Autómatas lógicos y lenguaje natural

M. Vilares Ferro M.A. Alonso Pardo D. Cabrero Souto

Resumen

Presentamos un modelo de implementación para los analizadores sintácticos que sirven de base al tratamiento del lenguaje natural. El núcleo de nuestra propuesta es un esquema de evaluación ascendente, guiado por un autómata lógico. El formalismo incorpora un mecanismo de predicción estática, lo que permite reducir el espacio de búsqueda sin sobrecargar el proceso de resolución. El generador de autómatas subyacente toma como entrada gramáticas de cláusulas definidas, sin restricciones. Una interfaz gráfica integra el conjunto.

A diferencia de trabajos anteriores, es posible la compartición óptima tanto de las estructuras sintácticas como del proceso de cálculo, lo que permite mejorar las prestaciones. La completud del análisis está garantizada en el caso de ausencia de símbolos de función.

Tópicos: Análisis sintáctico del lenguaje natural, autómatas lógicos, programación dinámica.

1 Introducción

Buena parte de los formalismos utilizados para el análisis sintáctico en el procesamiento del lenguaje natural (PLN) encuentra su fuente de inspiración en la teoría de lenguajes formales. La idea central consiste en adaptar a la dinámica operativa del algoritmo clásico un mecanismo de unificación que permita la recuperación de la información asociada a las categorías léxicas y sintácticas.

Tradicionalmente, los analizadores sintácticos usados en PLN son intérpretes que aplican un esquema descendente de evaluación [10]. Ello se debe a la facilidad de implementación de algoritmos de recuperación eficaz de memoria, una necesidad acuciante en PLN debido a la complejidad del problema abordado. Sin embargo, las implementaciones prácticas de este tipo de arquitectura presentan carencias graves que afectan tanto al rendimiento como a la potencia de evaluación exhibida. Así, es conocida la ineficacia de los sistemas basados en PROLOG, debida en parte a la consideración del retroceso como mecanismo de simulación del no determinismo, pues facilita la consideración de cálculos redundantes. Podemos añadir los problemas de no completud operacional cuando las gramáticas son recursivas por la izquierda o cíclicas, así como la carencia de mecanismos propios de limitación del espacio de búsqueda, lo que favorece la proliferación de cálculos inútiles.

Algunos autores han propuesto soluciones alternativas, todas ellas incluyendo algoritmos de subsumisión que garantizan la completud del análisis, así como la eliminación de cálculos duplicados. La reducción del espacio de búsqueda se ha centrado en la consideración de técnicas de reescritura, que introducen el control mediante la adición de nuevas reglas [1, 11], pero que en contrapartida aumentan la complejidad del proceso de análisis. En cuanto a la compartición de cálculos, la solución pasa por la aplicación de técnicas de programación dinámica tanto en lo que se refiere a algoritmos interpretados [12] como compilados [3, 7], aunque estos últimos se muestran mejor adaptados al tratamiento computacional en PLN.

M. Vilares es profesor del Departamento de Computación de la Universidad de A Coruña, Campus de Elviña S/N, 15071 A Coruña, España. E-mail: vilares@dc.fi.udc.es. Fax: +34 81 132736. Teléfono: +34 81 132552

M.A. Alonso trabaja actualmente en el Institut National de Recherche en Informatique et en Automatique, Domaine de Voluceau, Rocquencourt, B.P.105, 78153 Le Chesnay Cedex, Francia. E-mail: Miguel.Alonso-Pardo@inria.fr.

D. Cabrero trabaja actualmente en el Centro de Investigacións Lingüísticas e Literarias Ramón Piñeiro, Estrada Santiago-Noia, Km. 3, A Barcia, 15896 Santiago de Compostela, España. E-mail: dcabrero@cirp.es.

Este trabajo ha sido parcialmente financiado en diferentes convocatorias por el proyecto europeo Eureka Software Factory, así como por la Xunta de Galicia mediante los proyectos XUGA10501A93 y XUGA20403B95.

Nuestro trabajo se sitúa en el contexto de los autómatas lógicos de pila (ALPs), un formalismo operacional propuesto por Lang en [9] para el tratamiento de programas en lógica de Horn, y por tanto aplicable a las gramáticas de cláusulas definidas (GCDs). La construcción generaliza los aspectos dinámicos de técnicas anteriores [1, 12, 14] a la vez que garantiza la completud en el caso de GCDs sin símbolos de función. El objetivo ahora es el de optimizar el comportamiento general de la arquitectura desde tres puntos de vista distintos: la generación del ALP, la recuperación de la memoria disponible y la reducción del espacio de búsqueda.

En la sección 2, damos una visión general del modelo ALP en programación dinámica. La sección 3 describe el algoritmo de análisis sintáctico, al tiempo que justifica las diferentes estrategias de implementación en función de los problemas planteados. Una estimación de la complejidad del algoritmo de evaluación aparece en la sección 4. En la sección 5 introducimos la interfaz del sistema. Un amplio abanico de resultados experimentales se incluye en la sección 6. Finalmente, la sección 7 es una conclusión acerca del trabajo realizado.

2 El modelo ALP

El modelo fue introducido por Lang in [9]. Básicamente, un ALP es un autómata de pila que almacena átomos y sustituciones, y que utiliza la unificación para aplicar sus transiciones. Nosotros consideramos una variante simple de la arquitectura original. Formalmente, un ALP 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 el conjunto finito de símbolos de predicado en la base de datos extensiva.

Δ es el conjunto finito de símbolos de predicado en el alfabeto de la pila.

$\$$ es el *predicado inicial*, el único literal sobre la pila del autómata en el momento de comienzo del proceso de cálculo.

