la binarización implícita de las cláusulas. Esta binarización nos asegura una complejidad temporal $\mathcal{O}(n^3)$ en ausencia de símbolos de función.

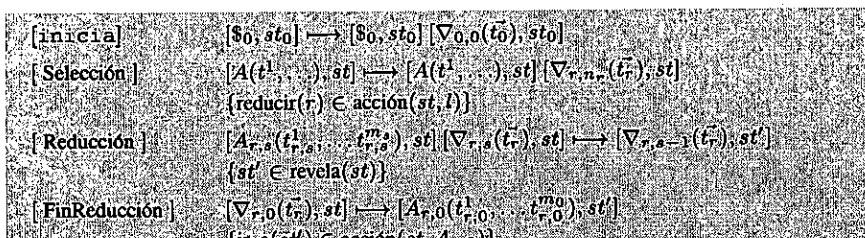


Figura 6.5: Esquema de compilación en ICE para GCDs

Sobre las ventajas derivadas del cálculo del control finito a partir del esqueleto IC de la gramática, señalaremos:

- Se garantiza la terminación del cálculo de dicho control, y desde el punto de vista computacional su coste es irrelevante.
- La indexación de la tabla del autómata es trivial, y el acceso se realiza en tiempo constante.

- El dominio determinista de la familia LR(k) es amplio, lo cual se traduce en un tratamiento eficiente del determinismo local[8].
- El fenómeno de división de estados, típico de las estrategias de determinización por predicción estática, se mantiene dentro de límites razonables.

Tal y como indicábamos para el caso de los APs, las técnicas de control estático no tienen porqué limitarse a autómatas LR(k), ni son exclusivas, siendo posible su combinación. En concreto, aunque mantenemos el uso del autómata de tipo LR, señalaremos una mejora sobre la construcción del esqueleto IC. Esta mejora se basa en las implementaciones más comunes en las máquinas abstractas de PROLOG tipo WAM [4] para la indexación de las cabezas de la cláusulas, $A_{k,0}(t_1,\ldots,t_n)$, cuyo índice se construye como alguna de las siguientes alternativas:

1. El símbolo de predicado y su aridad: $A_{k,0}/n$
2. El símbolo de predicado y el tipo o una descripción del primer argumento $@(t_1)$:

t_1	a	X	$f(s_1,\ldots,s_m,)$	$[t_1,\ldots,t_m\|t]$
$@(t_1)$	CST	$*$	FUN	LIST

t_1	a	X	$f(s_1,\ldots,s_m,)$	$[t_1,\ldots,t_m\|t]$
$@(t_1)$	a	$*$	f/m	LIST

Sobre la segunda opción señalaremos dos inconvenientes. En primer lugar la arbitrariedad en la elección del primer argumento. En segundo lugar, al igual que en la propuesta de [95], el uso de variables, implica la relación de subsumción para acceder a las entradas de la tabla del control finito. Por tanto, para conservar el acceso a la tabla en tiempo constante en el sistema ICE, optamos por la primera opción.

DEFINICIÓN 6.13 (esqueleto IC)
Dada una GCD $\mathcal{D} = (V_T, V_N, \mathcal{V}, \mathcal{F}, \gamma, \overset{\circ}{\mathcal{P}})$ *definimos su* esqueleto IC *como una GIC* $\mathcal{G} = (N, \Sigma, P, S)$ *cuyo conjunto de producciones viene dado por:*

$$\forall\ A_{r,0} \rightarrow A_{r,1}\cdots A_{r,n_r} \in \gamma,\ \ A^{IC}_{r,0} \rightarrow A^{IC}_{r,1}\cdots A^{IC}_{r,n_r} \in P$$

donde $A^{IC}_{r,i} = A/m_i,\ A_{r,i} = A(t^1_{r,i},\ldots,t^{m_i}_{r,i})$. □

Esta nueva definición del esqueleto IC nos permite discriminar entre símbolos como $p(X,Y)$ y $p(c(N,G))$, que sabemos *a priori* que no son unificables.

6.5.2 Tabulación y sincronización

Para la tabulación de los ALPs en ICE, continuamos con el entorno dinámico S^1, al mismo tiempo que seguimos aplicando el mecanismo de sincronización descrito para el caso de los APs. El esquema de compilación del ALP resultante se muestra en la figura 6.6.

[8] En la práctica, el proceso es determinista durante buena parte del análisis.

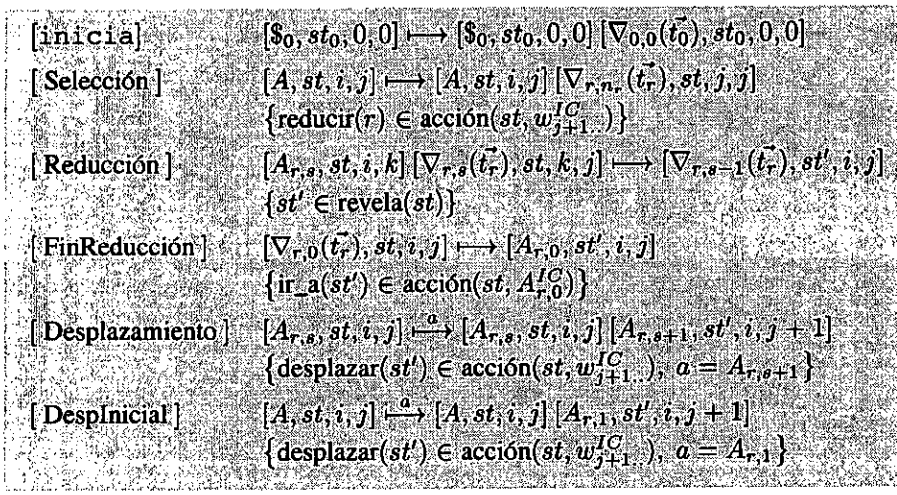


Figura 6.6: Esquema de compilación en ICE para GCDs, con sincronización

En lo que respecta a la comprobación de redundancias dentro de la tabla de items, no es suficiente con establecer una condición de igualdad entre éstos como en el caso de las GICs y APs. En su lugar es necesario emplear la relación de subsumción, la cual, dados un ítem recién generado $[B, st, i, j]$ y algún ítem de la tabla $[A, st, i, j]$, establece dos condiciones en que el nuevo ítem resulta redundante [81]:

- *Admisibilidad fuerte*, $A \preceq B$. Es decir, el nuevo ítem es una instancia de otro que ya existe en la tabla, y por lo tanto no es necesario continuar con su análisis.
- *Admisibilidad débil*, $B \preceq A$. Es decir, el nuevo ítem es una versión más general de otro que ya existe en la tabla. En este caso, podemos optar por:
  - Continuar el análisis de $[A, st, i, j]$, obteniendo de esta forma un análisis más general que otro ya existente.
  - Por contra, podemos reemplazar $[B, st, i, j]$ por $[A, st, i, j]$, para lo cual deberemos propagar el cambio a todos los cálculos realizados a partir del primer ítem.

    En general, esta opción se obvia debido a su coste temporal. Como alternativa, De la Clergerie [179] propone una estrategia de análisis que evita en lo posible la situación descrita. Otra opción es realizar una compartición de estructuras adecuada de manera que la propagación de los cambios se realice automáticamente, tal y como muestra la figura 6.7.

Para completar la aplicación de la tabulación en el sistema ICE para GCDs, resta por definir el mecanismo de indexación de la tabla de items. Dado el carácter infinito del conjunto de símbolos gramaticales no es posible emplear una indexación directa[9]

[9]Nos referimos, por ejemplo, a la construcción de una tabla de dispersión.

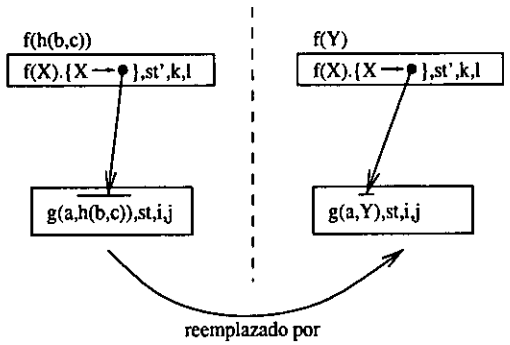


Figura 6.7: Propagación de cambios

como en el caso de las GICs. Recordemos que la comprobación de redundancias, es decir, la operación de subsumción, se apoya en el mecanismo de indexación. Para que ésta sea efectiva, tiene que ser posible, a partir de un ítem B, discriminar el mayor número posible de items que no subsumen o son subsumidos por B. Sobre el desarrollo de mecanismos de indexación con estas propiedades se han propuesto diversas aproximaciones:

- El mecanismo de indexación de la WAM [4] para PROLOG, que ya hemos descrito anteriormente.
- Uso de la misma estructura tanto para almacenar los términos, como para calcular los índices. Estas estructuras pueden ser árboles abstractos [100], árboles discriminantes y árboles de sustitución [54, 55], o *tries* [108]. Las figuras 6.8 y 6.9 muestra dos ejemplos. La idea común consiste en calcular los índices mediante un recorrido de la estructura de datos, árboles o autómatas, que contiene los términos incluidos en la tabla. Durante dicho recorrido, se aplican las sustituciones indicadas por los nodos de la estructura, obteniendo al alcanzar un nodo hoja o estado final el término almacenado. A modo de ejemplo, el cálculo del índice de $(a, f(a,b), a)$ supondría recorrer los estados $s_1, s_2, s_3, s_4, s_5, s_6$ en el *trie* de la figura 6.9, comparando cada uno de los términos con la transición correspondiente. Si en lugar de una comparación de igualdad, usásemos la relación de subsumción, también obtendríamos $s_1, s_2, s_3, s_4, s_7, s_8$, es decir el índice de un término que subsume $(a, f(a,b), a)$. Como desventajas señalaremos que:
  - El acceso a la tabla no se realiza en tiempo constante.
  - Las operaciones de inserción en la tabla resultan complejas puesto que deben mantener el árbol o autómata correspondiente.

Para el sistema ICE, optaremos por continuar la línea de los demás componentes, construyendo el índice de un ítem $[A, st, i, j]$ como una función $h(A^{IC}, st, i, j)$, donde $h(x)$ puede ser una función de dispersión que nos proporcione acceso en tiempo constante a la tabla. Sobre esta aproximación hemos de advertir el hecho de que diferentes items pueden tener el mismo índice. En efecto:

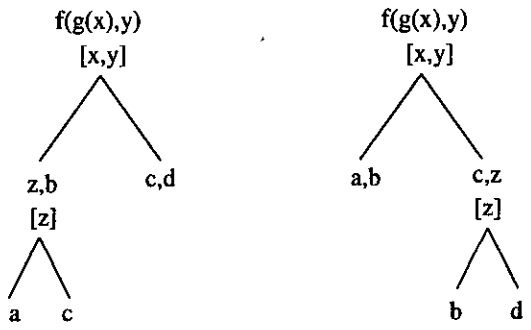


Figura 6.8: Ejemplo de árbol abstracto

$$h([sn(agr(N,G)),st,i,j]) = h([sn(agr(\text{sing},G)),st,i,j])$$

Esta característica nos obliga a implementar la tabla de items empleando técnicas como las listas de desbordamiento [72]. Sobre el sistema de indexación seleccionado, destacaremos:

- Reciben distintos índices, y por lo tanto son discriminados a la hora de realizar una comprobación de redundancia:
  - Los items pertenecientes a diferentes conjuntos. Estos items cubren el análisis de diferentes porciones de la cadena de entrada.
  - Los items con distinto símbolo de predicado y aridad, $A^{IC} \neq B^{IC}$.
  - Los items cuyo estado del control finito es distinto. Al igual que ocurre en el caso de las GICs, esta característica puede evitar en algunos casos que se detecten redundancias.
- Reciben el mismo índice, y por lo tanto se encuentran en la misma lista de desbordamiento, los items que corresponden al análisis de la misma porción de la cadena de entrada, y cuyo estado del control finito es el mismo y, por lo tanto, es posible que cumplan la relación de subsumción. Consecuentemente la comprobación de redundancias se limita a los elementos de dicha lista[10].

6.5.3 Implementación

A continuación describiremos las decisiones de diseño tomadas en la implementación del sistema ICE. El sistema consta de dos fases, una fase de compilación y una fase de análisis propiamente dicha. La fase de análisis utiliza un algoritmo de punto fijo apoyándose en las técnicas de tabulación desarrolladas.

[10]En la práctica, dadas las condiciones, mismo estado, mismo conjunto de items y mismo puntero hacia atrás, la mayoría de las veces esta lista contiene un único elemento.

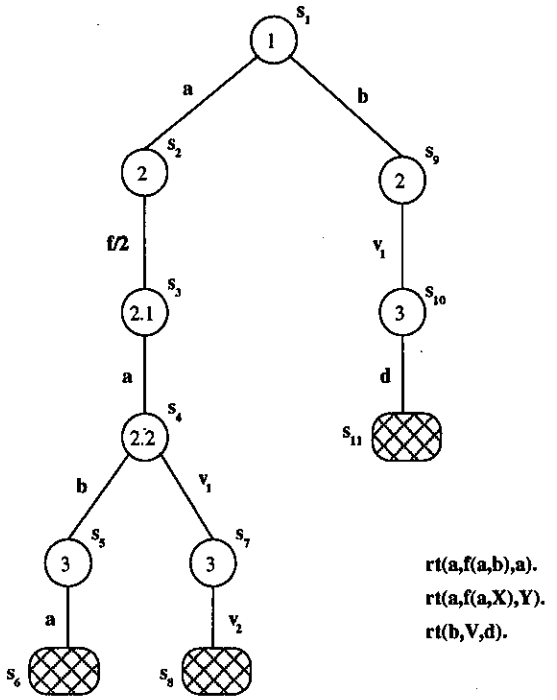


Figura 6.9: Ejemplo de trie

Compilación

La fase de compilación se encarga de extraer el esqueleto IC de la GCD y las estructuras de datos asociadas a los términos lógicos. A partir de dicho esqueleto se construye el control finito asociado a la gramática, que será un autómata LALR(1). Para la obtención de dicho autómata se emplea una versión modificada[11] de BISON [40].

Bucle de control

Como ya hemos indicado, el algoritmo es del tipo punto fijo. Partiendo de un ítem inicial se crean nuevos items hasta que no es posible crear ninguno más. Para ello disponemos de un bucle de control asistido por:

- Una agenda. Será una estructura de datos donde almacenar las acciones de análisis que todavía no han sido realizadas.
- La tabla de objetos. Gracias a las características de comprobación de redundancias, evita la repetición de items.

Sobre este punto resaltaremos el hecho de que el bucle de control debe adecuarse al mecanismo de sincronización. Para ello procede de forma iterativa sobre un conjunto

[11] Las modificaciones son necesarias puesto que originalmente BISON sólo genera autómatas deterministas.

de items y no pasa al siguiente hasta que no es posible añadir más elementos al conjunto actual. Esto implica que los items resultado de una operación de desplazamiento son almacenados en el núcleo del siguiente conjunto y su procesamiento se interrumpe temporalmente. Así, los pasos del bucle de control, ilustrados por la figura 6.10, son:

1. Mientras el núcleo del siguiente conjunto de items no esté vacío: proceder con el siguiente conjunto de items.
   (a) Mientras podamos añadir items nuevos al conjunto actual.
      i. Seleccionar de la agenda una acción para un ítem del conjunto actual.
      ii. Ejecutar la acción y para cada uno de los items resultantes, si es un ítem nuevo:
         - Insertar en la tabla de items.
         - Si es el resultado de una acción de desplazamiento, añadir al núcleo del siguiente conjunto de items.
         - Consultar en el control finito las acciones a realizar sobre él y añadirlas a la agenda.

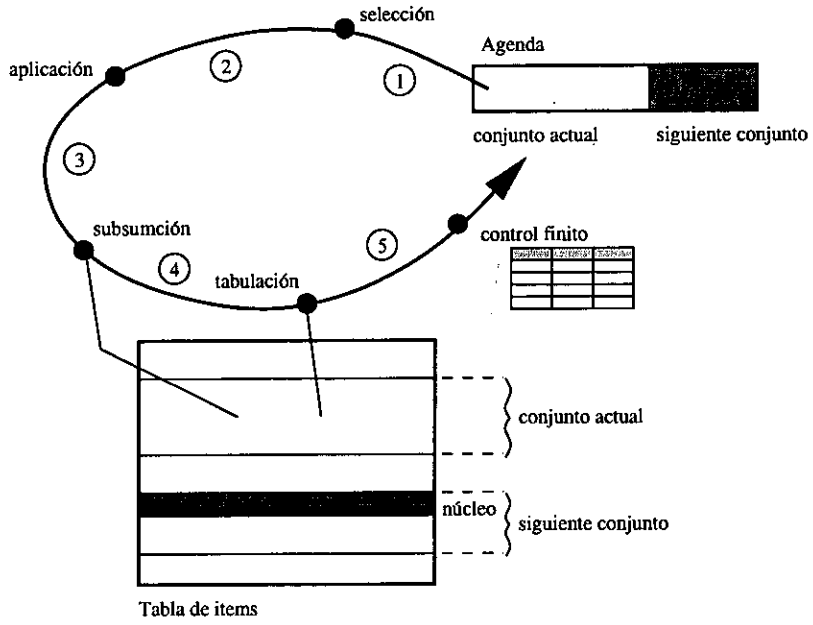


Figura 6.10: Bucle de control del sistema ICE

Agenda

Como hemos visto, la agenda juega un papel fundamental dentro del bucle de control. Sobre su uso e implementación realizaremos las siguientes consideraciones:

- Debido a que el bucle de control selecciona acciones únicamente para los items del conjunto actual, y tal y como se aprecia en la figura 6.10, la agenda está dividida en dos partes, una correspondiente a dicho conjunto y otra para el siguiente.
- El algoritmo de análisis funciona concurrentemente examinando al mismo tiempo todas las ramas de análisis alternativas. De forma implícita, la agenda mantiene la lista de acciones pendientes en cada rama.
- Para evitar búsquedas innecesarias en la tabla de items, la agenda guarda tanto la acción a realizar como una referencia al ítem sobre el que se aplica. De esta forma evitamos considerar aquellos items para los cuales ya hemos realizado todas las acciones posibles.
- La estructura de datos empleada para la implementación de la agenda determina la estrategia del análisis, y viceversa. Por ejemplo, la implementación de la agenda como una cola FIFO, primero dentro primero fuera, supone una estrategia primero en anchura, mientras que una cola LIFO, último dentro primero fuera, supondría una estrategia primero en profundidad.

Asimismo, la agenda debe cumplir dos propiedades importantes:

- Una acción no se puede seleccionar más que una vez. Para ello simplemente una vez seleccionada, la eliminamos de la agenda.
- Ha de existir un orden de selección equitativo. No puede haber acciones que no se seleccionen nunca. Siempre y cuando el análisis termine, esta propiedad está garantizada por la anterior y el mecanismo de tabulación que impide generar dos veces la misma acción para el mismo ítem.

En relación a la última propiedad, destacaremos la posibilidad de realizar una comprobación de admisibilidad débil sobre los elementos de la agenda. De esta forma favorecemos las acciones sobre items más generales, reduciendo, posiblemente, el número de cálculos necesarios.

Tabla de items

El último elemento a considerar es la tabla de items. Como ya hemos indicado anteriormente, para un ítem $[A, st, i, j]$ su índice en la tabla se construye como $h(A^{IC}, st, i, j)$. En concreto h es una función de dispersión [72] donde los conflictos de claves se resuelven mediante una lista de desbordamiento. Como también indicábamos anteriormente, de esta forma los items susceptibles de cumplir la relación de subsumción, se encuentran en la misma lista de desbordamiento.

CAPÍTULO 7

UNIFICACIÓN Y ESTRUCTURAS CÍCLICAS EN GCDs

Independientemente de la eficacia del esquema de análisis elegido, las implementaciones de la operación de unificación y de las estructuras de datos asociadas son, generalmente, costosas tanto en tiempo como en espacio. Así, algunos autores indican ratios de hasta un 90% del tiempo total de análisis dedicado únicamente a la operación de unificación [144].

Por otra parte, la extensión de las GICs a GCDs no sólo aumenta el poder descriptivo del formalismo gramatical resultante, también aumenta la complejidad de los analizadores sintácticos asociados. Para afrontar este problema en el capítulo 6 incluíamos en el sistema ICE técnicas de tabulación, control estático y un mecanismo de unificación eficiente. No obstante, la complejidad del análisis de las GCDs también se ve reflejada en la aparición de estructuras cíclicas dentro de los términos lógicos que hemos empleado para la extensión de los símbolos gramaticales. Estas estructuras dan lugar a nuevos problemas de no-terminación, que abordaremos en el presente capítulo.

7.1 Implementación de la unificación

La implementación de la unificación es un aspecto importante, ya que se trata de una operación costosa que influye notablemente en el rendimiento del análisis. En lo que concierne a los ALPs, recordemos que está presente en la aplicación de cualquier transición:

- $B \overset{a}{\longmapsto} C(\xi\, E.\rho) = \xi\, C\sigma.\rho\sigma,\ \sigma = \mathrm{umg}(B, E)$.
- $B \overset{a}{\longmapsto} BC(\xi\, E.\rho) = \xi\, B.\rho\, C\sigma.\sigma,\ \sigma = \mathrm{umg}(B, E)$.
- $BD \overset{a}{\longmapsto} C(\xi\, E'.\rho'\, E.\rho) = \xi\, C\sigma.\rho'\rho\sigma,\ \sigma = \mathrm{umg}(\langle E, E'\rho\rangle, \langle D, B\rangle)$.

Así, por cada transición es necesario calcular el umg de los items correspondientes e instanciar un nuevo ítem a partir de dicho unificador. En este sentido es necesario abordar una implementación que:

- Realice de forma eficiente el cálculo del umg.
- Represente los términos lógicos en base a una estructura de datos adecuada para:

- Facilitar el cálculo del umg y la instanciación, mejorando la eficiencia temporal.
- Permitir la compartición, mejorando la eficiencia espacial.

A continuación pasamos a tratar los dos puntos expuestos anteriormente.

7.1.1 Algoritmo de unificación

En primer lugar, señalaremos que el algoritmo de unificación clásico de Robinson [113] se define de forma recurrente. Sin embargo, una implementación de ese estilo penaliza, en general, los tiempos de ejecución del mismo, por lo que recurrimos al uso de una pila explícita:

ALGORITMO 7.1 (unificación)
Entrada: Dos términos lógicos t_1 y t_2

Salida: El umg de t_1 y t_2, o `fallo` si no son unificables

Pasos:

1. Inicialización: $\Theta = \{\}$; Pila=$\emptyset$
2. Añadir $t_1 = t_2$ a la Pila
3. **mientras** Pila$\neq \emptyset$
   (a) Extraer $X = Y$ de la Pila
   (b) **caso**

$X \notin Y$:	sustituir(X, Y) en la Pila añadir$(\Theta, X \leftarrow Y)$
$Y \notin X$:	sustituir(Y, X) en la Pila añadir$(\Theta, Y \leftarrow X)$
$X \leftarrow Y$:	**continuar**
$X \leftarrow f(X_1, \ldots, X_n)$ y $Y \leftarrow f(Y_1, \ldots, Y_n)$:	**desde** i=1 **hasta** n Añadir $X_i = Y_i$ a la Pila
sino :	**devolver** `fallo`

4. **devolver** Θ □

Existen múltiples trabajos enfocados en la mejora del algoritmo de unificación [6, 88]. En nuestro caso, partiendo de la definición básica que acabamos de realizar, nos centraremos en la compartición de las estructuras de datos asociadas como medida para mejorar la eficiencia. A continuación trataremos la representación tanto de los términos lógicos, como de los unificadores obtenidos mediante el algoritmo.

7.1.2 Estructuras de datos

En relación a la estructura de datos empleada para representar los términos lógicos, el primer aspecto a considerar es la elección entre copia o compartición. Durante la operación del autómata, es necesario unificar los símbolos de sus transiciones con los items producidos, dando lugar a un nuevo símbolo si la unificación es posible. Llamaremos a los términos originales, del ítem y la transición, *términos argumento* y al término resultante de la unificación, *término resultado*.

Algunas implementaciones de la unificación, para reducir su complejidad, funcionan de forma destructiva sobre los términos argumento. Puesto que dichos términos forman parte de la definición de la gramática o de items ya tabulados, es necesario realizar antes una copia de los mismos. Este proceso de copia penaliza en gran medida la eficiencia, tanto desde el punto de vista del coste temporal como espacial. Por tanto buscaremos evitar en la medida de lo posible la copia innecesaria de argumentos [144], que se produce en los siguientes casos:

- *Copia extrema.* Los términos argumento se copian antes de comenzar la unificación. Ciertas partes son redundantes y podrían, simplemente, ser compartidas.

- *Copia prematura.* La copia se realiza antes de saber si la unificación tendrá éxito. Si no es así, la copia fue innecesaria.

Por tanto, hemos de preferir la compartición de términos antes que la copia. Esta compartición puede venir dada por:

- La propia gramática, como es el caso en la siguiente instanciación:

$$f(X,X).\{X/a\}$$

  donde los dos argumentos de la función f pueden compartir la representación de la constante a, en lugar de realizar una copia de la misma, tal y como se muestra en la figura 7.1. Nos referiremos a este tipo de compartición como *compartición de estructuras*.

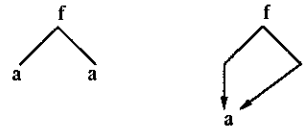


Figura 7.1: $f(X,X).\{X/a\}$ sin y con compartición de estructuras

- La implementación del algoritmo de unificación y de las propias estructuras de datos. Para distinguirla del tipo de compartición anterior, que refleja la estructura de la gramática, nos referiremos a ella como *compartición de estructuras ocultas* o *compartición de estructuras de datos*.

Para poder reflejar convenientemente ambos tipos de compartición, la estructura de datos elegida será un grafo dirigido[1]. Si además optamos por un grafo cíclico, podremos representar términos de profundidad infinita, por ejemplo: $f(f(f(\ldots)))$.

A continuación señalamos algunas de las ideas empleadas a la hora de permitir la compartición de estructuras y evitar la copia durante la unificación:

- Uso de *entornos de unificación* [26, 103, 182]. Durante la unificación los cambios se almacenan en un entorno, que en el caso de los términos lógicos será un conjunto de asociaciones entre variables y términos[2]. De esta forma no es necesario destruir los términos argumento y es posible la compartición de estructuras asociando a cada variable una referencia en vez de una copia del término correspondiente, tal y como mostrábamos en la figura 7.1.

  El uso de entornos para la compartición de estructuras en un analizador con estrategia descendente introduce el problema del renombramiento [103] a la hora de unir diferentes ramas de análisis. En efecto, cada rama tendrá asociado un entorno con distintas variables lógicas que, sin embargo, pueden tener el mismo nombre. La propuesta de *compartición de capas* [179, 180, 150] presenta una estructura de datos para la representación de entornos que permite realizar la operación de renombramiento de forma eficiente.

- La unificación se implementa manipulando directamente los grafos, y empleando alguna técnica que evite en lo posible la copia:

  - Copia incremental [186], sólo se copia un nodo del grafo cuando es estrictamente necesario.
  - Unificación reversible [67, 52, 144], similar a la copia incremental, los cambios se realizan de forma que sea posible deshacerlos si la unificación falla, o crear una nueva estructura si tiene éxito.
  - Unificación perezosa [52], la copia se pospone hasta que es necesario realizar un cambio destructivo. En relación a esta aproximación, el uso de entornos también puede entenderse como una instanciación perezosa de términos lógicos.

En lo que respecta al sistema ICE, hemos optado por una aproximación basada en el concepto de entorno de unificación. Dado que la estrategia de evaluación es ascendente, no es necesario recurrir a la estructura de compartición de capas. Siguiendo la terminología de [180], definiremos el conjunto $\mathcal{S}$ de *esqueletos* como un subconjunto del álgebra de términos lógicos $T_\Delta[\mathcal{F} \cup \mathcal{X}]$. Así, cada término lógico queda definido por un esqueleto y por un entorno aplicado sobre él. Como ya hemos dicho, de esta forma posponemos la instanciación efectiva de los términos, evitando las operaciones de copia.

Para facilitar su tratamiento, los entornos se almacenan en memoria como un vector de referencias. Por su parte el esqueleto del término, que representamos como un

[1] Habitualmente, las *estructuras de rasgos* [31] también se representan como grafos, por lo que existen resultados comunes en lo que respecta a sus algoritmos de unificación.

[2] Es decir, una sustitución.

árbol, para su almacenamiento en memoria se transforma en un vector de celdas. Para esta transformación realizamos un recorrido en preorden del árbol y por cada nodo almacenamos la siguiente información:

- Si es una función, el símbolo de funtor y su aridad. Nótese que las constantes son funciones de aridad 0. En el caso de las funciones no triviales, necesitamos almacenar una referencia a su siguiente nodo hermano en el recorrido del árbol.
- Si es una variable lógica, un índice en el vector de entorno. De esta forma podemos acceder directamente al valor asociado a la variable.

