Procesamiento del Lenguaje Natural, nº39 (2007), pp. 13-20 recibido 16-05-2007; aceptado 22-06-2007

Técnicas deductivas para el análisis sintáctico con corrección de errores*

Carlos Gómez-Rodríguez y **Miguel A. Alonso**
Departamento de Computación
Universidade da Coruña
Campus de Elviña, s/n
15071 A Coruña, Spain
{cgomezr, alonso}@udc.es

Manuel Vilares
E. S. de Ingeniería Informática
Universidad de Vigo
Campus As Lagoas, s/n
32004 Ourense, Spain
vilares@uvigo.es

Resumen: Se presentan los *esquemas de análisis sintáctico con corrección de errores*, que permiten definir algoritmos de análisis sintáctico con corrección de errores de una manera abstracta y declarativa. Este formalismo puede utilizarse para describir dichos algoritmos de manera simple y uniforme, y proporciona una base formal para demostrar su corrección y otras propiedades. Además, mostramos cómo se puede utilizar para obtener distintas implementaciones de los algoritmos de análisis sintáctico, incluyendo variantes basadas en corrección regional.
Palabras clave: análisis sintáctico robusto, corrección de errores, esquemas de análisis sintáctico

Abstract: We introduce error-correcting parsing schemata, which allow us to define error-correcting parsers in a high-level, declarative way. This formalism can be used to describe error-correcting parsers in a simple and uniform manner, and provides a formal basis allowing to prove their correctness and other properties. We also show how these schemata can be used to obtain different implementations of the parsers, including variants based on regional error correction.
Keywords: robust parsing, error correction, parsing schemata

1. Introducción

Cuando se utilizan técnicas de análisis sintáctico en aplicaciones reales, es habitual encontrarse con frases no cubiertas por la gramática. Esto puede deberse a errores gramaticales, errores en los métodos de entrada, o a la presencia de estructuras sintácticas correctas pero no contempladas en la gramática. Un analizador sintáctico convencional no podrá devolver un árbol de análisis en estos casos. Un *analizador sintáctico robusto* es aquél que puede proporcionar resultados útiles para estas frases agramaticales. Particularmente, un *analizador sintáctico con corrección de errores* es un tipo de analizador sintáctico robusto que puede obtener árboles sintácticos completos para frases no cubiertas por la gramática, al suponer que estas frases agramaticales son versiones corruptas de frases válidas.

* Parcialmente financiado por Ministerio de Educación y Ciencia (MEC) y FEDER (TIN2004-07246-C03-01, TIN2004-07246-C03-02), Xunta de Galicia (PGIDIT05PXIC30501PN, PGIDIT05PXIC10501PN, Rede Galega de Procesamento da Linguaxe e Recuperación de Información) y Programa de Becas FPU (MEC).

En la actualidad no existe un formalismo que permita describir de manera uniforme los analizadores sintácticos con corrección de errores y probar su corrección, tal y como se hace con los *esquemas de análisis sintáctico* para los analizadores convencionales. En este artículo, se propone un formalismo que cubre esta necesidad al tiempo que se muestra cómo se puede utilizar para obtener implementaciones prácticas.

2. Esquemas de análisis sintáctico convencionales

Los esquemas de análisis sintáctico (Sikkel, 1997) proporcionan una manera simple y uniforme de describir, analizar y comparar distintos analizadores sintácticos. La noción de esquema de análisis sintáctico proviene de considerar el análisis como un proceso deductivo que genera resultados intermedios denominados *ítems*. Se parte de un conjunto inicial de ítems obtenido directamente de la frase de entrada, y el proceso de análisis sintáctico consiste en la aplicación de reglas de inferencia (*pasos deductivos*) que producen nuevos ítems a partir

ISSN: 1135-5948

de los ya existentes. Cada ítem contiene información sobre la estructura de la frase, y en cada análisis sintáctico satisfactorio se obtiene al menos un *ítem final* que garantiza la existencia de un árbol sintáctico completo para la frase.

Sea $G = (N, \Sigma, P, S)$[1] una gramática independiente del contexto[2]. El conjunto de árboles válidos para G, denotado $Trees(G)$, se define como el conjunto de árboles finitos donde los hijos de cada nodo están ordenados de izquierda a derecha, los nodos están etiquetados con símbolos de $N \cup \Sigma \cup (\Sigma \times \mathbb{N}) \cup \{\epsilon\}$, y cada nodo u satisface alguna de las siguientes condiciones:

- u es una hoja,
- u está etiquetado A, los hijos de u están etiquetados $X_1, \ldots, X_n$ y hay una producción $A \rightarrow X_1 \ldots X_n \in P$,
- u está etiquetado A, u tiene un único hijo etiquetado ϵ y existe una producción $A \rightarrow \epsilon \in P$,
- u está etiquetado a y u tiene un único hijo etiquetado (a, j) para algún j.

