DETECCIÓN DE ERRORES SINTÁCTICOS BAJO EL ALGORITMO DE EARLEY FASE II

Juan C. Vázquez¹, Leticia Constable¹, Fanny Montoya¹, Germán Parisi¹, Marcelo Arcidiacono¹,
Valentina Brocanelli¹

Grupo GAALA / Departamento de Ingeniería en Sistemas de Información /
Facultad Regional Córdoba / Universidad Tecnológica Nacional
¹{jcvazquez, leticiaconstable, fmontoya, germanparisi, marceloarcidiacono,
valebroca}@gmail.com

RESUMEN

Se presenta la segunda fase de un proyecto y la línea de investigación a la cual pertenece, que revisa el algoritmo de análisis sintáctico presentado por Jay Earley a principios de 1970, el cual estudia su aplicabilidad efectiva en el desarrollo de compiladores, sobre todo en lo referido a la especificidad (localización y causa) con que pueden ser detectados e informados errores sintácticos de programas analizados.

En esta parte del proyecto se pretende, por un lado traducir las herramientas ya realizadas en la primera fase a lenguaje Python y diseñar un analizador sintáctico y semántico operativo para las técnicas ya diseñadas de construcción de derivaciones y detección de errores usando el algoritmo de Earley; por otro lado, se pone en foco la enseñanza de teoría de lenguajes y autómatas, por lo que se intenta que los programas desarrollados se ajusten a la teoría lo más posible y sean claros y sencillos de entender, explicando dónde la práctica real indica alguna modificación de la teoría pura subyacente.

Palabras clave: análisis sintáctico, algoritmo Earley, errores sintácticos.

CONTEXTO

La segunda fase del presente proyecto, al igual que la primera de 2014-2017 [Vázquez-2015], es parte de una línea de investigación que se inició hace años sobre el estudio de los lenguajes formales y las máquinas abstractas, tanto en el aspecto teórico-conceptual como

aplicativo y en su enseñanza en la carrera de Ingeniería en Sistemas de Información.

La línea llevó a cabo indagaciones sobre herramientas didácticas automatizadas para la enseñanza, simuladores, distintos tipos de autómatas (finitos, con pila, máquinas de Turing, autómatas celulares), algoritmos de análisis léxico y sintáctico, y colateralmente estudios sobre aprendizaje automático y procesamiento de lenguaje natural, todos ellos desde la carrera de Ingeniería en Sistemas de Información de la Facultad Regional Córdoba de la Universidad Tecnológica Nacional.

Todos los proyectos realizados se evaluaron y acreditaron por la Secretaría de Ciencia, Tecnología y Posgrado de la UTN.

Estos proyectos han participado del grupo de investigación en inteligencia artificial (GIA), del grupo de desarrollo, investigación y transferencia de sistemas de información (GIDTSI, hoy centro CIDS), y dieron origen al grupo de investigación en aprendizaje automático, lenguajes y autómatas (GAALA) que aglutina nuestra actual línea.

Los integrantes de esta fase del proyecto son docentes en la asignatura Sintaxis y Semántica de los Lenguajes de la carrera, por lo que el objetivo perseguido está fuertemente orientado a la profundización de saberes, a la construcción de explicaciones y algoritmos con códigos claros que puedan ser entendidos por estudiantes y transferidos a la cátedra,

1. INTRODUCCIÓN

El estudio de lenguajes formales sigue las ideas de Noam Chomsky quien desarrolló la teoría de gramáticas generativas [Chomsky-1957] en lingüística, la cual fue rápidamente adoptada para la especificación de lenguajes

de programación de computadores desde fines de 1950 [Backus-1959, Naur-1960]. La teoría de la computación, que puede considerarse iniciada por Alonzo Church, Emil Post y en especial Alan Turing en 1936, sirvió como herramienta conceptual de base para el diseño de los algoritmos que implementan el análisis y la traducción actuales de estos lenguajes; se construyeron así intérpretes y compiladores que traducen un código fuente en un código ejecutable.

Las primeras computadoras construidas se programaron físicamente (cableado), luego se permitió el ingreso de instrucciones en código binario que la máquina entendía y ejecutaba, posteriormente esto se mejoró con el diseño de lenguajes mnemónicos (ensambladores) simples cuyas instrucciones constituían una función uno a uno con el binario de máquina y, cuando la complejidad de los programas creció, se hizo necesario diseñar lenguajes más expresivos y al alcance de una mayor cantidad de gente que los pudiera utilizar. Pero ahora estos lenguajes no tenían una relación directa con el hardware, por lo que los programas de computación descritos con ellos (programas fuente), requirieron de un complejo procesamiento antes de poder ser ejecutados en un computador.

