https://repository.uaeh.edu.mx/revistas/index.php/xikua/issue/archive

XIKUA Boletín Científico de la Escuela Superior de Tlahuelilpan

Publicación semestral, Vol. 13, No. 25 (2025) 31-38

Xikua

ISSN: 2007-4948

Algoritmo de conversión de una gramática libre de contexto a un autómata de pila

Conversion algorithm from a context-free grammar to a push down automata

Ofelia Gutiérrez Giraldi [a] , Martha Martínez Moreno [b], Gabriela Clavel Martínez [c], Pedro J Pérez Hernández [d].

Abstract:

Context-Free Grammars (CFG) and Pushdown Automata (PDA) are fundamental in the theory of computation and have various applications in this field, such as formal language analysis, compiler and interpreter development, natural language processing, structure modeling, eTherefore, the Computer Systems Engineer should have a solid foundation to address the variety of challenges they will face. However, CFGs present certain issues, with the main one being the difficulty of analysis and the ambiguity they exhibit. This is where PDA comes in, being a structure equivalent to CFG but equally powerful, yet easier to represent and analyze. Based on this concept, we propose developing an algorithm to convert CFG to PDA. The user will provide a CFG, and the algorithm will return its equivalent ADP, allowing the user to better understand the relationship between both structures and realize that there is an alternative to CFG that is equally useful but more flexible. The challenge of this project lies in generating an optimal and simple algorithm capable of understanding any type of grammar to construct its ADP. For this purpose, an algorithm with various functionalities was developed and implemented. It interprets simple grammars that can be solved with finite automata, corresponding to Regular Languages, as well as those inherently of the context-free type, both levels represented in Chomsky's Hierarchy.

Keywords:

Algorithm, Context-Free Grammars, Pushdown Automata, Context-Free Language.

Resumen:

Las Gramáticas Libres de Contexto (GLC) y los Autómatas de Pila (ADP) son fundamentales en la teoría de la computación tienen diversas aplicaciones en esta área, tales como: Análisis de lenguajes formales, desarrollo de Compiladores e Intérpretes, Procesamiento de Lenguaje Natural, Modelado de Estructuras, etc. Por lo tanto, el Ingeniero en Sistemas Computacionales deberá tener una base sólida para abordar la variedad de desafíos a los cuales se enfrentará. Sin embargo, las GLC presentan ciertas problemáticas, siendo la principal la dificultad para analizarlas, así como la ambigüedad que presentan. Es aquí donde entra el ADP, que al ser una estructura equivalente a la GLC, es igual de poderoso que ésta, con la diferencia de que son más fáciles de representar así como de analizar. Tomando como base esta idea, se propone desarrollar un algoritmo de conversión de GLC a ADP, donde el usuario proporcionará una GLC y el algoritmo regresará su ADP equivalente, con la finalidad de que el usuario pueda comprender mejor la relación entre ambas estructuras, así como saber que existe una alternativa a las GLC que es igual de útil pero con más flexibilidad. La dificultad de este proyecto está presente en generar un algoritmo óptimo y simple que pueda entender cualquier tipo de gramática para construir su ADP. Para esto se desarrolló e implementó un algoritmo con diferentes funcionalidades: por un parte interpreta gramáticas simples que se pueden resolver con autómatas finitos, correspondiente a los Lenguajes Regulares, pero también aquéllas que son por excelencia del tipo de libres de contexto, ambos niveles representados en la Jerarquía de Chomsky.

[a] Ofelia Gutiérrez Giraldi, Tecnológico Nacional de México/Instituto Tecnológico de Veracruz | Veracruz-Veracruz | México, https://orcid.org/0000-0002-9544-4784, Email: ofelia.gg@veracruz.tecnm.mx

[b] Martha Martínez Moreno, , Tecnológico Nacional de México/Instituto Tecnológico de Toluca| Toluca-México | México, https://orcid.org/0000-0003-3793-6315, Email:Martha.mm@toluca.tecnm.mx.