EJEMPLO 7.1
La figura 7.2 muestra un ejemplo de representación en memoria de dos términos: $g(b, s(c, a, a), b)$ *y* $f(a, b)$. *El segundo de ellos no contiene variables por lo que hemos omitido el entorno. La figura también muestra tanto la compartición de estructuras dada por* $g(X, s(c, N, N), Y)$. $\{N/a\}$, *como la compartición de estructuras ocultas. En este caso ambos términos comparten en memoria la constante b.* ■

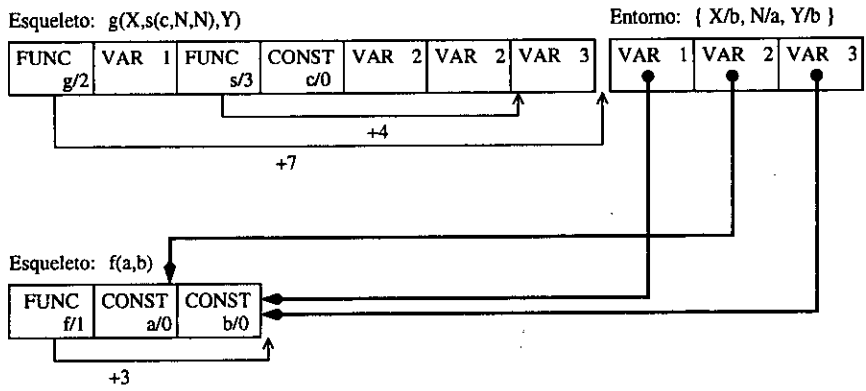


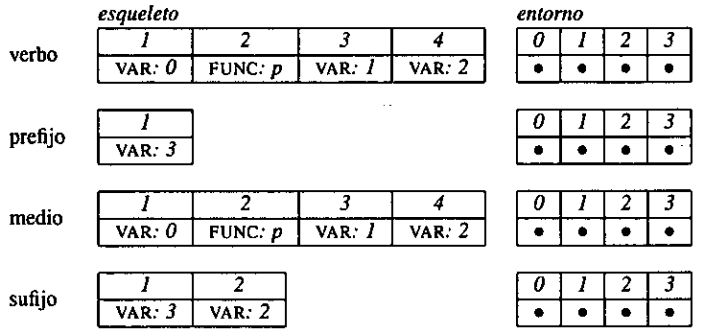
Figura 7.2: Ejemplo de almacenamiento en memoria y compartición de estructuras

Por último, hemos de realizar algunas consideraciones adicionales sobre el entorno y su uso. En primer lugar, tal y como decíamos anteriormente, en los ALPs que constituyen la base del formalismo operacional de ICE, las sustituciones resultantes de aplicar las transiciones del autómata se almacenan en la pila junto con el término implicado, tal y como mostrábamos en el ejemplo 6.7. En lugar de emplear una estructura de datos adicional para ellas, ampliaremos el entorno del término. Esta ampliación no supone una sobrecarga importante en el espacio de almacenamiento, puesto que las variables son locales a las cláusulas y por tanto podemos obviar todas aquellas que no ocurran en ninguno de los símbolos de la cláusula implicada. Para evitar la necesidad de reordenar las variables, los entornos son idénticos para todos los términos de la cláusula.

EJEMPLO 7.2
Supongamos la cláusula:

verbo(V,p(G,N)) → prefijo(T), medio(V,p(G,N)), sufijo(T,N)

Los entornos asociados a los términos que ocurren en la cláusula contendrán espacio para cuatro variables: V, T, G, y N, tal y como se muestra a continuación:

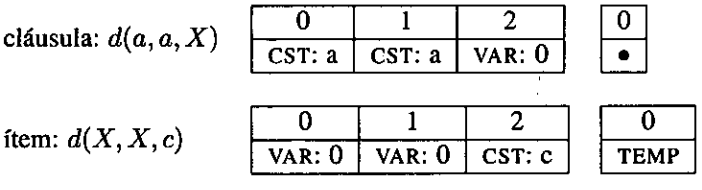


■

En segundo lugar, un término puede contener variables sin instanciar. Estas variables influyen en el proceso de unificación. Supongamos como ejemplo la cláusula

$$a(X) \rightarrow d(a, b, X), \ldots$$

y un ítem cuyo término es $d(X, X, c)$. Recordemos que las variables son locales a las cláusulas, por tanto aunque nos encontramos con dos variables con el mismo nombre, X, éstas son distintas y el algoritmo de unificación debe tenerlo en cuenta, para lo cual es necesario renombrar las variables implicadas. A continuación se muestra la representación en memoria de los dos términos.



Como se observa, en el caso de $d(X, X, c)$ la variable X no ha sido instanciada y por tanto en el entorno correspondiente se ha marcado como temporal. Para todos los efectos, el algoritmo de unificación, trata las variables temporales como si se hubiesen añadido al final de los entornos implicados en la unificación, lo que equivale a su renombramiento automático.

7.2 Estructuras cíclicas

En general, distinguiremos dos tipos de estructuras cíclicas dentro de los términos lógicos en una GCD:

- Aquellas producidas por la operación de unificación, a las que nos referiremos como *términos cíclicos*. La parte izquierda de la figura 7.3 muestra un ejemplo.

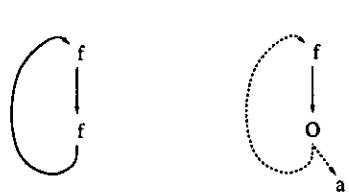


Figura 7.3: Ejemplos de estructuras cíclicas

Como característica de este tipo de términos, destacaremos que se contienen a si mismos. La aparición de términos cíclicos afecta a la terminación de la operación de unificación tal y como veremos a continuación.

- Aquellas producidas por una secuencia infinita de derivaciones, a las que nos referiremos como *derivaciones cíclicas* o *términos infinitos*. Constituyen un conjunto infinito de términos simples, y para su representación hemos optado por un grafo Y/O dirigido. La parte derecha de la figura 7.3 muestra un ejemplo para:

$$\{f(a), f(f(a)), f(f(f(a))), \ldots\}$$

La derivación de este tipo de términos afecta a la terminación del proceso de análisis, tal y como vimos en el capítulo 6.

7.2.1 Términos cíclicos

Como hemos dicho, estos términos se contienen a si mismos, creando una estructura cíclica. Generalmente, resultan de la unificación de dos términos cuando una variable X se asocia a un término que contiene dicha variable. El ejemplo de la figura 7.3 podría haber sido producido por una unificación como la siguiente:

$$\text{umg}(a(X, f(X)), a(Y, Y))$$

Un algoritmo de unificación convencional podría entrar en un bucle infinito al intentar recorrer o unificar términos cíclicos.

Una primera solución al problema es evitar que se produzcan este tipo de términos como resultado de la unificación. Para ello es necesario incorporar un *test de ocurrencia*[3], que antes de asignar $X \leftarrow t$, compruebe que la variable X no está incluida en el término t.

DEFINICIÓN 7.1 (test de ocurrencia)
Dada una GCD $\mathcal{D} = (V_T, V_N, \mathcal{V}, \mathcal{F}, \gamma, \overset{\circ}{\mathcal{P}})$ *definiremos el* test de ocurrencia *como la función:*

$$ocurre: \quad \mathcal{V} \times T_{\mathcal{P}}[\mathcal{F} \cup \mathcal{V}] \longrightarrow \{verdadero, falso\}$$

$$ocurre(X, X) = verdadero, X \in \mathcal{V}$$
$$ocurre(X, Y) = falso, X \neq Y,\ X, Y \in \mathcal{V}$$
$$ocurre(X, f(t_1, \ldots, t_n)) = ocurre(X, t_1)\ \acute{o}\ \ldots \acute{o}\ ocurre(X, t_n) \qquad \square$$

[3]En terminología anglosajona, *occur-check*

El algoritmo 7.2 muestra el pseudocódigo para una versión simplificada del algoritmo de unificación con test de ocurrencia.

ALGORITMO 7.2 (unificación con test de ocurrencia)
Entrada: Dos términos t_1 y t_2

Salida: El umg de t_1 y t_2, o `fallo` si no son unificables

Pasos:

1. Inicialización: `unificador={}`
2. Si t_1 es una variable, y t_2 no contiene a t_1, entonces `unificador=`$\{t_1/t_2\}$, terminar
3. Si t_2 es una variable, y t_1 no contiene a t_2, entonces `unificador=`$\{t_2/t_1\}$, terminar
4. Si $t_1 = t_2$, entonces terminar
5. Si $t_1 = f(a_1, \ldots, a_n)$ y $t_2 = f(b_1, \ldots, b_n)$, entonces por cada a_i y b_i `unificar(`a_i,b_i`)`
6. En caso contrario, `fallo` □

Sin embargo, los intérpretes y analizadores convencionales no implementan el test de ocurrencia [17] por ser una operación costosa en tiempo debido a su aplicación continua en el proceso de resolución lógica. Una primera solución consiste en una implementación alternativa que realice de forma eficiente dicho test [122]. No obstante, si bien se evita la creación de términos cíclicos, no se posibilita su uso.

En nuestro caso optaremos por la adaptación del algoritmo de unificación para el manejo de términos cíclicos, basándonos en el trabajo de Filgueiras [51]. La idea es tratar los funtores de forma análoga al tratamiento de las variables, esto es, una vez comprobada la igualdad de dos funtores, se crea una referencia entre ellos, y, por otro lado, antes de comprobar dicha igualdad, se siguen las referencias que puedan existir. Esta aproximación es equivalente a la adoptada en [144, 186, 71] mediante el uso de punteros hacia adelante en la unificación de estructuras de rasgos.

ALGORITMO 7.3 (unificación con términos cíclicos)
Entrada: Dos términos t_1 y t_2

Salida: El umg de t_1 y t_2, o `fallo` si no son unificables

Pasos:

1. Inicialización: `unificador={}`
2. Si t_1 es una variable, entonces `unificador=`$\{t_1/t_2\}$, terminar
3. Si t_2 es una variable, entonces `unificador=`$\{t_2/t_1\}$, terminar

4. Si t_1 contiene una referencia,
   entonces t_1=`resolver-referencia`(t_1)
5. Si t_2 contiene una referencia,
   entonces t_2=`resolver-referencia`(t_2)
6. Si $t_1 = t_2$, entonces terminar
7. Si $t_1 = f(a_1, \ldots, a_n)$ y $t_2 = f(b_1, \ldots, b_n)$,
   entonces `establecer-referencia`($t_1 \rightarrow t_2$), y
   por cada a_i y b_i `unificar`(a_i,b_i)
8. En caso contrario, `fallo` □

A continuación mostramos el funcionamiento del algoritmo mediante un ejemplo.

EJEMPLO 7.3

Para ilustrar las ideas expuestas, desarrollaremos la unificación de los dos términos cíclicos $f^*()$ y $f(f^*())$ que se muestran en la figura 7.4. En el resto del ejemplo, emplearemos arcos dirigidos para denotar las referencias y etiquetaremos los funtores con un número de secuencia que nos permita seguir la evolución del proceso.

Al comienzo del algoritmo, t_1=f-1 y t_2=f-2. Ambos términos son funciones con el mismo funtor y aridad por lo que añadimos a t_1 una referencia a t_2, y continuamos con la unificación de sus argumentos. La situación actual se muestra en el paso 1 de la figura 7.4.

A continuación, t_1=f-1 y t_2=f-3, sin embargo, t_1 contiene una referencia a f-2, por lo que actualizamos t_1=f-2. Tal y como ocurría en el paso anterior, ambos términos son compatibles, así que creamos la referencia correspondiente y continuamos con la unificación de sus argumentos. El paso 2 de la figura 7.4 se corresponde con la situación actual.

Finalmente, t_1=f-3 y t_2=f-2. Tras resolver la referencia de f-2 a f-3, actualizamos t_2=f-3, con lo cual tenemos que $t_1 = t_2$ y la unificación termina. ■

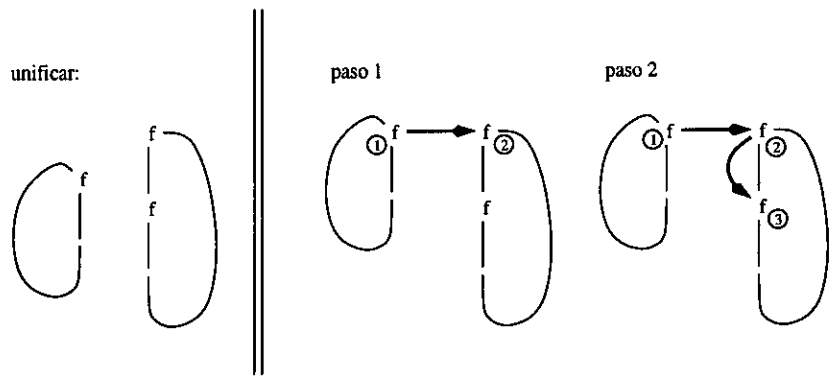


Figura 7.4: Unificación de términos cíclicos

Una vez visto el problema de los términos cíclicos, continuamos con el tratamiento de las derivaciones cíclicas, que requieren una aproximación distinta.

7.2.2 Derivaciones cíclicas

Como ya hemos dicho, es posible que el proceso de análisis produzca una secuencia infinita de derivaciones como la del ejemplo que mostrábamos en el capítulo 6 para el SDG de Earley, figura 6.1:

$$
\begin{array}{l}
[s \rightarrow \bullet r(0,N),0,0] \\
[r(0,N) \rightarrow \bullet r(s(0),N)\ b,0,0] \\
[r(s(0),N) \rightarrow \bullet r(s(s(0)),N)\ b,0,0] \\
[r(s(s(0)),N) \rightarrow \bullet r(s(s(s(0))),N)\ b,0,0] \\
\ldots
\end{array}
$$

Al contrario que el caso anterior, en este tipo de estructura cíclica se crea un conjunto infinito de términos. A pesar de este carácter infinito, se puede representar mediante una GIC, que, a su vez, como hemos indicado anteriormente, representaremos como un grafo Y/O dirigido tal y como mostrábamos en la parte derecha de la figura 7.3. Opcionalmente usaremos la siguiente representación:

$$f^1([a|^1]) = \{f(a), f(f(a)), f(f(f(a))), \ldots\}$$

Donde los superíndices constituyen arcos del grafo y la construcción [|] representa una alternativa o nodo O.

Dado que es posible que se produzcan este tipo de derivaciones durante el análisis, resulta conveniente establecer los mecanismos necesarios para su detección. Esta detección se remite al hecho de que en una derivación cíclica el proceso de análisis vuelve repetidamente al mismo estado, sin avanzar en el reconocimiento de la cadena de entrada. En el caso del sistema ICE para GICs es suficiente con comprobar que cada ítem generado sea distinto de los anteriores. Gracias a la forma en que se aplicó el mecanismo de tabulación, en combinación con el mecanismo de sincronización, la mera inserción de un ítem en la tabla nos asegura esta condición. Sin embargo, para el caso de las GCDs, la redundancia de los nuevos items se comprueba mediante la condición de admisibilidad fuerte. Esta condición es equivalente al *bloqueo*[4] en el algoritmo de deducción de Earley [105] y contribuye a evitar, en parte, la creación de derivaciones cíclicas, sin embargo no evita ejemplos como el visto al comienzo de esta sección. En efecto, en la secuencia de items dada:

$$s^1([0|^1]) = \{s(0), s(s(0)), s(s(s(0))), \ldots\}$$

ningún término subsume a otro. De forma análoga, la condición de admisibilidad débil no es suficiente para detectar este tipo de derivaciones cíclicas.

Detección y representación de ciclos

La detección de derivaciones cíclicas revierte en comprobar si el proceso de análisis está volviendo continuamente al mismo estado [122]. De la definición de esqueleto IC

[4] *Blocking* en la terminología original.

se sigue inmediatamente que para cualquier derivación cíclica sobre una GCD también existe una derivación sobre su esqueleto IC:

$$\text{derivación GCD} \Rightarrow \text{derivación esqueleto IC}$$

Por lo tanto, cualquier comprobación sobre los términos de la gramática se pospone mientras no se haya detectado un ciclo sobre el esqueleto IC. Las ventajas de esta aproximación son las siguientes:

- Se facilita la detección de derivaciones cíclicas.
- Se reduce el número de comprobaciones, mejorando la eficiencia.

Una vez detectado un ciclo en el esqueleto IC, se cotejan los términos creados al comienzo y final del ciclo, comprobando si el primero está incluido en el segundo. En caso afirmativo, se ha producido una vuelta al mismo estado del análisis y, por lo tanto, hemos detectado una derivación cíclica en la GCD. A destacar:

- El ciclo en el esqueleto IC implica que hemos obtenido dos items con el mismo estado del control finito, el mismo símbolo de predicado, y analizando la misma porción de la cadena de entrada.
- La comprobación de la inclusión de un término en otro viene facilitada por la compartición de estructuras descrita en apartados anteriores.

EJEMPLO 7.4
Como ejemplo supondremos la siguiente GCD:

$$a(nil) \rightarrow \text{“}b\text{”} \qquad a(f(X)) \rightarrow a(X)$$

cuyo esqueleto IC es:

$$a \rightarrow \text{“}b\text{”} \qquad a \rightarrow a$$

El análisis de la cadena de entrada “b” produciría la siguiente secuencia de términos:

$$a(f^1([nil|^1])) = \{a(f(nil)), a(f(f(nil))), a(f(f(f(nil)))), \ldots\}$$

Sin embargo, tras crear los items $[a(f(nil)), st_2, 1, 1]$, *y* $[a(f(f(nil))), st_2, 1, 1]$, *se detecta un ciclo en el esqueleto IC, dado por la repetición del ítem:*

$$[a, st_2, 1, 1]$$

A continuación se recorren los términos correspondientes, que se muestran en la figura 7.5, concluyendo la detección del ciclo. ■

Seguidamente desarrollaremos un ejemplo en el que mostraremos no sólo los términos implicados en la detección de la derivación cíclica, sino todas las configuraciones implicadas en el proceso de análisis.

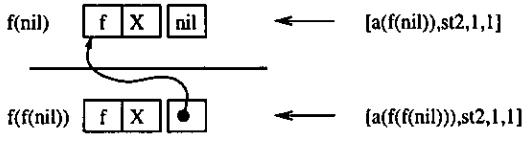


Figura 7.5: $f(nil)$ y $f(f(nil))$ después de un ciclo en el esqueleto IC

EJEMPLO 7.5
Para el presente ejemplo consideraremos una GCD para la secuencialización de nombres en inglés, como en el caso de "North Atlantic Treaty Organization". *Las cláusulas son las siguientes:*

$\gamma_1 : s(X) \rightarrow fn(X).$ $\quad \gamma_2 : fn(fn(X,Y)) \rightarrow fn(X)\, fn(Y).$
$\gamma_3 : fn(X) \rightarrow nombre(X).$ $\quad \gamma_4 : fn(nil).$

En este caso, el esqueleto IC viene dado por las producciones:

(0) $\Phi \rightarrow S \dashv$ $\quad$ (1) $S \rightarrow FN$ $\quad$ (2) $FN \rightarrow FN\ FN$
(3) $FN \rightarrow nombre$ $\quad$ (4) $FN \rightarrow \varepsilon$

cuya máquina de estados finita característica mostramos en la figura 7.6.

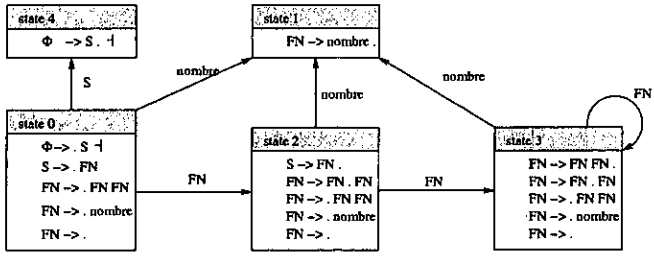


Figura 7.6: Máquina de estados característica para la gramática de ejemplo

Describiremos el proceso de análisis para la cadena de entrada "North Atlantic", *centrándose en la evolución de las configuraciones del autómata. A partir del símbolo inicial* \$, *y teniendo en cuenta que el autómata está en el estado inicial 0, la primera acción es el desplazamiento de la palabra* "North", *que implica el apilamiento del ítem* $[nombre(\text{“}North\text{”}), 0, 1, st_1]$, *que indica el reconocimiento del término* $nombre(\text{“}North\text{”})$ *entre las posiciones 0 y 1 de la cadena de entrada, siendo el estado 1 el estado del control estático. Esta configuración se muestra en la figura 7.7.*

$$[\$, 0, 0, st_0] \vdash \begin{array}{|c|}\hline [nombre(\text{“}North\text{”}), 1, 0, st_1] \\ \hline [\$, 0, 0, st_0] \\ \hline \end{array}$$

Figura 7.7: Configuraciones durante el reconocimiento de "North"

Llegados a este punto, podemos aplicar las transiciones [*Selección*], [*Reducción*], *y* [*FinReducción*] *para reducir la cláusula* γ_3. *Las configuraciones resultantes se muestran en la figura 7.8.*

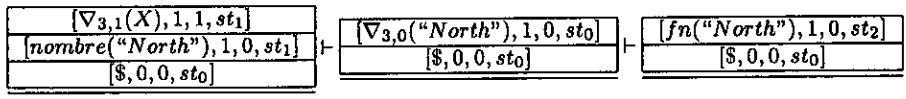


Figura 7.8: Configuración durante la reducción de la cláusula γ_3

Ahora podemos desplazar la palabra "Atlantic", *como resultado reconocemos el término* $nombre("Atlantic")$ *entre las posiciones 1 y 2 de la cadena de entrada, con el control LALR(1) en el estado 1. Como en el caso de la palabra anterior, podemos reducir la cláusula* γ_3. *Este proceso se desglosa en la figura 7.9.*

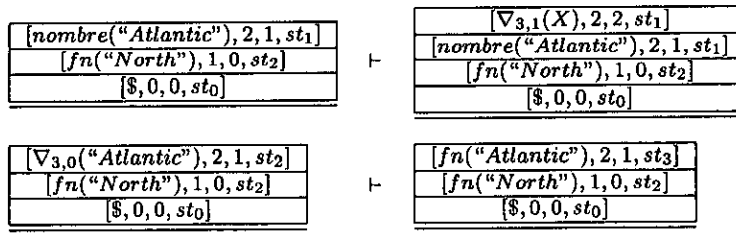


Figura 7.9: Configuraciones durante el procesamiento de la palabra "Atlantic"

Después de reconocer dos predicados fn, *podemos reducir la cláusula* γ_2 *para obtener un nuevo predicado* fn *que representa el sintagma nominal* "North Atlantic". *Esta reducción se muestra en la figura 7.10. El reconocimiento de la cadena de entrada completa finaliza con la reducción de la cláusula* γ_1, *obteniendo el término*

$$s(fn(fn("North", "Atlantic")))$$

que constituye una representación abstracta del análisis de "North Atlantic". *El estado del control LALR(1) será 4, que es un estado final, lo cual significa que el análisis de esta rama ha concluido. Las configuraciones resultantes se desglosan en la figura 7.11.*

Sin embargo, la gramática define un número infinito de posibles análisis para cada cadena de entrada. Si observamos el autómata LALR(1), en los estados 0, 2 y 3, además de las acciones realizadas, siempre podemos reducir la cláusula γ_4, *que tiene una parte derecha vacía. En particular, en el estado 3 el predicado* fn *se puede generar un número indefinido de veces sin avanzar en el reconocimiento de la cadena de entrada, tal y como se muestra en la figura 7.12. En dicha figura, la parte izquierda representa la derivación cíclica en el esqueleto IC, la parte central, el proceso de análisis de la GCD en el estado 3, y la parte derecha la representación finita del recorrido del término infinito. Las cajas representan el reconocimiento de una categoría gramatical en el estado dado del control LALR(1).* ■

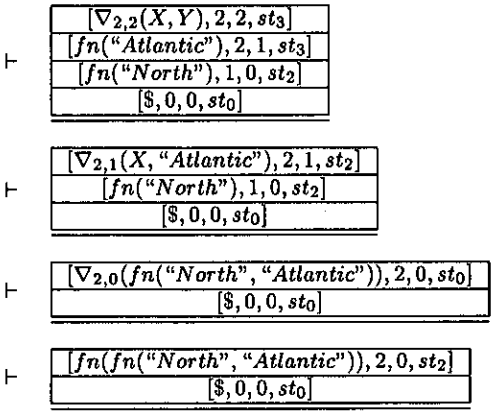


Figura 7.10: Reconocimiento del sintagma nominal "North Atlantic"

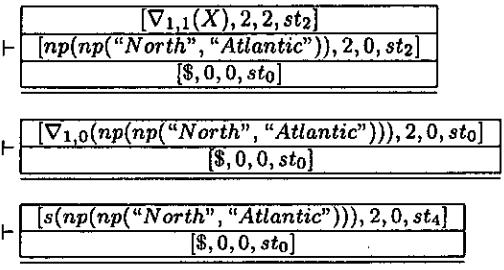


Figura 7.11: Configuraciones en el reconocimiento de "North Atlantic"

Tal y como mostrábamos en el ejemplo 7.5, una vez detectado un ciclo en el esqueleto IC, procedemos a comprobar la existencia de un ciclo en la GCD. Sobre este aspecto, resaltaremos el hecho de que, tal y como hemos definido el esqueleto IC, esta situación implica la existencia de dos items con el mismo símbolo de predicado y la misma aridad. En consecuencia sólo es necesario examinar sus argumentos en busca de bucles.

Para comprobar la existencia de bucles verificaremos los argumentos, uno por uno, comprobando la existencia de términos repetidos. Gracias a la compartición de estructuras, esta comprobación se limita a una comparición de referencias.

Volviendo a la figura 7.12 en el ejemplo 7.5, una vez detectado el bucle independiente del contexto, comprobamos posibles derivaciones cíclicas en la GCD original. Tal y como indicábamos, la parte central de la figura muestra la familia de términos:

$$fn(nil), fn(fn(nil,nil)), fn(fn(fn(nil,nil),nil)), \ldots, fn(fn^1([nil|^1],nil))$$

De forma análoga, también podríamos generar:

$$fn(nil), fn(fn(nil,nil)), fn(fn(nil,fn(nil,nil))), \ldots, fn(fn^1(nil,[nil|^1]))$$

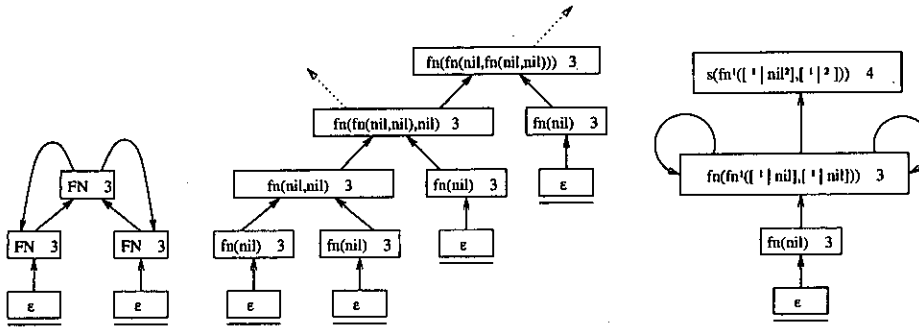


Figura 7.12: Derivación cíclica en la gramática de ejemplo

En ambos casos, los términos son resultado de la aplicación sucesiva de las cláusulas γ_2 y γ_4, y cada uno de los términos se construye a partir de los anteriores. De hecho, también podemos construir los términos de la segunda sucesión a partir de los de la primera, produciendo el término[5] de la parte derecha de la figura:

$$fn(fn^1([nil^2|^1],[^2|^1]))$$

Sobre la detección de la derivación cíclica en los términos, destacaremos que la existencia de un bucle sobre el esqueleto IC implica un bucle en la construcción de los términos. En efecto, si consideramos la representación[6] de términos como GICs, propuesta al comienzo del presente apartado, y los términos del ejemplo 7.5:

- Para $fn(fn(nil, nil))$ tendremos las producciones:

$$\begin{array}{ll} fn(X,Y): & r_1 \rightarrow r_2\ r_3 \\ (X/nil): & r_2 \rightarrow nil \\ (Y/nil): & r_3 \rightarrow nil \end{array}$$

- Para $fn(fn(fn(nil, nil)))$ tendremos las producciones:

$$fn(X,Y): \quad r_4 \rightarrow r_1\ r_3$$

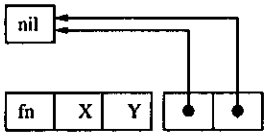
Sobre los términos anteriores consideraremos que

- Se repite la misma instanciación dentro del bucle.
- Dentro de dicha repetición cada nuevo término se construye a partir del anterior.

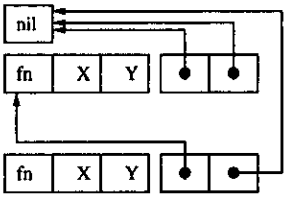
[5]Nótese que podríamos unir las estructuras $fn(nil)$ y $fn(fn^1([nil^2|^1],[^2|^1]))$ en $fn(^1[nil|fn(^{1,1})])$, pero esto requeriría un tratamiento adicional no trivial.