$\$_f$ es el *predicado final*. Los literales construidos con este predicado son los únicos sobre el predicado inicial $\$$ en la pila del autómata, una vez el proceso de análisis ha finalizado con éxito.

Θ es un conjunto finito de *transiciones*. Pueden ser de tres tipos:

1. *Horizontal:* $B \quad \{A\} \longmapsto C$
2. *Extracción:* $BD \quad \{A\} \longmapsto C$
3. *Inserción:* $B \quad \{A\} \longmapsto CB$

donde B,C y D son literales en el álgebra de términos $T_\Delta[\mathcal{F} \cup \mathcal{X}]$, y A está en $T_\Sigma[\mathcal{F} \cup \mathcal{X}]$ representando una condición adicional de control.

En adelante, usaremos los términos *pila* o *configuración del ALP* para referirnos a sucesiones finitas de pares *literal/sustitución*. La notación utilizada para un par será $A.\sigma$, donde σ representa la sustitución.

En cuanto a las reglas que rigen la aplicación de las transiciones, éstas son:

- *Caso horizontal:* $B\{A\} \longmapsto C$. Aplicable a una pila $E.\rho\ \xi$, sii existe una sustitución σ tal que:
  - $\sigma = \text{mgu}(E, B)$.
  - $F\sigma = A\sigma$, para un hecho F de la base de datos extensiva.

  En este caso, obtenemos la nueva configuración $C\sigma.\rho\sigma\ \xi$.
- *Caso extracción:* $BD\{A\} \longmapsto C$. Aplicable a pilas $E.\rho E'.\rho'\ \xi$, sii existe σ tal que:
  - $\sigma = \text{mgu}(< E, E'\rho >, < B, D >)$.
  - $F\sigma = A\sigma$, para un hecho F en la base de datos extensiva.

  El resultado obtenido es la configuración $C\sigma.\rho'\rho\sigma\ \xi$.

- *Caso inserción:* $B\{A\} \longmapsto CB$. Podemos aplicarlo a configuraciones $E.\rho\ \xi$, sii existe σ tal que:
  - $\sigma = \mathrm{mgu}(E, B)$
  - $F\sigma = A\sigma$, para un hecho F en la base de datos extensiva.

  Como resultado, obtenemos la nueva pila $C\sigma.\sigma\ B.\rho\ \xi$.

Hablamos de *control estático* de la evaluación cuando la condición A unifica con un hecho en la base de datos extensiva, y de *control dinámico* en otro caso.

Una *interpretación en programación dinámica* de un ALP viene definida por la exploración sistemática de un espacio de elementos denominados *estados*. Este *espacio de búsqueda* es la representación condensada de todos los cálculos posibles del ALP. La interpretación debe garantizar que todos los estados útiles para el cálculo sean explorados, al tiempo que se evitan los redundantes, para lo que suele utilizarse una relación de subsumisión en el conjunto de estados. El entorno de trabajo así definido se conoce como *entorno dinámico*. Utilizaremos la notación S^n para referenciar el entorno dinámico estandar, en el que la pila del autómata se representa por la totalidad de sus literales componentes.

Una vez definido el entorno dinámico, el ALP construye una colección de estados a partir de la configuración inicial. Los nuevos estados se generan a partir de los ya existentes por aplicación de las transiciones. La dinámica de transiciones se aplicará mientras el mecanismo de subsumisión lo permita.

3 El núcleo operativo

El núcleo operativo del sistema es un esquema de evaluación ascendente con un nivel de predicción intermedio. El punto de partida para su implementación es el entorno ICE[1] [5], un generador de analizadores sinácticos incrementales para gramáticas algebraicas arbitrarias que ha mostrado su adaptabilidad para el diseño de evaluadores en lógica de Horn de primer orden [7].

3.1 Recuperación de la memoria

La evaluación en profundidad, de izquierda a derecha, típica de los analizadores basados en PROLOG; permite una implementación a la vez simple y potente de técnicas de recuperación de memoria. En esencia, se trata de aprovechar el mecanismo de retroceso del algoritmo de resolución subyacente. Cuando el retroceso se hace efectivo, la memoria correspondiente a los nodos situados a la izquierda de la nueva posición alcanzada pueden ser liberados.

Aún cuando el algoritmo de evaluación difiera de la arquitectura PROLOG clásica, el comportamiento descrito puede extenderse a cualquier método de resolución en programación dinámica. Esta es una posibilidad apuntada en trabajos precedentes [13], aunque no aplicada hasta ahora al caso de ALPs. La idea consiste en indexar el proceso de análisis, agrupando en conjuntos los estados generados entre el reconocimiento de dos palabras consecutivas en el texto de entrada [4]. La asociación a cada estado de un *puntero de retroceso* hacia la posición del texto en la que ha comenzado el reconocimiento de la categoría sintáctica actualmente en fase de análisis, permite detectar la zona de memoria susceptible de ser recuperada. Así, basta con liberar aquellos estados generados entre la posición más cercana de entre las señaladas por los punteros de retroceso de las categorías sintácticas no triviales reconocidas justo antes de una acción de lectura, y la posición de lectura actual en el texto.

[1] por **I**ncremental **C**ontext-free **Environment**.

3.2 El esquema de evaluación

Antes de formalizar el esquema de evaluación, introducimos algunas notaciones adicionales con el fin de simplificar la exposición. Dada una regla γ_k de la forma $A_{k,0} \rightarrow A_{k,1}, \ldots, A_{k,n_k}$, en una GCD $\gamma_{1..m}$, consideramos:

- Un vector $\vec{T}_k$ con las variables que aparecen en γ_k.
- El símbolo de predicado $\bigtriangledown_{k,i}$, para el que una instancia de $\bigtriangledown_{k,i}(\vec{T}_k)$ indica que todos los literales a partir del situado en la posición i de la parte derecha de la regla γ_k, han sido analizados.