A los pares (a, j) les llamaremos *terminales marcados*, y cuando trabajemos con una cadena $a_1 \ldots a_n$, escribiremos $\underline{a}_j$ como notación abreviada para (a_j, j). El número natural j se utiliza para indicar la posición del símbolo a en la entrada, de modo que la frase de entrada $a_1 \ldots a_n$ pueda verse como un conjunto de árboles de la forma $a_j(\underline{a}_j)$ en lugar de como una cadena de símbolos. A partir de ahora, nos referiremos a los árboles de esta forma como *seudoproducciones*.

Sea $Trees(G)$ el conjunto de árboles para una gramática independiente del contexto G. Un *conjunto de ítems* es un conjunto $\mathcal{I}$ tal que $\mathcal{I} \subseteq \Pi(Trees(G)) \cup \{\emptyset\}$, donde Π es una partición de $Trees(G)$. Si el conjunto contiene como elemento a $\emptyset$, llamaremos a este elemento el *ítem vacío*.

Los análisis válidos de una cadena en el lenguaje definido por una gramática G están representados por ítems que contienen *árboles sintácticos marcados* para esa cadena. Dada una gramática G, un árbol sintáctico marcado para una cadena $a_1 \ldots a_n$ es cualquier árbol $\tau \in Trees(G) / root(\tau) = S \wedge yield(\tau) = \underline{a}_1 \ldots \underline{a}_n$. Llamaremos *ítem final* a todo ítem que contenga un árbol sintáctico marcado para una cadena cualquiera. Llamaremos *ítem final correcto* para una cadena concreta $a_1 \ldots a_n$ a todo ítem que contenga un árbol sintáctico marcado para esa cadena.

Ejemplo: El conjunto de ítems de Earley (Earley, 1970), $\mathcal{I}_{Earley}$, asociado a una gramática $G = (N, \Sigma, P, S)$ es:

$\mathcal{I}_{Earley} = \{[A \rightarrow \alpha \bullet \beta, i, j] / A \rightarrow \alpha\beta \in P \wedge 0 \leq i \leq j\}$

donde la notación $[A \rightarrow \alpha \bullet \beta, i, j]$ usada para los ítems representa el conjunto de árboles de raíz A, tales que los hijos directos de A son $\alpha\beta$, los nodos frontera de los subárboles con raíz en los nodos etiquetados α forman una cadena de terminales marcados de la forma $\underline{a}_{i+1} \ldots \underline{a}_j$, y los nodos etiquetados β son hojas. El conjunto de ítems finales en este caso es

$\mathcal{F}_{Earley} = \{[S \rightarrow \gamma \bullet, 0, n]\}$. □

Un esquema de análisis sintáctico es una función que, dada una cadena $a_1 \ldots a_n$ y una gramática G; permite obtener un conjunto de *pasos deductivos*. Los pasos deductivos son elementos de $(H \cup \mathcal{I}) \times \mathcal{I}$, donde $\mathcal{I}$ es un conjunto de ítems y H (que llamaremos conjunto de *ítems iniciales* o *hipótesis*) contiene un conjunto $\{a_i(\underline{a}_i)\}$ por cada seudoproducción asociada a la cadena. Los pasos deductivos establecen una relación de inferencia $\vdash$ entre ítems, de modo que $Y \vdash x$ si $(Y', x) \in D$ para algún $Y' \subseteq Y$. Llamaremos *ítems válidos* en un esquema dado a todos aquellos que puedan deducirse de las hipótesis por medio de una cadena de inferencias.

Un esquema de análisis sintáctico se dice *sólido* si verifica, para cualquier gramática y cadena de entrada, que todos los ítems finales válidos son correctos. Si verifica que todos los ítems finales correctos son válidos (es decir, si existe un árbol sintáctico marcado para una cadena, el sistema puede deducirlo) se dice que es *completo*. De un esquema que es a la vez sólido y completo se dice que es *correcto*.

Un esquema correcto puede usarse para obtener una implementación ejecutable de un analizador sintáctico mediante el uso de máquinas deductivas como las que se describen en (Shieber, Schabes, y Pereira, 1995; Gómez-Rodríguez, Vilares, y Alonso, 2006) para obtener los ítems finales válidos.

[1] Donde N es el conjunto de símbolos no terminales, Σ el alfabeto de símbolos terminales, P el conjunto de reglas de producción, y S el axioma o símbolo inicial de la gramática.

[2] Aunque en este trabajo nos centraremos en gramáticas independientes del contexto, los esquemas de análisis sintáctico (convencionales y con corrección de errores) pueden definirse análogamente para otros formalismos gramaticales.

