UNIVERSIDAD AUTÓNOMA DE CHIAPAS

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN
CAMPUS 1

Licenciatura en Ingeniería en Desarrollo y Tecnologías de Software

Actividad. 2.2 Investigar la siguiente investigación en formato APA

6 “M”

Materia: Compiladores

Docente: Luis Gutiérrez Alfaro
ALUMNO:
A 211387
Steven de Dios Montoya Hernández
GIT:

TUXTLA GUTIÉRREZ, CHIAPAS
Lunes, 11 de Septiembre De 2023, 23:59

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

Método Sintáctico Descendente Predictivo Recursivo:

El análisis predictivo recursivo es un método de análisis sintáctico que utiliza funciones recursivas para analizar una cadena de símbolos y determinar si sigue la gramática especificada. En este enfoque, cada función se asocia con un no terminal y busca producciones de la gramática que coincidan con la entrada actual.

Ejemplo de Gramática:

E -> T + E | T
T -> int * T | int

Método Sintáctico Descendente Predictivo Directo:

El análisis predictivo directo es una variante del análisis predictivo recursivo en la que se utiliza una tabla de análisis para guiar las decisiones de análisis. Cada entrada en la tabla contiene un conjunto de símbolos que se esperan en ese punto del análisis.

Ejemplo de Gramática:

S -> iEtS | iEtSeS | a
E -> b

Tabla de Análisis Predictivo Directo:

	i		a	b	e	$
S	1		3			
E				2		
t					4	
e						acc

Ejemplo de Análisis Predictivo Directo:

Supongamos que tenemos la entrada "i b e a $":

Paso 1: S
- Predicción: iEtS
- Avanzamos: i b e a $

Paso 2: E
- Predicción: b
- Avanzamos: b e a $

Paso 3: t
- Coincide con la entrada: b
- Avanzamos: e a $

Paso 4: S
- Predicción: eS
- Avanzamos: e a $

Paso 5: E
- Predicción: ε (producción vacía)
- Avanzamos: a $

Paso 6: S
- Predicción: ε (producción vacía)
- Avanzamos: $

Paso 7: Fin de entrada ($)
- Analizador concluye: Aceptación

Métodos sintácticos descendentes - predictivo no recursivo de un ejemplo

Los métodos sintácticos descendentes no recursivos son un enfoque menos común pero muy interesante para el análisis sintáctico. A diferencia de los métodos predictivos recursivos que utilizan funciones recursivas para el análisis, los métodos no recursivos emplean una pila para rastrear el proceso de análisis. Aquí tienes un ejemplo de un analizador sintáctico descendente predictivo no recursivo utilizando una pila:

Gramática:

E -> T E'
E' -> + T E' | ε
T -> F T'
T' -> * F T' | ε
F -> (E) | id

Cadena de entrada:
Supongamos que tenemos la siguiente cadena de entrada: id + id * id.

Análisis Sintáctico Descendente Predictivo No Recursivo:

Inicializa una pila con el símbolo inicial E y el símbolo de fin de entrada $.

Inicializa una cadena de entrada con la cadena proporcionada concatenada con el símbolo de fin de entrada $.

Comienza el proceso de análisis:
Mientras la pila no esté vacía:

Toma el símbolo en la cima de la pila y el símbolo actual en la cadena de entrada.
Si son iguales:

Retirar ambos de la pila y la cadena de entrada.

Si el símbolo en la pila es un no terminal (E, T, E', T', F):
Usa la regla correspondiente para ese no terminal y el símbolo actual para decidir qué producción aplicar.

Reemplaza el no terminal en la pila con la secuencia de símbolos de la producción.
Si no se cumple ninguna de las condiciones anteriores, se trata de un error sintáctico.

Una vez que la pila esté vacía y también la cadena de entrada, la cadena es válida según la gramática. Si queda algo en la cadena de entrada o en la pila, la cadena no es válida.

Pasos para el Ejemplo de Cadena id + id * id:

‘E’ (en la pila) y ‘id’ (en la cadena) -> Coinciden y se eliminan.

‘T E' (en la pila) y ‘+ id * id’(en la cadena) -> Aplicamos reglas de producción para ‘T E'.