A riesgo de ser reiterativos con lo expuesto para la primera fase del proyecto, pero para que se entienda la problemática, se explican a continuación las ideas y problemas del tema.

La traducción de programas fuentes para que sean ejecutables en máquina, es realizada por programas llamados compiladores, que deben poder *entender* el algoritmo descrito en el lenguaje de alto nivel para generar luego otro que haga lo mismo (sea semánticamente equivalente) pero en lenguaje de máquina.

La técnica ideada para que esto se pueda hacer (*entender* un algoritmo no es sencillo), es expresar el programa fuente como un árbol (de análisis sintáctico) o representaciones equivalentes (una derivación en la gramática o un código en un lenguaje abstracto sencillo).

Este proceso constituye la fase de análisis del compilador; involucra un estudio léxico

que identifica secuencias de símbolos válidas del lenguaje (componentes léxicos o tokens), uno sintáctico durante el cual se verifica si las frases formadas por los tokens cumplen las reglas de estructura del lenguaje (ajustan a su gramática) y un estudio semántico, donde se asignan significados a las frases correctas del programa; el resultado es usualmente un árbol, una derivación o una representación intermedia en un lenguaje abstracto sencillo y de fácil conversión a código de máquina. Desde esta representación del algoritmo, la fase de síntesis del compilador optimizará el código intermedio y generará el programa objeto semánticamente equivalente al fuente.

[Aho-1990] indica que “si todo lo que tuviera que hacer un compilador es traducir programas correctos, se simplificaría en gran medida su construcción”.

Pero los programas fuente frecuentemente contienen errores de variados tipos: léxicos, sintácticos, semánticos, de lógica, etc., por lo que además de realizar la traducción, una función secundaria pero muy importante del análisis del compilador, es detectar e informar sobre los errores tan claramente y tan pronto como sea posible, para lograr *especificidad* (la localización y la causa), con el objetivo de orientar al programador en su corrección.

Análisis Sintáctico y Semántico

Las gramáticas generativas de Chomsky, son un formalismo que permite especificar un lenguaje mediante una colección de símbolos terminales (que constituirán las palabras del lenguaje que generan), de símbolos auxiliares o no terminales (usados para formar reglas de reescritura entre palabras), un símbolo inicial llamado axioma (un no terminal distinguido) y un conjunto de producciones (reglas de reescritura que establecen la estructura del lenguaje); con esta notación se especifican lenguajes de cuatro tipos distintos (Jerarquía de Chomsky), los que se diferencian sólo por el formato y las restricciones de sus reglas. Todos los lenguajes de programación actuales están especificados con gramáticas tipo 2 de Chomsky y se denominan lenguajes de contexto libre o independientes (LIC).

El llamado *problema de análisis sintáctico* es un problema de membresía: determinar si una secuencia finita de símbolos terminales constituye o no una palabra del lenguaje descrito por la gramática dada.

Este problema es resuelto por la teoría de la computación y los lenguajes formales con un formalismo matemático llamado autómatas con pila (AP), una máquina secuencial abstracta de estados finitos, que usando una memoria de pila *decide* la pertenencia o no de palabras a un LIC; su funcionamiento es en esencia no determinista (varios caminos de análisis posibles). Muchas técnicas para implementar como algoritmos estas máquinas han sido desarrolladas y para tratar con el no determinismo (análisis descendente recursivo, tabular descendente (LL), tabular ascendente (LR), técnicas predictivas y variaciones).

En general, estas técnicas se restringieron a subclases de gramáticas independientes del contexto que cumplen con restricciones sobre el formato de sus producciones o en sus características (llamadas deterministas), cuyos lenguajes especificados podían procesarse con algoritmos eficientes.

Pero no solo la pertenencia de una palabra a un lenguaje es lo que busca un compilador; una vez determinado que la palabra está bien escrita es necesario saber *qué significa*, cuál es la acción que esa palabra debe realizar al ejecutarse en una máquina; hace falta realizar *análisis semántico* que muestre el significado del programa.

La técnica usada consiste en asociar a cada producción de la gramática un conjunto de reglas semánticas que especifiquen qué debe ejecutarse efectivamente en cada paso del reconocimiento de los programas analizados. Pero al modificar las gramáticas para que cumplan las restricciones necesarias para su implementación mediante una de las técnicas señaladas, suelen cambiarse las producciones con lo cual, de alguna forma, se rompe con la asignación de reglas semánticas.