[c] Gabriela Clavel Martínez, Tecnológico Nacional de México/Instituto Tecnológico de Boca del Río | Veracruz-Boca del Río | México, https://orcid.org/0000-0002-2733-4145, Email: gabriela.cm@boca.tecnm.mx

Pedro Jesús Pérez Hernández, Tecnológico Nacional de México/Instituto Tecnológico de Veracruz|| Veracruz-Veracruz | México, https://orcid.org/0009-0003-3539-9488 , Email: L19021131@veracruz.tecnm.mx

Fecha de recepción: 28/09/2024, Fecha de aceptación: 24/10/2024, Fecha de publicación: 05/01/2025

DOI: https://doi.org/10.29057/xikua.v13i25.13821

***Palabras Clave**:*

Algoritmo, Gramáticas de Libres de Contexto, Autómatas de Pila, Lenguajes Libres de Contexto.

Introducción

Las Gramáticas Libres de Contexto (GLC) representan una piedra angular en la teoría de la computación, con aplicaciones multifacéticas que abarcan desde el análisis de lenguajes formales hasta el procesamiento de lenguaje natural y la creación de compiladores. Aunque poderosas, las GLC plantean desafíos significativos debido a su complejidad y ambigüedad, lo que ha dado paso al surgimiento de los Autómatas de Pila (ADP) como una alternativa más flexible y de igual poder.

En esta propuesta, se busca desarrollar un algoritmo de conversión de GLC a ADP, permitiendo a los ingenieros en sistemas computacionales comprender mejor la relación entre ambas herramientas. La idea subyacente es que, si bien las GLC son fundamentales, los ADP ofrecen una solución más accesible y comprensible para representar y analizar gramáticas.

El reto radica en crear un algoritmo óptimo y simple que pueda manejar cualquier tipo de gramática.

Para la implementación del software, primero se recopiló la información correspondiente, haciendo un comparativo de aquellos algoritmos encontrados para tal fin; al no asegurarse la funcionalidad que se buscaba, se optó por el diseño paso a paso de este. Posteriormente, se procedió al diseño de un entorno gráfico que permitiera la comprensión y facilitara el manejo de las estructuras, buscando que sea amigable e intuitivo para los estudiantes. Una vez logrado el producto deseado, se realizaron pruebas que permitieran encontrar errores y corregir los mismos hasta obtener una herramienta que aporte un apoyo en la resolución de problemas.

Con este trabajo se pretende impactar a los estudiantes del área de ingeniería en sistemas computacionales, dándoles las facilidades para que desarrollen su capacidad de análisis, así como la aplicación de la lógica en la resolución de problemas en el ámbito de los lenguajes formales, ayudándolos a obtener el nivel de madurez y alcanzando la competencia establecida en el marco del perfil del egresado, que dice: "Definir, diseñar, construir y programar compiladores." que se refleja también en las competencias a desarrollar en las materias de Lenguajes y Autómatas I y II.

Metodología

El desarrollo del proyecto se dividió en tres etapas:

- Investigación teórica
- Desarrollo–Implementación del software
- Resultados

Durante la primera etapa se investigaron fundamentos teóricos, principalmente definiciones formales matemáticas de cada una de las estructuras a implementar en el software, así como algoritmos para la conversión de GLC a ADP.

Investigación Teórica

Durante el proceso de análisis, se consideraron algoritmos de conversión ya existentes, en el comparativo realizado se encontraron los siguientes inconvenientes:

Complejidad y Eficiencia: Los algoritmos clásicos como LR(1), LALR(1), y LL pueden ser complejos y pesados en términos de recursos computacionales cuando se aplican a gramáticas extensas o complicadas. Esto puede resultar en una ineficiencia significativa en términos de tiempo de ejecución y uso de memoria, lo cual es particularmente crítico en aplicaciones que requieren alta velocidad y eficiencia, como en entornos de tiempo real.

Dificultad de Implementación: Implementar estos algoritmos de forma eficiente puede ser un desafío, especialmente para desarrolladores que no están profundamente inmersos en la teoría de la computación. Esto puede conducir a errores en la implementación o a un rendimiento óptimo.

Restricciones de la Gramática: Algoritmos como LR y LL tienen restricciones en cuanto a los tipos de gramáticas que pueden analizar sin conflictos. Por ejemplo, las gramáticas LR(0) y LL(1) solo pueden manejar un subconjunto de

todas las gramáticas libres de contexto. Esto puede ser una limitación importante si la gramática de la aplicación no se ajusta a estas restricciones.
Especificidad de la Aplicación: En algunos casos, la gramática que se necesita analizar puede tener características únicas que los algoritmos generales no manejan bien. Esto puede incluir construcciones sintácticas específicas, necesidades de optimización particulares, o integración con otras partes del sistema de software que los algoritmos estándar no pueden abordar eficientemente.

