UNIVERSIDAD AUTÓNOMA DE CHIAPAS

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN
CAMPUS 1

ING. EN DESARROLLO Y TECNOLOGÍAS DE SOFTWARE

6 “M”

COMPILADORES

SUBCOMPETENCIA II.- ANÁLISIS SINTÁCTICO

ACTIV. 2.2 INVESTIGAR LA SIGUIENTE INVESTIGACIÓN EN FORMATO APA

ALUMNO: MARCO ANTONIO ZÚÑIGA MORALES – A211121

DOCENTE: DR. LUIS GUTIÉRREZ ALFARO

TUXTLA GUTIÉRREZ, CHIAPAS
LUNES, 11 DE SEPTIEMBRE DE 2023

ÍNDICE

Introducción 3
Desarrollo 4
Métodos sintácticos descendentes - predictivo recursivo o directo de un ejemplo. 4
Métodos sintácticos descendentes - predictivo no recursivo de un ejemplo 5
Definir el concepto de Autómata Push-Down (Push Down Automata PDA) 5
Explicar la notación gráfica de PDA. 7
Explicar las características de los lenguajes de PDA. 7
Explicar el proceso de determinación de equivalencia de PDA y CFG. 8
Explicar el proceso de construcción de PDA de acuerdo a gramáticas. 10
Conclusión 11
Referencia bibliográfica 12

Introducción

En el mundo de la programación y la teoría de la computación, se encuentran conceptos y técnicas cruciales relacionados con el análisis sintáctico, la representación de lenguajes formales y la equivalencia entre estructuras computacionales.

En este trabajo, abordaremos los métodos sintácticos descendentes, ya sean predictivos recursivos o no recursivos, que son fundamentales para la validación de la estructura de programas. También exploraremos el concepto de Autómata Push-Down (PDA), una máquina abstracta que amplía la capacidad de reconocimiento de lenguajes y se representa mediante notación gráfica para una comprensión más visual. Además, se abordarán las características distintivas de los lenguajes reconocidos por PDAs y cómo determinar su equivalencia con Gramáticas Libres de Contexto (CFG).

Los PDA son una herramienta poderosa en la teoría de la computación para reconocer y generar lenguajes contextuales. Su notación gráfica es una representación visual de su funcionamiento, lo que facilita la comprensión de su comportamiento.

Por último, se analizará el proceso de construcción de PDAs a partir de gramáticas, destacando su importancia en el diseño de analizadores sintácticos. Estos temas se entrelazan para proporcionar una base sólida en la teoría de la computación y la programación.

A continuación, se presentan de una forma mas amplia los siguientes temas antes mencionados:

Desarrollo

Métodos sintácticos descendentes - predictivo recursivo o directo de un ejemplo.

Los métodos sintácticos descendentes son un enfoque común en el análisis sintáctico de lenguajes de programación. Estos métodos comienzan desde el símbolo inicial de la gramática y avanzan hacia abajo en la jerarquía de la gramática para reconocer la estructura sintáctica de una cadena de entrada

Los métodos sintácticos descendentes son un enfoque para el análisis sintáctico que se basa en la construcción de un árbol de análisis sintáctico. Este árbol representa la estructura sintáctica de la entrada, y se construye a partir de la cima hacia abajo, siguiendo las reglas de la gramática del lenguaje.

Los métodos recursivos utilizan recursiones para construir el árbol de análisis sintáctico.

Este método utiliza funciones o procedimientos recursivos para analizar la entrada. Cada no terminal de la gramática se mapea a una función o procedimiento específico que se encarga de reconocer esa parte de la estructura. El analizador sintáctico comienza llamando a la función correspondiente al símbolo inicial y, a medida que avanza, invoca otras funciones para los no terminales encontrados en la cadena de entrada.

Ejemplo:

Supongamos que tenemos una gramática para expresiones aritméticas simples:

Expresión -> Término + Expresión | Término
Término -> Factor * Término | Factor
Factor -> (Expresión) | número

Un analizador sintáctico predictivo recursivo podría tener funciones como Expresión (), Termino (), y Factor (), cada una de las cuales corresponde a un no terminal de la gramática. El analizador comenzaría llamando a Expresión () y seguiría invocando las funciones apropiadas para reconocer la estructura de la cadena de entrada.