[6]Esta representación es análoga a la empleada para la construcción del bosque de análisis en ICE.

Reducción γ_2:
$t_1 \equiv fn(X,Y) \cdot \{X/nil^2, Y/nil^2\}$



Reducción γ_2:
$fn(X,Y) \cdot \{X/t_1, Y/nil^2\}$



$t_3 \equiv fn(X,Y) \cdot \{X/fn^1([nil^2|^1], nil^2), Y/nil^2\}$

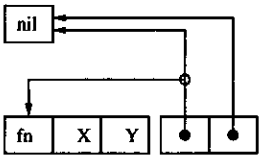


Figura 7.13: Creación de estructuras cíclicas (1/2)

Por tanto, este proceso se puede repetir un número ilimitado de veces, dando lugar a términos infinitos de la forma $fn(fn^1([nil|^1], nil))$. La figura 7.13 muestra la representación en memoria de los términos implicados en el ejemplo. El segundo término se obtiene tras aplicar un paso de unificación sobre el primero, t_1, tal y como muestra la compartición de estructuras. Finalmente, el tercer término, t_3, de la figura es el resultado de representar la derivación cíclica detectada. A continuación, la figura 7.14 muestra la representación en memoria de los términos cíclicos construidos de forma análoga a partir de t_3.

7.2.3 Unificación y subsumción con términos cíclicos

Hasta el momento hemos tratado la detección y representación de estructuras cíclicas. Además, hemos de considerar cómo estas estructuras afectan a la relación de subsumción y a la operación de unificación.

En general, una función subsume otra función si ambas son compatibles, mismo funtor y misma aridad, y cada uno de sus argumentos, o bien son iguales, o bien cumplen la relación de subsumción. Sin embargo, cuando tratamos estructuras cíclicas dichos argumentos pueden estar constituidos por una alternativa. Suponiendo que queremos comprobar si $t_1 \preceq t_2$:

1. Ni t_1 ni t_2 presentan alternativas. La relación de subsumción no cambia.
2. t_1 presenta varias alternativas y t_2 no. t_1 subsume a t_2 si lo hace al menos una de sus alternativas.
3. t_2 presenta varias alternativas y t_1 no. t_1 subsume a t_2 si subsume todas sus alternativas.

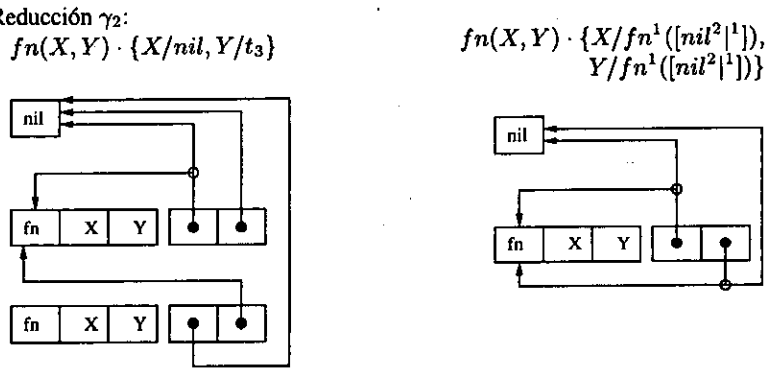


Figura 7.14: Creación de estructuras cíclicas (2/2)

4. t_1 y t_2 presentan varias alternativas. Este caso es una combinación de los dos anteriores, t_1 subsume a t_2 si al menos una de sus alternativas subsume todas las de t_2.

A continuación presentamos el algoritmo de subsumción resultante, definido de forma recurrente.

ALGORITMO 7.4 (subsumción)
Entrada: Dos términos lógicos t_1 y t_2

Salida: `Verdadero` si $t_1 \preceq t_2$, `falso` en caso contrario son unificables

Pasos:

1. Inicialización:Pila=$\emptyset$
2. Añadir t_1, t_2 a la Pila
3. **mientras** Pila$\neq \emptyset$
   (a) Extraer X, Y de la Pila
   (b) **caso**

esVariable(X):	**continuar**
esVariable(Y):	**devolver** `falso`
alternativas(X):	**devolver** $X_1 \preceq Y$ **o** ... **o** $X_n \preceq Y$
alternativas(Y):	**devolver** $X \preceq Y_1$ **y** ... **y** $X \preceq Y_n$
$X \leftarrow f(X_1, \ldots, X_n)$ y $Y \leftarrow f(Y_1, \ldots, Y_n)$:	**devolver** $X_1 \preceq Y_1$ **y** ... **y** $X_n \preceq Y_n$
sino :	**devolver** `falso`

4. **devolver** `verdadero` □

A continuación mostramos un ejemplo.

EJEMPLO 7.6
Volviendo al ejemplo de la figura 7.13, podemos concluir que $fn^1([nil|^1],^1)$ *subsume* $fn^1([nil|^1], nil)$. *El funtor y la aridad,* $fn/2$, *coinciden, al igual que el primer argumento. En lo que respecta al segundo,* $[nil|^1] \preceq nil$, *ya que para la primera alternativa se cumple* $nil \preceq nil$. ■

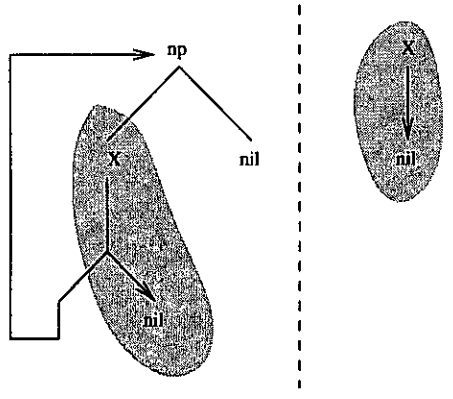


Figura 7.15: umg con estructuras cíclicas

En relación a la operación de unificación, el algoritmo para el cálculo del umg debe considerar todas las alternativas presentes en una estructura cíclica, descartando aquellas que no unifican. Por lo tanto:

$$\text{umg}(\{Y \leftarrow [a|b]\}, \{Y \leftarrow a\}) = \{Y \leftarrow a\}$$

y consecuentemente, continuando con el ejemplo anterior:

$$\text{umg}(fn(X, X), fn^1([nil|^1], nil)) = \{X \leftarrow \text{nil}\}$$

Dicha unificación está ilustrada en la figura 7.15, donde las alternativas que unifican están sombreadas. Por último, no debemos olvidar que las variables son los términos más generales y por lo tanto subsumen cualquier otro, y unifican, también, con cualquier otro, incluidas las alternativas en los términos cíclicos. A modo de ejemplo:

$$\text{mgu}(fn(X), fn^1([a|^1])) = \{X \leftarrow [a|fn^1([a|^1])\}$$

7.2.4 Ámbito de aplicación

Este apartado tiene un sentido práctico ya que, como se ha establecido [105], las GCDs en presencia de símbolos funcionales son sólo semi-decidibles. La propuesta de tratamiento de estructuras cíclicas presentada es capaz de detectar ciclos con una estructura sintáctica regular. En el peor de los casos, es decir en presencia de otro tipo de ciclos, es equivalente a una estrategia de evaluación clásica. Obviaremos la extensión de la técnica descrita para el tratamiento de estructuras más complejas, debido a la dificultad para representar información útil de estructuras no regulares. Informalmente, las estructuras regulares consideradas son las siguientes:

- *Términos conjuntivos.* En el caso más sencillo, todos los términos incluidos en el ciclo se generan a partir de cláusulas que no comparten la parte derecha. Esta es la situación del siguiente ejemplo:

$$\gamma_1 : a(nil) \rightarrow \epsilon \qquad \gamma_2 : a(g(X)) \rightarrow b(X) \qquad \gamma_3 : b(f(X)) \rightarrow a(X)$$

Las figuras 7.16 y 7.17 muestran la evolución del analizador en la detección del ciclo. Finalmente el término producido es: $a(g^1(f([nil|^1])))$

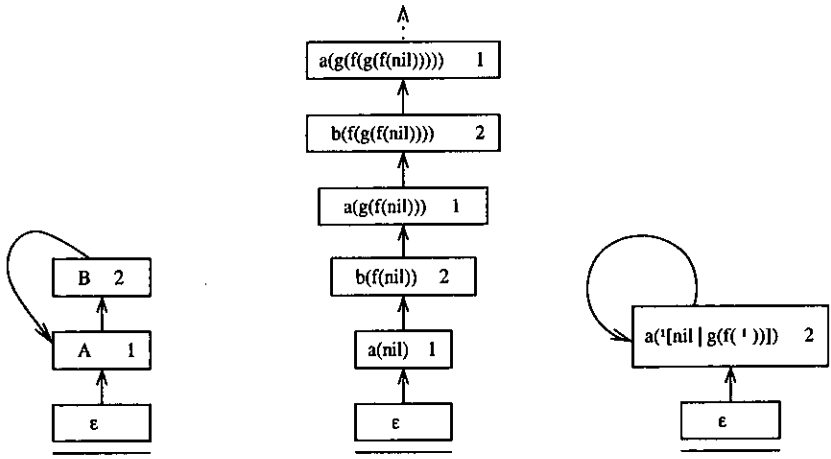


Figura 7.16: Derivación cíclica en un contexto conjuntivo

- *Términos disyuntivos.* En esta ocasión, los términos incluidos en el ciclo se generan a partir de cláusulas que sí comparten su parte derecha, como en el siguiente ejemplo:

$$\gamma_1 : a(nil) \rightarrow \epsilon \qquad \gamma_2 : a(f(X)) \rightarrow a(X) \qquad \gamma_3 : a(g(X)) \rightarrow a(X)$$

Las figuras 7.18, 7.19 y 7.21 muestran la evolución del analizador en la detección del ciclo. Para evitar una extensión excesiva, las dos últimas figuras referenciadas sólo muestran los primeros pasos de la detección de ciclos. Finalmente los términos producidos son:

$$a(g^1[nil|[f(^1)|g(^1)]]) \quad \text{y} \quad a(f^1[nil|[g(^1)|f(^1)]])$$

- *Estructuras infinitas no cíclicas.* Es el caso del siguiente ejemplo:

$$\gamma_1 : a(nil) \rightarrow \epsilon \qquad \gamma_2 : a(f(Y, X)) \rightarrow a(X)$$

donde en cada reducción de la cláusula γ_2 se crea una nueva instancia de la variable Y. Como consecuencia, se genera una sucesión infinita de términos

$t_1 \equiv g(f(X)) \cdot \{X/nil\}$ $\quad g(f(X)) \cdot \{X/t_1\}$

$t_3 \equiv g(f(X)) \cdot \{X/g^1(f([nil|^1]))\}$

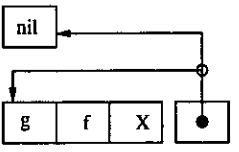


Figura 7.17: Creación de estructuras cíclicas en un contexto conjuntivo

que no siguen una estructura regular. Sin embargo, considerando la variable Y como una variable anónima sí se puede aplicar la detección de estructuras cíclicas propuesta, dando lugar al término:

$$a(f^1(\_, [^1|nil]))$$

Las figuras 7.20 y 7.22 muestran la evolución del analizador en la detección del ciclo.

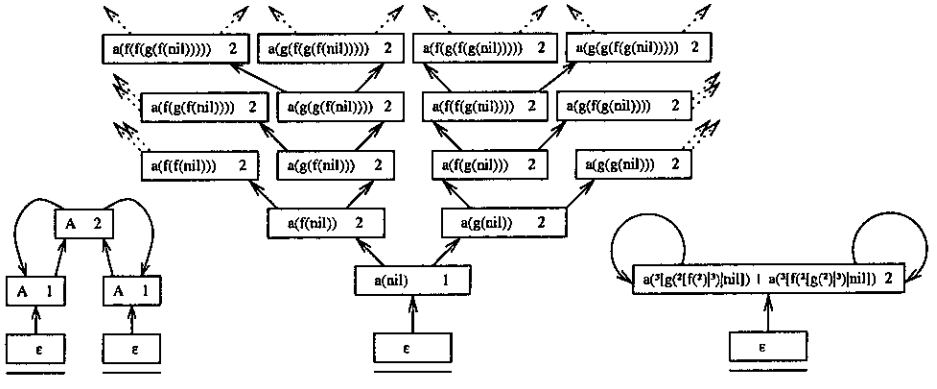


Figura 7.18: Derivación cíclica en un contexto disyuntivo

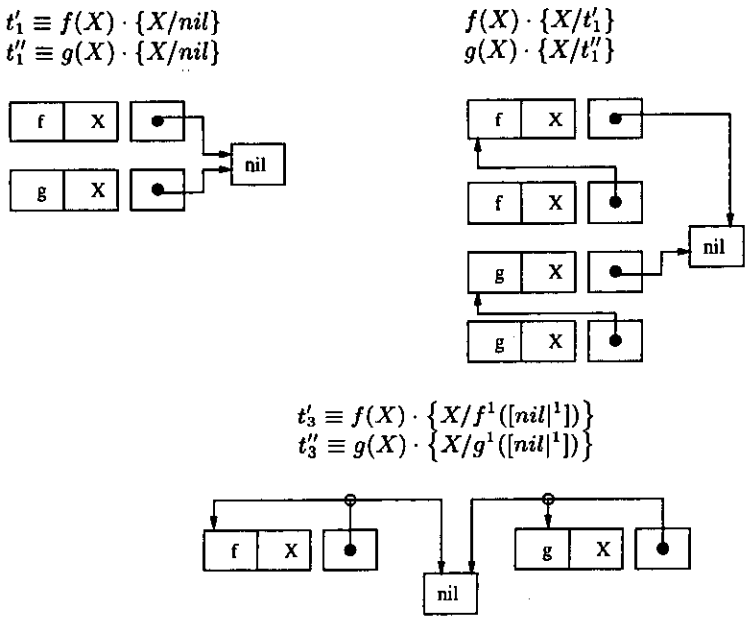


Figura 7.19: Creación de estructuras cíclicas en un contexto disyuntivo (1/2)

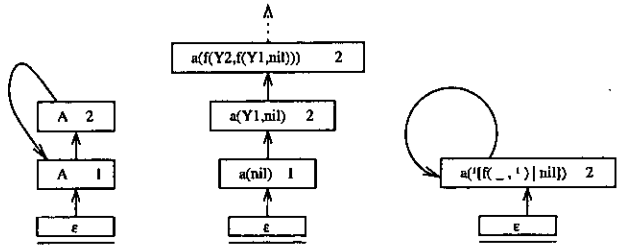
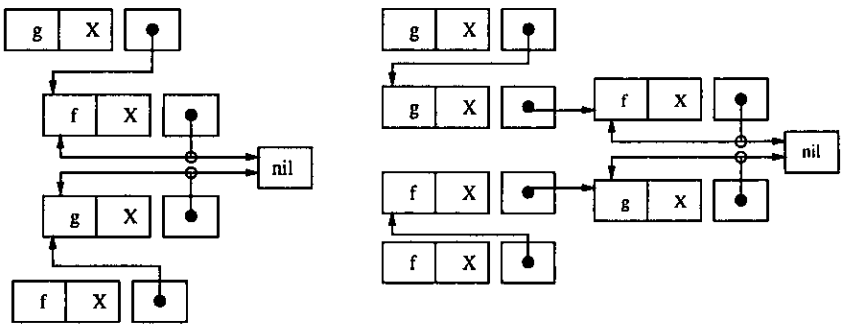


Figura 7.20: Una estructura infinita

$s_1' \equiv f(X) \cdot \{X/t_3''\}$
$s_1'' \equiv g(X) \cdot \{X/t_3'\}$

$f(X) \cdot \{X/s_1'\}$
$g(X) \cdot \{X/s_1''\}$



$s_3' \equiv f(X) \cdot \{X/f^1([t_3''|^1])\}$
$s_3'' \equiv g(X) \cdot \{X/g^1([t_3'|^1])\}$

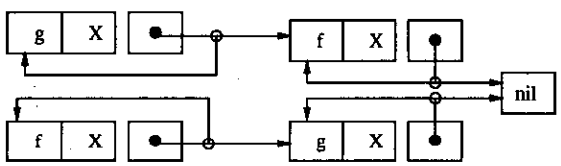
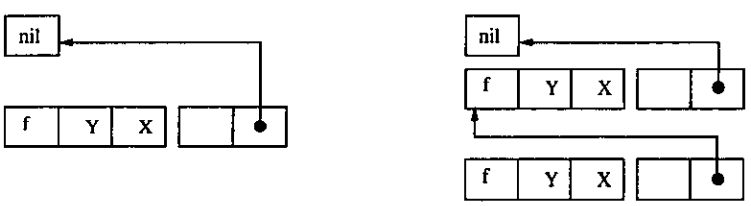


Figura 7.21: Creación de estructuras cíclicas en un contexto disyuntivo (2/2)

$t_1 \equiv f(Y, X) \cdot \{X/nil\}$

$f(Y, X) \cdot \{X/t_1\}$



$t_3 \equiv f(Y, X)) \cdot \{X/f^1(\_, [nil|^1])\}$

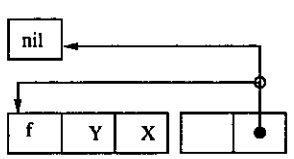


Figura 7.22: Creación de estructuras cíclicas en una estructura infinita

CAPÍTULO 8

ADECUACIÓN AL ANÁLISIS DEL LENGUAJE NATURAL

El campo del análisis de las *lenguas naturales* es más complejo que el análisis de los lenguajes formales, debido a su naturaleza poco estructurada y frecuentemente ambigua, imponiendo un nivel de exigencia mayor sobre la eficacia de los analizadores y el poder descriptivo de las gramáticas. Estos problemas ya han sido tratados en capítulos anteriores, sin embargo también nos encontramos con nuevos requerimientos que debemos abordar:

- Integración con el *análisis morfológico*. En efecto, el problema del análisis a nivel morfológico es suficientemente complejo y particular como para ser tratado mediante técnicas específicas. Esta separación desemboca en la necesidad de integrar ambos tipos de analizadores.

- *Análisis sintáctico robusto*. Al contrario que ocurre con los lenguajes formales, en muchos casos no es posible conseguir o bien una cadena de entrada correcta, o bien una gramática que cubra todas las posibles cadenas de entrada. Esta situación nos obliga a realizar el análisis sintáctico en presencia de lagunas gramaticales e, incluso, errores.

8.1 Integración con el análisis morfológico

Suponiendo la cadena de entrada como un flujo de caracteres, habitualmente el proceso de análisis sintáctico viene precedido por una fase de *análisis léxico* donde dicha entrada se divide en palabras[1], a las cuales se les asigna una categoría. Estas categorías se corresponden con los símbolos terminales de la gramática empleada en el análisis sintáctico y la cadena de categorías constituye la cadena de entrada del analizador sintáctico. Esta es la aproximación adoptada, por ejemplo, en la combinación de entornos producidos por los generadores de analizadores FLEX (léxico) y BISON (sintáctico) [102, 40].

En el caso del análisis de las lenguas naturales, la fase de análisis léxico comprende al análisis morfológico, aunque se extiende a aspectos sintácticos e incluso semánticos. Como ya hemos indicado, la complejidad del análisis morfológico difiere en gran

[1] Entendiendo palabra como *token*.

medida de la que hemos encontrado en el análisis sintáctico. A modo de ejemplo, señalaremos las principales características que han dificultado la definición de reglas que describan los procesos flexivos [57] para el caso del castellano y el gallego:

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

- Gran variedad de raíces y terminaciones verbales irregulares. Hasta un 30% de los verbos se clasifican como irregulares, presentando, por ejemplo en el caso del verbo `hacer`, hasta 7 raíces diferentes (`hac-er`, `hag-o`, `hic-e`, `ha-ré`, `hiz-o`, `haz`, `hech-o`). Se han identificado hasta 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 existencia de pronombres enclíticos. Dichos pronombres pueden añadirse a las formas verbales, hasta tres en castellano y cuatro en gallego: `tráe-te-me-lo`, `déu-lle-lo`. En el caso del gallego los pronombres se pueden contraer, modificando además la raíz verbal: `vaichemo` (`me` + `o` = `mo`) `buscar`.

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

- No unicidad de la forma de participio, 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: género gramatical, género heredado, misma forma para masculino y femenino (`azul`), ...

- Análogamente, 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[2] al añadir sufijos de género y número como en `luz`, `luc-es`. En gallego, los alomorfos también aparecen debido al uso de contracciones, `a` + `os` = `ós`.

Para tratar estos problemas es deseable un formalismo compacto, y eficiente. En esta línea se ha mostrado interés por los modelos basados en AFs [74, 111, 112, 114], en modelos estadísticos [35], o en una combinación de ambos [53]. Otra aproximación

[2]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.

es la morfología de dos niveles [76, 75] y su transformación en TFs [68], incluida la emisión de probabilidades asociadas a cada análisis [93].

En conclusión, los formalismos desarrollados para el análisis morfológico difieren de los empleados en el análisis sintáctico, por lo que es necesario establecer los mecanismos necesarios para la integración de ambos. Para ello, consideraremos que el resultado del análisis morfológico se expresa en base a *etiquetas*, que pasamos a describir.

8.1.1 Etiquetas

Cada analizador emplea un conjunto finito de *etiquetas*, esto es, valores asignados a cada una de las palabras analizadas. Nos referiremos a dicho conjunto como *juego de etiquetas*[3]. Cada etiqueta es una tupla cuyos elementos denominaremos *atributos de la etiqueta*. Dichos atributos expresan información relevante de la palabra como pueda ser: género, número, tiempo verbal, ... y están compuestos por un par identificador/valor.

EJEMPLO 8.1
A continuación se muestran algunas palabras y las posibles etiquetas asociadas a cada una de ellas:

yo	*categoría:pronombre, persona:1ª, número:singular, género:neutro*
veo	*categoría:verbo, persona:1ª, número:singular, tiempo:presente*
hombre	*categoría:sustantivo, número:singular, género: masculino* ■

En general cuanto más grande es el conjunto de etiquetas considerado, existe más ambigüedad potencial. Por ejemplo, en inglés algunos juegos de etiquetas hacen una distinción entre "*to*" como preposición y "*to*" como marca de infinitivo, lo que da lugar a que la palabra "*to*" tenga dos etiquetas potencialmente correctas. Además, destacaremos que:

- No existe un juego de etiquetas universalmente aceptado, por lo que diferentes analizadores pueden emplear juegos diferentes. Como referencia indicaremos algunos juegos de etiquetas:
  - El estándar EAGLES: *Expert Advisory Group on Language Engineering Standards* es una iniciativa de la Comisión Europea iniciada en 1993 dentro del marco del programa de Investigación e Ingeniería Lingüística (LRE), y su objetivo es acelerar la provisión de estándares para la construcción de recursos lingüísticos a gran escala (*corpora* de textos y de habla, léxicos informatizados, etc.), así como para la manipulación de este conocimiento (formalismos lingüísticos, lenguajes de marcado y herramientas informáticas), y establecer mecanismos de evaluación de los recursos, herramientas y productos [30].
  - El proyecto Crater [107] tiene su propio conjunto de etiquetas, inspirado en el estándar EAGLES.

[3] En terminología anglosajona, *tagset*.

  - El corpus Susanne [120], uno de cuyos resultados es un corpus de análisis sintácticos[4] en inglés, disponible libremente, utiliza también su propio juego de etiquetas.
  - Por último, nos referiremos a los proyectos *GALENA* [172, 177, 57, 56] y ERIAL [20, 19] relacionados con el sistema ICE, que también tienen su propio juego de etiquetas inspirado en EAGLES.

- Incluso dentro del mismo juego de etiquetas, éstas no incluyen siempre la misma cantidad de información, tal y como se observa en el ejemplo 8.1.

Etiquetador vs multietiquetador

A los problemas señalados al comienzo de la sección hemos de añadir la naturaleza ambigua del análisis que estamos tratando. Así, a algunas palabras le pueden corresponder diferentes análisis, sólo discriminables en función de información sintáctica o semántica adicional.

EJEMPLO 8.2
Las siguiente etiquetas representan análisis válidos de la palabra sobre*:*

verbo	*presente de subjuntivo, tercera persona singular*
verbo	*presente de subjuntivo, primera persona singular*
sustantivo	*común, masculino singular*
preposición	

■

Cuando el analizador es capaz de deshacer esta ambigüedad, eliminando las etiquetas incorrectas en un contexto dado, nos referiremos a él como *etiquetador*. Si por el contrario, devuelve como resultado del análisis más de una[5] etiqueta por palabra, usaremos el término *multietiquetador*.

8.1.2 Integración

A continuación trataremos la integración de un etiquetador o multietiquetador con el analizador sintáctico descrito en el sistema ICE. Para ello nos centraremos en el desarrollo de dos aspectos:

- Un formalismo descriptivo de enlace que sirva de paso intermedio entre los formalismos empleados en ambos analizadores.
- Una implementación que cumpla las siguientes características:
  - Desacoplamiento del analizador sintáctico y la definición del juego de etiquetas
  - Transferencia de la información dada por los atributos de las etiquetas
  - En el caso de un multietiquetador, tratamiento de la ambigüedad no resuelta

[4] Es decir, un *treebank*.
[5] Puede devolver todas las etiquetas posibles, las que tengan mayor probabilidad de ser correctas, ...

Respecto al formalismo descriptivo, dado que para el sistema ICE hemos empleado GCDs, necesitamos establecer una correspondencia entre etiquetas y términos lógicos. Aunque el número de atributos de una etiqueta es variable, en general, todas las que presentan la misma categoría morfosintáctica contienen el mismo. Así, emplearemos dichas categorías como símbolos de predicado. Para el resto de atributos de la etiqueta, ordenados según un criterio definido[6], añadiremos sus valores como argumentos.

EJEMPLO 8.3
A continuación se muestran los términos lógicos asociados a las etiquetas del ejemplo 8.1:

yo	*categoría:pronombre, persona:1ª, número:singular, género:neutro*
	pronombre(1ª,singular,neutro)
veo	*categoría:verbo, persona:1º, número:singular, tiempo:presente*
	verbo(1ª,singular,presente)
hombre	*categoría:sustantivo, número:singular, género: masculino*
	sustantivo(singular,masculino) ■

Los términos lógicos construidos de esta forma pasarían a constituir los símbolos terminales de la GCD. Sin embargo, aún debemos reducir el grado de acoplamiento entre los analizadores morfológico y sintáctico, que se refleja en los siguientes problemas:

- Los símbolos terminales incluyen información de todos los atributos de las etiquetas, aunque dicha información sea irrelevante, tal y como ocurre en la siguiente cláusula:

  sn(Número) → pronombre(Persona,Número,Género)

- Cualquier cambio en el juego de etiquetas supone cambiar la GCD correspondiente para añadir, eliminar o reordenar los argumentos de los símbolos terminales.

Para solucionar los problemas descritos, optaremos por un formalismo similar a los *términos-ψ* [5], que se pueden entender como términos lógicos de primer orden extendidos. Dicha extensión consiste en añadir a cada argumento un identificador y un tipo, como en el siguiente ejemplo:

persona(id=>nombre, nacimiento=>fecha(día=>nombre,mes=>nombre,año=>número))

Tras lo cual, la operación de unificación se redefine convenientemente [5]. En nuestro caso obviaremos los tipos, y nos limitaremos a asociar un identificador con cada argumento. Cada identificador determina de forma unívoca uno de los atributos de la etiqueta correspondiente.

EJEMPLO 8.4
La figura 8.1 muestra la correspondencia entre dos etiquetas y los argumentos de los símbolos terminales:

indefinido(número=>N, género=>G, palabra=>P)
verbo(número=>N, palabra=>P) ■

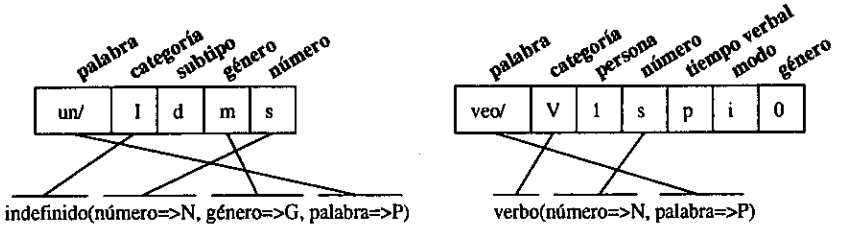


Figura 8.1: Correspondencia entre etiquetas y símbolos terminales

Una vez seleccionado un formalismo descriptivo, resta por establecer los mecanismos necesarios para su implementación.

8.1.3 Implementación

Sobre la implementación realiza para el sistema ICE, destacaremos los siguientes aspectos:

- Para conseguir el mayor desacoplamiento entre los juegos de etiquetas y las GCDs, la correspondencia entre categorías y atributos con símbolos de predicado y argumentos se realiza dinámicamente antes de comenzar el análisis sintáctico. De esta forma, un cambio en la GCD no implica un cambio en la definición de las etiquetas, y viceversa.