Además, asociamos a cada estado la posición actual de lectura en el texto, así como la del puntero de retroceso. Para ello, utilizamos dos superíndices sobre el término involucrado, en la forma

$$A^{\text{posición,retroceso}}$$

De este modo, el esquema de evaluación es el definido por las transiciones:

$$\begin{array}{llcll}
1. & A^{it,bp}_{k,n_k} & \{E_k\} & \longmapsto & \bigtriangledown^{it,it}_{k,n_k}(\vec{T}_k) \quad A^{it,bp}_{k,n_k} \\
2. & \bigtriangledown^{it,r}_{k,i}(\vec{T}_k) & A^{r,bp}_{k,i} & \longmapsto & \bigtriangledown^{it,bp}_{k,i-1}(\vec{T}_k), \quad i \in [1, n_k] \\
3. & \bigtriangledown^{it,bp}_{k,0}(\vec{T}_k) & & \longmapsto & A^{it,bp}_{k,0}
\end{array}$$

para el *modo de reducción*, y

$$\begin{array}{llcll}
4. & A^{it,bp}_{k,i} & \{E^{n,it}_{k,i}\} & \longmapsto & E^{n,it}_{k,i} \quad A^{it,bp}_{k,i}, \; i \in [0, n_k) \\
5. & A^{it,bp}_{k,0} & \{E^{n,it}_{m,0}\} & \longmapsto & E^{n,it}_{m,1} \quad A^{it,bp}_{k,0} \\
6. & \$^{it,bp} & \{E^{n,it}_{\$}\} & \longmapsto & E^{n,it}_{\$} \quad \$^{it,bp}
\end{array}$$

para el de lectura. Brevemente, pasamos a interpretar cada una de las transiciones:

1. *Selección de una regla:* Cuando la condición $\{E_k\}$ se verifica, seleccionamos la regla γ_k si un literal A_{k,n_k} está en la cima de la pila. En ese caso, insertamos $\bigtriangledown_{k,n_k}(\vec{T}_k)$ para indicar que ninguno de los literales en la parte derecha de la regla ha sido todavía reconocido.
2. *Reducción en la parte derecha de una regla γ_k:* El literal de posición $\bigtriangledown_{k,i}(\vec{T}_k)$ indica que todos los literales en la parte derecha de γ_k situados más allá de la posición i, han sido ya reconocidos. En ese momento, todas las pilas que contengan al literal $A_{k,i}$ justo debajo de su cima pueden incrementar el literal de posición, indicando de esta forma el avance del proceso de análisis en la parte derecha de la regla.
3. *Reducción de la parte izquierda de una regla γ_k:* El literal $\bigtriangledown_{k,0}(\vec{T}_k)$ indica que todos los literales en la parte derecha de la regla γ_k han sido reconocidos. Reemplazar el literal de posición por el literal $A_{k,0}$ que constituye la parte izquierda de la regla, indicando de esta forma que ha sido reducido.
4. *Inserción de literales:* El literal $E_{k,i}$, correspondiente a la lectura de una categoría lexical, se inserta en la pila.
5. *Inserción del primer literal de la parte derecha de una regla:* El literal $E_{m,1}$ es insertado en la pila, marcando el inicio del reconocimiento de la parte derecha de la reglaγ_m.
6. *Inserción inicial:* El predicado inicial marca el inicio del proceso de análisis mediante acciones de inserción exclusivamente.

En consecuencia el esquema descrito es ascendente, aunque admite la consideración de mecanismos predictivos a través de la condición de control opcional. Ello nos permitirá delimitar el espacio de búsqueda.

3.3 Reducción del espacio de búsqueda

La difícil determinización del análisis, tanto léxico como sintáctico, en PLN hace que el espacio de búsqueda pueda alcanzar una complejidad considerable, incluso exponencial en la longitud del texto analizado. Es por ello primordial su reducción, con el fin no sólo de ganar espacio útil, sino de acelerar el proceso de cálculo. Dos técnicas se muestran adaptadas a este fin:

- La utilización de entornos dinámicos que aseguren una representación lo más compacta posible de los estados, lo que favorece la compartición de configuraciones.
- La aplicación de técnicas que eviten la generación de estados inviables.

3.3.1 Entornos dinámicos

En la práctica, el entorno dinámico mejor adaptado a nuestros fines es S^1, introducido por Villemonte de la Clergerie en [3] en lógica de Horn y por Vilares en [5] en el caso de los lenguajes algebraicos. Este entorno está caracterizado por estados de la forma $[A.u] := \{A.u\xi\}$.

En consecuencia, S^1 no proporciona información acerca del contexto sintáctico, por lo que es necesario en las transiciones que extraen elementos de la pila, preveer un mecanismo capaz de solventar esta limitación. En esta línea, definimos un operador Op que traduce transiciones de S^n, el entorno dinámico estandar, a S^1:

- *Caso horizontal:* $Op(B \longmapsto C)([A]) = [C\sigma]$, donde $\sigma = \mathrm{mgu}(A, B)$
- *Caso extracción:* $Op(BD \longmapsto C)([A]) = \{D\sigma \longmapsto C\sigma\}$, donde $\sigma = \mathrm{mgu}(A, B)$, y $D\sigma \longmapsto C\sigma$ es una *transición dinámica* generada por la extracción. La nueva transición es aplicable no sólo a la configuración resultante de la aplicación de la extracción, sino también a aquellas todavía no generadas, pero que comparten el mismo entorno sintáctico.
- *Caso inserción:* $Op(B \longmapsto CB)([A]) = [C\sigma]$, donde $\sigma = \mathrm{mgu}(A, B)$

En relación a los inconvenientes, S^1 es incompatible con los esquemas de análisis descendente [3, 5]. El origen es la falta de información acerca del contexto sintáctico. Un esquema de evaluación de este tipo puede predecir más información de la que luego puede recuperar, con lo que la corrección del proceso de cálculo no está asegurada. Este no es el caso que nos ocupa, pues el nuestro es un esquema ascendente.