3. Esquemas con corrección de errores

El formalismo de esquemas de análisis sintáctico descrito en la sección anterior no basta para definir analizadores sintácticos con corrección de errores que muestren un comportamiento robusto en presencia de entradas agramaticales, ya que los ítems finales se definen como aquéllos que contienen árboles sintácticos marcados que pertenecen a $Trees(G)$. Sin embargo, en un análisis sintáctico con corrección de errores, será necesario obtener ítems que representen "análisis aproximados" para frases que no tengan un análisis sintáctico exacto. Los análisis aproximados de estas frases agramaticales no pueden pertenecer a $Trees(G)$, pero deberían ser *similares* a algún elemento de $Trees(G)$. En este contexto, si medimos la "similaridad" mediante una función de distancia, podemos dar una nueva definición de ítems que permita generar análisis aproximados, y así extender los esquemas de análisis para soportar la corrección de errores.

Dada una gramática independiente del contexto $G = (N, \Sigma, P, S)$, llamaremos $Trees'(G)$ al conjunto de árboles finitos en los que los hijos de cada nodo están ordenados de izquierda a derecha y cada nodo está etiquetado con un símbolo de $N \cup \Sigma \cup (\Sigma \times \mathbb{N}) \cup \{\epsilon\}$. Nótese que $Trees(G) \subset Trees'(G)$.

Sea $d : Trees'(G) \times Trees'(G) \rightarrow \mathbb{N} \cup \{\infty\}$ una función de distancia que verifique los axiomas usuales de positividad estricta, simetría y desigualdad triangular.

Llamaremos $Trees_e(G)$ al conjunto $\{t \in Trees'(G) / \exists t' \in Trees(G) : d(t, t') \leq e\}$, es decir, $Trees_e(G)$ es el conjunto de árboles que tengan distancia e o menos a algún árbol válido de la gramática. Nótese que, por el axioma de positividad estricta, $Trees_0(G) = Trees(G)$.

Definición 1. *(árboles aproximados)*
Se define el conjunto de *árboles aproximados* para una gramática G y una función de distancia entre árboles d como $ApTrees(G) = \{(t, e) \in (Trees'(G) \times \mathbb{N}) / t \in Trees_e(G)\}$. Por lo tanto, un árbol aproximado es el par formado por un árbol y su distancia a algún árbol de $Trees(G)$. □

Este concepto de árboles aproximados nos permite definir con precisión los problemas que pretendemos resolver con el análisis sintáctico con corrección de errores. Dada una gramática G, una función de distancia d y una cadena $a_1 \ldots a_n$, el *problema del reconocimiento aproximado* consiste en determinar el mínimo $e \in \mathbb{N}$ tal que exista un árbol aproximado $(t, e) \in ApTrees(G)$ donde t es un árbol sintáctico marcado para la cadena. A un árbol aproximado así le llamaremos *árbol sintáctico marcado aproximado* para $a_1 \ldots a_n$.

Análogamente, el problema del *análisis sintáctico aproximado* consiste en encontrar el mínimo $e \in \mathbb{N}$ tal que exista un árbol sintáctico marcado aproximado $(t, e) \in ApTrees(G)$ para la cadena de entrada, y encontrar todos los árboles marcados aproximados de la forma (t, e) para la cadena.

Así, del mismo modo que el problema del análisis sintáctico se puede ver como un problema de encontrar árboles, el problema del análisis sintáctico aproximado se puede ver como un problema de encontrar árboles aproximados, que puede ser resuelto por un sistema deductivo análogo a los usados para el análisis sintáctico convencional, pero cuyos ítems contengan árboles aproximados.

Definición 2. *(ítems aproximados)*
Dada una gramática G y una función de distancia d, definimos *conjunto de ítems aproximados* como un conjunto $\mathcal{I}'$ tal que
$\mathcal{I}' \subseteq ((\bigcup_{i=0}^{\infty} \Pi_i) \cup \{\emptyset\})$
donde cada Π_i es una partición del conjunto $\{(t, e) \in ApTrees(G) / e = i\}$. □

Nótese que el concepto está definido de manera que cada ítem aproximado contiene árboles aproximados con un único valor de la distancia e. Definir directamente un conjunto de ítems aproximados usando una partición de $ApTrees(G)$ no sería práctico, dado que necesitamos que nuestros analizadores tengan en cuenta cuánta discrepancia acumula cada análisis parcial con respecto a la gramática, y esa información se perdería si nuestros ítems no estuviesen asociados a un único valor de e. Este valor concreto de e es lo que llamaremos *distancia de análisis* de un ítem ι, o $dist(\iota)$:

Definición 3. *(distancia de análisis)*
Sea $\mathcal{I}' \subseteq ((\bigcup_{i=0}^{\infty} \Pi_i) \cup \{\emptyset\})$ un conjunto de ítems aproximados tal como se ha definido arriba, y $\iota \in \mathcal{I}'$. La *distancia de análisis* asociada al ítem aproximado no vacío ι, $dist(\iota)$, se define como el (trivialmente único) valor de $i \in \mathbb{N} / \iota \in \Pi_i$.