- Para la integración de los términos lógicos de primer orden y los términos-ψ [5], es necesario bien:

  - Modificar convenientemente la operación de unificación.
  - Transformar alguno de los dos tipos de términos.

  En nuestro caso, optaremos por transformar los términos-ψ en términos de primer orden. Para ello, se examina la gramática, y por cada símbolo de predicado se construye una lista con todos los identificadores de argumento posibles. Cada uno de estos identificadores ocupa una posición en la lista que se hace corresponder con su posición dentro de los argumentos del término de primer orden correspondiente.

EJEMPLO 8.5
Dada una GCD con las siguientes cláusulas:

sn(N) $\rightarrow$ *nombre(número=>N)*
sn(N) $\rightarrow$ *determinante(número=>N,género=>G), nombre(número=>N,género=>G)*

Los términos correspondientes a nombre y determinante serían:

[6]Por ejemplo, según su orden de aparición en la etiqueta.

nombre(N,G) y determinante(N,G) ■

Como ventaja de esta aproximación, destacaremos su sencillez, lo cual nos permite realizar la transformación de forma implícita durante el propio proceso de análisis, sin necesidad de modificar realmente la gramática. Como desventaja, en teoría el número de argumentos de los términos resultantes no está limitado. En la práctica, este número viene limitado por el número de atributos presente en el juego de etiquetas.

8.1.4 Multietiquetador

Finalmente, debemos tratar el caso en que el analizador asigna a cada palabra varios análisis posibles, es decir, varias etiquetas. La información proveniente de cada etiqueta individual será tratada tal y como hemos descrito para el caso de los etiquetadores. La diferencia estriba en el hecho de que por cada una de estas etiquetas debemos lanzar una nueva rama de análisis. La técnica a emplear será la misma que la que describiremos en el siguiente apartado para el caso de las palabras desconocidas.

8.2 Análisis sintáctico robusto

Al realizar un análisis sintáctico, es posible que la cadena de entrada contenga errores, es decir que no pertenezca al lenguaje generado por la gramática. En algunas aplicaciones, en especial el análisis de lenguas naturales, debido a los motivos que provocan dichos errores, es deseable completar en la medida de lo posible el análisis. A modo de ejemplo, destacaremos los siguientes motivos:

- Debido a su complejidad, las gramáticas no son exhaustivas y existen partes del lenguaje sin cubrir.
- Igualmente debido a su complejidad, el análisis morfológico en ocasiones o no es capaz de completar el análisis de una palabra, o le asigna un análisis incorrecto.
- Tal y como ocurre durante el uso habitual de las lenguas naturales, la cadena de entrada puede estar incompleta, o ser sintácticamente incorrecta.

En relación a los problemas expuestos, trataremos los siguientes aspectos dentro del analizador sintáctico del sistema ICE:

- *Palabras desconocidas.* El analizador morfológico no es capaz de etiquetar alguna palabra, en este caso el analizador sintáctico tratará de establecer dicho análisis en función de su contexto sintáctico.
- *Análisis sintácticos parciales.* Aún cuando la cadena de entrada sea errónea, el analizador obtendrá, en la medida de lo posible, los análisis de diversas porciones de la cadena de entrada.

Nótese que los cambios expuestos están dirigidos a obtener la mayor cantidad de información posible a partir de una cadena de entrada con errores. Otra aproximación diferente sería intentar corregir dichos errores para obtener un análisis sintáctico completo. La aplicación de esta aproximación al sistema ICE constituye un trabajo complementario que puede ser consultado en [152, 153]. En lo que resta, para no alargar en exceso la exposición, se han omitido algunos SDGs intermedios que pueden ser consultados en el apéndice A.

8.2.1 Palabras desconocidas

En adelante representaremos las palabras desconocidas mediante un símbolo nuevo, *desc*, que no pertenece al conjunto de símbolos de la gramática. Tal y como indicábamos anteriormente, trataremos dichas palabras como un caso especial de múltiples etiquetas, es decir, consideraremos todos sus análisis posibles dentro del contexto en que se encuentren. Para el caso de las GICs esto significa que, potencialmente, dicho análisis contendrá como alternativas todos los símbolos terminales de la gramática. En la figura 8.2 se muestran los SDGs resultantes.

Para el SDG del analizador descendente hemos modificado la regla de inferencia, reconocimiento, encargada del análisis de los símbolos de la cadena de entrada. Puesto que asignamos a una palabra desconocida todos los análisis posibles, la nueva regla simplemente continúa con el reconocimiento de cualquier símbolo terminal cuando la siguiente palabra a analizar es desconocida.

De forma similar en el caso de los analizadores ascendentes, modificamos la regla reconocimiento del SDG correspondiente. En esta ocasión, la nueva regla creará un ítem por cada uno de los posibles análisis de la palabra desconocida, es decir todos los símbolos terminales, $a \in \Sigma$.

En lo que respecta al algoritmo de Earley, la modifición de la regla reconocimiento en el SDG correspondiente, es análoga a la presentada para el algoritmo descendente.

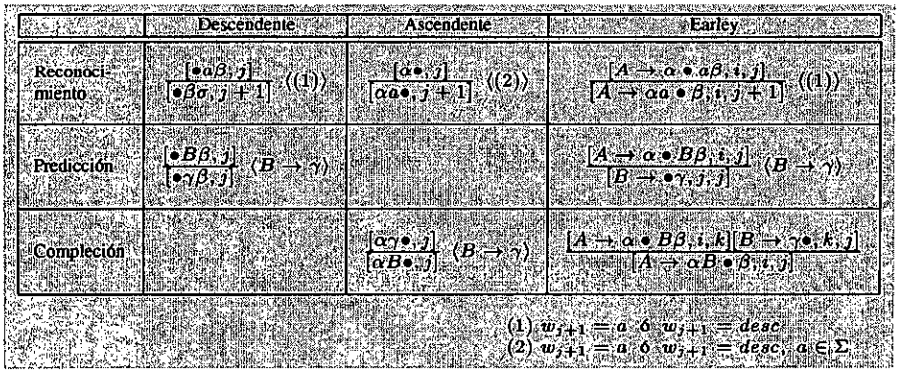


Figura 8.2: Pasos deductivo de los SDGs para GICs con palabras desconocidas

Siguiendo la línea de anteriores capítulos, a continuación desarrollaremos la evolución del algoritmo de Earley a algoritmos tipo LR(1). En el caso que nos ocupa,

mantendremos la evolución presentada en el capítulo 4, de tal forma que únicamente debemos modificar el SDG del intérprete, y no la construcción del control finito. Para ello ante la presencia de una palabra desconocida, consideraremos todos los símbolos terminales para los cuales el control finito establezca una acción válida [145]. El SDG resultante se muestra en la figura 8.3. A continuación ilustraremos mediante un ejemplo los cambios propuestos para el caso de un analizador LR.

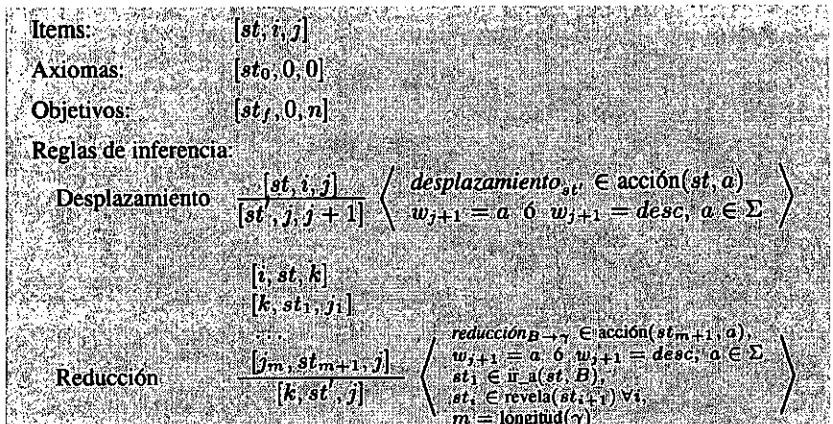


Figura 8.3: SDG del algoritmo LR(1) con palabras desconocidas

EJEMPLO 8.6

En este ejemplo se muestran diversas acciones del intérprete LR ante palabras desconocidas. Emplearemos la GIC aumentada del ejemplo 3.13, con el control finito de la tabla 3.1. La cadena de entrada está formada por palabras desconocidas. Ante la configuración (0) *la única acción válida es desp*(1) *para el símbolo "a". La configuración resultante es* $(0a1)$.

En la configuración $(0F3)$ *existen dos acciones válidas: red*(4) *y desp*(5) *para los símbolos "+" y "*", respectivamente. Las configuraciones resultantes son* $(0T2)$ *y* $(0F3 * 5)$. ■

Una vez tratada la adaptación de los algoritmos de análisis de GICs anteriores al manejo de palabras desconocidas, procederemos a su extensión al análisis de GCDs. Como principal diferencia, señalaremos que no es posible sustituir las palabras desconocidas por el conjunto de símbolos terminales, ya que dicho conjunto es potencialmente infinito. En su lugar, seleccionaremos un conjunto finito de símbolos , $\mathcal{T}$, que subsuman a todos los demás. Una vez seleccionado, la extensión de los algoritmos es análoga a la realizada en el capítulo 6, tal y como se muestra en el figura 8.4 para el SDG del algoritmo de Earley.

Puesto que estamos desarrollando los algoritmos como una extensión de los presentados para el caso de las GICs, dicho conjunto finito de símbolos terminales contendrá un símbolo por cada símbolo de predicado terminal. Como argumentos de dichos símbolos consideraremos los términos más generales, es decir, variables. Finalmente para una GCD:

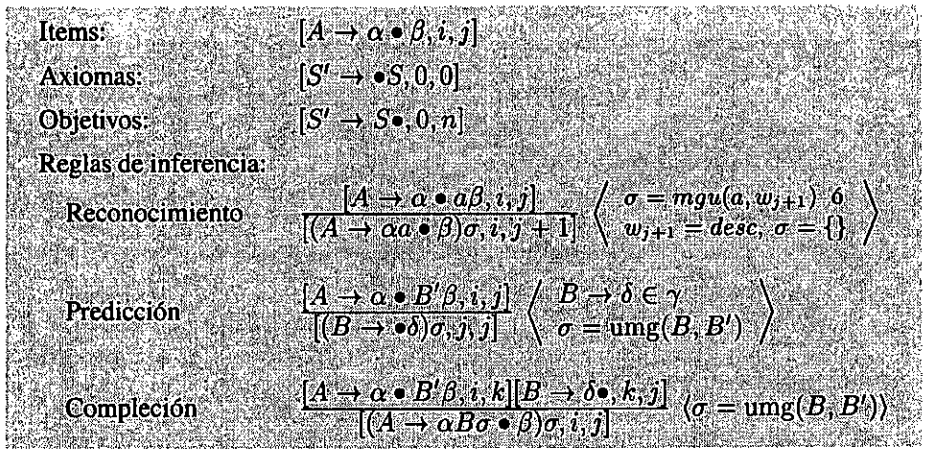


Figura 8.4: SDG del algoritmo de Earley para GCDs y palabras desconocidas

$$\mathcal{D} = (V_T, V_N, \mathcal{V}, \mathcal{F}, \gamma, \overset{\circ}{\mathcal{P}}) \quad \text{tenemos que} \quad \mathcal{T} = T_{V_T}[\mathcal{X}]$$

El siguiente ejemplo ilustra la definición del conjunto de símbolos descrito.

EJEMPLO 8.7
Dado el siguiente conjunto de símbolos terminales:

nombre(singular,masculino),	*nombre(singular,femenino),*
nombre(singular,masculino),	*determinante(singular,masculino),*
determinante(singular,femenino),	*verbo(transitivo,3ª,singular),*
verbo(intransitivo,3ª,singular),	*preposición*

asignaremos como análisis válidos de una palabra desconocida:

nombre(_,_), determinante(_,_), verbo(_,_,_), preposición ■

Finalmente, hemos tratado como formalismo operacional los APs y ALPs, que también adaptaremos de forma similar a como hemos hecho con los SDGs anteriores, tal y como se describe en [85]. En general, recordamos que las transiciones de un ALP son de la forma:

- $B \overset{a}{\longmapsto} C(\xi\, E.\rho) = \xi\, C\sigma.\rho\sigma$, si y sólo si $\sigma = \text{umg}(B, E)$.
- $B \overset{a}{\longmapsto} BC(\xi\, E.\rho) = \xi\, B.\rho\, C\sigma.\sigma$, si y sólo si $\sigma = \text{umg}(B, E)$.
- $BD \overset{a}{\longmapsto} C(\xi\, E'.\rho'\, E.\rho) = \xi\, C\sigma.\rho'\rho\sigma$, si y sólo si $\sigma = \text{umg}(\langle E, E'\rho\rangle, \langle D, B\rangle)$.

En caso de que el siguiente símbolo de la cadena de entrada sea una palabra desconocida, para cualquier símbolo terminal de la GCD, $A_i(t_1, \ldots, t_{m_i})$, aplicaremos las transiciones existentes para todos los símbolos "a":

$$a = A_i(\_, \overset{m_i}{\cdots}, \_)$$

Por último, en el caso concreto del sistema ICE, aplicaremos las modificaciones realizadas en los algoritmos tipo LR(k) sobre el mecanismo de control estático. El esquema de compilación resultante se muestra en la figura 8.5, donde hemos modificado las transiciones de desplazamiento, [Desplazamiento] y [DespInicial], y la transición de [Selección]. En el caso de las operaciones de desplazamiento, la propia transición del autómata permite el desplazamiento únicamente del símbolo a. Este desplazamiento se realizará tanto si el siguiente símbolo de la cadena de entrada es compatible, como si es una palabra desconocida. Por su parte, las transiciones de selección, cuando el símbolo adelantado es una palabra desconocida, se realizarán para todas las acciones de reducción compatibles con algún símbolo de $\mathcal{T}$.

A continuación, una vez establecido el tratamiento de palabras desconocidas, abordaremos la obtención de análisis parciales.

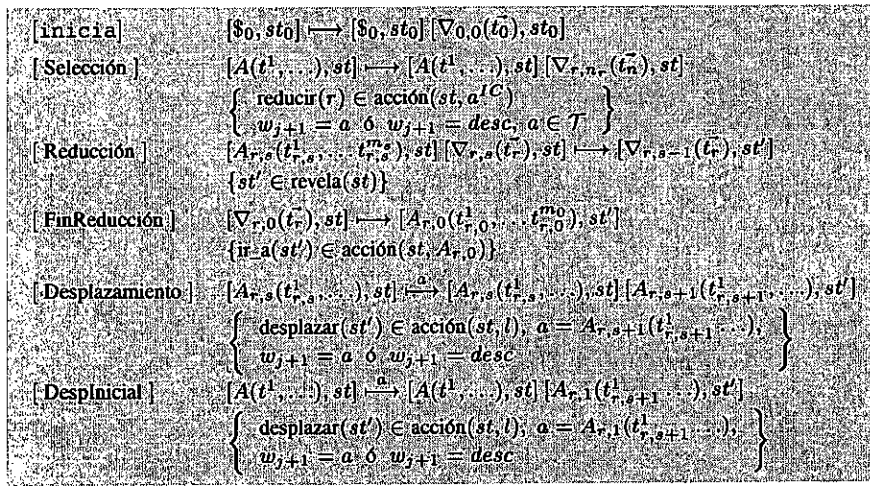


Figura 8.5: Esquema de compilación en ICE para GCDs y palabras desconocidas

8.2.2 Análisis sintácticos parciales

Las computaciones redundantes son habituales cuando tratamos formalismos gramaticales complejos, donde la ambigüedad es un fenómeno frecuente. Esta situación ha motivado el desarrollo de técnicas de análisis que representan las computaciones y los árboles en algún tipo de estructura compartida. Un área de aplicación extensa es el análisis de las lenguas naturales, donde la consideración de la programación dinámica se conoce desde hace tiempo [48]. En particular, el tratamiento de las lenguas naturales añade un nuevo desafío, ya que, a menudo la información gramatical es insuficiente. Como señalábamos anteriormente, el problema de la información parcial es debido a

errores en etapas anteriores del procesamiento junto con el hecho de que las gramáticas y lexicones existentes son incompletos e incluso incorrectos. En el presente apartado desarrollaremos una extensión de las técnicas de análisis sintáctico presentadas, con el objeto de obtener la mayor cantidad posible de información de dicho proceso, al mismo tiempo que preservamos la compactabilidad de su representación.

En adelante, nos referiremos al análisis sintáctico convencional, tal y como lo hemos descrito hasta ahora, como *análisis sintáctico completo*, mientras que el emplearemos el término *análisis sintáctio parcial* para referirinos a sus posibles subcomputaciones. Nuestro objetivo es obtener todos los análisis parciales correctos incluso cuando no existe un análisis completo, es decir, cuando existen lagunas.

EJEMPLO 8.8

Consideremos como ejemplo el lenguaje, $\mathcal{P}$, de los palíndromos no vacíos sobre el alfabeto $\Sigma = \{a, b\}$, generado por la GIC:

Palin $\rightarrow a$	Palin $\rightarrow b$
Palin $\rightarrow a$ Palin a	Palin $\rightarrow b$ Palin b

Obsérvese que, aunque la cadena de entrada aababab $\notin \mathcal{P}$, contiene subcadenas que sí pertenecen al lenguaje, tal y como se muestra en la figura 8.6. ■

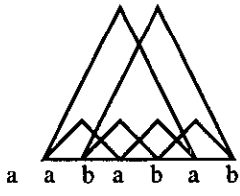


Figura 8.6: Análisis parciales de aababab

La aproximación al problema que adoptaremos será la extensión de los modelos empleados hasta el momento preservando los beneficios de la programación dinámica. En el caso del sistema ICE, dicho modelo incluye una mejora del rendimiento mediante un control estático encargado de eliminar computaciones innecesarias. Aplicado al análisis sintáctico parcial, este control puede eliminar algunas ramas de análisis que, si bien no son necesarias en un análisis completo, sí lo son en un análisis parcial. Para abordar este problema, trataremos el comienzo y final de un análisis parcial como otra forma de no determinismo, modificando el control estático en consecuencia.

En contraste con esta aproximación, otros autores optan por modificar un modelo de análisis particular [66, 138, 115]. Otra aproximación [2, 1], consiste en realizar el análisis en dos fases. En una primera se llevan a cabo todos los análisis parciales posibles, y a continuación se intentan unir en un único análisis completo.

Análisis sintáctico parcial de GIC

Para obtener una definición pragmática de análisis parcial, generalizaremos las condiciones que venimos aplicando sobre el concepto de gramática. Para una GIC tenemos

un símbolo inicial o axioma, S, a partir del cual podemos generar todas las cadenas del lenguaje correspondiente. En su lugar usaremos un conjunto de símbolos iniciales, $\mathcal{S}$, siguiendo el concepto de *punto de entrada* [65], una estructura clásica en sintaxis abstracta, que permite analizar fragmentos de programas. Cada uno de estos símbolos iniciales puede generar o bien cadenas del lenguaje o bien partes de ellas, dando lugar tanto a análisis completos como parciales. A continuación discutiremos la extensión de los SDGs vistos anteriormente.

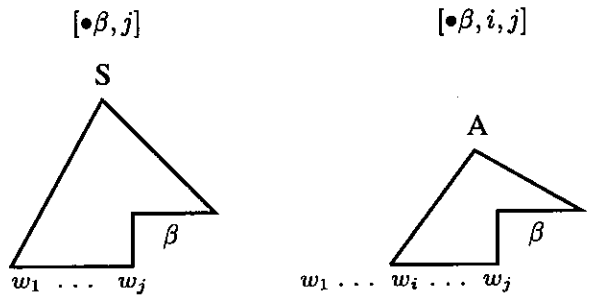


Figura 8.7: Representación gráfica de los items. Esquema descendente

En el SDG para el caso descendente, figura 4.2, teníamos un único axioma que predecía el análisis del símbolo inicial, y un ítem objetivo que representaba el análisis de la cadena de entrada completa. Para adaptar este esquema, en primer lugar debemos considerar que un análisis parcial puede cubrir cualquier porción de la cadena de entrada. Para considerar este caso, añadiremos a los items una referencia a la posición inicial. La figura 8.7 muestra una representación gráfica de los nuevos items, en contraposición con los empleados para el análisis completo. Igualmente, modificaremos los axiomas del SDG de forma que predigan cualquier análisis parcial válido, comenzando en cualquier posición de una cadena de entrada de longitud n:

$$\text{Axiomas} = \{[\bullet A, i, i], A \in \mathcal{S},\ 0 \leq i < n\}$$

Análogamente, extenderemos el conjunto de items objetivo para considerar los análisis parciales que finalizan en cualquier posición de la cadena de entrada:

$$\text{Objetivos} = \{[\bullet, i, j], 0 \leq i < j \leq n\}$$

Por último, los pasos deductivos se comportan como en el esquema original, y a mayores conservan la posición de inicio.

- Reconocimiento: $\dfrac{[\bullet w_{j+1}\beta, i, j]}{[\bullet\beta, i, j+1]}$
- Predicción: $\dfrac{[\bullet B\beta, i, j]}{[\bullet\gamma\beta, i, j]}\ \langle B \to \gamma\rangle$

Nótese que la extensión presentada difiere del caso en que simplemente añadimos nuevas producciones a la gramática $\mathcal{G} = (N, \Sigma, P, S)$:

$$
\begin{aligned}
&S' \rightarrow S_c \\
&S' \rightarrow S_p \\
&S_c \rightarrow S \\
&S_p \rightarrow A,\ A \in \mathcal{S} \\
&\vdots
\end{aligned}
$$

Este cambio permite analizar cualquiera de los símbolos A del conjunto de símbolos iniciales $\mathcal{S}$, pero no permite que el análisis cubra porciones arbitrarias de la cadena de entrada, es decir que comiencen y terminen en cualquier posición.

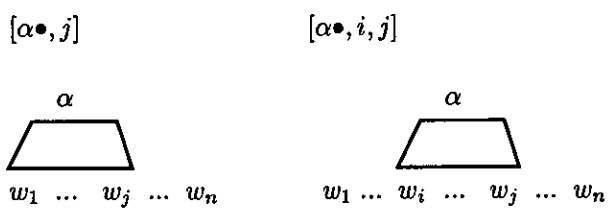


Figura 8.8: Representación gráfica de los items. Esquema ascendente

Otra consideración a realizar sobre la extensión de los algoritmos para el tratamiento de análisis parciales, es el aumento de la complejidad. Con respecto al esquema original en que teníamos un único axioma, ahora necesitaremos $m \times n$, donde $m = \|\mathcal{S}\|$ y n es la longitud de la cadena de entrada.

	Descendente	Ascendente	Earley
Axiomas	$[\bullet A, i, i]$ (1)	$[\bullet, i, i]$ (1)	$[S' \rightarrow \bullet A, i, i]$ (1)
Objetivos	$[\bullet, i, j]$ (1)	$[A\bullet, i, j]$ (1)	$[S' \rightarrow A\bullet, i, j]$ (1)
Reconocimiento	$\dfrac{[\bullet w_{j+1}\beta, i, j]}{[\bullet\beta\sigma, i, j+1]}$	$\dfrac{[\alpha\bullet, i, j]}{[\alpha w_{j+1}\bullet, i, j+1]}$	$\dfrac{[A \rightarrow \alpha \bullet w_{j+1}\beta, i, j]}{[A \rightarrow \alpha w_{j+1} \bullet \beta, i, j+1]}$
Predicción	$\dfrac{[\bullet B\beta, i, j]}{[\bullet\gamma\beta, i, j]}$ ((2))		$\dfrac{[A \rightarrow \alpha \bullet B\beta, i, j]}{[B \rightarrow \bullet\gamma, j, j]}$ ((2))
Compleción		$\dfrac{[\alpha\gamma\bullet, i, j]}{[\alpha B\bullet, i, j]}$ ((2))	$\dfrac{[A \rightarrow \alpha \bullet B\beta, i, k]\,[B \rightarrow \gamma\bullet, k, j]}{[A \rightarrow \alpha B \bullet \beta, i, j]}$
			(1) $A \in \Sigma,\ 0 \leq i \leq j \leq n$ (2) $B \rightarrow \gamma$

Figura 8.9: SDGs para análisis parcial de GICs

Los cambios realizados son extrapolables al caso del SDG para el análisis ascendente, figura 4.5. De nuevo extendemos los items con una referencia a la posición de

inicio, tal y como ilustra la figura 8.8. Al mismo tiempo el conjunto de axiomas incluye los análisis que comienzan en cualquier posición:

$$\text{Axiomas} = \{[\bullet, i, i,],\ 0 \leq i < n\}$$

Los items objetivo incluyen aquellos que terminan en cualquier posición de la cadena de entrada, una vez reconocido algún símbolo inicial:

$$\text{Objetivos} = \{[A\bullet, i, j], A \in \mathcal{S},\ 0 \leq i \leq j \leq n\}$$

De la misma forma, la única modificación en los pasos deductivos es la conservación de la posición de inicio del análisis:

- Desplazamiento: $\dfrac{[\alpha\bullet, i, j]}{[\alpha w_{j+1}\bullet, i, j+1]}$
- Reducción: $\dfrac{[\alpha\gamma\bullet, i, j]}{[\alpha B\bullet, i, j]}\ \langle B \rightarrow \gamma \rangle$

A diferencia de la extensión realizada para el análisis descendente, el número de axiomas necesario es simplemente n, la longitud de la cadena de entrada.