Métodos sintácticos descendentes - predictivo no recursivo de un ejemplo

El método predictivo no recursivo utiliza una pila (stack) para rastrear la estructura sintáctica. En lugar de funciones recursivas, utiliza una tabla de análisis sintáctico que determina las acciones a tomar en función del símbolo actual de entrada y la parte superior de la pila. El analizador sintáctico sigue las reglas de la tabla para realizar desplazamientos y reducciones hasta reconocer la estructura sintáctica completa.

Ejemplo:

Expresión -> Término + Expresión | Término
Término -> Factor * Término | Factor
Factor -> (Expresión) | número

Para el mismo ejemplo de expresiones aritméticas, el analizador predictivo no recursivo usaría una tabla que indica qué hacer en función del símbolo actual y la parte superior de la pila. Por ejemplo, si la parte superior de la pila es un estado que representa Término y el símbolo de entrada es +, la tabla podría indicar que se debe realizar una reducción usando la regla Término -> Factor * Término para reconocer el término completo.

Definir el concepto de Autómata Push-Down (Push Down Automata PDA)

Un autómata push-down (PDA) es un modelo matemático de máquina abstracta que se utiliza para reconocer lenguajes formales. Un PDA tiene un número finito de estados, una cinta de entrada y una pila. La pila se utiliza para almacenar información temporal, lo que permite al PDA reconocer lenguajes que no pueden ser reconocidos por autómatas finitos.

Fue propuesto por primera vez por el matemático Stephen Cole Kleene en la década de 1950.

Un Autómata Push-Down (PDA) es un modelo teórico de computación que extiende la capacidad de una Máquina de Estados Finitos (MEF) al incluir una pila (stack) como un componente adicional. La pila permite al PDA realizar operaciones de almacenamiento y recuperación de datos de manera más sofisticada que una MEF, lo que lo convierte en una máquina capaz de reconocer lenguajes más poderosos, conocidos como lenguajes contextuales.

Elementos

El PDA consta de los siguientes elementos:

1. **Estado Inicial:**
   Un estado desde el cual comienza la computación.

2. **Conjunto de Estados:**
   Un conjunto finito de estados, que incluye el estado inicial y posiblemente estados finales (de aceptación).

3. **Alfabeto de Entrada:**
   El conjunto de símbolos de entrada que puede procesar el PDA.

4. **Alfabeto de Pila:**
   El conjunto de símbolos que se pueden almacenar en la pila.

5. **Transiciones:**
   Funciones de transición que indican cómo se mueve el PDA entre estados en función del símbolo actual de entrada y la parte superior de la pila.

6. **Pila:**
   Una estructura de datos de tipo pila que permite almacenar y recuperar símbolos. La pila se utiliza para recordar información sobre la historia de la entrada procesada.

7. **Estado Final:**
   Un conjunto de estados que indica si el PDA ha alcanzado un estado de aceptación.

El funcionamiento principal de un PDA implica leer un símbolo de entrada, consultar la parte superior de la pila y realizar una transición basada en estos elementos. La pila puede ser modificada al realizar operaciones de "empujar" (push) o "pop" en la parte superior. Esto permite al PDA recordar información y manejar estructuras anidadas en la entrada, como paréntesis equilibrados en expresiones matemáticas o etiquetas anidadas en lenguajes de marcado como HTML o XML.

Los PDA son capaces de reconocer y generar lenguajes contextuales, lo que los convierte en un modelo poderoso para describir y analizar la estructura de programas y lenguajes de programación más complejos. Son especialmente relevantes en la teoría de la computación, en la construcción de analizadores sintácticos en compiladores y en el procesamiento de lenguajes formales en general.

Explicar la notación gráfica de PDA.

La notación gráfica de un PDA se utiliza para representar visualmente su estructura y las transiciones entre estados.

1. **Estados:**
   Se representan como nodos en el grafo. El estado inicial suele denotarse con una flecha que lo señala claramente.

2. **Transiciones:**
   Se muestran como flechas etiquetadas que conectan dos estados. Cada flecha tiene una etiqueta que indica la entrada que activa la transición y los símbolos de pila involucrados en la operación. Por ejemplo, "a, X/Y" podría indicar una transición que lee 'a', desplaza 'X' de la pila y coloca 'Y' en la pila.

3. **Pila:**
   A menudo se representa verticalmente cerca de los estados para mostrar el contenido de la pila. Esto ayuda a visualizar cómo cambia la pila en cada transición.

4. **Estado de Aceptación:**
   En algunos casos, se utiliza un estado de aceptación especial para indicar que el PDA ha aceptado la cadena de entrada con éxito. Este estado puede estar marcado de manera distintiva.