Actualización y Mantenimiento: Los algoritmos tradicionales, siendo establecidos y ampliamente utilizados, pueden no adaptarse bien a los cambios rápidos o a las necesidades específicas de actualización que un proyecto de software particular puede requerir.
Dado estos desafíos, el desarrollo de un algoritmo de conversión propio se justifica en la medida en que un algoritmo personalizado puede ser diseñado para manejar eficientemente las particularidades de la gramática y los requisitos de rendimiento del proyecto específico, además puede ser más eficiente en términos de uso de memoria y tiempo de ejecución, especialmente en hardware o entornos específicos donde se implementará el software, por último tenemos que un algoritmo propio permite una mayor flexibilidad en términos de ajustes, actualizaciones y mantenimiento, adaptándose mejor a los cambios en los requisitos del proyecto a lo largo del tiempo.
Se concluyó que, aunque los algoritmos tradicionales han sido fundamentales en el avance de la teoría de la computación y siguen siendo útiles en muchos contextos, el desarrollo de un algoritmo de conversión personalizado puede ofrecer ventajas significativas en términos de rendimiento, adaptabilidad y eficacia para proyectos de software específicos.

Definición formal GLC

De manera formal la definición matemática de una Gramática Libre de Contexto (GLC) es: (MARTIN, 1996)
Una gramática G se define como una cuádrupla:
G = (V, ∑, S, P)

Donde:

- *V es un conjunto finito de objetos llamados variables, algunos autores las identifican también como NoTerminales*
- *∑ es un conjunto finito de objetos llamados símbolos terminales*
- *S es un símbolo especial llamado la variable de inicio*
- *P es un conjunto finito de producciones*

Las reglas de producción son el corazón de una gramática; especifican cómo la gramática transforma una cadena en otra y, a través de ella, definen un lenguaje asociado con la gramática. En nuestra discusión asumimos que todas las reglas de producción son de la forma.
x à y

*Donde x es un elemento de (V ? T) + y y está en (V ? T) *. Las producciones se aplican de la siguiente manera:*
Dada una cadena w de la forma.
W = uxv

Definición Formal de ADP

Un autómata de pila se define como una séptupla: (MARTIN, 1996)
AP = {Q, ∑, ґ , q0, Z0,A, δ } donde:
Q = {Conjunto de estados finitos del autómata}
∑ = {Conjunto de símbolos o alfabeto de entrada del autómata}
ґ = {Conjunto de símbolos o alfabeto de la pila}
q0 = Estado inicial, q0 ϵ Q
Z0 = Símbolo inicial de la pila, marca el fondo de la pila Z ϵ ґ
A⊆ Q = {Conjunto de estados finales} donde
δ = Función de transición que sigue la estructura (autómata) y que representa una regla
δ : Q x (Q ∪{
*Λ}) x ґ -> (Q x ґ *)*

Equivalencia entre GLC y ADP

Las GLC y los ADP son estructuras formales equivalentes, lo cual queda demostrado, en los siguientes términos: (John, E. Hopcroft, Motwani, & D. Ullman, 2002)

Dada una GLC, se construye un ADP que simule las derivaciones más a la izquierda de la gramática. Cualquier forma sentencial izquierda que no sea una cadena terminal puede

escribirse como xA∞, donde es la variable más a la izquierda, x son los símbolos terminales que aparecen a su izquierda, y ∞ es la cadena de símbolos terminales y variables que aparecen a la derecha de A. Se llama A∞ a la cola de esta forma sentencial izquierda. Si una forma sentencial izquierda consta únicamente de símbolos terminales, entonces su cola es ε.

La idea para la construcción de un ADP a partir de una GLC, es que el primero simule la secuencia de formas senténciales izquierdas que usa la CFG para generar una cadena determinada de símbolos terminales.