En la línea de los dos esquemas anteriores, y a diferencia de éstos, el SDG para el algoritmo de Earley, figura 4.7, precisa únicamente la extensión de los conjuntos de axiomas:

$$\text{Axiomas} = \{[S' \rightarrow \bullet A, i, i], A \in \mathcal{S},\ 0 \leq i < n\}$$

e items objetivo:

$$\text{Objetivos} = \{[S' \rightarrow A\bullet, i, j], A \in \mathcal{S},\ 0 \leq i \leq j \leq n\}$$

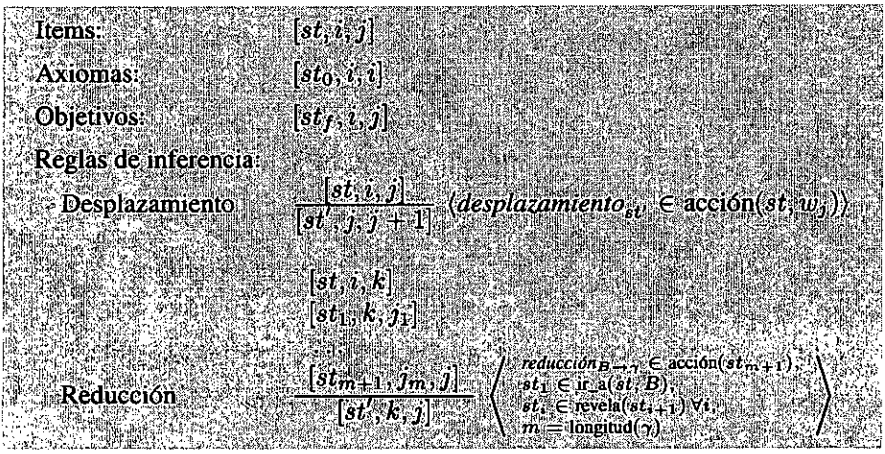


Figura 8.10: SDG del algoritmo LR(0) para análisis parcial

La figura 8.9 resume los SDGs para análisis parcial presentados hasta el momento. A continuación, y siguiendo la exposición del capítulo 4, transformaremos el SDG para el algoritmo de Earley en el SDG para un algoritmo de tipo LR(0), a partir del cual, para conseguir una mayor eficiencia, compilábamos la gramática obteniendo un control finito que guía el proceso de análisis. Recordemos que la construcción de dicho control comienza con un estado inicial $st_0 = [S \rightarrow \bullet\alpha]$, y se basa en la aplicación iterativa de las operaciones `cierre` y `X-sucesor`. Para extender este esquema en el caso del análisis parcial, debemos realizar varias modificaciones. En primer lugar, tenemos que permitir que comience en cualquier posición de la cadena de entrada, para lo cual redefiniremos el conjunto de axiomas:

$$\text{Axiomas} = \{[st_0, i, i], 0 \leq i < n]\}$$

También ampliamos el conjunto de items objetivo para permitir la finalización del análisis en cualquier posición de la cadena de entrada:

$$\text{Objetivos} = \{[st_f, i, j], 0 \leq i \leq j \leq n]\}$$

El SDG resultante se muestra en la figura 8.10. Finalmente, tenemos que adaptar la construcción del control finito. La inicialización del mismo, $st_0 = [S' \rightarrow \bullet S]$, permite los análisis únicamente a partir del símbolo inicial S, en su lugar debemos considerar el conjunto de símbolos iniciales $\mathcal{S}$:

$$st_0 = \{[S' \rightarrow \bullet A] \mid A \in \mathcal{S}\}$$

Además del estado inicial, debemos modificar el estado final, $st_f = \{[S' \rightarrow S\bullet]\}$, que extenderemos a:

$$st_f = \{[S' \rightarrow A\bullet] \mid A \in \mathcal{S}\}$$

Nótese que sobre la extensión anterior existe una alternativa en la que consideramos por cada análisis parcial posible, un estado inicial diferente en el control finito. El resultado es el siguiente conjunto de axiomas en el SDG:

$$\{[st_0, i, i] \mid st_0 = closure(\{S' \rightarrow \bullet S\})\} \cup$$
$$\{[st_j, i, i] \mid st_j = closure(\{[S' \rightarrow \bullet A\}),\ A \in \mathcal{S}\}$$

Sin embargo, podemos considerar que ambas soluciones son intercambiables puesto que el cierre del estado inicial en el primer caso coincide con la unión de los cierres de los estados iniciales en el segundo caso:

$$closure(\{S' \rightarrow \bullet A\}) = \bigcup_A closure(\{S' \rightarrow \bullet A\}),\ A \in \mathcal{S}$$

Siguiendo con la línea propuesta, la eficacia del analizador LR(0) se puede mejorar aumentando la capacidad de determinización del control finito. Para ello añadiremos información de los símbolos adelantados, obteniendo un analizador LR(k). Para el

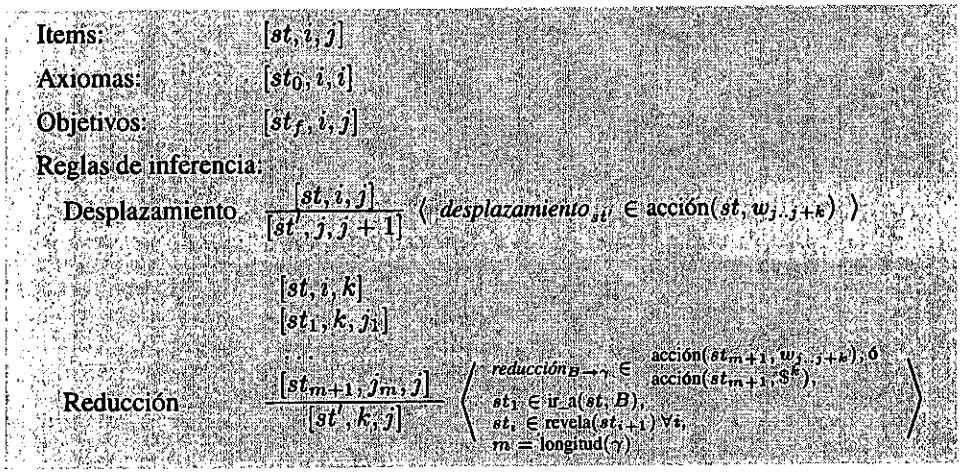


Figura 8.11: SDG del algoritmo LR(k) para análisis parcial

SDG correspondiente, figura 4.10, consideraremos la extensión de los conjuntos de axiomas e items objetivo para el análisis parcial:

$$\text{Axiomas} = \{[st_0, \_, i, i], 0 \leq i < n, A \in \mathcal{S}\}$$
$$\text{Objetivos} = \{[st_j, \_, i, j], 0 \leq i \leq j \leq n, A \in \mathcal{S}\}$$

Nótese que a diferencia de los esquemas anteriores, y puesto que un análisis parcial puede comenzar y terminar en cualquier posición de la cadena de entrada, la acción de reconocimiento es compatible con cualquier símbolo terminal y no sólo con el símbolo de fin de cadena, lo cual hemos indicado mediante el símbolo "_". Adicionalmente, debemos tener en cuenta este factor con respecto a la construcción del control finito, para lo cual consideraremos dos opciones:

- Modificar la construcción del control finito mediante el uso del concepto de *símbolo terminal comodín* tal y como hemos realizado en la extensión de los axiomas e items objetivo del SDG anterior:

  $$st_0 = \{[A \to \bullet\alpha, \_] \mid A \in \mathcal{S}\}$$

  Como aspecto negativo de esta aproximación destacaremos el incremento en el número de estados del control finito.

- Modificar el SDG correspondiente al interprete del autómata LR(k). De esta forma, en cada momento del análisis consideraremos tanto los símbolos adelantados como el símbolo de fin de cadena, tal y como se muestra en la figura 8.11. Los primeros símbolos son compatibles con las acciones que continúan el proceso de análisis, mientras que los segundos son compatibles con la finalización de un análisis parcial.

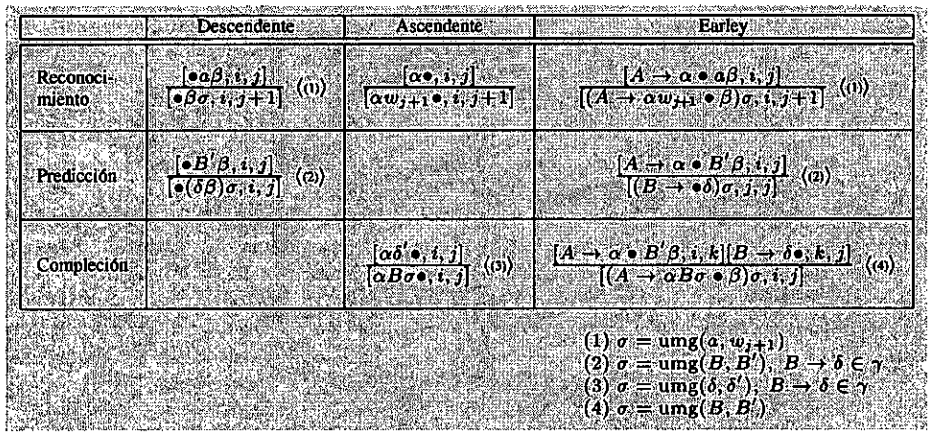


Figura 8.12: Pasos deductivos de SDGs para el análisis parcial GCDs

Análisis parcial de GCD

En lo que respecta al análisis parcial de GCDs, realizaremos un desarrollo paralelo al presentado para las GICs. En primer lugar, para una GCD, en lugar de considerar un único símbolo inicial, consideraremos un conjunto de símbolos iniciales, $\mathcal{P}_I$, cada uno de los cuales puede derivar un análisis parcial válido. A continuación debemos adaptar, tal y como hicimos para las GICs, la forma de los items y extender los conjuntos de axiomas e items objetivo:

- Para el caso descendente:

$$\begin{aligned}\text{Axiomas} &= \{[\bullet A, i, i], 0 \leq i < n, A \in \mathcal{P}_I\}\\ \text{Objetivos} &= \{[\bullet, i, j], 0 \leq i \leq j \leq n\}\end{aligned}$$

- Para el caso ascendente:

$$\begin{aligned}\text{Axiomas} &= \{[\bullet, i, i], 0 \leq i < n\}\\ \text{Objetivos} &= \{[A\bullet, i, j], 0 \leq i \leq j \leq n, A \in \mathcal{P}_I\}\end{aligned}$$

- Para el algoritmo de Earley:

$$\begin{aligned}\text{Axiomas} &= \{[S' \to \bullet A, i, i], 0 \leq i < n, A \in \mathcal{P}_I\}\\ \text{Objetivos} &= \{[S' \to A\bullet, i, j], 0 \leq i \leq j \leq n, A \in \mathcal{P}_I\}\end{aligned}$$

Una vez modificados los items, axiomas y objetivos, adaptaremos los pasos deductivos tal y como se muestra en la figura 8.12.

En lo que respecta a los analizadores de tipo LR(k), tal y como indicábamos en el capítulo 6, debido a los problemas de no-terminación en la construcción del control finito hemos optado por construir dicho control a partir del esqueleto IC trasladando la operación de unificación únicamente al intérprete del autómata. A continuación se ilustra la aplicación de esta idea al analizador sintáctico del sistema ICE.

$$
\begin{array}{ll}
[\texttt{inicia}] & [\$_0, st_0, i, i] \longmapsto [\$_0, st_0, i, i]\,[\nabla_{0,0}(\vec{t_0}), st_0, i, i] \\
[\,\text{Selección}\,] & [A, st, i, j] \longmapsto [A, st, i, j]\,[\nabla_{r,n_r}(\vec{t_r}), st, j, j] \\
 & \left\{ \begin{array}{l} \text{reducir}(r) \in \text{acción}(st, w^{IC}_{j+1}\ldots), \text{ ó} \\ \text{reducir}(r) \in \text{acción}(st, \$^k) \end{array} \right\} \\
[\,\text{Reducción}\,] & [A_{r,s}, st, i, k]\,[\nabla_{r,s}(\vec{t_r}), st, k, j] \longmapsto [\nabla_{r,s-1}(\vec{t_r}), st', i, j] \\
 & \{st' \in \text{revela}(st)\} \\
[\,\text{FinReducción}\,] & [\nabla_{r,0}(\vec{t_r}), st, i, j] \longmapsto [A_{r,0}, st', i, j] \\
 & \{\text{ir\_a}(st') \in \text{acción}(st, A^{IC}_{r,0})\} \\
[\,\text{Desplazamiento}\,] & [A_{r,s}, st, i, j] \xrightarrow{a} [A_{r,s}, st, i, j]\,[A_{r,s+1}, st', i, j+1] \\
 & \{\text{desplazar}(st') \in \text{acción}(st, w^{IC}_{j+1}\ldots),\ a = A_{r,s+1}\} \\
[\,\text{DespInicial}\,] & [A, st, i, j] \xrightarrow{a} [A, st, i, j]\,[A_{n,1}, st', i, j+1] \\
 & \{\text{desplazar}(st') \in \text{acción}(st, w^{IC}_{j+1}\ldots),\ a = A_{r,1}\}
\end{array}
$$

Figura 8.13: Esquema de compilación en ICE para análisis parcial de GCDs

Análisis sintáctico parcial en el sistema ICE

Para la adaptación de analizador del sistema ICE al análisis parcial de GCDs, tendremos en cuenta que el control estático se construye a partir del esqueleto IC como un autómata de tipo LR(k). Tras aplicar los cambios descritos anteriormente a la construcción del autómata y la extensión de los items iniciales y finales, obtenemos el esquema de compilación de la figura 8.13.

CAPÍTULO 9

RESULTADOS EXPERIMENTALES

En el presente capítulo describiremos los resultados obtenidos tras la realización de diversos experimentos sobre las técnicas y algoritmos de análisis sintáctico tratadas en los capítulos anteriores. Dichos experimentos versan sobre cuestiones derivadas de la complejidad temporal de los algoritmos, tiempos de ejecución de sus implementaciones o eficacia del control estático.

9.1 Complejidad temporal

Los cálculos con un alto grado de redundancia constituyen un problema característico en el análisis sintáctico no determinista, especialmente en el caso del PLN. Así, a medida que crece el número de ambigüedades, si no se considera ninguna técnica de compartición de estructuras, crece exponencialmente el número de cálculos necesarios [83]. En este sentido, a continuación se describen experimentos cuyo objetivo es comparar diversos analizadores en base a su complejidad temporal.

9.1.1 Analizadores de GICs

En este apartado compararemos la eficacia de varios de los analizadores descritos en capítulos anteriores. Para conseguir la mayor fidelidad posible en dicha comparación hemos seleccionado como marco descriptivo común los SDGs, capítulo 4. El comportamiento y la comparación entre los analizadores se realizará en función del número de cálculos realizados durante el análisis. Como medida del número de dichos cálculos tomaremos el número de items generados. Los analizadores estudiados son los siguientes:

- Descendente, el SDG se muestra en la figura 4.2. Este analizador es representativo de una estrategia de análisis descendente.
- Ascendente, el SDG se muestra en la figura 4.5. Este analizador es representativo de una estrategia de análisis ascendente.
- Earley [47], el SDG se muestra en la figura 4.7. Este analizador es representativo de una estrategia mixta con control dinámico.
- ICE [12], el SDG se muestra en la figura 5.15. Este analizador es representativo de una estrategia mixta con control estático.

A su vez, para los tests de análisis se han empleado las siguientes gramáticas:

- *palíndromo*. Genera el lenguaje de los palíndromos no vacíos sobre el alfabeto $\Sigma = \{a, b\}$. Como principal característica señalaremos que favorece una estrategia de análisis predictiva o descendente. Viene dada por las producciones:

$$\begin{array}{ll} Palin \rightarrow a & Palin \rightarrow b \\ Palin \rightarrow a\ Palin\ a & Palin \rightarrow b\ Palin\ b \end{array}$$

- *expr-arit*. Presentada en el ejemplo 3.2. Genera el lenguaje de expresiones aritméticas, y viene dada por las producciones:

$$\begin{array}{ll} S \rightarrow S + S & S \rightarrow (S) \\ S \rightarrow S * S & S \rightarrow a \end{array}$$

  Como principal característica destacaremos el hecho de que la gramática presenta recursividad por la izquierda. Esta característica invalida la aplicabilidad del esquema de análisis descendente. Para solucionar este problema hemos añadido un restrictor [127, 128] al esquema de análisis descendente que evita la creación de derivaciones infinitas. Dicho restrictor se obtuvo tras la observación de la gramática, quedando expresado como una restricción sobre el conjunto de items del esquema para una cadena de entrada de longitud n:

$$\text{Items}: \quad \{[\bullet\beta, j],\ \text{longitud}(\beta) < n - j\}$$

- *expr-arit-no-rec*. Al igual que *expr-arit* genera el lenguaje de expresiones aritméticas. A diferencia de ésta, las producciones han sido modificadas para eliminar la recursividad por la izquierda:

$$\begin{array}{ll} E \rightarrow E + T & E \rightarrow T \\ T \rightarrow T * F & T \rightarrow F \\ F \rightarrow (E) & F \rightarrow a \end{array}$$

  Sobre esta gramática destacaremos que la supresión de la recursividad por la izquierda permite la aplicación directa del esquema descendente, sin la necesidad de la aplicación de restrictores. A mayores, la transformación aplicada favorece el análisis predictivo.

- *ln-1*. Dento del PLN, esta gramática maneja la adjunción de sintagmas preposicionales. De nuevo contiene recursividades por la izquierda por lo que ha sido necesaria la aplicación de un restrictor al análisis descendente como en el caso de la gramática *expr-arit*. El conjunto de producciones es:

$$\begin{array}{lll} S \rightarrow Frase & FN \rightarrow \text{pronombre} & FP \rightarrow prep\ FN \\ Frase \rightarrow FN\ FV & FN \rightarrow det\ nombre & FV \rightarrow verb\ FN \\ Frase \rightarrow Frase\ FP & FN \rightarrow FN\ FP & \end{array}$$

Por cada gramática hemos empleado un juego de prueba formado por todas las cadenas de entrada pertenecientes al lenguaje. A su vez, hemos agrupado estas cadenas en conjuntos según su longitud, calculando, para cada uno de ellos, la media aritmética de los resultados obtenidos. La figura 9.1 recapitula dichos resultados, cada tabla se corresponde con una gramática y cada columna con una longitud de las cadenas de entrada. Por su parte, la figura 9.2 muestra una representación gráfica de dichas tablas. Finalmente, la figura 9.3 sintetiza las anteriores.

palíndromo	1	3	5	7	9	11	13	15	17	19
descendente	11	24	38	52	67	82	97	112	127	142
ascendente	3	16	72	313	1333	5408	-	-	-	-
Earley	12	28	46	65	84	104	124	144	164	184
ICE	4	14	25	37	49	62	75	87	100	113

expr-arit	1	3	5	7	9	11	13	15	17	19
descendente	1	2	11	35	103	297	854	2417	-	-
ascendente	3	9	21	44	90	178	345	663	2244	-
Earley	9	20	33	47	62	78	95	112	128	146
ICE	4	7	12	18	25	31	39	46	53	61

expr-arit-no-rec	1	3	5	7	9	11	13	15	17	19
descendente	19	56	132	282	584	1186	1614	2144	-	-
ascendente	5	22	77	256	827	-	-	-	-	-
Earley	13	26	39	52	66	80	93	107	121	135
ICE	6	10	15	20	24	29	34	38	43	48

ln-1	3	4	5	6	7	8	9	10	11	12	13	14
descendente	6	15	32	57	91	140	199	276	369	480	613	766
ascendente	13	14	37	52	93	162	259	450	731	1215	2009	3282
Earley	24	25	34	39	44	53	58	67	74	82	91	100
ICE	11	12	16	19	23	29	34	40	47	54	62	71

Figura 9.1: Tablas de resultados experimentales en GICs

Como se aprecia en la figura 9.1, el esquema descendente presenta el peor rendimiento para gramáticas con un alto grado de ambigüedad, *expr-arit*, y, consecuentemente, repetición de cálculos. Esta falta de rendimiento se produce aún a pesar del empleo del restrictor, que favorece el proceso de análisis eliminando items inútiles. Este beneficio del restrictor se observa para las cadenas de entrada de menor longitud en las que el analizador descendente muestra los mejores resultados.

Por su parte, el esquema ascendente muestra un comportamiento uniforme en todos los ejemplos, tendiendo a un crecimiento exponencial[1] del número de items generados a medida que aumenta el tamaño de la cadena de entrada. Esta característica hace que su rendimiento sólo sea aceptable para las entradas de menor tamaño.

En lo que respecta a las estrategias mixtas, éstas presentan, en general, los mejores resultados independientemente de la longitud de la cadena de entrada y de la gramática empleada. De entre las dos estrategias mixtas, la que incorpora el control estático consigue el mejor rendimiento, ratificando la conveniencia del empleo de dicho control.

[1] De hecho, este crecimiento exponencial nos ha impedido la obtención de resultados para longitudes de la cadena de entrada mayores, que sí fueron obtenidos con los otros esquemas.

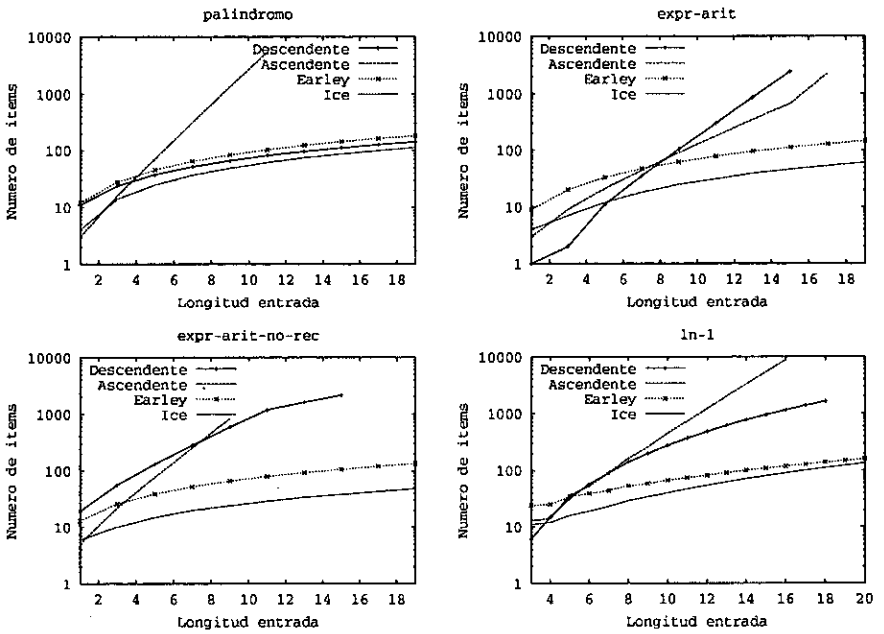


Figura 9.2: Resultados experimentales en GICs

9.1.2 Comparación de analizadores basados en unificación

Diferenciaremos dos dominios: analizadores inmediatos y diferidos. Los primeros se corresponden con los presentados en el capítulo 6, en los que la operación de unificación se aplica durante el proceso de análisis. En los segundos existe una fase preliminar en la que se realiza el análisis sobre un esqueleto de la gramática, sin tener en cuenta la unificación. A continuación, y sobre los resultados de esta fase preliminar, aplicaremos la operación de unificación. Como resultado de esta segunda fase, por un lado eliminaremos análisis incorrectos, y por otro, dividiremos contextos sintácticos dando lugar a nuevos análisis. En el caso que nos ocupa, nos centraremos en las GCDs para el caso de los analizadores inmediatos y en las GICs como esqueleto de la gramática para el caso de los analizadores diferidos.

Para comparar diferentes estrategias de análisis convenientemente, en primer lugar debemos seleccionar un marco descriptivo común, tanto para GICs como para GCDs. En esta ocasión elegimos los ALPs, y los APs como un caso particular de los primeros, y las técnicas de tabulación presentadas en el capítulo 6, en la forma de los entornos dinámicos S^T, S^2y S^1. A continuación, para mostrar cómo mejora la eficiencia por la aplicación de técnicas de programación dinámica, nos centraremos en dos aspectos: el número de cálculos en contraste con las técnicas clásicas basadas en retroceso, y una comparación simple entre diferentes esquemas.

Con la intención de simplificar al presentación, en el presente apartado consideraremos una única gramática representativa de los resultados obtenidos:

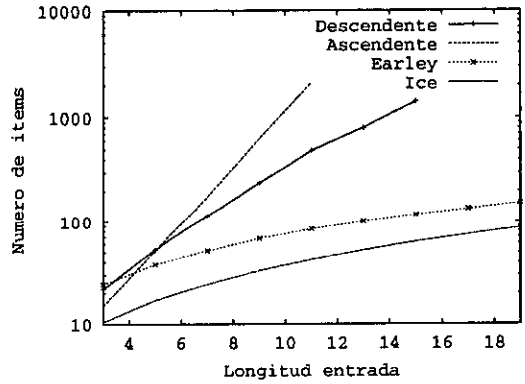


Figura 9.3: Resumen de resultados en GICs

s → *frase*
frase → *fn(concor(N,G)), fv(N)*
frase → *frase, fp*
fn(concor(N,G)) → *"nombre(N,G)"*
fn(concor(N,G)) → *"pronombre(N,G)"*
fn(concor(N,G)) → *"determinante(N,G)", "nombre(N,G)"*
fn(concor(N,G)) → *fn(concor(N,G)), fp*
fp → *"preposición", fn(concor(N,G))*
fv(N) → *"verbo(transitivo,_,N)", fn(_)*

cuyo esqueleto independiente del contexto es el siguiente:

S	→	*Frase*	*Frase*	→	*FN FV*	*Frase*	→	*Frase FP*
FN	→	*nombre*	*FN*	→	*pronombre*	*FN*	→	*determinante nombre*
FN	→	*FN FP*	*FP*	→	*preposición FN*	*FV*	→	*verbo FN*

Y como entrada hemos seleccionado las cadenas de la forma:

$$\text{Yo veo un padre (de un hijo de un padre)}^i$$

donde $i \geq 0$ es el número de repeticiones de la subcadena "*de un hijo de un padre*". El número de análisis diferentes, C_i, crece exponencialmente con i y viene dado por:

$$C_0 = C_1 = 1 \quad \text{y} \quad C_i = \binom{2i}{i} \frac{1}{i+1}, \text{ si } i > 1$$

favoreciendo el estudio del comportamiento de los analizadores en presencia de cálculos con un alto grado de redundancia. En relación a los analizadores, hemos seleccionado los siguientes:

- *Analizadores diferidos.*

  Compararemos varios esquemas de análisis independiente del contexto. En concreto consideraremos tres generadores de analizadores: el entorno AGFL[92], el

algoritmo clásico de Earley [47], y el sistema ICE [166, 164]. Estos tres generadores son representativos, respectivamente, de las siguientes estrategias: clásica descendente con retroceso, mixta con predicción dinámica y mixta con predicción estática basada en un autómata LALR(1). Hemos seleccionado el sistema ICE como el representante más eficiente de los analizadores de tipo LR generalizados, entre los que se encuentran SDF [61] y GLR [109], ambos basados en el algoritmo de Tomita [146].

- *Analizadores inmediatos.*

  En lo que respecta a las GCDs y los analizadores inmediatos, consideraremos una vez más el entorno AGFL[2] como representativo de un esquema de análisis descendente, el esquema de deducción de Earley [105] y finalmente el sistema ICE [166]. En esta ocasión hemos seleccionado ICE como el representante más eficiente de la familia de entornos de inferencia de tipo LR [99, 117].

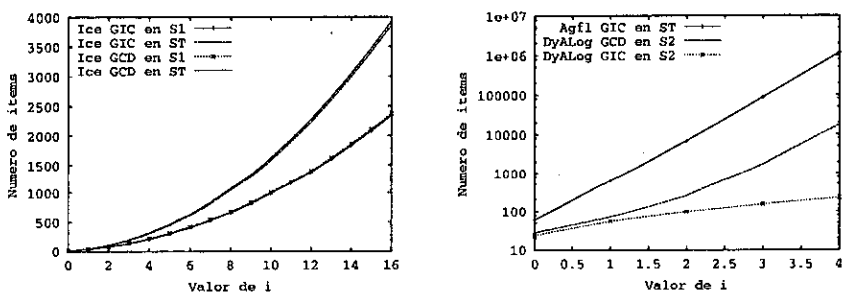


Figura 9.4: Resultados experimentales con ICE y análisis descendente para GCDs

Los análisis descritos se han llevado a cabo en los entornos dinámicos S^T con AGFL, Earley e ICE, S^2 con AGFL e ICE, y S^1 con Earley e ICE. Los tests en S^T y S^2 para el algoritmo de Earley no tienen mayor interés, ya que el algoritmo clásico se describe de forma natural en S^1. En el caso de AGFL los tests en S^1 están fuera de lugar ya que dicho entorno no es compatible con una estrategia descendente, y en lo que respecta a S^2, la herramienta original no incorpora las técnicas de programación dinámica, por lo que los resultados se han obtenido mediante una adaptación alternativa del esquema descendente dada por DYALOG [179]. Asimismo, en el caso de las GCDs y AGFL ha sido necesario eliminar los símbolos de función. Para evitar la distorsión de los resultados debido a esta última característica, la gramática ha sido seleccionada de manera que el número de cálculos en S^T es el mismo, tanto para la GCD como para la GIC correspondiente.

Como medida del número de cálculos realizados hemos seleccionado el número de items generados durante el análisis de las cadenas de entrada para diferentes valores

[2] AGFL no es realmente un analizador basado en unificación puesto que está construido sobre el concepto de gramática de afijos. Sin embargo, puede ser asimilado a un analizador de GCDs donde los símbolos de función no están permitidos. De esta forma podemos considerar una de las estrategias de análisis más conocidas.

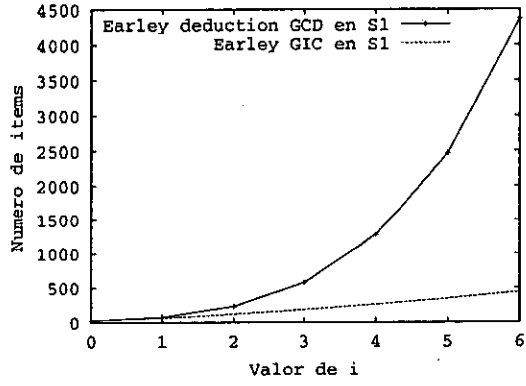


Figura 9.5: Resultados experimentales con una estrategia Earley

de i. Los resultados correspondientes a ICE y AGFL se muestran en la figura 9.4. En el caso de Earley, los resultados se muestran en la figura 9.5. En los casos anteriores, la programación dinámica produce un comportamiento computacional mejor que el mostrado en las aproximaciones clásicas basadas en S^T. A mayores, el sistema ICE presenta los mejores resultados.

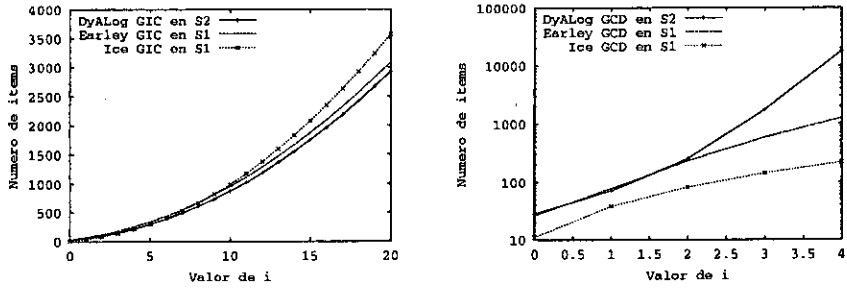


Figura 9.6: Comparación de estrategias para GICs y GCDs

La figura 9.6 recopila los resultados tanto para los análisis inmediatos de la GCD como para el análisis de su esqueleto IC. Los resultados corresponden al entorno dinámico natural para cada analizador, en ellos se aprecia que el incremento en el número de items generados para el caso de la GCD respecto a la GIC es menor en el caso de ICE. En particular, este resultado indica la capacidad del esquema de análisis para evitar unificaciones infructuosas. Finalmente el aparente menor rendimiento de ICE respecto a las otras aproximaciones, en el caso de las GICs, se debe al hecho de que el análisis de la gramática no precisa un número elevado de predicciones, tal y como se indica en [164].