3.3.2 Control estático

Nuestra propuesta trata de evitar conflictos de evaluación entre términos del ALP mediante la extensión del concepto de ventana, ampliamente utilizado en la determinización de los lenguajes algebraicos, a las GCDs. Para ilustrar el proceso, describiremos el caso más sencillo, en el que la ventana es calculada directamente a partir del esqueleto algebraico de la GCD. La técnica a seguir es entonces extremadamente simple. En primer lugar, extraemos la base algebraica, $\mathcal{G}^f$, de la GCD. Para ello, recuperamos a partir de cada literal un símbolo en el esqueleto algebraico, diferenciando en razón de la firma del funtor principal. El esquema de evaluación se define entonces, dada una regla γ_k, por el siguiente conjunto de transiciones:

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

para el modo de reducción, y

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

para el modo de lectura. En cada caso, las notaciones $first_1^f$, $follow_1^f$ y $lookahead_k^f$ representan respectivamente los conceptos $first_1$, $follow_1$ y $lookahead_k^f$ para la gramática $\mathcal{G}^f$. Representamos por $A_{k,i}^f$ el término en $\mathcal{G}^f$ obtenido a partir de $A_{k,i}$. La notación n referencia la posición siguiente a la actual de lectura it cuando el símbolo reconocido no es ε, e it en otro caso. Dado que tanto $first_1$ como $follow_1$ pueden ser calculados *a priori*, no existe ninguna objeción para considerar las condiciones auxiliares de control como hechos en la GCD. Esto es, el control será estático. Intuitivamente, en este caso el esquema de evaluación se corresponde a la adaptación a nuestro modelo de la dinámica de un autómata LALR(1) clásico.

Pese a la sencillez de la técnica presentada, ésta posee un sentido práctico real. En efecto, tenemos que:

- El coste de cálculo de los hechos que componen el conjunto de las condiciones de control es, desde el punto de vista computacional, irrelevante.
- El dominio determinista de la familia LALR(1) es amplio, lo cual se traduce en un tratamiento eficiente del determinismo local[2]. Ello conlleva una mejora de las prestaciones del sistema.
- El fenómeno de división de estados, típico de las estrategias de determinización por predicción estática, se mantiene dentro de límites razonables. En consecuencia, la diferenciación entre contextos sintácticos irrelevantes es mínima, por lo que la calidad de compartición de cálculos y estructuras no se ve gravemente afectada.

La dinámica descrita puede extenderse en tres sentidos distintos:

1. Ampliando las nociones de *first*, *follow* y *lookahead* a GCDs, tal y como ha sido descrito en [5].
2. Aumentando el tamaño de la ventana, lo que nos lleva a generalizar el tratamiento LALR(k) a los ALPs.
3. Introduciendo técnicas de control estático más complejas.

Estas estrategias no son exclusivas, sino complementarias. El programador puede combinarlas a voluntad. En cualquier caso, la elección final debe ser el resultado de un equilibrio entre los costes temporales y espaciales. En este punto, disponemos de algunas referencias coincidentes tanto en el campo de los lenguajes algebraicos como en el de la lógica de Horn [5, 3]:

- Los algoritmos ascendentes puros muestran mejores resultados desde el punto de vista de la compartición de cálculos y estructuras.
- Un nivel de predicción elevado en un algoritmo ascendente no garantiza un mejor comportamiento computacional. Si bien se reduce el espacio de búsqueda, también se favorece el fenómeno de división de estados [5], con una incidencia desfavorable en la calidad de compartición. Sería, por ejemplo, el caso de las técnicas de control inspiradas en el modelo LR.
- Las técnicas situadas entre los modelos ascendente puro y Earley [4] muestran los mejores resultados prácticos.

4 Complejidad

Diferenciamos [12] entre análisis sintáctico *online* y *offline*, en función de si las restricciones derivadas del tratamiento de los atributos se aplican en el momento del análisis o en un proceso posterior. La razón es que el problema de la evaluación de GCDs no es en general resoluble, aunque sí en el caso *offline*, que además parece ser el único lingüísticamente relevante [2].

De este modo, estimamos la complejidad de la evaluación para GCDs analizables *offline*. Partiendo de un texto de entrada de longitud n, la complejidad temporal es $\mathcal{O}(n^3)$ y la espacial $\mathcal{O}(n^2)$, en el peor de los casos. Las razones son las siguientes:

[2] en la práctica, el proceso es determinista durante buena parte del análisis.