Algoritmo de conversión

De la siguiente forma, se demuestra que, dada una GLC, se construye un ADP: (Brookshear, 1993)

1. Designe el alfabeto del ADP como los símbolos terminales de la GLC, y los símbolos de la pila como los símbolos terminales y no terminales de la gramática, junto con el símbolo especial (que no es un símbolo terminal o no terminal)

2. Designe los estados del ADP como i,p,q y f, identificando el estado inicial y el estado final o de aceptación

3. Introduzca la transición (i,λ,λ;p,#)

4. Introduzca una transición donde (p, λ, λ;q,S)donde S es el símbolo inicial de GLC

5. Introduzca una transición de forma(q, λ,N;q,w) para cada regla de reescritura N-> w en GLC (aquí empleamos nuestra nueva conversión que permite que una sola transición inserte más de un símbolo de pila. Específicamente ,w puede ser una cadena de cero o más símbolos , incluyendo terminales y no terminales)

6. Introduzca una transición de forma para cada terminal de una gramática (q,x,x;q, λ) para cada terminal x de GLC (es decir para cada símbolo del alfabeto ADP)

7. Introduzca una transición (q, λ,#;f, λ)

Desarrollo-Implementación

Para la elaboración de esta estructura se consideró lo siguiente:

- lógica del Autómata de pila:
- Se toma como estado inicial 0
- El alfabeto puede ser cualquier símbolo (exceptuando ? dicho símbolo se le da a épsilon)
- La pila inicia con un valor de fondo Z

En cuanto al desarrollo e implementación del software, se aplicó la siguiente metodología, considerando primeramente el diseño de ambas estructuras (GLC y ADP) y posteriormente el análisis, diseño e implementación del algoritmo, que es finalmente a dónde se quiere llegar.

Creación de una conversión

Para poder crear un algoritmo de conversión, primero se debe crear por separado esta estructura gramatical formal de la GLC, es decir, programar su funcionamiento lógico. Para realizar la programación de esta estructura se consideró lo siguiente:

- Se toma como no terminal inicial al carácter S.
- Los no-terminales pueden ser cualquier letra del alfabeto escrita en mayúscula (exceptuando la letra ñ)
- Los terminales puede ser cualquier letra minúscula (exceptuando la letra ñ)
- La cabeza de producción puede tener relacionadas múltiples producciones.

Respecto a la programación se siguió el paradigma de programación orientada a objetos, con el fin de obtener una manipulación más fácil en las funciones que existen. Para mantener el control de flujo (la construcción de hojas) el algoritmo se diseñó de manera recursiva. Los parámetros para hacer funcionar la simulación constan de dos partes: el ingreso de las producciones que forman parte de la gramática y las cadenas a evaluar, dando como resultado un mensaje que indique si la cadena pertenece o no a la gramática.

Como inicio y para analizar la magnitud y capacidad óptima, no se plantean límites con la longitud y cantidad de producciones, el primer paso es analizar el amplio espectro de GLC así que a partir de eso se desarrolla la lógica del cómo debería funcionar.

En el caso de los ADP, la programación consideró lo siguiente:

a) Estado: representa los estados del autómata de pila. Esta clase cuenta con los siguientes atributos:

- Nombre: Identifica el nombre del estado
- Aceptación: Indica si el estado es un estado final o de aceptación

b) Transición: define las transiciones entre los estados del autómata. Esta clase posee los siguientes atributos:

- Nombre del Estado: Representa el estado desde el cual se realiza la transición
- Caracter a Leer: Es el símbolo que se lee desde la entrada
- Pop: Indica el elemento que se saca de la pila durante la transición
- Push: Representa el elemento que se introduce en la pila durante la transición

c) Autómata: La clase Autómata es responsable de llevar a cabo la validación de las cadenas en el autómata de pila. Esta clase incluye los siguientes aspectos principales:

- Método de Validación: Comprueba si existe una cadena
- Resultado y Contenido de la Pila: Imprime el resultado de la validación de la cadena y muestra el contenido final de la pila transición para cada símbolo de la cadena y realiza los cambios correspondientes en la pila
- Operaciones en la Pila: Maneja las operaciones de sacar elementos de la pila (pop) y agregar nuevos elementos (push) según las transiciones definida
- Cambio de Estado: Cambia de estado de acuerdo con las transiciones definidas hasta que se completa el recorrido de la cadena.