9.2 Eficacia del control estático

Los resultados de la sección 9.1 ya han mostrado la eficacia del control estático, en tanto que aquellos esquemas que lo incorporan ofrecen un mayor rendimiento en el análisis de las cadenas de entrada correctas. En el presente apartado, trataremos el aspecto concerniente al análisis de cadenas que no pertenecen al lenguaje generado por la gramática. En este caso, deseamos que el proceso de análisis detecte cuanto antes esta situación, evitando la mayor cantidad de cálculos posible. De nuevo, consideraremos el número de items generados como medida del esfuerzo realizado.

En primer lugar consideraremos un juego de prueba consistente en conjuntos de cadenas de entrada generadas aleatoriamente a partir de los terminales de la gramática. Al igual que en apartados anteriores dichos conjuntos agrupan cadenas de idéntica longitud. Para las pruebas, hemos considerado las mismas gramáticas y analizadores de la subsección 9.1.1. Los resultados obtenidos se muestran en la figura 9.7. Tal y como ocurría en apartados anteriores, el uso del control estático proporciona los mejores rendimientos.

palíndromo	1	2	3	4	5	6	7	8	9	10	11	12
descendente	11	17	23	29	36	43	50	57	64	70	78	85
ascendente	3	7	15	33	69	147	305	632	1316	2723	5725	-
Earley	12	19	26	34	43	52	61	69	78	86	97	105
ICE	4	7	12	16	22	27	33	38	44	49	56	62

expr-arit	1	2	3	4	5	6	7	8	9	10	11	12	13	14
descendente	1	2	2	8	8	21	22	51	55	123	112	260	237	541
ascendente	2	3	5	7	9	12	17	22	28	34	46	57	60	78
Earley	6	7	7	8	7	7	9	8	9	8	7	8	7	7
ICE	1	1	1	2	1	1	2	2	2	2	1	2	1	1

expr-arit-no-rec	1	2	3	4	5	6	7	8	9	10	11	12	13
descendente	28	40	43	59	55	51	62	48	56	-	-	-	-
ascendente	2	5	7	16	27	41	67	131	175	296	-	-	-
Earley	9	10	10	11	10	10	12	11	12	12	11	11	10
ICE	2	2	2	2	2	2	2	2	2	2	2	2	2

ln-1	1	2	3	4	5	6	7	8	9	10	11	12	13	14
descendente	1	2	5	12	22	37	55	76	107	138	164	194	241	283
ascendente	2	3	5	8	12	16	21	34	40	61	63	85	125	167
Earley	8	9	8	9	8	9	9	8	9	9	8	8	8	8
ICE	2	2	1	2	2	2	2	1	2	2	1	1	1	1

Figura 9.7: Resultados experimentales con cadenas erróneas

9.3 Derivaciones cíclicas

Para el siguiente experimento emplearemos la siguiente gramática:

γ_1 : s(X) $\rightarrow$ fn(X). $\quad$ γ_2 : fn(fn(X, Y)) $\rightarrow$ fn(X) fn(Y).

γ_3 : fn(X) $\rightarrow$ nombre(X). $\quad$ γ_4 : fn(nil).

con la cual se han analizado cadenas de entrada de diversa longitud. Dado que la gramática contiene la cláusula fn(fn(X, Y)) $\rightarrow$ fn(X) fn(Y), el número de análisis cíclicos crece exponencialmente con la longitud de la cadena de entrada, n. Este número es, como en ocasiones anteriores:

$$C_0 = C_1 = 1 \quad \text{y} \quad C_n = \left(\begin{array}{c} 2n \\ n \end{array} \right) \frac{1}{n+1}, \text{ si } n > 1$$

Para cada uno de los análisis se han calculado:

- Número de variables instanciadas en varios entornos dinámicos.
- Número de comprobaciones de derivaciones cíclicas realizadas.
- Número de comprobaciones de derivaciones cíclicas que sería necesario realizar si eliminásemos el control estático.

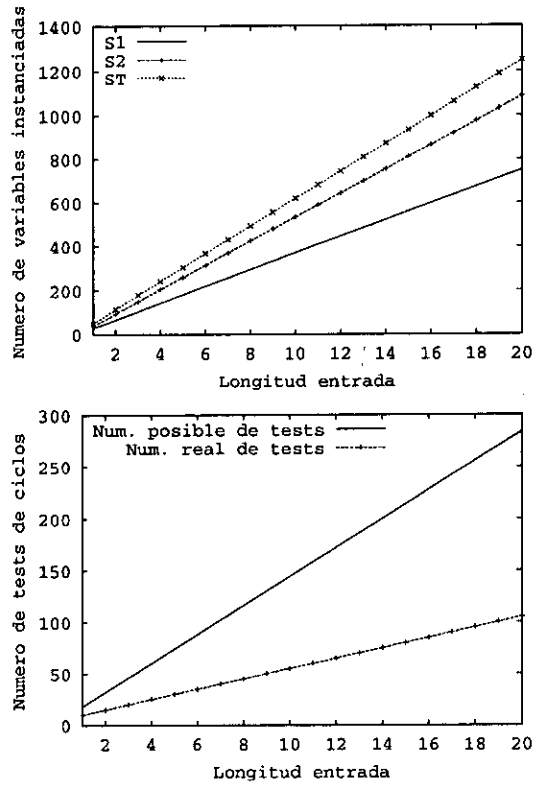


Figura 9.8: Resultados experimentales con estructuras cíclicas

Resulta difícil comparar el sistema ICE con otros analizadores de GCDs debido a los problemas de estos últimos en el tratamiento de estructuras cíclicas. Sin embargo,

podemos considerar los resultados sobre el entorno dinámico S^T como referencia de los métodos tipo SLR(1) [99, 117], y los métodos ascendentes simples [83, 179] se pueden asimilar a los analizadores en un entorno S^1 sin sincronización. La figura 9.8 ilustra los resultados obtenidos, comparando, en la parte superior, el número de variables instanciadas en los tres entornos dinámicos[3], S^1, S^2 y S^T. Por su parte, en la parte inferior de la figura se trata la mejora de eficiencia en la detección de ciclos gracias al uso del autómata LALR(1) como control estático.

9.4 Análisis sintáctico parcial

En este apartado deseamos estudiar los distintos analizadores abordados en el capítulo 8 para el tratamiento del análisis parcial. Para ello, en primer lugar repetiremos los tests realizados en la subsección 9.1.1 empleando los esquemas de análisis parcial mencionados. La figura 9.9 muestra directamente la representación gráfica de los resultados obtenidos. Como se puede observar, se mantiene la relación existente para el caso de los analizadores completos. Las estrategias mixtas, y en concreto el sistema ICE, que incorpora un control estático presentan los mejores rendimientos, mientras que la estrategia ascendente únicamente presenta un buen comportamiento para cadenas de entrada pequeñas. La figura 9.10 sintetiza los resultados.

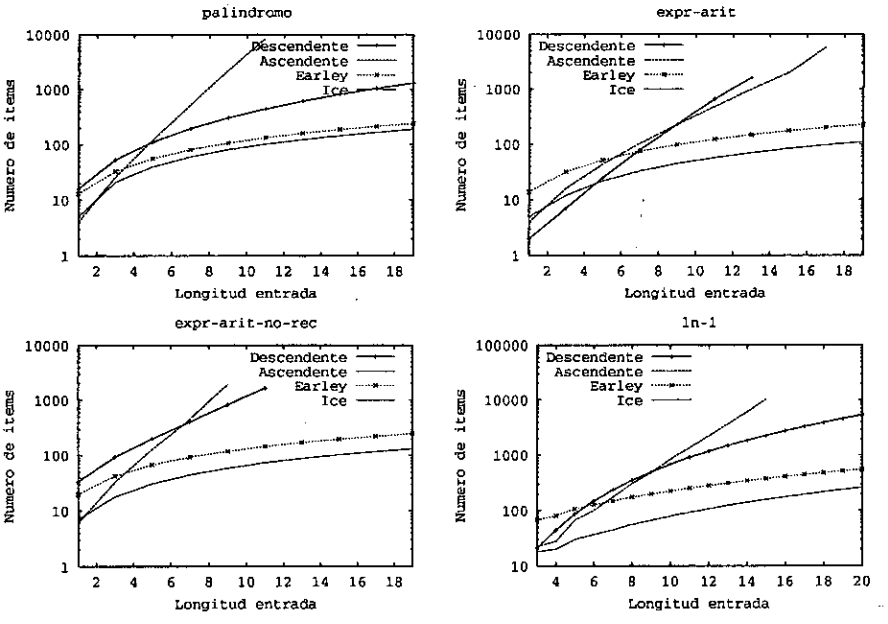


Figura 9.9: Resultados en análisis parciales

[3]Debido a la problemática señalada, en los entornos dinámicos S^2 y S^T se ha realizado una estimación del número mínimo de variables a instanciar en dichos entornos.

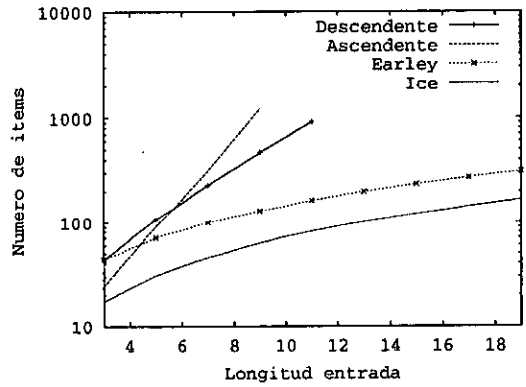


Figura 9.10: Resumen de resultados con análisis parcial

A continuación hemos relacionado los resultados de las figuras 9.9 y 9.2 para los análisis parcial y completo, respectivamente, de los tests descritos en la subsección 9.1.1. Esta relación indica el incremento en tanto por ciento del número de items generados con un esquema de análisis parcial frente a su correspondiente esquema de análisis completo. La figura 9.11 muestra estos incrementos y supone una estimación práctica del incremento del coste computacional que supone sustituir un esquema de análisis completo por su correspondiente extensión al análisis parcial. La parte izquierda recoge los resultados para los esquemas ascendente, Earley e ICE. Como se observa, el esquema ascendente presenta un crecimiento lineal, mientras que los esquemas mixtos, Earley e ICE, tienden a estabilizarse en torno al 200%. Por su parte, el esquema descendente presenta un comportamiento totalmente irregular, tal y como se observa en la parte derecha de la figura. Esto es debido en parte al comportamiento irregular de los restrictores, y en parte a la interpretación tabulada del esquema de análisis.

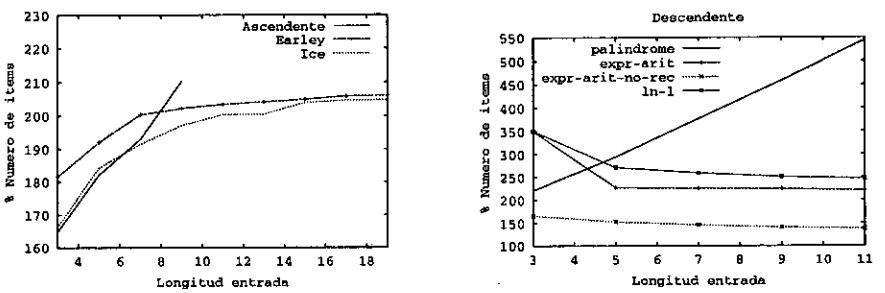


Figura 9.11: Coste de la extensión al análisis parcial

Continuando con los experimientos anteriores, hemos repetido los análisis de la subsección 9.1.1, pero en esta ocasión sobre un juego de prueba compuesto por conjuntos de cadenas de entradas generados aleatoriamente. De esta forma comprobamos el comportamiento de los esquemas parciales ante cadenas erroneas. La figura 9.12

muestra los resultados para cada una de las gramáticas, que se resumen en la figura 9.13. En dichas figuras se observa cómo los esquemas con una estrategia mixta continúan siendo los más eficaces, destacando el sistema ICE sobre el algoritmo de Earley. Por su parte, el esquema ascendente presenta una vez más un crecimiento exponencial, mientras que el descendete presenta un crecimiento lineal, un tanto irregular debido a la aleatoriedad de las cadenas de entrada.

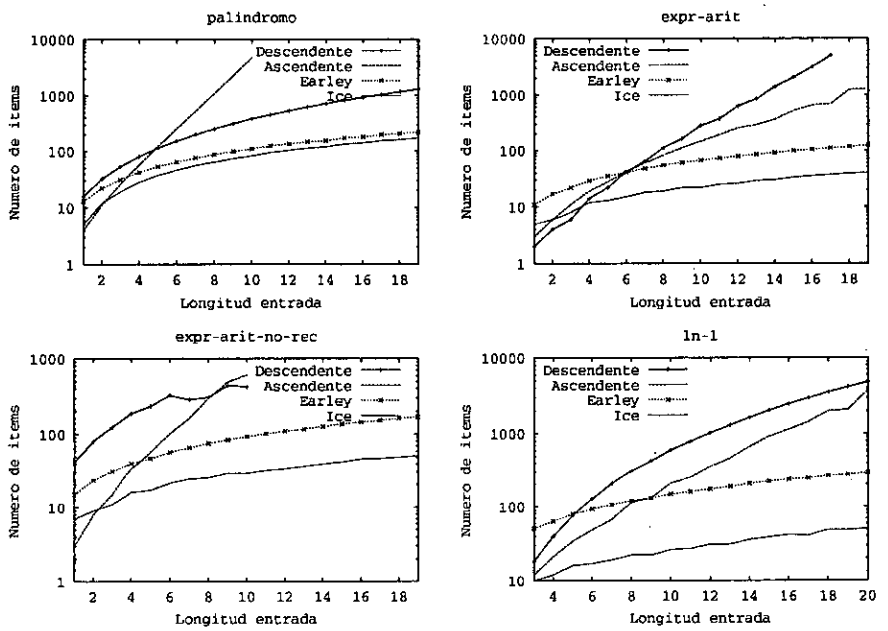


Figura 9.12: Análisis parcial de cadenas erróneas

9.5 Tabulación

En este apartado realizaremos un breve experimento puesto que los resultados de apartados anteriores ya destacaban el beneficio derivado de la aplicación de las técnicas de tabulación. Para ello emplearemos analizadores con una estrategia similar: SLR Inference, AID e ICE. Todos ellos se basan en el uso de un autómata de tipo LR como control estático, pero únicamente ICE incorpora técnicas de tabulación. La gramática empleada contiene únicamente dos cláusulas:

$$\text{expr}(\text{plus}(X,Y)) \rightarrow \text{expr}(X), [+], \text{expr}(Y)$$
$$\text{expr}(\text{num}) \rightarrow [a]$$

Y las cadenas de entrada analizadas son de la forma:

$$a\,(+\,a)^i$$

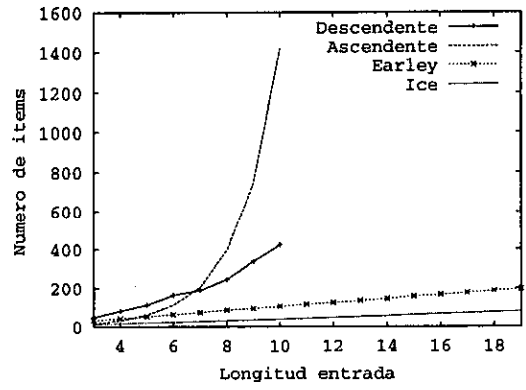


Figura 9.13: Resumen: análisis parcial de cadenas erróneas

El número de items generados en cada sistema para distintos valores de i se muestra a continuación y el gráfico correspondiente, en la figura 9.14. Los sistemas sin tabulación equivalen a un entorno dinámico S^T, mientras que el sistema ICE emplea un entorno dinámico S^1. Como se observa, en S^T la complejidad temporal es exponencial.

	1	2	3	4	5	6	7	8	9	10	11
AID	6	15	39	109	325	1018	3306	11028	37548	129926	455510
SLR inf	13	34	91	259	781	2464	8041	26917	91891	318637	1119031
ICE	8	15	25	38	54	73	95	120	148	179	213

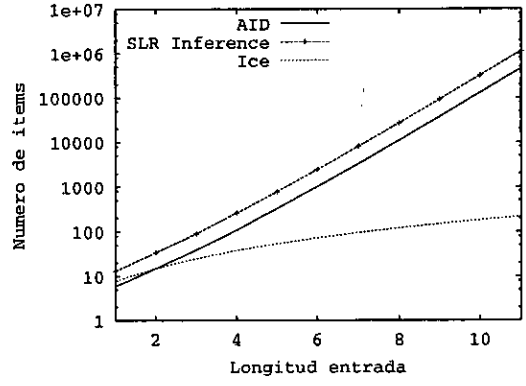


Figura 9.14: Resultados en S^T y S^1

Por último, si nos centramos únicamente en el sistema ICE, para los análisis descritos en la sección 9.3, también se han calculado:

- Número de items generados en el entorno dinámico S^1, y una estimación de una cota inferior para el número que hubiese sido necesario generar en S^2 y S^T.
- Número de transiciones dinámicas que sería necesario generar si se empleasen técnicas de tabulación sin sincronización.

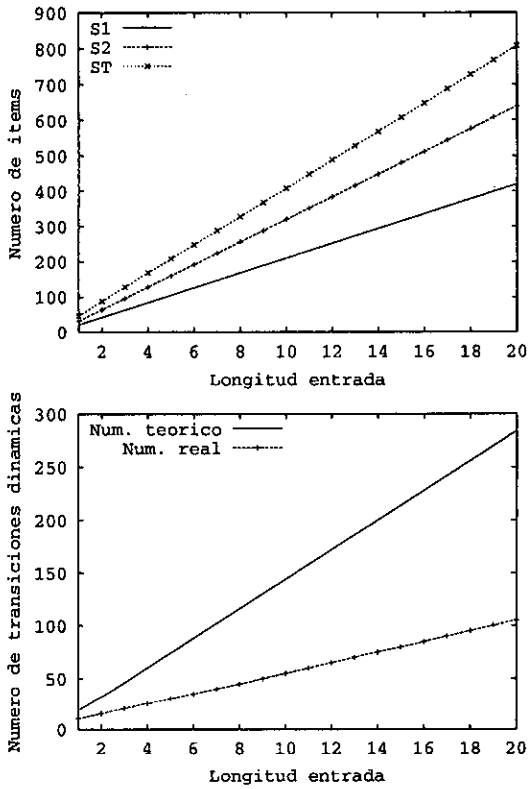


Figura 9.15: Rendimiento de la tabulación en ICE

- Número de transiciones dinámicas generadas con la técnica de tabulación con sincronización.

Los resultados se muestran en la figura 9.15, y vienen a corroborar la eficacia de las técnicas de tabulación implantadas.

CAPÍTULO 10

CONCLUSIONES Y TRABAJO FUTURO

En este capítulo expondremos las conclusiones sobre el trabajo realizado y trazaremos las posibles líneas a seguir para su mejora y ampliación.

10.1 Conclusiones

En el presente trabajo de tesis doctoral hemos introducido y desarrollado de forma progresiva diversas técnicas de análisis sintáctico con el objeto de mejorar la eficiencia de los analizadores. Para conseguir una aproximación utilizable en la práctica, hemos centrado nuestro trabajo en tres ejes principales:

- Reducir el espacio búsqueda, evitando la realización de cálculos inútiles.
- Incorporar técnicas de programación dinámica, mejorando la compartición de cálculos redundantes.
- Extender el dominio de terminación en el análisis de formalismos gramaticales lógicos.

El algoritmo de Earley es un buen punto de partida para derivar otros algoritmos de análisis más complejos. En este sentido, hemos derivado un algoritmo GLR para el análisis de gramáticas independientes del contexto arbitrarias usando varios esquemas intermedios, aplicando en cada paso transformaciones simples e intuitivas. El resultado [11] es un algoritmo en programación dinámica que se integra de forma natural en el marco común propuesto por Lang, con una complejidad temporal de $\mathcal{O}(n^3)$ en el peor caso y un comportamiento lineal en el mejor de los casos, tal y como indican los resultados experimentales. Esta eficiencia ha sido obtenida tanto por la eliminación de cálculos redundantes, como por el alto grado de compartición obtenido.

El análisis de formalismos de evaluación lógica presenta dificultades tanto por los problemas de terminación como de eficacia. La técnica propuesta puede ser extendida para tratar formalismos gramaticales que contienen un esqueleto IC. Tal y como demuestra De la Clergerie [179], existe una extensión directa de las técnicas de análisis sintáctico independiente del contexto basado en autómatas al análisis de cláusulas de Horn. El formalismo gramatical basado en cláusulas de Horn más conocido son las GCDs [104, 105]. Podemos entender las GCDs como un esqueleto IC con atributos

asociados a cada símbolo gramatical. En esta línea, hemos descrito una implementación de un analizador para GCDs basado en la especificación LALR(1) del sistema ICE [12].

En un sentido más amplio, hemos descrito una estrategia para implementar analizadores para GCDs. Nuestro marco operacional son los ALPs en programación dinámica. La arquitectura es un esquema de evaluación ascendente optimizado con un control predictivo dado por un autómata LALR(1). El sistema asegura tanto un tratamiento óptimo de la compartición de cálculos como la completud y corrección para las GCDs sin símbolos de función.

La propuesta desarrollada para el sistema ICE persigue un compromiso entre la compartición en el bosque de análisis y la compartición de cálculos. En este sentido, el uso de un esquema de evaluación de naturaleza ascendente posibilita la introducción de un entorno dinámico S^1, que garantiza una compartición óptima tanto del espacio de búsqueda como del proceso de cálculo. Dentro del entorno S^1, agrupar los items en conjuntos y sincronizar los mismos nos permite reducir la generación de transiciones dinámicas. En lo que respecta al bosque de análisis, la elección de un grafo Y/O nos asegura la mejor compartición dados un esquema de análisis y un entorno dinámico cualesquiera.

Recapitulando, hemos desbrozado aspectos teóricos y metodológicos de un analizador de GCDs, discutiendo la posibilidad de explotar de forma eficiente las herramientas formales de análisis previamente desarrolladas en la teoría del análisis independiente del contexto.

Un inconveniente serio en los formalismos lógicos es el problema de la no-terminación. Hemos abordado este problema sobre la base del analizador presentado en el sistema ICE, sin que ello suponga una penalización apreciable de su rendimiento. Es decir, hemos tratado dos causas posibles de no-terminación, sin detrimento de los casos en que no existen estructuras cíclicas, tanto en la eficiencia como en el grado de compartición de cálculos:

- El primer problema es la detección, representación y manejo de términos cíclicos. Este problema, estrechamente relacionado con la programación lógica, es común en aquellos algoritmos de unificación que implican el recorrido de estructuras potencialmente infinitas. Una solución, muy extendida por razones de eficiencia, consiste en prohibir y evitar este tipo de estructuras. En nuestro caso hemos adaptado el algoritmo de unificación para su tratamiento haciendo uso de las propiedades de compartición de la programación dinámica, reduciendo de esta forma la complejidad computacional asociada.

- El segundo problema implica el tratamiento de derivaciones cíclicas y guarda una estrecha relación con la estrategia de análisis. Aquí hemos enfocado la cuestión desde el punto de vista de la programación dinámica, explotando la sincronización y ordenación del dominio, mejorando la técnica de tabulación y explotando la analogía con el análisis independiente del contexto clásico.

En el marco del análisis natural, la separación de la etiquetación en una fase previa reduce el coste del análisis sintáctico. Esta decisión se justifica simplemente por

la complejidad flexiva de las lenguas naturales. La separación abordada evita gran cantidad de predicciones tanto en estrategias con predicción dinámica como estática, respectivamente durante el análisis o durante la creación del control estático.

En cuanto al análisis parcial, hemos enfocado el problema desde un marco descriptivo basado en la noción de esquema de deducción, sobre el que hemos desarrollado extensiones para los esquemas de análisis tratados a lo largo de la memoria. El empleo de este marco descriptivo común nos ha permitido comparar de forma transparente la eficiencia de dichas extensiones.

En lo que respecta a la aplicabilidad práctica de la programación dinámica en el tratamiento de formalismos gramaticales complejos hemos revisado, dentro de un marco descriptivo común, las propiedades computacionales de varios de los esquemas más conocidos en el análisis de formalismos en el *continuum* de formalismos de tipo Horn, que va desde las GICs hasta las GCDs. Hemos experimentado con el problema de la compartición de cálculos en un entorno en el cual el no determinismo es habitual, mostrando el interés práctico de la programación dinámica independientemente del esquema de análisis sintáctico empleado.

Como resultado adicional, en los ejemplos tratados hemos observado la superioridad de los esquemas de análisis ascendente con algún tipo de control estático frente a los descendentes puros o ascendentes con control dinámico. En nuestro caso particular, la indexación del análisis permite compensar la diferencia con aquellos casos particulares en los que la técnica de retroceso maneja de forma eficiente el espacio de búsqueda. Finalmente, los resultados prácticos parecen mostrar un rendimiento por encima de lo previsto en las estimaciones teóricas.

10.2 Trabajo futuro

Los resultados del trabajo presentado en la memoria que nos ocupa no terminan en este punto sino que marcan diversas líneas en las que continuar el esfuerzo de investigación y desarrollo planteado. Entre estas líneas resaltaremos:

- Implantación de un entorno integrado de desarrollo que facilite las tareas de creación, depuración y puesta en producción de las gramáticas desde el punto de vista del usuario.

- Inclusión de las estructuras de rasgos [31] como mecanismo de generalización de las GICs.

- Mejorar la obtención del esqueleto IC de las gramáticas de manera que el control estático resultante sea más eficaz, por ejemplo instanciado parcialmente los símbolos de la gramática [32].

- Extensión de los formalismos gramáticales con resultados provenientes de la programación lógica:

  - El tratamiento de la negación [7, 41] por parte del analizador permite la extensión de los formalismos gramaticales con características como la inclusión de constituyentes por defecto.
  - Otros resultados como la satisfacción de restricciones [136, 80] y las lógicas de segundo orden y de orden superior [63, 137] permitirán acercar el analizador sintáctico a formalismos gramaticales como HPSG [106].

- Desarrollo e implantación de una interfaz que permita el intercambio de información y la cooperación con analizadores semáticos.

10.3 Trabajos relacionados

Dentro del propio grupo de investigación CoLe, se mantienen diversas líneas de investigación de forma paralela a la descrita en el presente trabajo:

- Aplicación de las técnicas de análisis sintáctico consideradas al caso de las *gramáticas lineales de índices* y *gramáticas de adjunción de árboles* [8].
- Adaptación de los algoritmos de análisis sintáctico para la reparación automática de errores en la cadena de entrada [158, 159].
- Desarrollo de algoritmos para el encaje aproximado de árboles y su integración de los algoritmos de análisis sintáctico [160].
- Análisis morfosintáctico [56].
- Aplicación de las técnicas anteriores, y otras nuevas, a la recuperación de información [20, 19, 163].

APÉNDICE A

SDGs ADICIONALES

En este apéndice se recopilan los SDGs que no se incluyeron en el desarrollo de la memoria por motivos de brevedad.

A.1 Palabras desconocidas

En esta sección incluimos los SDGs correspondientes a la transformación del SDG para el análisis con palabras desconocidas del algoritmo de Earley al SDG de los analizadores LR(k). Sobre la notación empleada, recordaremos que $w_1 \cdots w_n$ es la cadena de entrada y *desc* representa una palabra desconocida. Los SDGs se muestran en las figuras A.1, A.2 y A.3.

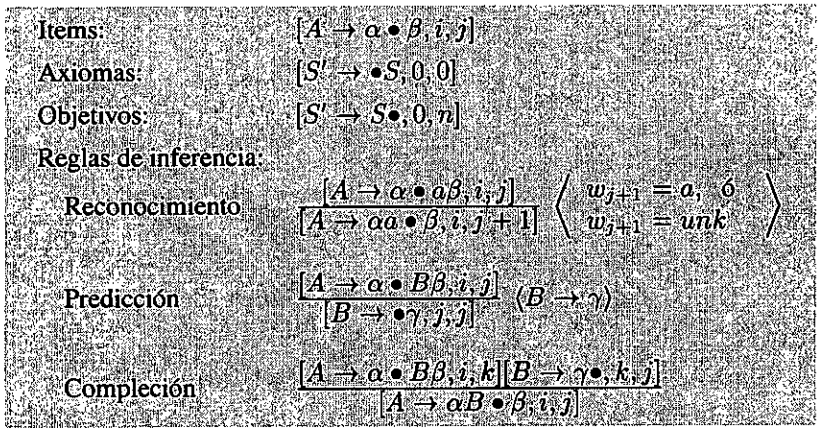


Figura A.1: SDG del algoritmo de Earley con palabras desconocidas

A.2 Análisis sintáctico parcial de GICs

En esta sección incluimos los SDGs correspondientes a la transformación del SDG para el análisis parcial del algoritmo de Earley al SDG de los analizadores LR(k). Sobre la notación empleada, recordaremos que $w_1 \cdots w_n$ es la cadena de entrada y $\mathcal{S}$ el conjunto de símbolos iniciales. Los SDGs se muestran en las figuras A.4, A.5 y A.6.