En el caso del ítem aproximado vacío $\emptyset$, diremos que $dist(\emptyset) = \infty$. □

Definición 4. *(esquema de análisis sintáctico con corrección de errores)*
Sea d una función de distancia. Llamamos *esquema de análisis sintáctico con corrección de errores* a una función que asigna a cada gramática independiente del contexto G una terna $(\mathcal{I}', \mathcal{K}, D)$, donde $\mathcal{K}$ es una función tal que $(\mathcal{I}', \mathcal{K}(a_1 \dots a_n), D)$ es un *sistema de análisis instanciado con corrección de errores* para cada $a_1 \dots a_n \in \Sigma^*$. Un *sistema de análisis instanciado con corrección de errores* es una terna $(\mathcal{I}', H, D)$ tal que $\mathcal{I}'$ es un conjunto de ítems aproximado con función de distancia d, H es un conjunto de hipótesis tal que $\{a_i(\underline{a}_i)\} \in H$ para cada $a_i, 1 \leq i \leq n$, y D es un conjunto de pasos deductivos tal que $D \subseteq \mathcal{P}_{fin}(H \cup \mathcal{I}') \times \mathcal{I}'$.

Definición 5. *(ítems finales)*
El conjunto de *ítems finales* para una cadena de longitud n en un conjunto de ítems aproximados se define como $\mathcal{F}(\mathcal{I}', n) = \{\iota \in \mathcal{I} / \exists (t, e) \in \iota : t$ es un árbol sintáctico marcado para alguna cadena $a_1 \dots a_n \in \Sigma^\star\}$.

El conjunto de *ítems finales correctos* para una cadena $a_1 \dots a_n$ en un conjunto de ítems aproximados se define como $\mathcal{CF}(\mathcal{I}', a_1 \dots a_n) = \{\iota \in \mathcal{I} / \exists (t, e) \in \iota : t$ es un árbol sintáctico marcado para $a_1 \dots a_n\}$. □

Definición 6. *(distancia mínima de análisis)*
La distancia mínima de análisis para una cadena $a_1 \dots a_n$ en un conjunto de ítems aproximados $\mathcal{I}'$ se define como $MinDist(\mathcal{I}', a_1 \dots a_n) = min\{e \in \mathbb{N} : \exists \iota \in \mathcal{CF}(\mathcal{I}', a_1 \dots a_n) : dist(\iota) = e\}$. □

Definición 7. *(ítems finales mínimos)*
El conjunto de ítems finales mínimos para una cadena $a_1 \dots a_n$ en un conjunto de ítems aproximados $\mathcal{I}'$ se define como $\mathcal{MF}(\mathcal{I}', a_1 \dots a_n) = \{\iota \in \mathcal{CF}(\mathcal{I}', a_1 \dots a_n) / dist(\iota) = MinDist(\mathcal{I}', a1..an)\}$. □

Los conceptos de *ítems válidos*, *solidez*, *completitud* y *corrección* son análogos al caso de los esquemas de análisis convencionales. Nótese que los problemas de *reconocimiento aproximado* y *análisis aproximado* definidos con anterioridad para cualquier frase y gramática pueden resolverse obteniendo el conjunto de ítems finales mínimos en un conjunto de ítems aproximados. Cualquier esquema con corrección de errores correcto puede deducir estos ítems, dado que son un subconjunto de los ítems finales correctos.

4. Una función de distancia basada en la distancia de edición

Para especificar un analizador sintáctico mediante un esquema de análisis sintáctico con corrección de errores, es necesario decidir primero qué función de distancia utilizar para definir el conjunto de ítems aproximados.

Un esquema correcto obtendrá los análisis aproximados cuya distancia a un análisis correcto sea mínima. Por lo tanto, la función de distancia debe elegirse dependiendo del tipo de errores que se quiera corregir.

Supongamos una situación genérica donde nos gustaría corregir errores según la distancia de edición. La distancia de edición o distancia de Levenshtein (Levenshtein, 1966) entre dos cadenas es el número mínimo de inserciones, borrados o sustituciones de un único terminal que hacen falta para transformar cualquiera de las cadenas en la otra.

Una distancia d adecuada para este caso viene dada por el número de transformaciones sobre árboles que necesitamos para convertir un árbol en otro, si las transformaciones permitidas son insertar, borrar o cambiar la etiqueta de nodos frontera etiquetados con terminales marcados (o con ϵ). Por lo tanto, $d(t_1, t_2) = e$ si t_2 puede obtenerse a partir de t_1 mediante e transformaciones sobre nodos correspondientes a terminales marcados en $t1$, y $d(t_1, t_2) = \infty$ en los demás casos.