El algoritmo de conversión que se llevó a cabo es el siguiente:

1. Inicialización:
   - Selecciona un conjunto de reglas gramaticales que describan la GLC
   - Inicia un nuevo autómata de pila (ADP) vacío
   - Inicializa un contador para asignar identificadores a los estados
2. Conversión de GLC a ADP:

Para cada regla gramatical en la GLC:

- Analiza la regla para entender cómo se relacionan los símbolos no terminales y terminales
- Empieza a construir el ADP siguiendo las reglas gramaticales

2. Proceso de Construcción del ADP:
   - Por cada regla gramatical en la GLC:
   - Identifica los símbolos no terminales y terminales en la regla
   - Crea transiciones en el ADP basadas en estos símbolos:
   - Si el símbolo es un no terminal:
   - Crea un estado nuevo para representar este símbolo
   - Explora todas las reglas que involucren este símbolo no terminal
   - Asocia transiciones desde este estado hacia los estados correspondientes a los símbolos en la regla
   - Si el símbolo es un terminal:
   - Crea transiciones desde el estado actual hacia los estados que representan este terminal
3. Marcado de Estados Finales:
   - Después de seguir todas las reglas gramaticales, identifica los estados finales del ADP
   - Marca los estados que corresponden al final de una regla gramatical como estados finales

En la figura 1 se muestra la conversión de una GLC a ADP.

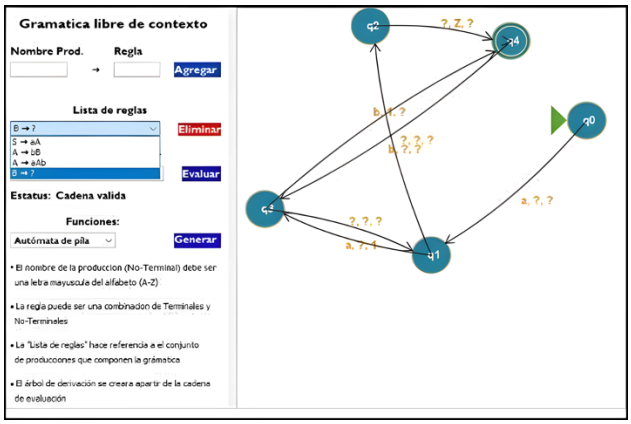

Figura 1. Conversión de GLC a ADP

Análisis de resultados

Para recabar información con relación al impacto de la conversión de GLC a ADP en las materias de lenguajes y autómatas del periodo AGO2023-FEB2024 se realizó lo siguiente:

- Se diseñó una herramienta (cuestionario) para que fuera contestado por los estudiantes.
- Anterior a esto, se les dio un curso a 23 estudiantes de dichas materias, donde se les explicó la definición formal matemáticas de ambas estructuras (GLC y ADP)
- Se les solicitó probar una herramienta similar, en la sección correspondiente a la conversión motivo de este artículo (se sugirió JFlap) y posteriormente usar el algoritmo de conversión.
- Una vez hecho lo anterior, se les aplicó el cuestionario.

Estadística

El cuestionario se diseñó utilizando la herramienta Google Form, que da la facilidad de generar las gráficas que se muestran a continuación. Para cada una de ellas, se generó una interpretación en relación con lo respondido por los estudiantes encuestados.

1. ¿Qué característica de la gramática le gustaría comprender mejor?

Fuente directa: Un significativo 41.2% muestra interés en reafirmar su conocimiento de todo el manejo de gramáticas libres de contexto.

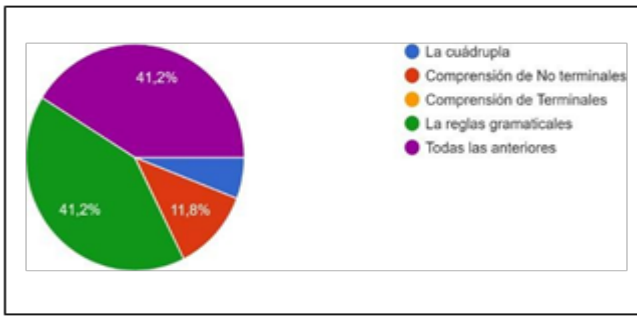

Figura 2. Estadística pregunta 1

2. En qué grado calificarías las facilidades que brinda la conversión de GLC a ADP Se consideran tres aspectos:Incomprensible/Comprensible,Lento/Rápido ,Ineficiente/Eficiente. De cada uno, se muestra una gráfica a continuación