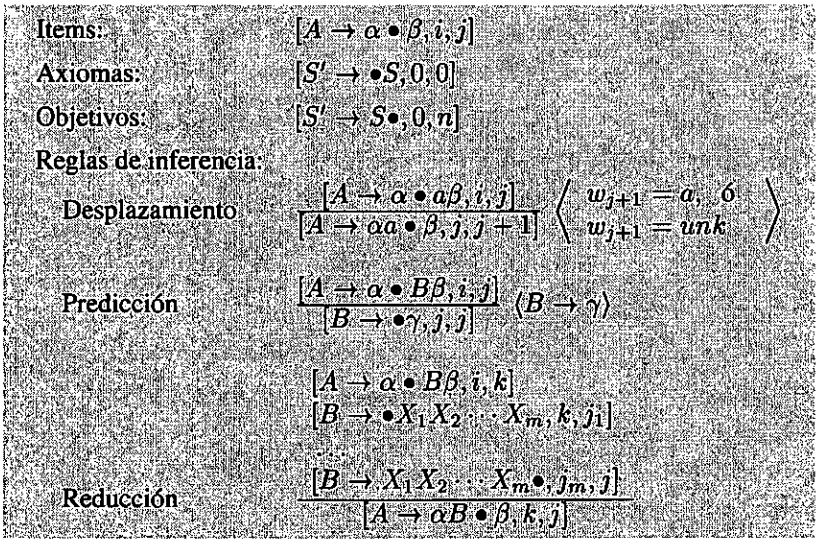


Figura A.2: SDG del algoritmo LR(0) sin control finito y palabras desconocidas

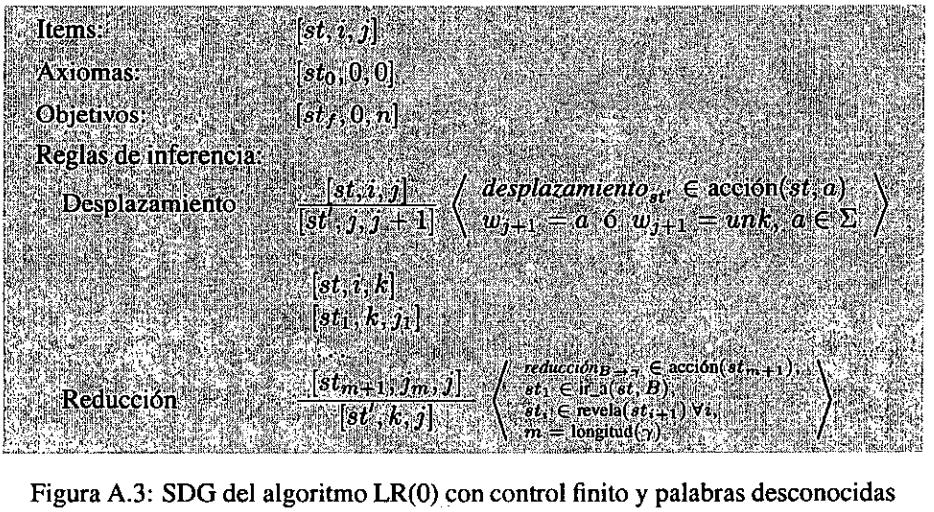


Figura A.3: SDG del algoritmo LR(0) con control finito y palabras desconocidas

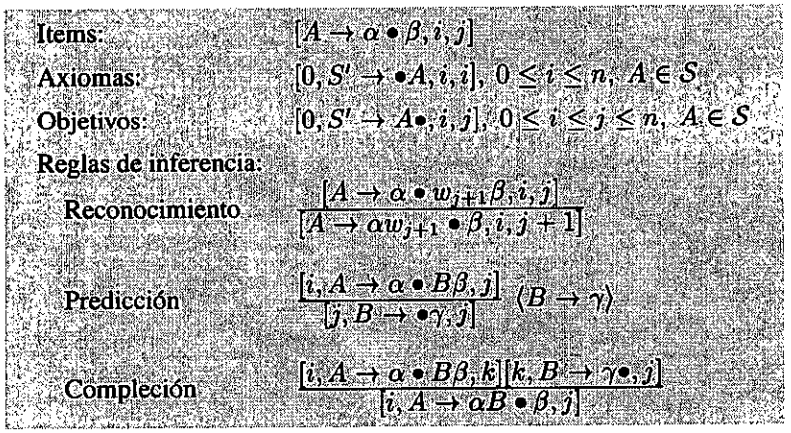


Figura A.4: SDG del algoritmo de Earley para análisis parcial

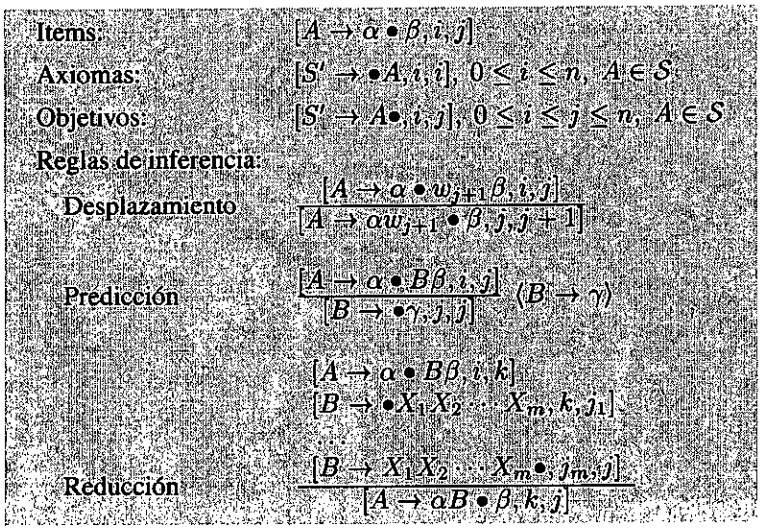


Figura A.5: SDG del algoritmo LR(0) sin control finito para análisis parcial

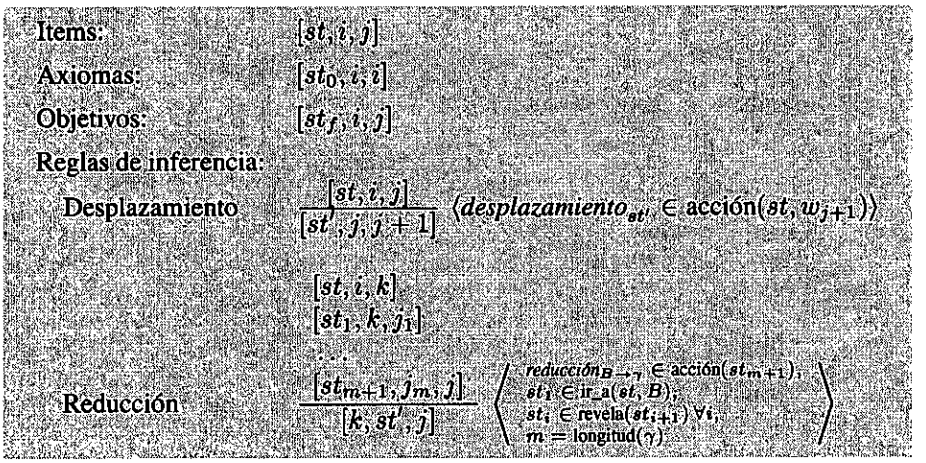


Figura A.6: SDG del algoritmo LR(0) con control finito para análisis parcial

APÉNDICE B

PUBLICACIONES DEL AUTOR

2002

1. Fco. Mario Barcala, Eva M. Domínguez, Miguel A. Alonso, David Cabrero, Jorge Graña, Jesús Vilares, Manuel Vilares, Guillermo Rojo, M. Paula Santalla, y Susana Sotelo. **El sistema ERIAL: LEIRA, un entorno para RI basado en PLN**. En *Actas de las I Jornadas de Tratamiento y Recuperación de Información (JOTRI 2002)*, Valencia, 2002.

2. Fco. Mario Barcala, Eva M. Domínguez, Miguel A. Alonso, David Cabrero, Jorge Graña, Jesús Vilares, Manuel Vilares, Guillermo Rojo, M. Paula Santalla, y Susana Sotelo. **Una aplicación de RI basada en PLN: el proyecto ERIAL**. En *Actas de las I Jornadas de Tratamiento y Recuperación de Información (JOTRI 2002)*, Valencia, 2002.

2001

3. David Cabrero Souto, Jesús Vilares Ferro, y Manuel Vilares Ferro. **Dynamic programming of partial parses**. En *Actas de las Primeras Jornadas sobre Programación y Lenguajes*, páginas 63–76, Almagro (Ciudad Real), España, 2001.

4. Jesús Vilares Ferro, David Cabrero Souto, y Miguel A. Alonso Pardo. **Applying productive derivational morphology to term indexing of Spanish texts**. En Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, volumen 2004 de *Lecture Notes in Computer Science*, páginas 336–348. Springer-Verlag, Berlin-Heidelberg-New York, 2001. ISBN 3-540-41687-0.

5. Manuel Vilares, Jesús Vilares, y David Cabrero. **Dynamic programming of partial parses**. En Galia Angelova, Kalina Bontcheva, Ruslan Mitkov, Nicolas Nocolov, y Nokolai Nikolov, editores, *EuroConference Recent Advances in Natural Language Processing*, páginas 291–293, Tzigov Chark, Bulgaria, 2001. ISBN 954-90906-1-2.

6. Manuel Vilares, David Cabrero, y Miguel A. Alonso. **On non-termination in DCGs**. En Bruce W. Watson y Derick Wood (eds.), editores, *Proc. of the*

6th Conference on Implementations and Applications of Automata, páginas 267–278, Pretoria, South Africa, 2001.

7. Jesús Vilares Ferro, David Cabrero Souto, y Miguel A. Alonso Pardo. **Generación automática de familias morfológicas mediante morfología derivativa productiva**. *Procesamiento del Lenguaje Natural*, 27:181–188, Septiembre 2001.

2000

8. Manuel Vilares, David Cabrero, y Miguel A. Alonso. **On non-termination in DCGs**. En Alexander Gelbukh, editor, *International Conference CICLing-2000, Conference on Intelligent text processing and Computational Linguistics, Proceedings*, páginas 150–165. Centro de Investigación en Computación of the Instituto Politécnico Nacional, Mexico City, Mexico, 2000. ISBN 970-18-4206-5.

9. Manuel Vilares, David Cabrero, y Franciso J. Ribadas. **On integration of parsing and tree matching schemes**. En Alexander Gelbukh, editor, *International Conference CICLing-2000, Conference on Intelligent text processing and Computational Linguistics, Proceedings*, páginas 257–272. 2000. ISBN 970-18-4206-5.

10. Manuel Vilares Ferro, David Cabrero Souto, y Francisco J. Ribadas Pena. **Computing the editing distance in shared forest**. En *Proc. of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, páginas 323–324, Trento, Italy, February 2000.

11. Miguel A. Alonso Pardo, David Cabrero Souto, y Manuel Vilares Ferro. **Generalized LR parsing for extensions of context-free grammars**. En Nicolas Nicolov y Ruslan Mitkov, editores, *Recent Advances in Natural Language Processing II*, volumen 189 de *Current Issues in Linguistic Theory*, páginas 81–92. John Benjamins Publishing Company, Amsterdam & Philadelphia, 2000. ISBN 90-272-3695-X.

1999

12. Miguel A. Alonso Pardo, David Cabrero Souto, Eric de la Clergerie, y Manuel Vilares Ferro. **Tabular algorithms for TAG parsing**. En *Proc. of EACL'99, Ninth Conference of the European Chapter of the Association for Computational Linguistics*, páginas 150–157, Bergen, Norway, Jun 1999. ACL.

13. Manuel Vilares, David Cabrero, y Miguel A. Alonso. **Dealing with non-termination in DCGs**. En *Proc. of CACIC'99*, Tandil, Buenos Aires, Argentina, 1999.

14. Manuel Vilares, David Cabrero, y Miguel A. Alonso. **Some questions about non-termination in DCGs**. En Maria Chiara Meo y Manuel Vilares Ferro (eds.), editores, *APPIA-GULP-PRODE'99 Joint Conference on Declarative Programming, Proceedings*, páginas 545–558, L'Aquila, Italy, 1999.

15. Manuel Vilares Ferro, Miguel A. Alonso Pardo, y David Cabrero Souto. **An operational model for parsing definite clause grammars with infinite terms**. En Alain Lecomte, François Lamarche, y Guy Perrier, editores, *Logical Aspects of Computational Linguistics*, volumen 1582 de *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin-Heidelberg-New York, 1999.

16. Manuel Vilares, David Cabrero, y Franciso J. Ribadas. **Pattern matching in shared forest**. En *Proc. of Sixth International Workshop on Natural Language Understanding and Logic Programming (NLULP'99)*, Las Cruces, New Mexico, USA, 1999.

17. Miguel A. Alonso Pardo, Eric de la Clergerie, y David Cabrero Souto. **Tabulation of automata for tree adjoining languages**. En *Proc. of the Sixth Meeting on Mathematics of Language (MOL 6)*, páginas 127–141, Orlando, Florida, USA, Jul 1999.

1998

18. Miguel A. Alonso Pardo, David Cabrero Souto, Eric de la Clergerie, y Manuel Vilares Ferro. **Algoritmos tabulares para el análisis de TAG**. *Procesamiento del Languaje Natural*, 23:157–164, September 1998.

19. Miguel A. Alonso Pardo, David Cabrero Souto, y Manuel Vilares Ferro. **Construction of efficient generalized LR parsers**. En Derick Wood y Sheng Yu, editores, *Automata Implementation*, volumen 1436 de *Lecture Notes in Computer Science*, páginas 7–24. Springer-Verlag, Berlin-Heidelberg-New York, 1998. ISBN 3-540-64694-9.

20. David Cabrero Souto. **Integración de herramientas para el análisis automático de los lenguajes naturales**. Tesis de licenciatura, Facultade de Informática, Universidade da Coruña, Corunna, Spain, July 1998.

21. Manuel Vilares Ferro, David Cabrero Souto, y Miguel A. Alonso Pardo. **Dynamic programming as frame for efficient parsing**. En *SCCC'98, XVIII International Conference of the Chilean Computer Science Society*, páginas 68–75. IEEE Computer Society Press, Los Alamitos, California, USA, 1998. November 9-14, Antofagasta, Chile, ISBN 0-8186-8616-2.

22. Manuel Vilares Ferro, Miguel Angel Alonso Pardo, Jorge Graña Gil, y David Cabrero Souto. **GALENA: Tabular DCG parsing for natural languages**. En *Proc. of First Workshop on Tabulation in Parsing and Deduction (TAPD'98)*, páginas 44–51, Paris, France, Apr 1998.

23. Manuel Vilares Ferro, David Cabrero Souto, y Miguel A. Alonso Pardo. **A comparison for unification-based parsers**. En M. Falaschi, J. L. Freire, y M. Vilares, editores, *Proc. of APPIA-GULP-PRODE'98 Joint Conference on Declarative Programming*, páginas 113–123, A Coruña, Spain, Jul 1998.

24. Manuel Vilares Ferro, David Cabrero Souto, y Miguel A. Alonso Pardo. **Exploring parsing efficiency in computational linguistics**. En *Utrecht Congress on Storage & Computation in Linguistics*, página 65, Utrecht, The Netherlands, October 1998.

25. Manuel Vilares Ferro, Jorge Graña Gil, T. Araujo, David Cabrero Souto, y I. Diz. **A tagger environment for galician**. En *Proc. of Workshop on Language Resources for European Minority Languages*, Granada, Spain, May 1998.

26. Eric de la Clergerie, Miguel A. Alonso Pardo, y David Cabrero Souto. **A tabular interpretation of bottom-up automata for TAG**. En *Proc. of Fourth International Workshop on Tree-Adjoining Grammars and Related Frameworks (TAG+4)*, páginas 42–45, Philadelphia, PA, USA, August 1998.

27. David Cabrero Souto, Manuel Vilares Ferro, Luis Docampo Gutiérrez, y Susana Sotelo Docío. **Web-surfing the lexicon**. En *Proc. of Workshop on Distributing and Accessing Linguistic Resources*, páginas 40–46, Granada, Spain, May 1998.

1997

28. Miguel Angel Alonso Pardo, David Cabrero Souto, y Manuel Vilares Ferro. **Construction of efficient generalized LR parsers**. En *Proc. of Second International Workshop on Implementing Automata (WIA'97)*, páginas 131–140, London, Ontario, Canada, Sep 1997.

29. Miguel Angel Alonso Pardo, David Cabrero Souto, y Manuel Vilares Ferro. **A new approach to the construction of Generalized LR parsing algorithms**. En Ruslan Mitkov, Nicolas Nicolov, y Nikolai Nikolov, editores, *Proc. of Recent Advances in Natural Language Processing (RANLP'97)*, páginas 171–178, Tzigov Chark, Bulgaria, Sep 1997.

30. Manuel Vilares Ferro, Miguel Angel Alonso Pardo, y David Cabrero Souto. **Efficient parsing of fixed-mode DCGs**. En Geert-Jan M. Kruijff, Glyn V. Morrill, y Richard T. Oehrle, editores, *Proc. of Formal Grammar'97*, Aix-en-Provence, France, September 1997.

31. Manuel Vilares Ferro, Miguel Angel Alonso Pardo, y David Cabrero Souto. **An operational model for parsing fixed-mode DCGs**. En *Proc. of Logical Aspects of Computational Linguistics (LACL'97)*, páginas 61–64, Nancy, France, Sep 1997.

32. Manuel Vilares Ferro, David Cabrero Souto, y Miguel Angel Alonso Pardo. **An approach to infinite term traversal in DCGs**. En M. Falaschi, M. Navarro, y A. Policriti, editores, *Proc. of APPIA-GULP-PRODE'97 Joint Conference on Declarative Programming*, páginas 307–317, Grado, Italy, Jun 1997.

1996

33. Manuel Vilares Ferro, Miguel Angel Alonso Pardo, y D. Cabrero Souto. **An experience on natural language parsing**. En C. Martín Vide, editor, *Lenguajes Naturales y Lenguajes Formales*, volumen XII, páginas 555–562, Barcelona, Spain, September 1996. PPU. ISBN 84-477-0575-7.

34. Manuel Vilares Ferro, Miguel Angel Alonso Pardo, y David Cabrero Souto. **Autómatas lógicos y lenguaje natural**. En *V Congreso Iberoamericano de Inteligencia Artificial. Iberamia'96*, páginas 341–351, Puebla, Clolula, México, November 1996. Editorial Limusa, S. A. de C. V. ISBN 968-18-5429-2.

35. David Cabrero Souto, Fidel Cacheda, y Alberto Pan. **Entorno gráfico para el análisis automático del lenguaje natural**. En *Actas del IV Congreso Científico de Alumnos*, Lugo, España, 1996.

36. David Cabrero Souto, Fidel Cacheda, y Alberto Pan. **Un entorno de generación de etiquetadores**. En *Actas da Primeira Xuntanza de Xoves Investigadores*, Gandarío, A Coruña, España, 1996.

Bibliografía

[1] Steven Abney. Parsing by chunks. En Robert Berwick, Steven Abney, y Carol Tenny, editores, *Principle-based parsing*. 1990.

[2] Steven Abney. Partial parsing, 1994. A tutorial presented at ANLP-94, Stuttgart, DE.
http://www.sfs.nphil.uni-tuebingen.de/~abney.

[3] A.V. Aho y J.D. Ullman. *The Theory of Parsing, Translation and Compiling*, volumen 1-2. Prentice-Hall, Englewood Cliff, New Jersey, U.S.A., 1973.

[4] H. Ait-Kaci. *Warren's Abstract Machine: A Tutorial Reconstruction.* MIT-Press, 1991.

[5] Hassan Aït-Kaci. Disjunctive ψ-term unification (summary). En Hans-Jürgen Bürckert y Werner Nutt, editores, *Extended Abstracts of the Third International Workshop on Unification*, SEKI-Report SR-89-17, páginas 179–182, Lambrecht, FRG, June 26–28, 1989. Fachbereich Informatik, Universität Kaiserslautern.

[6] Luc Albert, Rafael Casas, y François Fages. Average-case analysis of unification algorithms. *Theoretical Computer Science*, 113(1):3–34, 24 May 1993.

[7] José Júlio Alferes, Carlos Viegas Damásio, y Luís Pereira. SLX-A top-down derivation procedure for progams with explicit negation. En Maurice Bruynooghe, editor, *Logic Programming - Proceedings of the 1994 International Symposium*, páginas 424–438, Massachusetts Institute of Technology, 1994. The MIT Press.

[8] Miguel A. Alonso Pardo. *Interpretación tabular de autómatas para lenguajes de adjunción de árboles*. PhD thesis, Departamento de Computación, Universidade da Coruña, A Coruña, Spain, September 2000.

[9] Miguel A. Alonso Pardo, David Cabrero Souto, Eric de la Clergerie, y Manuel Vilares Ferro. Algoritmos tabulares para el análisis de TAG. *Procesamiento del Languaje Natural*, 23:157–164, September 1998.

[10] Miguel A. Alonso Pardo, David Cabrero Souto, Eric de la Clergerie, y Manuel Vilares Ferro. Tabular algorithms for TAG parsing. En *Proc. of EACL'99, Ninth Conference of the European Chapter of the Association for Computational Linguistics*, páginas 150–157, Bergen, Norway, Jun 1999. ACL.

[11] Miguel A. Alonso Pardo, David Cabrero Souto, y Manuel Vilares Ferro. Construction of efficient generalized LR parsers. En Derick Wood y Sheng Yu, editores, *Automata Implementation*, volumen 1436 de *Lecture Notes in Computer Science*, páginas 7–24. Springer-Verlag, Berlin-Heidelberg-New York, 1998. ISBN 3-540-64694-9.

[12] Miguel A. Alonso Pardo, David Cabrero Souto, y Manuel Vilares Ferro. Generalized LR parsing for extensions of context-free grammars. En Nicolas Nicolov y Ruslan Mitkov, editores, *Recent Advances in Natural Language Processing II*, volumen 189 de *Current Issues in Linguistic Theory*, páginas 81–92. John Benjamins Publishing Company, Amsterdam & Philadelphia, 2000. ISBN 90-272-3695-X.

[13] Miguel A. Alonso Pardo, Eric de la Clergerie, Jorge Graña Gil, y Manuel Vilares Ferro. New tabular algorithms for LIG parsing. En *Proc. of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, páginas 29–40, Trento, Italy, Feb 2000.

[14] Miguel A. Alonso Pardo, Jorge Graña Gil, y Manuel Vilares Ferro. Nuevos algoritmos tabulares para el análisis de LIG. *Procesamiento del Lenguaje Natural*, 25:7–14, Sep 1999.

[15] Miguel Angel Alonso Pardo, David Cabrero Souto, y Manuel Vilares Ferro. Construction of efficient generalized LR parsers. En *Proc. of Second International Workshop on Implementing Automata (WIA'97)*, páginas 131–140, London, Ontario, Canada, Sep 1997.

[16] Miguel Angel Alonso Pardo, David Cabrero Souto, y Manuel Vilares Ferro. A new approach to the construction of Generalized LR parsing algorithms. En Ruslan Mitkov, Nicolas Nicolov, y Nikolai Nikolov, editores, *Proc. of Recent Advances in Natural Language Processing (RANLP'97)*, páginas 171–178, Tzigov Chark, Bulgaria, Sep 1997.

[17] Krzysztof R. Apt y Alessandro Pellegrini. On the occur-check-free PROLOG programs. *ACM Transactions on Programming Languages and Systems*, 16(3):687–726, May 1994.

[18] François Bancilhon, David Maier, Yehoshua Sagiv, y Jeffrey D. Ullman. Magic sets and other strange ways to implement logic programs (extended abstract). En ACM, editor, *PODS'86. Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 24–26, 1986, Cambridge, MA*, páginas 1–15, New York, NY 10036, USA, 1986. ACM Press.

[19] Fco. Mario Barcala, Eva M. Domínguez, Miguel A. Alonso, David Cabrero, Jorge Graña, Jesús Vilares, Manuel Vilares, Guillermo Rojo, M. Paula Santalla, y Susana Sotelo. El sistema ERIAL: LEIRA, un entorno para RI basado en PLN. En *Actas de las I Jornadas de Tratamiento y Recuperación de Información (JOTRI 2002)*, Valencia, 2002.

[20] Fco. Mario Barcala, Eva M. Domínguez, Miguel A. Alonso, David Cabrero, Jorge Graña, Jesús Vilares, Manuel Vilares, Guillermo Rojo, M. Paula Santalla, y Susana Sotelo. Una aplicación de RI basada en PLN: el proyecto ERIAL. En *Actas de las I Jornadas de Tratamiento y Recuperación de Información (JOTRI 2002)*, Valencia, 2002.

[21] F. P. Barthélemy y E. Villemonte de la Clergerie. Building tabular interpretations for (generalized) push-down automata. Submitted to JICSLP'96, Feb. 1996.

[22] John Bear. A breadth-first parsing model. *IJCAI-83*, páginas 696–698, 1983.

[23] R.E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, U.S.A., 1957.

[24] Manuel E. Bermudez y George Logothetis. Simple computation of LALR(1) lookahead sets. *Information Processing Letters*, 31(5):233–238, June 1989.

[25] S. Billot y B. Lang. The structure of shared forest in ambiguous parsing. En *Proc. 27th Annual Meeting of the ACL*, 1989.

[26] R. S. Boyer y J. S. Moore. The sharing of structure in theorem-proving programs. En B. Meltzer y Donald Michie, editores, *Machine Intelligence*, páginas 101–116. Edinburgh University Press, 1972.

[27] T. Briscoe y J. Carroll. Generalized probabilistic LR parsing of natural language (corpora) with unification–based grammars. En *Computational Linguistics*, volumen 19(1), páginas 25–29. 1993.

[28] David Cabrero Souto. Integración de herramientas para el análisis automático de los lenguajes naturales. Tesis de licenciatura, Facultade de Informática, Universidade da Coruña, Corunna, Spain, July 1998.

[29] David Cabrero Souto, Jesús Vilares Ferro, y Manuel Vilares Ferro. Dynamic programming of partial parses. En *Actas de las Primeras Jornadas sobre Programación y Lenguajes*, páginas 63–76, Almagro (Ciudad Real), España, 2001.

[30] N. Calzolari y J. McNaught. Eagles document eag-eb-fr-1. Technical report, Expert Advisory Group on Language Engineering Standars, 1996.

[31] B. Carpenter. *The Logic of Typed Feature Structures*. Número 32 in Cambridge Tracts in Theorical Computer Science. Cambridge University Press, Camdridge, England, 1992.

[32] Bob Carpenter. Compiling typed attribute-value logic grammars. En *Proceedings of the Third International Workshop on Parsing Technologies*, páginas 39–48, Tilburg (The Netherlands), Durbuy (Belgium), 1993.

[33] John Andrew Carroll. *Practical Unificaction-based Parsing of Natural Language*. PhD thesis, University of Cambridge, 1993.

[34] J.-C. Chappelier y M. Rajman. A practical bottom-up algorithm for on-line parsing with stochastic context-free grammars. Technical Report 98/284, Département Informatique, EPFL, Lausanne (Switzerland), jul 1998.

[35] Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. En *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)*, páginas 598–603, Menlo Park, July 27–31 1997. AAAI Press.

[36] Noam Chomsky. On certain formal properties of grammars. *Information and Control*, 1:91–112, 1958.

[37] A. Colmerauer, H. Kahoui, y M. van Caneghem. Un systéme de communication homme-machine en français. Technical report, Gruppe Intelligence Artificielle, Université Aix-Marseille II, 1973.

[38] Alain Colmerauer. Les grammaires de metamorphose. Technical report, Groupe d'Intelligence Artificielle, Université de Marseille-Luminy, Noviembre 1975. Translated into English as Colmerauer [39].

[39] Alain Colmerauer. Opening the Prolog-III universe. *Byte Magazine*, 12(9):177–182, Agosto 1987.

[40] R. Corbett, R.M. Stallman, y W. Hansen. BISON, *Reference Manual*. Free Software Foundation, Inc., 675 Mass Avenue, Cambridge, MA 02139, U.S.A., 1.25 edición, 1996.

[41] C. Viegas Damásio y L. Moniz Pereira. A general tabulation procedure for extended constraint logic programs. En *Proceedings of Tabulation in Parsing and Deduction (TAPD'98)*, páginas 67–74, Paris (FRANCE), April 1998.

[42] Pierre Deransart y Jan Maluszyński. *A Grammatical View of Logic Programming*. MIT Press, 1993.

[43] Franklin L. DeRemer. Simple LR(k) grammars. *Communications of the ACM*, 14(7):453–460, July 1971.

[44] Franklin L. DeRemer y Thomas Pennello. Efficient computation of LALR(1) look-ahead sets. *ACM Transactions on Programming Languages and Systems*, 4(4):615–649, Octubre 1982.

[45] 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.