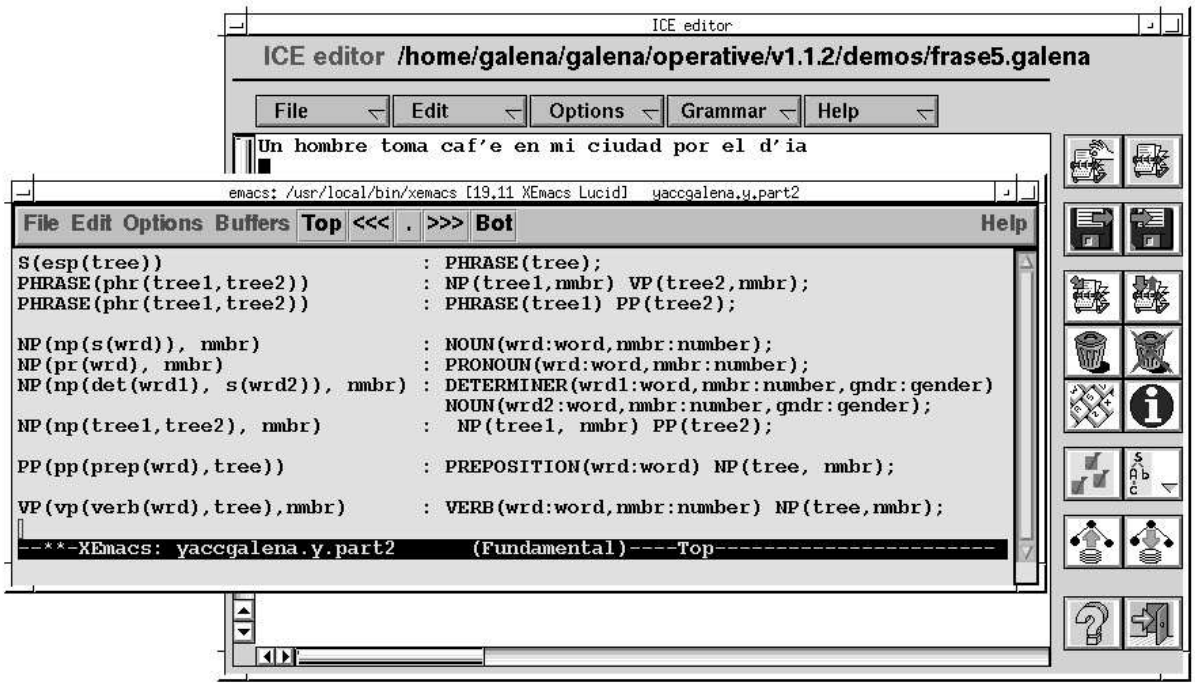

Figura 1: Los editores de la interfaz

- Los rangos del conjunto de reglas como de sustituciones están acotados, mientras el valor del puntero de retroceso depende de i, y está acotado por n. Así, el número de estados generados en la posición i de lectura es $\mathcal{O}(i)$, y el espacio requerido es $\mathcal{O}(\Sigma_{i=0}^{n} i) = \mathcal{O}(n^2)$.
- El número de variables accesibles en un estado, durante el proceso de unificación, es acotado.
- Las transiciones de inserción y horizontales, ejecutan cada una un número acotado de pasos por estado. Las de extracción ejecutan $\mathcal{O}(i)$ pasos por cada uno, puesto que pueden añadir $\mathcal{O}(l)$ estados por cada valor del puntero de retroceso l. Así, el tiempo consumido es $\mathcal{O}(i^2)$ en lo que se refiere a los estados en la posición i, y el total, incluyendo la unificación *online* y los tests de subsumción, es $\mathcal{O}(\Sigma_{i=0}^{n} i^2) = \mathcal{O}(n^3)$.

En gramáticas para las que el número de estados en cada posición de lectura está acotado, la complejidad es lineal en el tiempo y en el espacio. Dentro de éstas se incluye la familia LALR de los lenguajes algebraicos, lo que quiere decir que en la práctica es posible una complejidad lineal cuando el determinismo local esté presente.

5 La interfaz

El conjunto del sistema es accesible a través de una interfaz gráfica construida actualmente sobre la base de AIDA [8]. Esta integra las funcionalidades de edición y análisis de textos, así como de visualización y navegación en bosques sintácticos, amén de diversos útiles de traza y ayuda en línea. Los recursos de la interfaz son susceptibles de ser personalizados por el usuario. Para mostrar el funcionamiento del conjunto, utilizaremos el lenguaje de las frases nominales en español. En este caso, hemos utilizado como base léxica, el lematizador descrito en [6].

Tal y como se muestra en la figura 1, el usuario puede editar un texto de entrada y analizarlo según un lenguaje previamente definido conforme al formalismo de las GCDs. La localización de este último es automática a partir de la extensión del fichero editado. El acceso a la