Otras técnicas para gramáticas generales de LIC, fueron propuestas y en su momento se mostraron ineficientes para ser procesadas por

las computadoras de la época (Tomita -copias de LR en paralelo-, CYK (Cocke-Younger-Kasami -programación dinámica- [requiere formato especial], Earley, entre otras).

El Problema Que Nos Ocupa

El psicólogo Jay Earley presentó como su tesis doctoral [Earley-1970] el algoritmo de análisis sintáctico que lleva su nombre y que permite analizar cadenas de cualquier LIC, sin imponer restricciones a los formatos de sus reglas gramaticales, lo que en principio trae una fuerte ventaja sobre la semántica que se haya especificado anclada en las reglas de reescritura.

El algoritmo resulta no determinista y se consideró poco eficiente en su momento como para ser puesto a funcionar en computadores [Aho-1990]; Earley analizó su complejidad máxima como $O(n^3)$ al operar sobre gramáticas ambiguas. En su artículo a la comunidad publicado en ACM, el autor explicó el funcionamiento de su algoritmo, pero no cómo indicar específicamente los errores en las cadenas analizadas.

Posteriormente trabajos han propuesto formas más eficientes del algoritmo de Earley [McLean-1996, Aycock-2002, Trevor-2010], pero no se suelen enfocar respecto de su uso efectivo para un lenguaje de programación, y en especial, en el tema de cómo informar errores detectados. Por ello se llega al planteo de las siguientes preguntas:

- ¿Cómo se puede indicar la presencia de un error en una palabra de un LIC, usando el algoritmo de análisis sintáctico de Earley?
- ¿Qué tan específica (en cuanto a su localización y causa) puede ser esta indicación para que sirva para su reparación?
- ¿Cuán eficiente puede ser esta operación?

Motivación de Investigación y Docencia

El proyecto busca responder las preguntas anteriores y la comparación con la respuesta que dan a las mismas los usuales algoritmos de análisis sintáctico predictivo. Por ello también se quiere implementar los algoritmos

claramente para lograr contrastar su desempeño y funcionamiento entre ellos y en comparación con el de Earley.

Habiendo utilizado los compiladores de compiladores (lex, flex, jflex, yacc, byacc, bison, jcup y otros), se constató que al incorporar tantas alternativas debido a su gran versatilidad y a las mejoras y agregados realizados a lo largo del tiempo, que los analizadores lexicográficos y sintácticos que producen no son manejables por alumnos con poca experiencia en programación y son aún difíciles de analizar para aquellos con más experiencia. Por ello nuestro objetivo de propia programación de los algoritmos.

Siendo docentes de la cátedra de segundo año de la carrera, Sintaxis y Semántica de los Lenguajes, y coautores del libro [Giró-2013] actualmente en uso en la misma, durante años hemos enseñado en forma introductoria temas de lenguajes formales, teoría de autómatas, traductores y complejidad. Como ya se comentó en el informe de la primera fase del proyecto, frente a una asignatura de fuerte contenido conceptual y abstracto, con estudiantes noveles en programación, una estrategia para fijar conocimientos y bajar las abstracciones a la realidad ha sido el uso de simuladores de máquinas abstractas, para la comprobación de la ejercitación propuesta y trabajos prácticos; otra estrategia es hacer que programen sus propias soluciones, pero aún no están preparados para ello. Sin embargo, sí se cree que están preparados para entender un código, por lo que queremos mostrar código de soluciones ya desarrolladas en un lenguaje que los alumnos entiendan, tal vez con errores a corregir y con discusiones sobre su funcionamiento detallado.

Es por lo anterior, que un objetivo del proyecto sea también lograr la construcción de código sencillo y claro de analizadores léxicos y sintácticos de distinto tipo, en lenguajes que el alumno conozca, que se cree redundará en una mejor comprensión de los temas impartidos en la asignatura.

2. LÍNEAS DE INVESTIGACIÓN Y DESARROLLO

Como se anteriormente, el presente trabajo forma parte de la línea de investigación en Lenguajes Formales y Autómatas del grupo GAALA, tanto en sus aspectos teóricos y prácticos, como en la enseñanza de los temas relacionados en las carreras de Ingeniería.

Los integrantes del proyecto forman parte de la cátedra SSL en UTN-FRC y han participado de proyectos sobre la temática anteriores (GDH, RNA-AC, autómatas con pila, simuladores, etc.)