Nótese que, si bien en este trabajo utilizaremos esta distancia para ejemplificar la definición de analizadores con corrección de errores, el formalismo permite usar cualquier otra función de distancia entre árboles. Por ejemplo, en ciertas aplicaciones puede ser útil definir una distancia que compare todo el árbol (en lugar de sólo los nodos frontera) permitiendo la inserción, borrado o modificación de símbolos no terminales. Esto permite detectar errores sintácticos (como por ejemplo el uso de un verbo transitivo de forma intransitiva) independientemente de la longitud de los sintagmas implicados.

5. Algoritmo de Lyon

Lyon (1974) define un reconocedor con corrección de errores basado en el algoritmo de Earley. Dada una gramática G y una cadena $a_1 \dots a_n$, el algoritmo de Lyon devuelve la mínima distancia de edición a una cadena válida de $L(G)$.

En esta sección, usaremos nuestro formalismo para definir un esquema de análisis sintáctico con corrección de errores para el algoritmo de Lyon. Esto nos servirá como ejemplo de esquema con corrección de errores, y nos permitirá probar la corrección del algoritmo, implementarlo fácilmente y crear una variante con corrección regional de errores, como se verá más tarde.

El esquema para el algoritmo de Lyon está definido para la función de distancia d de la sección 4. Dada una gramática independiente del contexto G y una cadena de entrada $a_1 \dots a_n$, el esquema *Lyon* es el que nos proporciona un sistema de análisis instanciado $(\mathcal{I}', H, D)$ donde $\mathcal{I}'$ y D se definen como sigue:

$\mathcal{I}'_{Lyon} = \{[A \to \alpha \bullet \beta, i, j, e] / A \to \alpha\beta \in P \wedge i, j, e \in \mathbb{N} \wedge 0 \leq i \leq j\}$

donde usamos $[A \to \alpha \bullet \beta, i, j, e]$ como notación para el conjunto de árboles aproximados (t, e) tales que t es un árbol de análisis parcial con raíz A donde los hijos directos de A son los símbolos de la cadena $\alpha\beta$, y los nodos frontera de los subárboles con raíz en los símbolos de α forman una cadena de terminales marcados de la forma $\underline{a}_{i+1} \dots \underline{a}_j$, mientras que los nodos etiquetados β son hojas. Nótese que para definir este conjunto de ítems aproximados se utiliza la distancia d definida en la sección anterior, que es la que condiciona los valores de e en esta notación.

El conjunto de pasos deductivos, D, para el algoritmo de Lyon se define como la unión de los siguientes:

$$D^{Initter} = \{\vdash [S \to \bullet\gamma, 0, 0, 0]\}$$

$$D^{Scanner} = \{[A \to \alpha \bullet x\beta, i, j, e], [x, j, j+1] \vdash [A \to \alpha x \bullet \beta, i, j+1, e]\}$$

$$D^{Completer} = \{[A \to \alpha \bullet B\beta, i, j, e_1], [B \to \gamma\bullet, j, k, e_2] \vdash [A \to \alpha B \bullet \beta, i, k, e_1 + e_2]\}$$

$$D^{Predictor} = \{[A \to \alpha \bullet B\beta, i, j, e] \vdash [B \to \bullet\gamma, j, j, 0]\}$$

$$D^{ScanSubstituted} = \{[A \to \alpha \bullet x\beta, i, j, e], [b, j, j+1] \vdash [A \to \alpha x \bullet \beta, i, j+1, e+1]\}$$

$$D^{ScanDeleted} = \{[A \to \alpha \bullet x\beta, i, j, e] \vdash [A \to \alpha x \bullet \beta, i, j, e+1]\}$$

$$D^{ScanInserted} = \{[A \to \alpha \bullet \beta, i, j, e], [b, j, j+1] \vdash [A \to \alpha \bullet \beta, i, j+1, e+1]\}$$

$$D^{DistanceIncreaser} = \{[A \to \alpha \bullet \beta, i, j, e] \vdash [A \to \alpha \bullet \beta, i, j, e+1]\}$$

Los pasos *Initter*, *Scanner*, *Completer* y *Predictor* son similares a los del algoritmo de Earley, con la diferencia de que tenemos que llevar cuenta de la distancia asociada a los árboles aproximados de nuestros ítems. Nótese que el *Completer* suma las distancias en sus antecedentes, dado que su ítem consecuente contiene árboles construidos combinando los de los dos ítems antecedente, y que por lo tanto contendrán discrepancias provenientes de ambos.

Los pasos *ScanSubstituted*, *ScanDeleted* y *ScanInserted* son pasos de corrección de errores, y permiten leer símbolos no esperados de la cadena a la vez que se incrementa la distancia. *ScanSubstituted* sirve para corregir un error de substitución en la entrada, *ScanDeleted* corrige un error de borrado, y *ScanInserted* un error de inserción.