5. **Estado de Rechazo:**
   Similar al estado de aceptación, un PDA también puede tener un estado de rechazo especial para indicar que la cadena de entrada no es válida.

6. **Transiciones Épsilon:**
   A veces, se utilizan transiciones épsilon (ε) para indicar que no se consume ningún símbolo de entrada o pila durante la transición. Esto se denota mediante una flecha sin etiquetas o con la etiqueta ε.

La notación gráfica facilita la comprensión y la representación visual de cómo un PDA procesa una cadena de entrada y cómo se comporta en términos de su estructura y transiciones.

Explicar las características de los lenguajes de PDA.

Los lenguajes reconocidos por un Autómata Push-Down (PDA) tienen características específicas que los distinguen de otros tipos de lenguajes formales, como los lenguajes regulares.

Características:

- **Contextualidad:**
  Los lenguajes de PDA son lenguajes contextuales. Esto significa que pueden representar estructuras más complejas que los lenguajes regulares, ya que tienen la capacidad de recordar información sobre la historia de la entrada a través de una pila.

- **Anidamiento:**
  Los lenguajes de PDA pueden representar adecuadamente estructuras anidadas, como paréntesis equilibrados en expresiones matemáticas. Esto se debe a que la pila puede utilizarse para llevar un registro de los símbolos abiertos y cerrados.

- **Capacidad de Memoria Limitada:**
  Aunque los PDA tienen una pila para almacenar información, esta pila es de capacidad limitada. Esto significa que pueden recordar información sobre la historia de la entrada solo hasta cierto punto, lo que les permite representar contextos locales, pero no contextos infinitamente grandes.

- **No Determinismo:**
  Los PDA pueden ser no deterministas o deterministas. Un PDA no determinista puede tener múltiples transiciones posibles desde un estado dado para un símbolo de entrada y un símbolo de pila determinados. Esto permite una mayor expresividad en la representación de lenguajes contextuales.

- **Gramáticas Contextuales:**
  Existe una relación directa entre los lenguajes de PDA y las Gramáticas Libres de Contexto (CFG). Cada PDA puede asociarse con una CFG y viceversa, lo que significa que los lenguajes de PDA pueden describirse mediante reglas de producción gramaticales y viceversa.

Explicar el proceso de determinación de equivalencia de PDA y CFG.

Determinar si un PDA y una Gramática Libre de Contexto (CFG) son equivalentes implica mostrar que ambos pueden reconocer y generar el mismo lenguaje.

1. **De CFG a PDA:**
   Dado una CFG, puedes construir un PDA que reconozca el mismo lenguaje. El PDA utilizará la pila para seguir el proceso de derivación de la CFG.

Para cada regla de producción en la CFG, se deben definir transiciones en el PDA que simulan el proceso de aplicación de la regla de producción. Esto implica desplazar símbolos en la pila según las reglas de la gramática.

El primer paso implica la construcción de un PDA a partir de una Gramática Libre de Contexto (CFG). Esto se hace asignando componentes del PDA a elementos de la CFG:

- **Estados del PDA:**
  Cada no terminal de la CFG se convierte en un estado en el PDA.

- **Transiciones del PDA:**
  Las reglas de producción de la CFG se traducen en transiciones del PDA. Por ejemplo, si tienes una producción A -> α, donde "A" es un no terminal y α es una cadena de símbolos terminales y no terminales, puedes crear una transición en el PDA que, cuando se encuentra en el estado correspondiente a "A" y lee un símbolo de entrada que coincide con α, realiza una operación de desplazamiento en la pila para reemplazar "A" por α.

- **Inicio y Aceptación del PDA:**
  El estado inicial del PDA se establece en el estado correspondiente al símbolo inicial de la CFG, y los estados de aceptación se definen para reflejar los no terminales que se han derivado completamente en la CFG.

2. **De PDA a CFG:**
   Dado un PDA, puedes construir una CFG que reconozca el mismo lenguaje.

   Utiliza la técnica de "pila a pila" para generar las reglas de producción de la CFG. Por cada transición en el PDA, crea reglas de producción que describan cómo la pila cambia de un estado al siguiente en términos de símbolos de pila.