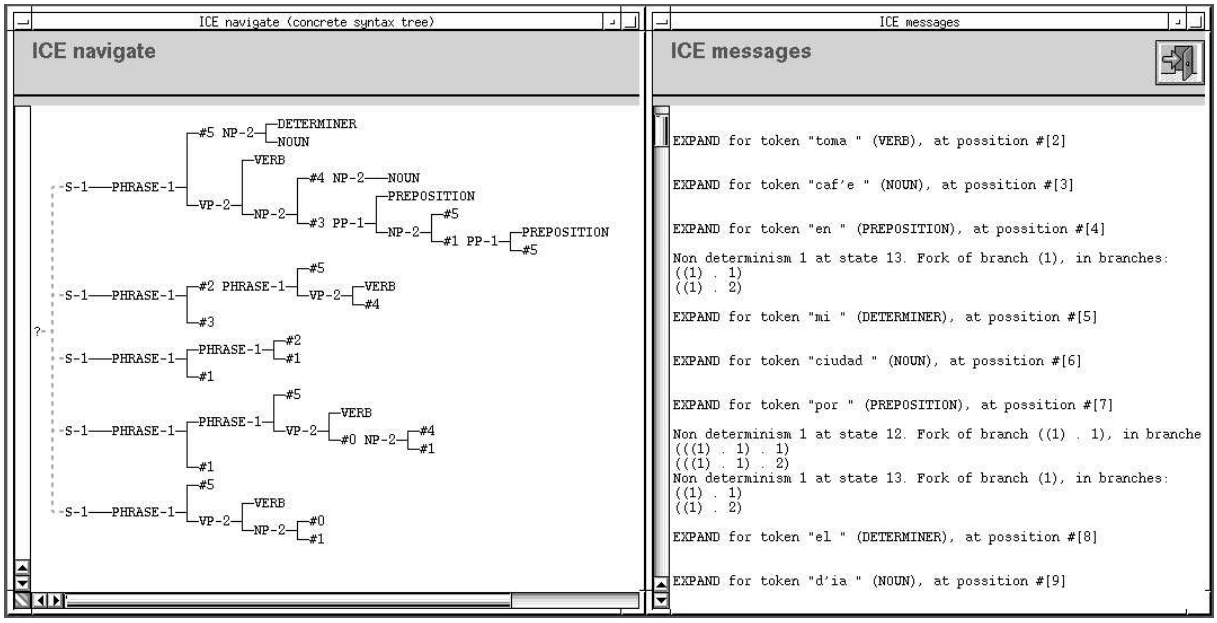

Figura 2: Un bosque sintáctico concreto compartido

información del lexical se realiza de forma automática a partir del nombre del atributo que se desea acceder. Así, en la tercera cláusula relativa al predicado NP, la morfología del determinante se accede mediante la combinación `wrd1:word`, lo que asigna a la variable `wrd1` la cadena de caracteres que conforma la categoría lexical en cuestión. El número de esa misma categoría se asigna al atributo `nmbr` mediante la combinación `nmbr:number`. El género del determinante se asigna a `gndr` mediante `gndr:gender`. De forma análoga se recuperaría la información relativa a la totalidad de los accidentes que permiten la derivación de las categorías léxicas, tales como el tiempo o la persona en los tiempos verbales.

En particular, la GCD utilizada en este caso introduce dos sintaxis: una concreta y la otra abstracta. La primera se define mediante los funtores de los predicados de forma natural. Así, la tercera cláusula del predicado `NP` establece una sintaxis concreta para los predicados nominales que viene dada por un árbol etiquetado `NP`, con dos hijos: uno etiquetado `DETERMINER` y otro `NOUN`. En contraste, la sintaxis abstracta indica que el árbol está etiquetado `np` y sus dos hijos `det` y `s`. Ambos nodos representan categorías lexicales cuyos contenidos son los valores de las variables `wrd1` y `wrd2` respectivamente.

En relación al proceso de análisis sintáctico, es posible obtener una traza del mismo sobre la ventana `ICE messages` en función de las opciones ofertadas por el sistema: ambigüedades detectadas, estadísticas sobre el espacio de búsqueda, compartición de cálculos, e incluso las acciones elementales del ALP. Un ejemplo de este comportamiento se muestra en la parte derecha de las figuras 2 y 3. Cualquier error detectado es comunicado al usuario a través de una ventana de diálogo generada a tal efecto.

Un punto importante es la posibilidad de visualizar el bosque sintáctico, y de navegar en su estructura mediante la herramienta `ICE navigate`. El usuario puede escoger el tipo de bosque sintáctico compartido que desea visualizar: el correspondiente a la gramática concreta o a la abstracta previamente descritas. En nuestro ejemplo, la parte izquierda de la figura 2 se corresponde con la gramática concreta, mientras que la parte izquierda de la figura 3 se corresponde a la abstracta. La herramienta evidencia, en cualquier caso, las comparticiones estructurales detectadas, y permite además elegir el nivel de detalle que deseamos en la visualización de dichas estructuras. Las ambigüedades se representan mediante líneas discontinuas, mientras que las expresiones del tipo $\#n$ indican la compartición de nodos.