El conjunto de ítems finales y el subconjunto de ítems finales correctos son:

$\mathcal{F} = \{[S \to \gamma\bullet, 0, n, e]\}$

$\mathcal{CF} = \{\iota = [S \to \gamma\bullet, 0, n, e] / \exists (t, e) \in \iota : t$ es un árbol sintáctico marcado para $a_1 \dots a_n\}$

El paso *DistanceIncreaser* asegura que todos los ítems finales no mínimos son generados (cosa que se requiere para la completitud). En implementaciones prácticas del analizador, como la propuesta original de Lyon (1974), normalmente no interesa la completitud estricta sino sólo el obtener los análisis de distancia mínima, así que el *DistanceIncreaser* no es necesario y puede simplemente omitirse.

Probar la solidez del esquema *Lyon* es demostrar que todos los ítems finales válidos en sus sistemas de análisis asociados son correctos. Esto se demuestra probando la proposición, más fuerte, de que todos los ítems válidos son correctos. Esto se puede demostrar analizando por separado cada paso deductivo y demostrando que si sus antecedentes son correctos, el consecuente también lo es.

Para probar la completitud del esquema *Lyon* (es decir, que todos los ítems finales correctos son válidos en el esquema), tenemos en cuenta que dichos ítems finales son de la forma $[S \to \alpha\bullet, 0, n, e]$, y lo demostramos por inducción en la distancia e. El caso base se prueba partiendo de la completitud del esquema *Earley* (Sikkel, 1998), y el paso inductivo

se demuestra mediante una serie de funciones de transformación de ítems que permiten inferir la validez de cualquier ítem final correcto con distancia $e+1$ a partir de la de un ítem con distancia e.

6. Implementación

Un esquema con corrección de errores completo permite deducir todos los análisis aproximados válidos para una cadena dada. Sin embargo, al implementar un analizador con corrección de errores en la práctica, no queremos obtener todos los posibles análisis aproximados (cosa que sería imposible en tiempo finito, dado que hay una cantidad infinita de análisis). Lo que buscamos, como mencionamos en la definición del problema del análisis sintáctico aproximado, son los análisis aproximados con distancia mínima.

Cualquier esquema correcto que verifique una propiedad que llamaremos *completitud finita* puede adaptarse para resolver el problema del análisis sintáctico aproximado en tiempo finito, generando sólo los análisis de distancia mínima, si le añadimos algunas restricciones. Para ello, definiremos algunos conceptos que nos llevarán a la noción de esquema finitamente completo.

Definición 8. *(esquema acotado)*
Sea S un esquema de análisis sintáctico con corrección de errores que asigna a cada gramática G una terna $(\mathcal{I}', \mathcal{K}, D)$. El *esquema acotado* asociado a S con cota b, denotado $B_b(S)$, es el que asigna a cada gramática G el sistema de análisis $Bound_b(S(G)) = Bound_b(\mathcal{I}', \mathcal{K}, D) = (\mathcal{I}', \mathcal{K}, D_b)$, donde $D_b = \{((a_1, a_2, \ldots, a_c), c) \in D : dist(c) \leq b\}$. □

En otras palabras, un esquema acotado es una variante de un esquema con corrección de errores que no permite deducir ítems con distancia asociada mayor que la cota b.

Definición 9. *(completitud hasta una cota)*
Diremos que un esquema de análisis con corrección de errores S es *completo hasta una cota* b si, para cualquier gramática y cadena de entrada, todos los ítems finales correctos cuya distancia asociada no sea mayor que b son válidos. □

Definición 10. *(completitud finita)*
Diremos que un esquema de análisis con corrección de errores S es *finitamente completo* si, para todo $b \in \mathbb{N}$, el esquema acotado $B_b(S)$ es completo hasta la cota b. □

Nótese que un esquema finitamente completo es siempre completo, ya que podemos hacer b arbitrariamente grande.

El esquema *Lyon* cumple la propiedad de ser finitamente completo, cosa que se puede demostrar de forma análoga a su completitud. Por otra parte, es fácil ver que, si disponemos de una máquina deductiva que pueda ejecutar esquemas de análisis sintáctico, cualquier esquema con corrección de errores S que sea finitamente completo puede utilizarse para construir un analizador que resuelva el problema del análisis sintáctico aproximado en tiempo finito, devolviendo todos los análisis aproximados válidos de distancia mínima sin generar ningún análisis de distancia no mínima. La manera más simple de hacerlo es la siguiente:

```
function AnalizadorRobusto ( str:cadena )
                              : conjunto de ítems
  b = 0; //máxima distancia permitida
  while ( true ) {
    computar validItems = v(Bound_b(S(G)),str);
    finalItems = {i ∈validItems /i es un ítem final };
    if ( finalItems ≠ ∅ ) return finalItems;
    b = b+1;
  }
```

donde la función `v(sys,str)` computa todos los ítems válidos en el sistema de análisis `sys` para la cadena `str` , y puede implementarse como en (Shieber, Schabes, y Pereira, 1995; Gómez-Rodríguez, Vilares, y Alonso, 2006).