[46] Víctor J. Díaz Madrigal y Miguel A. Alonso Pardo. Comparing tabular parsers for tree adjoining grammars. En David S. Warren, Manuel Vilares, Leandro Rodríguez Liñares, y Miguel A. Alonso, editores, *Proc. of Tabulation in Parsing and Deduction (TAPD 2000)*, páginas 91–100, Vigo, Spain, September 2000.

[47] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, Febrero 1970.

[48] Jay Earley. *An Efficient Context-Free Parsing Algorithm.* PhD thesis, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, PA, 1968.

[49] Gregor Erbach. *Bottom-Up Earley Deduction for Preference-Driven Natural Language Processing (draft).* PhD thesis, Universität des Saarlandes, Sthulsatzenhausweg, Saarbrücken, 1995.

[50] R. Falcone, P. Insinnamo, y O. Stock. Island parsing and bidirectional charts. En *Proc.* 12^{th} *International Conference on Computational Linguistics*, Budapest, Hungary, 1984.

[51] M. Filgueiras. A PROLOG interpreter working with infinite terms. *Implementations of* PROLOG, 1984.

[52] Kurt Godden. Lazy unificación. En 28^{th} *Annual Metting of Association for Computational Linguistics*, páginas 180–187, Pittsburgh, 1990.

[53] J. Graña Gil, F. M. Barcala Rodríguez, y M. Alonso Pardo. Compilation methods of minimal acyclic finite-state automata for large dictionaries. En *Proc. of the Sixth International Conference on Implementation and Application of Automata (CIAA-2001)*, páginas 116–129, Pretoria, South Africa, 2001.

[54] Peter Graf. Substitution tree indexing. Technical report, Saarbruecken, 1994.

[55] Peter Graf. *Term indexing*, volumen 1053 de *Lecture Notes in Artificial Intelligence and Lecture Notes in Computer Science.* Springer-Verlag Inc., New York, NY, USA, 1996.

[56] Jorge Graña Gil. *Técnicas de Análisis Sintáctico Robusto para la Etiquetación del Lenguaje Natural.* PhD thesis, Facultad de Informática, Universidad de A Coruña, 2000.

[57] Jorge Graña Gil, Miguel Angel Alonso Pardo, y 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, Spain, 1994.

[58] Dennis Grinberg, John Lafferty, y Daniel Sleator. A robust parsing algorithm for LINK grammars. Technical Report CMU-CS-TR-95-125, Carnegie Mellon University, Pittsburgh, PA, 1995.

[59] Michael A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, Reading, 1 edición, 1978.

[60] David G. Hays. *Introduction to Computational Linguistics.* American Elsevier, New York, 1967.

[61] J. Heering, P.R.H. Hendriks, P. Klint, y J. Rekers. The syntax definition formalism SDF - reference manual. *SIGPLAN Notices*, 24(11):43–75, 1989.

[62] J. Hopcroft y J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, N. Reading, MA, 1980.

[63] Gérard Huet. *Résolution d'equations dans les langages d'ordre 1,2, ...,ω*. Thèse de Doctorat d'Etat, Université de Paris 7 (France), 1976.

[64] Fred Ives. Unifying view of recent LALR(1) lookahead set algorithms. *ACM SIGPLAN Notices*, 21(7):131–135, July 1986.

[65] I. Jacobs. *The* CENTAUR *1.2 Manual*. INRIA, Sophia-Antipolis, France, 1992.

[66] Christian Jacquemin. Recycling terms into a partial parser. En *Proc. of the 4th Conf. on Applied Natural Language Processing. Stuttgart, DE, 13–15 Oct 1994*, páginas 113–118, 1994.

[67] L. Karttunen. D-PATR: A development environment for unification-based grammars. Technical report, SRI International and Center for the Study of Language and Information, 1986.

[68] Lauri Karttunen y Kenneth Beesley. Two-level rule compiler. Technical Report ISTL–92–2, Xerox Corporation, Palo Alto Research Center, Palo Alto, California, 1992.

[69] T. Kasami. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, Massachusetts, 1965.

[70] Martin Kay. Algorithm schemata and data structures in syntactic processing. En Barbara J. Grosz, Karen Sparck Jones, y B. L. Webber, editores, *Readings in Natural Language Processing*, páginas 35–70. M. Kaufmann, Los Altos, CA, 1986. Originally published as a Xerox PARC technical report CSL-80-12, 1980.

[71] Vlado Keselj y Nick Cercone. A graph unification machine for NL parsing. Technical Report CS-2002-01, Department of Computer Science, University of Waterloo, January 2002. `http://www.cs.uwaterloo.ca/ vkeselj/`.

[72] Knuth. *The Art of Computer Programming, Vol. 1*. Addison-Wesley, Reading, 1973.

[73] Donald E. Knuth. On the translation of languages from left to right. *Information and Control*, 8(6):607–639, Diciembre 1965.

[74] K. Koskenniemi. Compilation of automata from morphological two-level rules. En *Proc. of the 5th Scandinavian Conference of Computational Linguistics*, páginas 143–149, Helsinki, Finland, 1985.

[75] Kimmo Koskenniemi. Two-level model for morphological analysis. En Alan Bundy, editor, *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Los Altos, California, 1983. William Kaufmann, Inc.

[76] Kimmo Koskenniemi. *Two-level Morphology: a general computational model for word-form recognition and production.* University of Helsinki, Helsinki, 1983.

[77] C. H. A. Koster. A Technique for Parsing Ambiguous Languages. En D. Siefkes, editor, *GI – 4. Jahrestagung*, volumen 26 de *Lecture Notes in Computer Science*, Berlin, Germany, 1975. Springer-Verlag.

[78] Brigitte Krenn y Christer Samuelsson. The linguist's guide to statistics, 1994. A compendium for a course in Statistical Approaches in Computational Linguisticsheld at the University of Saarland 1994–1995.

[79] Bent Bruun Kristensen y Ole Lehrmann Madsen. Methods for computing LALR(k) lookahead. *ACM Transactions on Programming Languages and Systems*, 3(1):60–82, January 1981.

[80] Gabriel Kuper, Leonid Libkin, y eds. Jan Paredaens. *Constraint Databases.* Springer Verlag, 1999.

[81] B. Lang. Complete evaluation of Horn Clauses, an automata theoretic approach. Technical Report 913, INRIA, Rocquencourt, France, 1988.

[82] B. Lang. Datalog automata. En C. Beeri, J. W. Schmidt, y U. Dayal, editores, *Proc. of the* 3^{rd} *International Conference on Data and Knowledge Bases*, páginas 389–404, Jerusalem, Israel, 1988. Morgan Kaufmann Pub.

[83] B. Lang. Towards a uniform formal framework for parsing. En M. Tomita, editor, *Current Issues in Parsing Technology*, páginas 153–171. Kluwer Academic Publishers, 1991.

[84] Bernard Lang. Deterministic techniques for efficient non-deterministic parsers. En Jacques Loeckx, editor, *Automata, Languages and Programming*, volumen 14 de *Lecture Notes in Computer Science*, páginas 255–269. Springer Verlag, August 1974.

[85] Bernard Lang. Parsing incomplete sentences. En *Proc. of the* 12^{th} *Int. Conf. on Computational Linguistics. Budapest, HU, 22–27 Aug 1988*, volumen 1, páginas 365–371, 1988.

[86] Rene Leermakers. A recursive ascent Earley parser. *Information Processing Letters*, 41(2):87–91, February 1992.

[87] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 2 edición, 1987.

[88] Alberto Martelli y Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, April 1982.

[89] Y. Matsumoto, H. Tanaka, H. Hirakawa, H. Miyoshi, y H. Yasukawa. BUP: A bottom-up parser embedded in PROLOG. *NEW GENERATION COMPUT. (JAPAN) ISSN: 0288-3635*, 1(2):145–58, 1983. QA 76 N48.

[90] Yuji Matsumoto y Ryoichi Sugimura. A parsing system based on logic programming. *IJCAI-87*, 2:671–674, 1987.

[91] Philippe McLean y R. Nigel Horspool. A faster Earley parser. En Tibor Gyimothy, editor, *Compiler Construction, 6th International Conference*, volumen 1060 de *Lecture Notes in Computer Science*, páginas 281–293, Linköping, Sweden, 24–26 April 1996. Springer.

[92] H. Meijer. The project on extended affix grammars at Nijmegen. *Attribute Grammars and their Applications, SLNC*, 461:130–142, 1990.

[93] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

[94] Frank Morawietz. Chart parsing and constraint propagation. En *Coling 2000*, Saarbrücken, 2000.

[95] Tsuneko Nakazawa. Construction of LR parsing tables for grammars using feature-based syntatic categories. En Georgia M. Green Jennifer Cole y Jerry L. Morgan (eds.), editores, *Linguistic and Computation*, volumen 52 de *Lecture Notes in Computer Science*, Stanford, CA, USA, 1995. ISBN 1-881526-81-X.

[96] M.-J. Nederhof. *Linguistic Parsing and Program Transformations*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1994.

[97] M.-J. Nederhof y J.J. Sarbo. Increasing the applicability of LR parsing. En *Proc. of Third International Workshop on Parsing Technologies*, páginas 187–201, 1993.

[98] N.J. Nilsson. *Artificial Intelligence. A New Synstesis*. Morgan Kaufmann Publishers, 1998.

[99] U. Nilsson. AID: an alternative implementation of DCGs. *New Generation Computing*, 4, 1986.

[100] Hans Jürgen Ohlbach. Abstraction tree indexing for terms. En Hans-Jürgen Bürckert y Werner Nutt, editores, *Extended Abstracts of the Third International Workshop on Unification*, SEKI-Report SR-89-17, páginas 131–135, Lambrecht, FRG, June 26–28, 1989. Fachbereich Informatik, Universität Kaiserslautern.

[101] Joseph C. H. Park, K. M. Choe, y C. H. Chang. A new analysis of LALR formalisms. *ACM Transactions on Programming Languages and Systems*, 7(1):159–175, January 1985.

[102] V. Paxson. FLEX: *Reference Manual. Release 2.5.3*. Free Software Foundation, Inc., 675 Mass Avenue, Cambridge, MA 02139, U.S.A., 1995.

[103] Fernando C.N. Pereira. A structure-sharing representation for unification-based grammar formalisms. En *Proc. of the 23rd Annual Meeting of the Association for Computational Linguistics. Chicago, IL, 8–12 Jul 1985*, páginas 137–144, 1985.

[104] Fernando C.N. Pereira y David H.D. Warren. Definite clause grammars for language analysis - A survey of the formalism and a comparison with augmented transition network. *Artificial Intelligence*, páginas 231–278, 1980.

[105] F.C.N. Pererira y D.H.D. Warren. Parsing as deduction. En *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, páginas 137–144, Cambridge (Massachusetts), 1983.

[106] Carl Pollard y 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.

[107] CRATER project. Corpus resources and terminology extraction (mlap-93/20), creation of a set of tools and resources for multilingual corpus linguistics work. Technical report, Lancaster University, UK; Computers, Communications and Visions, France; Universidad Autónoma de Madrid, Spain, 1993. `http://www.lllf.uam.es/~fernando/projects/es_corpus.html`.

[108] I. V. Ramakrishnan, Prasad Rao, Konstantinos Sagonas, Terrance Swift, y David S. Warren. Efficient access mechanisms for tabled logic programs. *The Journal of Logic Programming*, 1999.

[109] J. Rekers. *Parser Generation for Interactive Environments*. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, 1992.

[110] Philip Resnik. Probabilistic tree-adjoining grammar as a framework for statistical natural language procesing. En *Proceedings of the 14th COLING*, páginas 418–424, Nantes, France, 1992.

[111] G. Ritchie. On the generative power of two-level morphological rules. En *European Chapter of the ACL*, páginas 51–57, Manchester, 1989.

[112] G. Ritchie, D. Pulman, Stephen, A.W. Black, y G.J Russ ell. *Computational Morphology*. The MIT Press, Cambridge, Massachusetts, U.S.A., 1991.

[113] J.A. Robinson. A machine-oriented logic based on resolution principle. *Journal of the ACM*, 12(1):23–49, Enero 1965.

[114] E. Roche y Y. Schabes. Deterministic part-of-speech tagging with finite-state transducers. *Computational Linguistics*, 21(2):227–253, 1995.

[115] V. Rocio y J. G. Lopes. Partial parsing, deduction and tabling. En *Proceedings of Tabulation in Parsing and Deduction (TAPD'98)*, páginas 52–61, Paris (FRANCE), April 1998.

[116] David A. Rosenblueth y Julio C. Peralta. SLR inference: An inference system for fixed-mode logic programs, based on SLR parsing.

[117] David A. Rosenblueth y Julio C. Peralta. LR inference: Inference systems for fixed-mode logic programs, based on LR parsing. En editor Maurice Bruynooghe, editor, *Proceedings of the International Logic Programming Symposium*, páginas 439–453, 1994.

[118] Rosenkrantz y Lewis. Deterministic left corner parsing. En *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1970.

[119] K. Sagonas, T. Swift, y D. S. Warren. XSB as an efficient deductive database engine. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 23(2):442–453, June 1994.

[120] G. Sampson. The SUSANNE corpus, release 3, 04/04/1994. Technical report, School of Cognitive & Computing Sciences, University of Sussex, Falmer, Brighton (England), 1994.

[121] Geoffrey Sampson. A stochastic approach to parsing. *COLING-86*, páginas 151–155, 1986.

[122] Christer Samuelsson. Avoiding non-termination in unification grammars. En *Proceedings of the 4th International Workshop on Natural Language Understanding and Logic Programming*, páginas 4–16, Nara, Japan, 1993.

[123] Yves Schabes. Polynomial time and space shift-reduce parsing of arbitrary context-free grammars. En *Proc. of the 29^{th} Annual Meeting of the Association for Computational Linguistics. Berkeley, CA, 18–21 Jun 1991*, páginas 106–113, 1991.

[124] Yves Schabes. Stochastic lexicalized tree-adjoining grammars. En *Proceedings of COLING-92*, páginas 426–432, Nantes, August 23–28 1992. Association for Computational Linguistics.

[125] Robert Sedgewick. *Algorithms*. Addison-Wesley Series in Computer Science. Addison-Wesley Publishing Company, Inc., 1988.

[126] B.A. Sheil. Observations on context-free grammars. En *Statistical Methods in Linguistics*, páginas 71–109. Stockholm, Sweden, 1976.

[127] S.M. Shieber. Using restriction to extend parsing algorithms for complex-feature-based formalisms. En *Proc. of the 23th Annual Meeting of the ACL*, páginas 145–152, 1985.

[128] S.M. Shieber. *Constraint-Based Grammar Formalism.* MIT Press, Cambridge, MA, 1992.

[129] S.M. Shieber, H. Uszkoreit, F.C.N. Pereira, J.J. Robinson, y M. Tyson. The formalism and implementation of PATR-II. Research on Interactive Acquisition and Use of Knowledge SRI Final Report 1894, SRI International, Menlo Park, California, U.S.A., 1983. Barbara Grosz and Mark Stickel, eds.

[130] Stuart M. Shieber. Sentence disambiguation by a shift-reduce parsing technique. Technical Report Technical Note 281, SRI International, 1983.

[131] Stuart M. Shieber, Yves Schabes, y Fernando C.N. Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 1-2:3–36, 1995.

[132] Klaas Sikkel. *Parsing schemata — A Framework for Sepecification and Analysis of Parsing Algorithms.* Texts in Theoretical Computer Science – An EATCS Series. Springer-Verlag, Berlin/Heidelberg/New York, 1997.

[133] Klass Sikkel. Parsing schemata and correctness of parsing algorithms. *Theoretical Computer Science*, 1-2(199):87–103, 1998.

[134] Klass Sikkel y Anton Nijholt. *The Handbook of Formal Languages*, volumen 2, capítulo Parsing of context-free languages, páginas 61–100. Springer Verlag, Berlin/Heidelberg/New York, 1997. Also available as Technical Report 96-32, Center of Telematics and Information Technology, University of Twente, Enschede, The Netherlands, 1996.

[135] Jonathan Slocum. A practical comparison of parsing strategies. En *Proc. of the 19th Annual Meeting of the Association for Computational Linguistics. Stanford, CA, June 1981*, páginas 1–6, 1981.

[136] Gert Smolka. The oz programming model. Research Report RR-95-10, Deutsches Forschungszentrum für Künstliche Intelligenz, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Erwin-Schrödinger Strasse, Postfach 2080r, 67608 Kaiserslautern, Germany, 1995.

[137] W. Snyder y J. Gallier. Higher order unification revisited: Complete sets of tranformations. Technical report, University of Pennsylvania, December 1987.

[138] Michael Sperber y Peter Thiemann. The essence of LR parsing. En *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, páginas 146–155, La Jolla, California, 21-23 June 1995.

[139] S. Steel y A. De Roeck. Bidirectional chart parsing. En John Wiley & Sons, editor, *Advances in Artificial Intelligence. Proc. of the AISB Conference*, páginas 223–235, Chichester, United Kingdom, 1987. University of Edinburgh.

[140] P Suppes. Probabilistic grammars for natural languages. *Synth&ea.se*, 22:95–116, 1970.

[141] H. Tamaki y T. Sato. Enumeration of success patterns in logic programs (PROLOG). *THEOR. COMPUT. SCI. (NETHERLANDS) ISSN: 0304-3975*, 34(1-2):227–40, November 1984.

[142] Hisao Tamaki y Taisuke Sato. OLD resolution with tabulation. En Ehud Shapiro, editor, *Proceedings of the Third International Conference on Logic Programming*, Lecture Notes in Computer Science, páginas 84–98, London, 1986. Springer-Verlag.

[143] H. Thompson. Chart parsing and rule schemata in GPSG. En *Proc. of* 19^{th} *Annual Meeting for Association for Computational Linguistics*, páginas 167–172, Standford, California, U.S.A., 1981.

[144] Hideto Tomabechi. Quasi-destructive graph unification. En *Proc. of the* 29^{th} *Annual Meeting of the Association for Computational Linguistics. Berkeley, CA, 18–21 Jun 1991*, página 8, 1991.

[145] Masaru Tomita. An efficient augmented context-free parsing algorithm. *Computational Linguistics*, 13(1-2):31–46, 1987.

[146] Matsuru Tomita. *Efficient Parsing for Natural Language. A Fast Algorithm for Practical Systems*. Kluwer Academic Publishers, Norwell, Massachusetts, U.S.A., 1986.

[147] Matsuru Tomita. *Generalized LR Parsing*. Matsuru Tomita, editor, Kluwer, Boston/Dordrecht/London, 1991.

[148] Arturo Trujillo. Computing first and follow functions for feature-theoretic grammars. En *Proceedings of the Fifteenth International Conference on Computational Linguistics (COLING'94)*, Kyoto, Japan, 1994.

[149] Kuniaki Uehora, Ryo Ochitani, Osamu Kakusho, y Junichi Toyoda. A bottom-up parser based on predicate logic: A survery of the formalism and its implementation technique. En *Proc. of the 1984 International Symposium on Logic Programming*, páginas 220–227, 1984.

[150] Ludovic Valter. Indexation de termes logiques au sein d'un evaluateur tabulaire de programmes logiques. Tesis de licenciatura, Université d'Orléans, 1997.

[151] N. Varile. Charts: a data structure for parsing. En Margaret King, editor, *Parsing Natural Language*, páginas 73–87. Academic Press, London, 1983.

[152] M. Vilares, V.M. Darriba, y F.J. Ribadas. A proposal on error repair. En *Proc. of the 2nd Workshop on Tabulation in Parsing and Deduction (TAPD'00)*, páginas 127–138, Vigo, Spain, 2000.

[153] M. Vilares, V.M. Darriba, y F.J. Ribadas. Regional least-cost error repair. En *Proc. of the 5th Int. Conf. on Implementation and Application of Automata (CIAA'00)*, London, Ontario, Canada, 2000.

[154] Manuel Vilares, David Cabrero, y Miguel A. Alonso. Dealing with non-termination in DCGs. En *Proc. of CACIC'99*, Tandil, Buenos Aires, Argentina, 1999.

[155] Manuel Vilares, David Cabrero, y Miguel A. Alonso. Some questions about non-termination in DCGs. En Maria Chiara Meo y Manuel Vilares Ferro (eds.), editores, *APPIA-GULP-PRODE'99 Joint Conference on Declarative Programming, Proceedings*, páginas 545–558, L'Aquila, Italy, 1999.

[156] Manuel Vilares, David Cabrero, y Miguel A. Alonso. On non-termination in DCGs. En Alexander Gelbukh, editor, *International Conference CICLing-2000, Conference on Intelligent text processing and Computational Linguistics, Proceedings*, páginas 150–165. Centro de Investigación en Computación of the Instituto Politécnico Nacional, Mexico City, Mexico, 2000. ISBN 970-18-4206-5.

[157] Manuel Vilares, David Cabrero, y Miguel A. Alonso. On non-termination in DCGs. En Bruce W. Watson y Derick Wood (eds.), editores, *Proc. of the 6th Conference on Implementations and Applications of Automata*, páginas 267–278, Pretoria, South Africa, 2001.

[158] Manuel Vilares, Víctor M. Darriba, y Alonso Miguel A. Searching for asymptotic error repair. En *Proc. of the Seventh International Conference on Implementation and Application of Automata (CIAA 2002)*, Tours, France, 2002.

[159] Manuel Vilares, Víctor M. Darriba, y Francisco J. Ribasdas. Regional least-cost error repair. En Sheng Yu y Andrei Paun (eds.), editores, *Implementation and Application of Automata*, volumen 2088 de *Lecture Notes in Computer Science*, páginas 293–301. Springer-Verlag, Berlin-Heidelberg-New York, 2001. ISSN 0302-9743 / ISBN 3-540-42491-1.

[160] Manuel Vilares, Francisco J. Ribadas, y Víctor M. Darriba. Approximate vldc pattern matching in shared forest. En Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, volumen 2004 de *Lecture Notes in Computer Science*, páginas 483–494. Springer-Verlag, Berlin-Heidelberg-New York, 2001. ISSN 0302-9743 / ISBN 3-540-41687-0.

[161] Manuel Vilares, Jesús Vilares, y David Cabrero. Dynamic programming of partial parses. En Galia Angelova, Kalina Bontcheva, Ruslan Mitkov, Nicolas Nocolov, y Nokolai Nikolov, editores, *EuroConference Recent Advances in Natural Language Processing*, páginas 291–293, Tzigov Chark, Bulgaria, 2001. ISBN 954-90906-1-2.

[162] Manuel Ferro Vilares, David Cabrero Souto, y Miguel A. Alonso Pardo. Dynamic programming as frame for efficient parsing. En *SCCC'98, XVIII International Conference of the Chilean Computer Science Society*, páginas 68–75. IEEE Computer Society Press, Los Alamitos, California, USA, 1998. November 9-14, Antofagasta, Chile, ISBN 0-8186-8616-2.

[163] Jesús Vilares Ferro, David Cabrero Souto, y Miguel A. Alonso Pardo. Applying productive derivational morphology to term indexing of Spanish texts. En Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, volumen 2004 de *Lecture Notes in Computer Science*, páginas 336–348. Springer-Verlag, Berlin-Heidelberg-New York, 2001. ISBN 3-540-41687-0.

[164] Manuel Vilares Ferro. *Efficient Incremental Parsing for Context-Free Languages*. PhD thesis, University of Nice, France, 1992.

[165] Manuel Vilares Ferro. Efficient sharing in ambiguos parsing. En *Proc. of SEPLN'94*, Córdoba, Spain, 1994.

[166] Manuel Vilares Ferro y Miguel A. Alonso Pardo. An LALR extension for DCGs in dynamic programming. En Carlos Martín Vide, editor, *Mathematical and Computational Analysis of Natural Language*, volumen 45 de *Studies in Functional and Structural Linguistics*, páginas 267–278. John Benjamins Publishing Company, Amsterdam & Philadelphia, 1998. ISBN 90-272-1554-5.

[167] Manuel Vilares Ferro, Miguel A. Alonso Pardo, y David Cabrero Souto. An operational model for parsing definite clause grammars with infinite terms. En Alain Lecomte, François Lamarche, y Guy Perrier, editores, *Logical Aspects of Computational Linguistics*, volumen 1582 de *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin-Heidelberg-New York, 1999.

[168] Manuel Vilares Ferro, Miguel Angel Alonso Pardo, y D. Cabrero Souto. An experience on natural language parsing. En C. Martín Vide, editor, *Lenguajes Naturales y Lenguajes Formales*, volumen XII, páginas 555–562, Barcelona, Spain, September 1996. PPU. ISBN 84-477-0575-7.

[169] Manuel Vilares Ferro, Miguel Angel Alonso Pardo, y David Cabrero Souto. Autómatas lógicos y lenguaje natural. En *V Congreso Iberoamericano de Inteligencia Artificial. Iberamia'96*, páginas 341–351, Puebla, Clolula, México, November 1996. Editorial Limusa, S. A. de C. V. ISBN 968-18-5429-2.

[170] Manuel Vilares Ferro, Miguel Angel Alonso Pardo, y David Cabrero Souto. Efficient parsing of fixed-mode DCGs. En Geert-Jan M. Kruijff, Glyn V. Morrill, y Richard T. Oehrle, editores, *Proc. of Formal Grammar'97*, Aix-en-Provence, France, September 1997.

[171] Manuel Vilares Ferro, Miguel Angel Alonso Pardo, y David Cabrero Souto. An operational model for parsing fixed-mode DCGs. En *Proc. of Logical Aspects of Computational Linguistics (LACL'97)*, páginas 61–64, Nancy, France, Sep 1997.

[172] Manuel Vilares Ferro, Miguel Angel Alonso Pardo, Jorge Graña Gil, y David Cabrero Souto. GALENA: Tabular DCG parsing for natural languages. En *Proc. of First Workshop on Tabulation in Parsing and Deduction (TAPD'98)*, páginas 44–51, Paris, France, Apr 1998.

[173] Manuel Vilares Ferro, Miguel Angel Alonso Pardo, y Alberto Valderruten Vidal. *Programación Lógica*. Editorial Tórculo, Santiago de Compostela, Spain, 1994. ISBN 84-88967-36-5.

[174] Manuel Vilares Ferro, David Cabrero Souto, y Miguel A. Alonso Pardo. A comparison for unification-based parsers. En M. Falaschi, J. L. Freire, y M. Vilares, editores, *Proc. of APPIA-GULP-PRODE'98 Joint Conference on Declarative Programming*, páginas 113–123, A Coruña, Spain, Jul 1998.

[175] Manuel Vilares Ferro, David Cabrero Souto, y Miguel A. Alonso Pardo. Exploring parsing efficiency in computational linguistics. En *Utrecht Congress on Storage & Computation in Linguistics*, página 65, Utrecht, The Netherlands, October 1998.

[176] Manuel Vilares Ferro, David Cabrero Souto, y Miguel Angel Alonso Pardo. An approach to infinite term traversal in DCGs. En M. Falaschi, M. Navarro, y A. Policriti, editores, *Proc. of APPIA-GULP-PRODE'97 Joint Conference on Declarative Programming*, páginas 307–317, Grado, Italy, Jun 1997.

[177] Manuel Vilares Ferro, Jorge Graña Gil, T. Araujo, David Cabrero Souto, y I. Diz. A tagger environment for galician. En *Proc. of Workshop on Language Resources for European Minority Languages*, Granada, Spain, May 1998.

[178] Eric Villemonte de la Clergerie. DyALog : une implantation des clauses de Horn en programmation dynamique. En *Proc. of the 9th Seminaire de Programmation en Logique*, páginas 207–228. CNET, May 1990.

[179] 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.

[180] Eric Villemonte de la Clergerie. Layer Sharing : an improved structure–sharing framework. En *Proc. of POPL'93*, páginas 345–356, 1993.

[181] Eric Villemonte de la Clergerie y Bernard Lang. LPDA: Another look at tabulation in logic programming. En Pascal Van Hentenryck, editor, *Proc. of the 11th International Conference on Logic Programming (ICLP'94)*, páginas 470–486, Massachusetts Institute of Technology, June 1994. The MIT Press.

[182] D. S. Warren. Efficient Prolog memory management for flexible control strategies. En *Proceedings of the International Symposium on Logic Programming*, páginas 198–202, Atlantic City, 1984. IEEE, Computer Society Press.

[183] D.A. Watt. The parsing problem for affix grammars. *Acta Informatica*, 8:1–20, 1977.

[184] M. Wirén. A control-strategy-independent parser for PATR. En *Proc. of the First Scandinavian Conference on Artificial Intelligence*, páginas 155–160, Tromsø, Norwa, 1988.

[185] Niklaus Wirth y Helmut Weber. EULER: A generalization of ALGOL, and its formal definition: Part I. *Communications of the ACM*, 9(1):13–23, January 1966.

[186] David A. Wroblewski. Nondestructive graph unification. En Howard Forbus, Kenneth; Shrobe, editor, *Proceedings of the 6th National Conference on Artificial Intelligence*, páginas 582–589, Seattle, WA, July 1987. Morgan Kaufmann.

[187] D. H. Younger. Recognition and parsing of context-free languages in time $O(n^3)$. *Information and Control*, 10(2):189–208, 1967.

Índice alfabético