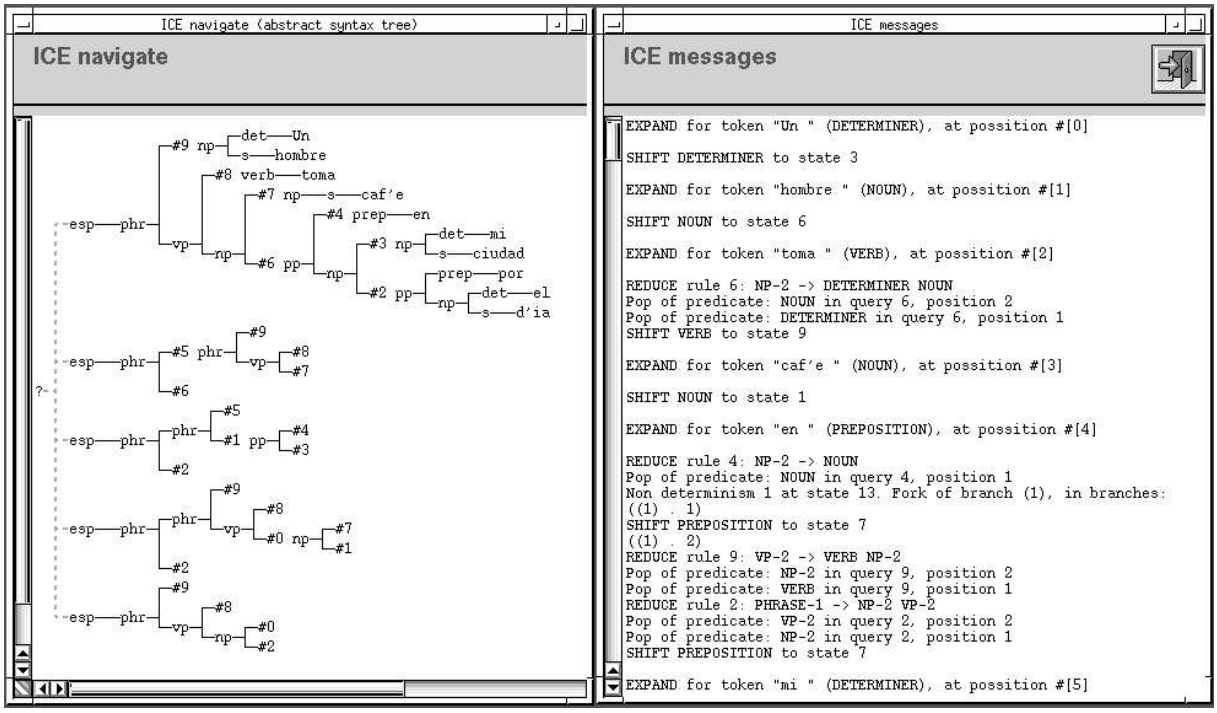

Figura 3: Un bosque sintáctico abstracto compartido

6 Resultados experimentales

Con el objeto de evidenciar la validez de nuestra propuesta, hemos seleccionado un escenario de experimentación que difícilmente puede calificarse como favorable. El punto de partida es la GCD que aparece en el editor de la figura 1. En este caso, la gramática $\mathcal{G}^f$ viene dada por las reglas:

S	→	PHRASE	PHRASE	→	NP VP	PHRASE	→	PHRASE PP
NP	→	noun	NP	→	pronoun	NP	→	determiner noun
NP	→	NP PP	PP	→	preposition NP	VP	→	verb NP

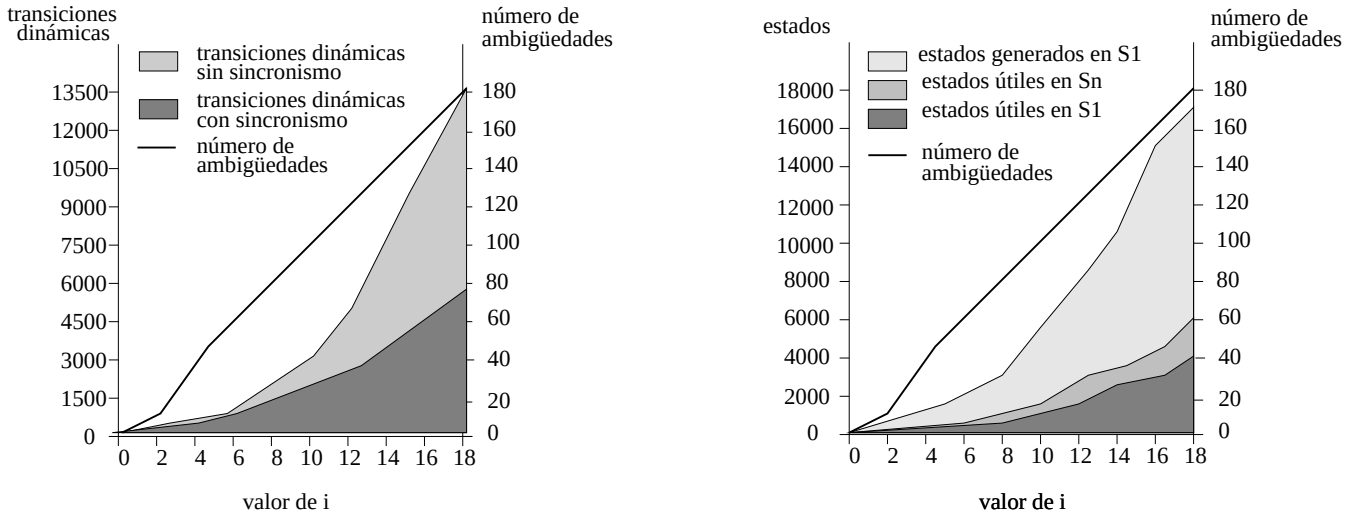

Figura 4: Transiciones dinámicas y estados

El ejemplo reune ciertas características que, a nuestro juicio, hacen de él un buen candidato para los tests prácticos. En primer lugar, se trata de un lenguaje conocido y conciso, lo que favorece su comprensión. Además, la gramática es ambigua, por lo que es fácil comprobar las aptitudes del algoritmo de análisis sintáctico en lo que se refiere a la compartición de cálculos y estructuras. En concreto, para textos del tipo

$$\text{Yo veo un padre (de un hijo de un padre)}^i$$

donde $i \geq 0$ expresa el número de repeticiones de la cadena *"de un hijo de un padre"*, el número de ambigüedades C_i crece exponencialmente con i en la forma:

$$C_i = \begin{cases} 1 & \textbf{si} \quad i = 0, 1 \\ \left(\begin{array}{c} 2i \\ i \end{array} \right) \frac{1}{\mathrm{i+1}} & \textbf{si} \quad i > 1 \end{cases}$$

En relación al espacio de búsqueda, la GCD propuesta no facilita la labor de su reducción y posterior recuperación:

- La técnica de predicción aplicada es la más simple posible, y en este caso concreto no representa un control adicional real sobre un esquema de evaluación ascendente puro. Basta comparar las máquinas LR(0) y LALR(1) de la gramática $\mathcal{G}^f$.
- Las reducciones implican a un número moderado de símbolos en lo que se refiere a la cadena *"de un hijo de un padre"*. En estas circunstancias, la indexación no permite la liberación de grandes zonas de memoria.

Finalmente, la presencia de recursivad por la izquierda permite mostrar la viabilidad de este tipo de gramáticas.