   El segundo paso implica la construcción de una Gramática Libre de Contexto (CFG) a partir de un PDA. Este proceso es más complejo y requiere la creación de reglas de producción basadas en las transiciones y la estructura del PDA:

   - **Producciones de CFG:**
     Cada estado del PDA se mapea a un no terminal en la CFG. Luego, las transiciones del PDA se utilizan para generar las reglas de producción de la CFG. Las transiciones del PDA determinan cómo se reemplazan los símbolos en la pila, lo que se traduce en reglas de producción en la CFG.

3. **Prueba de Equivalencia:**
   Una vez que hayas construido la CFG y el PDA a partir de uno u otro, debes demostrar que ambos reconocen el mismo lenguaje.

   Esto se logra mostrando que cualquier cadena generada por la CFG puede ser aceptada por el PDA, y viceversa, cualquier cadena aceptada por el PDA puede ser generada por la CFG.

Explicar el proceso de construcción de PDA de acuerdo a gramáticas.

El proceso de construcción de un PDA a partir de una Gramática Libre de Contexto (CFG) implica:

1. **Crear Estados:**
   Para cada no terminal en la CFG, crea un estado en el PDA. Estos estados representarán los no terminales en la derivación.

2. **Transiciones:**
   Define las transiciones del PDA de manera que sigan el proceso de derivación de la gramática. Cada transición debe considerar el símbolo actual de la entrada y el símbolo superior de la pila.

3. **Manejar la Pila:**
   Utiliza la pila para llevar un registro de los no terminales y símbolos en la derivación actual. Las transiciones deben empujar y sacar símbolos de la pila según las reglas de la gramática.

4. **Estado Inicial y Aceptación:**
   Define un estado inicial y un conjunto de estados de aceptación que reflejen las reglas de la gramática.

Conclusión

En conclusión, con respecto a la información recabada de libros y sitios web vemos que, al explorar los temas de análisis sintáctico, lenguajes formales y autómatas, se destaca la importancia de comprender las diferentes técnicas y conceptos en programación y teoría de la computación.

Los métodos sintácticos descendentes, ya sean predictivos recursivos o no recursivos, son fundamentales para validar la estructura de un programa. Los autómatas Push-Down (PDA) amplían la capacidad de reconocimiento de lenguajes a contextuales, permitiendo representar estructuras más complejas. La notación gráfica de PDA facilita la visualización de su funcionamiento.

Los lenguajes de PDA son contextuales y pueden manejar anidamientos. La determinación de equivalencia entre PDA y Gramáticas Libres de Contexto (CFG) es esencial y requiere demostrar que ambos reconocen el mismo lenguaje. La construcción de un PDA a partir de una gramática es un proceso clave en la creación de analizadores sintácticos. Estos conceptos son fundamentales tanto en la práctica de la programación como en la teoría de la computación.

Referencia bibliográfica

Aho, A. V., Sethi, R., & Ullman, J. D. (2006). Fundamentos de la teoría de autómatas, compiladores e intérpretes. Pearson Educación.

Navarro Colorado, B., & Palomar, M. (2008). Análisis sintáctico. Ingeniería del Lengueja Natural.

Pereyra, P., & Rosario, R. (2021). Desarrollo e implementación de un analizador sintáctico utilizando el compilador Javacc para el reconocimiento de errores sintácticos en el lenguaje PHP. Revista Ciencia y Tecnologia, 17(1), 85-96.

Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2007). Introducción a los algoritmos. Pearson Educación.

Gutiérrez Viedma, E. (2020). New Perspectives on Classical Automata Constructions (Doctoral dissertation, ETSI_Informatica).

Knuth, D. E. (2016). El arte de la programación con computadoras. Addison-Wesley.

Schmal, R. F., & Olave, T. Y. (2014). Optimización del proceso de atención al cliente en un restaurante durante periodos de alta demanda. Información tecnológica, 25(4), 27-34.

Sipser, M. (2012). Conceptos de teoría de la computación. Pearson Educación.

Jiménez Heredia, J. P., & Flores Meza, J. A. (2008). Estudio, diseño y desarrollo de un software de monitorización de red en dispositivos inalámbricos (PDA) para la administración y gestión de red (Bachelor's thesis, QUITO/EPN/2008).

Hopcroft, J. E., Motwani, R., & Ullman, J. D. (s.f.). Introduction to Automata Theory, Languages, and Computation. Editorial.

En Introducción a la computación y a la programación Estructurada, de Guillermo Levine, 115-118. Mexico, 1989.

En Sistema Operativos y Compiladores, de Jesus Salas Parilla, 149,150,173,174. México: Mc Graw-Hill, 1992.