3. RESULTADOS OBTENIDOS/ESPERADOS

El proyecto ha definido un pequeño y sencillo lenguaje de programación (el de la máquina RAM) y está afinando el desarrollo de otro lenguaje un tanto más complejo para pruebas de algoritmos.

El lenguaje RAM definido, además de ser utilizado como lenguaje de ejemplo sobre el cual efectuar análisis léxico, sintáctico y semántico, sirve como lenguaje intermedio de representación. Ya se tiene en funcionamiento un simulador de máquina RAM que a modo de intérprete podrá ejecutar el resultado de los análisis antes indicados.

Los objetivos pueden resumirse en dos grandes partes:

- a) Determinar con qué especificidad (su tipo y su ubicación) pueden informarse errores sintácticos, usando el método de Earley de análisis sintáctico.
- b) Generar código y explicaciones claras para acrecentar el material didáctico para la enseñanza de Sintaxis y Semántica de los Lenguajes.

En la primera fase ya se logró el primer objetivo desarrollando un algoritmo para la construcción de las posibles derivaciones que en paralelo genera el algoritmo de Earley al analizar un programa fuente, con lo cual se puede informar de errores con especificidad. En esta segunda fase se pretende obtener este algoritmo programado en lenguaje Python y con un análisis semántico inicial incluido.

El segundo objetivo, fue resuelto en lenguaje Java en la primera fase del proyecto, pero nuestros estudiantes ahora manejan otro

lenguaje (Python) por lo cual se debe traducir lo realizado a este lenguaje.

4. FORMACIÓN DE RECURSOS HUMANOS

El grupo del proyecto de investigación y está conformado por investigadores formados, investigadores noveles y estudiantes de grado de la UTN-FRC.

Los investigadores pertenecen actualmente a la cátedra SSL del Dpto. de Ingeniería en Sistemas de Información. Los estudiantes por su parte, son becados de los últimos años de la carrera, que aprenden a realizar actividades de investigación y desarrollo y se relacionan trabajando en equipo con sus ex profesores.

Como en proyectos anteriores, se espera que el proyecto genere temática para posibles tesis de grado (en el caso de los alumnos) y de posgrado para los ingenieros integrantes.

5. BIBLIOGRAFÍA

[Aho-1990] Aho A., Sethi R., Ullman J.; *Compiladores: principios, técnicas y herramientas*; Addison Wesley Iberoamericana S.A.; 1990; D.F., México.

[Aycock-2002] Aycock J., Horspool N.; *Practical Earley Parsing*; The Computer Journal, British Computer Society, Vol 45 – Nr. 6 – pp 620-630; Apr 2002; UK.

[Backus-1959] Backus J.; *The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference*; Proceedings of International Conference on Information Processing, UNESCO, pp 125-132; 1959; USA.

[Chomsky-1957] Chomsky N.; *Syntactic Structures*; Mouton, The Hague; 1957; Berlin, Alemania.

[Earley-1970] Earley J.; *An Efficient Context-Free Parsing Algorithm*; Communications of ACM, Vol. 13 – Nr. 2 – pp 94-102; Feb 1970; NY, USA.

[Giró-2013] Giró J., Vázquez J., Meloni B., Constable L.; *Elementos de Máquinas Abstractas y Gramáticas Formales*; Edición

Propia (en revisión por Alfaomega); Mar 2013; Córdoba, Argentina.

[Knuth-1965] Knuth D.; *On the translation of Language from Left to Right*; Information and Control, Elsevier; Vol. 8 – Nr. 6 – pp 607-639; Dec 1965; USA.

[McLean-1996] McLean P., Horspool N.; *A Faster Earley Parser*; Proceedings of International Conference on Compiler Construction, Springer, pp 281-293; 1996; Canada.

[Naur-1960] Naur P. (Editor); *Report on the Algorithmic Language ALGOL 60*; Communications of ACM, Vol. 3 – Nr. 5 – pp 299-314; May 1960; USA.

[Trevor-2010] Trevor J., Mandelbaum Y.; *Efficient Early Parsing with Regular Right-Hand Sides*; Electronic Notes in Theoretical Computer Science, Elsevier; Vol. 253 – Nr. 7 – pp 137-148; Sep 2010; Amsterdam, The Netherlands.

[Vázquez-2015] Vázquez J. et al.; *Detección de errores sintácticos bajo el algoritmo de Earley*, WICC 2015, Salta.

<http://sedici.unlp.edu.ar/handle/10915/45309>