Es fácil demostrar que, si el problema del análisis aproximado tiene alguna solución para una cadena dada (cosa que, bajo nuestra definición de distancia, siempre sucede), entonces este algoritmo la encuentra en tiempo finito. En la práctica, podemos hacerle varias optimizaciones para mejorar el tiempo de ejecución, como utilizar los ítems generados en cada iteración como hipótesis de la siguiente en lugar de inferirlos de nuevo. Nótese que esta variante de máquina deductiva puede ejecutar cualquier esquema con corrección de errores, no sólo el de Lyon.

6.1. Implementación con corrección regional

Si un analizador con corrección de errores es capaz de encontrar todos los análisis aproximados de distancia mínima para cualquier cadena dada, como el de la sección 6, se le llama analizador con corrección de errores *global*. En la práctica, los correctores globales pueden volverse muy ineficientes si queremos analizar cadenas largas o utilizar gramáticas

con miles de producciones, como es usual en el procesamiento del lenguaje natural.

Una alternativa más eficiente es la corrección de errores *regional*, que se basa en aplicar corrección de errores a una *región* que rodee al punto en que no se pueda continuar el análisis. Los analizadores regionales garantizan encontrar siempre una solución óptima; pero si existen varias no garantizan encontrarlas todas.

Los algoritmos con corrección regional basados en estados, como los que se definen en (Vilares, Darriba, y Ribadas, 2001), suelen estar asociados a una implementación particular. Los esquemas de análisis sintáctico con corrección de errores nos permiten definir analizadores regionales más generales, basados en ítems, donde las regiones son conjuntos de ítems. Los analizadores regionales pueden obtenerse de los globales de un modo general, tal que el analizador regional siempre devolverá una solución óptima si el analizador global del que proviene es correcto y finitamente completo. Para ello, utilizamos la noción de *función de progreso*:

Definición 11. *(función de progreso)*
Sea $\mathcal{I}'$ un conjunto de ítems aproximados. Una *función de progreso* para $\mathcal{I}'$ es una función $f_p : \mathcal{I}' \rightarrow \{p \in \mathbb{N} / 0 \leq p \leq k\}$, donde k es un número natural llamado el *progreso máximo.* □

Sea S un esquema de análisis sintáctico con corrección de errores correcto y finitamente completo, y f_p una función de progreso para su conjunto de ítems. Podemos implementar un analizador con corrección regional basado en S de esta manera:

```
function AnalizadorRegional ( str:cadena )
                              : conjunto de ítems
  b = 0; //distancia máxima permitida
  maxProgr = 0; //límite superior región
  minProgr = 0; //límite inferior región
  while ( true ) {
    computar validItems = v'(Bound_b(S(G)),str,
                                 minProgr,maxProgr);
    finalItems = {i ∈ validItems /i es un ítem final };
    if ( finalItems ≠ ∅ ) return finalItems;

    newMax = max{p ∈ N/∃i ∈ validItems /f_p(i) = p}

    if ( newmaxProgr > maxProgr ) {
        maxProgr = newMax; minProgr = newMax;
    }
    else if ( minProgr > 0 ) minProgr = minProgr−1;
    else b = b+1;
  }
```

donde la función `v'(ded,str,min,max)` computa todos los ítems válidos en el sistema deductivo `ded` para la cadena `str` con la restricción de que los pasos deductivos de corrección de errores sólo se lanzan si al menos uno de sus antecedentes, ι, verifica que $\texttt{minProgr} \leq f_p(\iota) \leq \texttt{maxProgr}$.

Este analizador regional devuelve siempre una solución óptima bajo la condición de que S sea correcto y finitamente completo. Para que además el analizador regional sea eficiente, debemos definir la función de progreso de modo que sea una buena aproximación de cuán "prometedor" es un ítem de cara a alcanzar un ítem final[3].

Una función simple pero adecuada en el caso del analizador *Lyon* es $f_{p_j}([A \rightarrow \alpha \bullet \beta, i, j, e]) = j$, que simplemente evalúa un ítem de acuerdo con su índice j. Otra alternativa es $f_{p_{j-i}}([A \rightarrow \alpha \bullet \beta, i, j, e]) = j - i$. Ambas funciones premian a los ítems que han llegado más a la derecha en la cadena de entrada, y toman valores máximos para los ítems finales.

7. Resultados empíricos

Para probar nuestros analizadores y estudiar su rendimiento, hemos usado el sistema descrito en (Gómez-Rodríguez, Vilares, y Alonso, 2006) para ejecutar el esquema *Lyon* con corrección global y regional. La función de progreso usada para el caso regional es la función f_{p_j} definida más arriba.