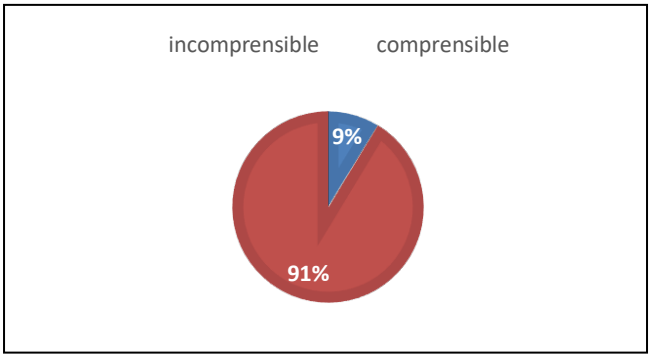

Figura 3. Estadistica pregunta 2a

Fuente directa: En relación a la comprensión de la conversión, un significativo 91% la considera en un rango de 5 a 10, de medio a alto.

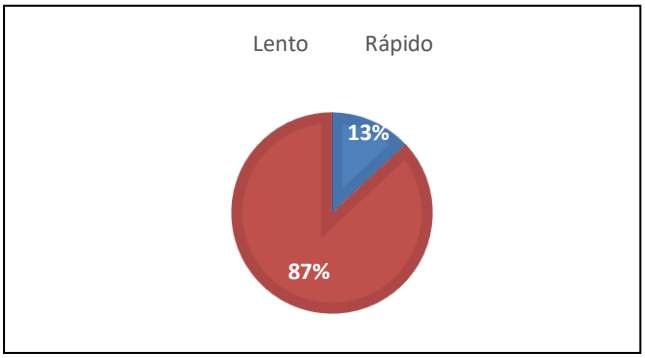

Figura 4. Estadística pregunta 2b

Fuente directa: Un 87% indica que la conversión rápida en un rango de medio a alto.

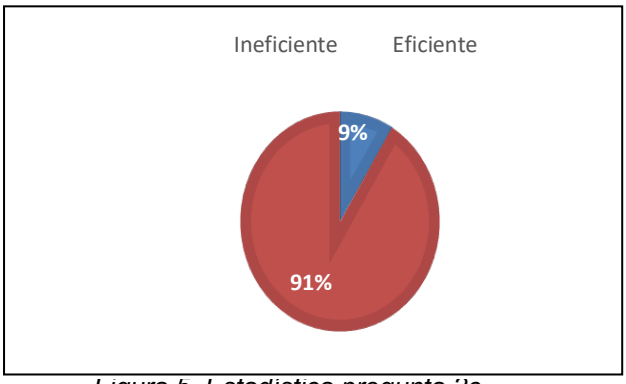

Figura 5. Estadistica pregunta 2c

Fuente directa: En cuanto a la eficiencia, el 91% piensa que es la eficiente.

3. La representación gráfica de la conversión de GLC a ADP es, Se consideran tres aspectos: Ilegible/Legible,Incomprensible/Comprensible,

Lento/Rápido. De cada uno, se muestra una gráfica a continuación.

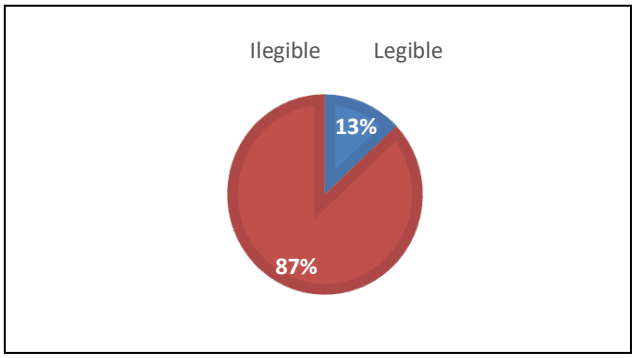

Figura 6. Estadística pregunta 3a

Fuente directa: En cuanto a la representación gráfica dela conversión, un significativo 87% lo considera legible