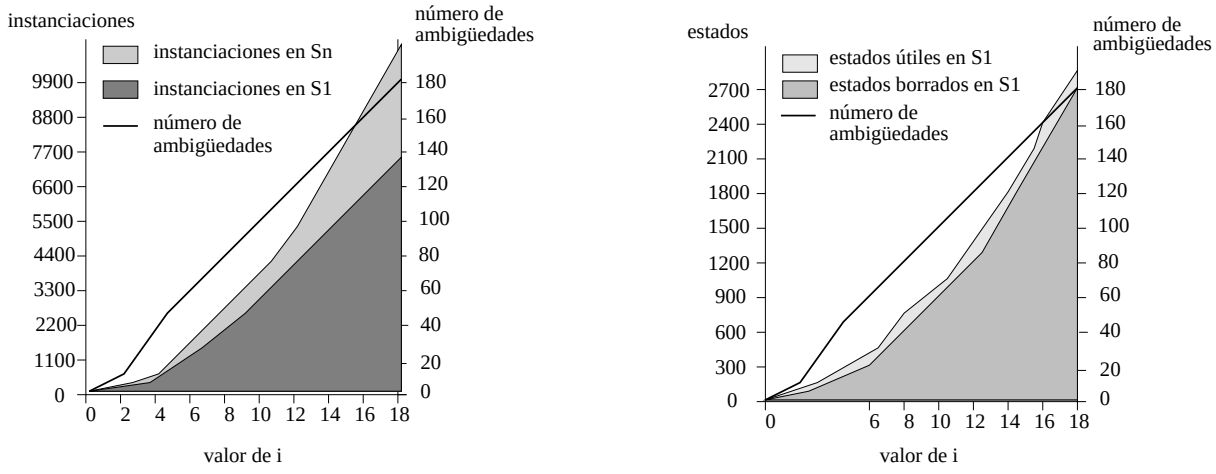

Figura 5: Pares unificados y recuperación de memoria

Sobre la base descrita, la gráfica de la izquierda en la figura 4 muestra el número de transiciones dinámicas generadas en S^1 indexando el texto de entrada y sin hacerlo. La gráfica de la derecha presenta el número de estados útiles y el de no viables generados en S^1, a la vez que compara el número de estados útiles generados en S^1 y S^n. La gráfica de la izquierda de la figura 5 representa el número de pares unificados durante el proceso de análisis, esto es, de cálculos duplicados detectados. La gráfica de la derecha muestra la memoria recuperada durante el proceso de análisis.

7 Conclusión

El análisis sintáctico de los lenguajes naturales no es un tema trivial. La presencia de ciclos, ambigüedades y, todo tipo de procesos contextuales dificultan tanto su tratamiento como la eficacia de los sistemas que lo implementan. Los formalismos de evaluación lógica han demostrado ser los mejor adaptados, extendiéndose su uso en los últimos años, fundamentalmente sobre la base de PROLOG. Sin embargo, la pobre terminación del algoritmo de resolución subyacente, y su no completud operacional limitan en la práctica la declaratividad que propició su popularidad.

Nuestro trabajo se orienta a la resolución de tales lagunas en las arquitecturas tradicionales. El formalismo operacional es un autómata, lo que favorece un buen comportamiento temporal y espacial frente a los algoritmos interpretados. El proceso de cálculo garantiza la completud del análisis en el caso de ausencia de símbolos de función. La reducción del espacio de búsqueda se realiza de forma estática, sin sobrecargar la evaluación. Un esquema ascendente de resolución posibilita la introducción de un entorno dinámico, S^1, que garantiza una compartición óptima tanto del espacio de búsqueda como del proceso de cálculo. Una interfaz gráfica integra el conjunto, haciendo transparentes aquellos aspectos no esenciales para el usuario.

Finalmente, los resultados prácticos parecen mostrar un rendimiento por encima de lo previsto en las estimaciones teóricas. Ello nos permite ser optimistas en cuanto a la extensión del modelo para el tratamiento de formalismos de unificación más complejos, tales como las TAGs.

Referencias

[1] F. Bancilhon, D. Maier, Y. Sagiv, and J. Ullman. Magic-set and other strange ways to implement logic programs. In *Proc. of the 5th ACM SIGMOD-SIGACT Symp. on Principles of Database Systems*, 1986.

[2] J. Bresnan and R. Kaplan. Lexical-functional grammar: A formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, 1982.

[3] E. Villemonte de la Clergerie. *Automates à Piles et Programmation Dynamique*. PhD thesis, University of Paris VII, France, 1993.

[4] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

[5] M. Vilares Ferro. *Efficient Incremental Parsing for Context-Free Languages*. PhD thesis, University of Nice, France, 1992.

[6] M. Vilares Ferro, J. Graña Gil, and A. Pan Bermúdez. Building friendly architectures for tagging. In *Proc. of SEPLN'96*, Sevilla, Spain, 1996.

[7] M. Vilares Ferro and M. A. Alonso Pardo. A predictive bottom-up evaluator. *Logic Programming Newsletter*, 8:9–10, 1995.

[8] Ilog S.A., 2 Avenue Galliéni, BP 85, 94253 Gentilly, France. AÏDA, *Reference Manual, Version 1.65*, 1992.

[9] B. Lang. Complete evaluation of Horn Clauses, an automata theoretic approach. Technical Report 913, INRIA, Rocquencourt, France, 1988.

[10] H. Meijer. The project on extended affix grammars at Nijmegen. *Attribute Grammars and their Applications, SLNC*, 461:130–142, 1990.

[11] U. Nilsson. Abstract interpretation: A kind of magic. In *Proc. of PLILP'91*, 1991.

[12] F.C.N. Pereira and D.H.D. Warren. Parsing as deduction. In 137-144, editor, *Proc. of the* 21^{st} *Annual Metting of the Association for Computational Linguistics*, Cambridge, Massachusetts, U.S.A., 1983.

[13] S.M. Shieber. Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *Proc. of the 23th Annual Meeting of the ACL*, pages 145–152, 1985.

[14] H. Tamaki and T. Sato. Old resolution with tabulation. In *Proc. of the* 3^{rd} *International Conference on Logic Programming*, pages 84–98, London, United Kingdom, 1986. Springer LNCS 225.