La gramática y frases utilizadas para las pruebas provienen del sistema DARPA ATIS3. En particular, hemos usado las mismas frases de prueba utilizadas en (Moore, 2000). Este conjunto de pruebas es adecuado para nuestros propósitos, dado que proviene de una aplicación real y contiene frases agramaticales. En particular, 28 de las 98 fras-

[3] Los criterios para determinar una buena función de progreso son similares a los que caracterizan a una buena heurística en un problema de búsqueda informada. Así, la función de progreso ideal sería una tal que $f(\iota) = 0$ si ι no fuese necesario para deducir un ítem final, y $f(\iota) > f(\kappa)$ si ι puede dar lugar a un ítem final en menos pasos que κ. Evidentemente esta función no se puede usar, pues hasta completar el proceso deductivo no sabemos si un ítem dado puede conducir o no a un ítem final; pero las funciones que proporcionen una buena aproximación a esta heurística ideal darán lugar a analizadores eficientes. En el caso degenerado en el que se devuelve $f(\iota) = 0$ para cualquier ítem, la función de progreso no proporciona ninguna información y el analizador con corrección regional equivale al global.

es del conjunto lo son. Al ejecutar nuestros analizadores con corrección de errores, encontramos que la distancia de edición mínima a una frase gramatical es 1 para 24 de ellas (es decir, estas 24 frases tienen una posible corrección con un solo error), 2 para dos de ellas, y 3 para las dos restantes.

Dist. Mín.	Nº de Frases	Long. Media	Ítems med. (Global)	Ítems med. (Regional)	Mejora (%)
0	70	11.04	37558	37558	0 %
1	24	11.63	194249	63751	65.33 %
2	2	18.50	739705	574534	22.33 %
3	2	14.50	1117123	965137	13.61 %
>3	0	n/a	n/a	n/a	n/a

Cuadro 1: Rendimiento de los analizadores globales y regionales al analizar frases del conjunto de prueba ATIS. Cada fila corresponde a un valor de la distancia mínima de análisis (o contador de errores).

Como podemos ver, la corrección regional reduce la generación de ítems en un factor de tres en frases con un único error. En frases con más de un error, las mejoras son menores: esto es porque, antes de devolver soluciones con distancia $d+1$, el analizador regional genera todos los ítems válidos con distancia d. De todos modos, debe tenerse en cuenta que el tiempo de ejecución crece más rápido que el número de ítems generados, así que estas mejoras relativas en los ítems se reflejan en mejoras relativas mayores en los tiempos de ejecución. Además, en situaciones prácticas es esperable que las frases con varios errores sean menos frecuentes que las que sólo tienen uno, como en este caso. Por lo tanto, los tiempos más rápidos hacen a los analizadores con corrección regional basados en ítems una buena alternativa a los correctores globales.

8. Conclusiones y trabajo actual

Hemos presentado los esquemas de análisis sintáctico con corrección de errores, un formalismo que puede utilizarse para definir, analizar y comparar fácilmente analizadores sintácticos con corrección de errores. Estos esquemas son descripciones sencillas y declarativas de los algoritmos que capturan su semántica y abstraen los detalles de implementación.

En este trabajo, los hemos utilizado para describir un analizador con corrección de errores basado en Earley — descrito por primera vez en (Lyon, 1974) —, para probar su corrección, para generar una implementación deductiva del algoritmo original, y para obtener un analizador más rápido, basado en corrección regional, a partir del mismo esquema. Los métodos utilizados para obtener estos resultados son genéricos y se pueden aplicar en otros analizadores.

En la actualidad, estamos trabajando en la definición de una función que transforma esquemas convencionales correctos que verifiquen ciertas condiciones en esquemas con corrección de errores correctos. Esta transformación permite obtener automáticamente analizadores sintácticos con corrección de errores regional y global a partir de esquemas convencionales como los correspondientes a los analizadores CYK o Left-Corner.

Bibliografía

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

Gómez-Rodríguez, C., J. Vilares, y M. A. Alonso. 2006. Automatic generation of natural language parsers from declarative specifications. En *Proc. of STAIRS 2006*, Riva del Garda, Italy. Long version available at http://www.grupocole.org/GomVilAlo2006a_long.pdf.

Levenshtein, V. I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710.

Lyon, G. 1974. Syntax-directed least-errors analysis for context-free languages: a practical approach. *Comm. ACM*, 17(1):3–14.

Moore, R. C. 2000. Improved left-corner chart parsing for large context-free grammars. En *Proc. of the 6th IWPT, pages 171–182, Trento, Italy*, páginas 171–182.

Shieber, S. M., Y. Schabes, y F. C. N. Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1–2):3–36, July-August.

Sikkel, K. 1998. Parsing schemata and correctness of parsing algorithms. *Theoretical Computer Science*, 199(1-2):87-103.

Sikkel, K. 1997. *Parsing Schemata — A Framework for Specification and Analysis of Parsing Algorithms*. Springer-Verlag, Berlin/Heidelberg/New York.

Vilares, M., V. M. Darriba, y F. J. Ribadas. 2001. Regional least-cost error repair. *Lecture Notes in Computer Science*, 2088:293–301.