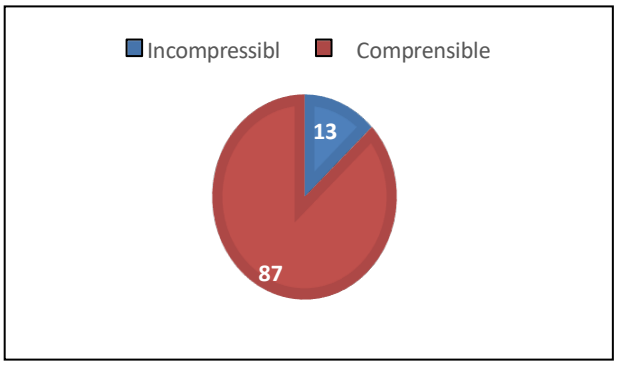

Figura 7 Estadística pregunta 3b

Fuente directa: La comprensión de la representación gráfica es considerada entre medio y alto en un 87%.

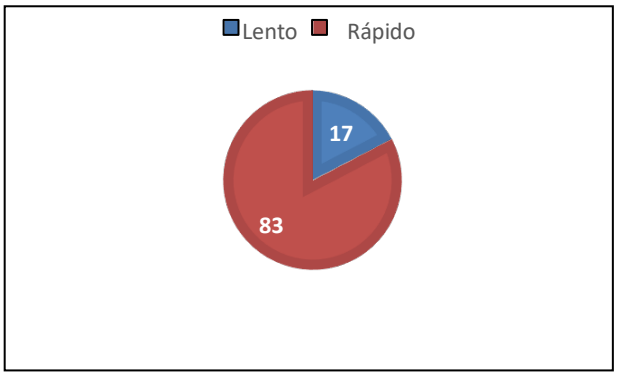

Figura 8. Estadística pregunta 3c

Fuente directa: El 83% de los encuestado indican que la rapidez de la conversión es buena.

Concluisión

Tomando como base que el perfil del Ingeniero en Sistemas Computacionales debe contener una base sólida en la resolución de problemas, aunado también a su enfoque en el análisis, diseño e implementación de compiladores, qué es precisamente lo que lo hace diferente del resto de los profesionales del área de la computación, se da un énfasis especial dentro de la carrera a las materias de Lenguajes y Autómatas; es en éstas donde se aplican conocimientos de los lenguajes formales que utilizan el manejo de las matemáticas discretas, las estructuras de datos y los lenguajes de programación, todos ellos involucrados en el desarrollo de lo que se llama software de base.

Estas estructuras ya aplicadas son el bagaje de conocimiento que requiere un profesional del área de ingeniería; sin embargo, y aunque la implementación de ellas permite que el estudiante cambie su forma de pensar y resolver los problemas, también se les complica en función de la forma que usualmente tienen de hacerlo.

Las Gramáticas Libres de Contexto y los Autómatas de Pila forman parte fundamental de estas estructuras aplicadas, llevando su enfoque directamente a la parte de la fase sintáctica de un compilador. Entonces, los estudiantes deben tener un manejo adecuado de las mismas para lograr los objetivos que se marcan en las materias arriba mencionadas.

Con base a todo lo anterior, se ha desarrollado una herramienta que apoye al estudiante en todo el proceso de comprensión para la realización del análisis, siendo el Algoritmo de Conversión de GLC a ADP una parte fundamental de esta.

Esta parte que se describe en el artículo se ha concluido con éxito considerando los resultados de la encuesta aplicada a los estudiantes objetivo; sin embargo, cabe mencionar que existe un porcentaje mínimo (pero existente) que tiene opiniones contrarias y que son dignas de analizar.

Es muy importante también dejar establecido, que hay estudiante que solicitan mejoras; algunas de ellas son: de lenguaje en español, trabaje máquinas de Turing, manuales de referencia (tanto técnico como del usuario) e interfaz amigable.

Referencias

Brookshear, J. G. (1993). Teoría de la Computación Lenguajes formales, autómatas y complejidad (Vol. Primera edición). Estados Unidos : Pearson Addison Wesley Longman

John, E. Hopcroft, J., Motwani, R., & D. Ullman, J. (2002). Introducción a la Teoría de Autómatas,Lenguajes y Computación (Vol. 2a Edición). Madrir, España: PEARSON Addison Wesly.

MARTIN, J. C. (1996). INTRODUCTION TO LANGUAGES AND THE THEORY OF COMPUTATION (Vol. SECOND EDITION). United States: McGraw-Hill