‘F T' E' (en la pila) y ‘+ id * id’ (en la cadena) -> Aplicamos reglas de producción para ‘F T'.

‘id T' E' (en la pila) y ‘+ id * id’(en la cadena) -> Coinciden y se eliminan.

‘T' E' (en la pila) y ‘+ * id’ (en la cadena) -> No se puede avanzar usando T', se busca otra producción.

‘E' (en la pila) y ‘+ * id’ (en la cadena) -> Coinciden y se eliminan.

‘$’ (en la pila) y ‘$’ (en la cadena) -> Coinciden y se eliminan. La cadena es válida según la gramática.

¿Qué es el Análisis Sintáctico Descendente Predictivo No Recursivo?

El análisis sintáctico descendente predictivo no recursivo es un método de análisis sintáctico que utiliza una pila y una tabla de análisis para determinar si una cadena de entrada sigue la gramática especificada. A diferencia del análisis predictivo recursivo, que utiliza funciones recursivas, el análisis predictivo no recursivo utiliza una pila para rastrear y controlar el proceso de análisis.

Pasos en el Análisis Sintáctico Descendente Predictivo No Recursivo:

Inicialización:

Inicializa una pila con el símbolo inicial de la gramática y el símbolo de fin de entrada.

Inicializa una cadena de entrada con la cadena a analizar concatenada con el símbolo de fin de entrada.

Proceso de Análisis:
Mientras la pila no esté vacía:

Toma el símbolo en la cima de la pila (llamémoslo A) y el símbolo actual en la cadena de entrada (llamémoslo a).

Si A es un símbolo terminal y coincide con a, retira ambos de la pila y la cadena.

Si A es un no terminal:
Consulta la tabla de análisis para encontrar la producción que corresponde a la combinación de A y a.

Reemplaza A en la pila con la secuencia de símbolos de la producción encontrada.
Si no se cumple ninguna de las condiciones anteriores, se trata de un error sintáctico.

Finalización:

Si la pila y la cadena de entrada están vacías, la cadena se considera válida según la gramática.

Si queda algo en la pila o en la cadena de entrada, la cadena no es válida.

Ventajas del Análisis Predictivo No Recursivo:

Eficiencia de memoria: A diferencia de los métodos recursivos, el análisis no recursivo puede ser más eficiente en términos de uso de memoria, ya que no depende de llamadas recursivas.

Evita Recursión Infinita: El análisis no recursivo evita problemas de recursión infinita que a veces pueden ocurrir en los métodos recursivos.

Más General: El análisis no recursivo puede manejar gramáticas más complejas y ambigüedades de manera más eficiente que los métodos recursivos.

Desafíos del Análisis Predictivo No Recursivo:

Tabla de Análisis: El análisis no recursivo requiere una tabla de análisis, que puede ser compleja de construir y mantener.

Conflictos y ambigüedades: Las gramáticas ambiguas o con conflictos pueden llevar a decisiones incorrectas en la tabla de análisis.

Complejidad de Implementación: La implementación del análisis no recursivo puede ser más compleja que los métodos recursivos.

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

Un autómata push down (PDA) es un tipo de autómata que emplea una pila. Los autómatas push down se utilizan en teorías sobre lo que pueden calcular las máquinas. Son más capaces que las máquinas de estado finito pero menos capaces que las máquinas de Turing. El autómata de pila (se abrevia PDA de sus siglas en inglés Push-Down Autómata) tiene una cinta de entrada, un control finito y una pila. Los autómatas Push-Down son los autómatas más importantes entre las máquinas de estados finitos y las de Turing

Un Autómata de Pila, también conocido como Autómata Push-Down (PDA por sus siglas en inglés, Push Down Automata), es un tipo de autómata finito extendido que incluye una pila, una estructura de datos similar a una pila en la que se pueden almacenar y recuperar símbolos. Los PDAs se utilizan para modelar y reconocer lenguajes contextuales, que son más poderosos en términos de capacidad de reconocimiento que los lenguajes regulares reconocidos por autómatas finitos.

Un PDA consiste en:

Alfabeto de Entrada: Un conjunto finito de símbolos de entrada que el autómata puede leer mientras realiza su proceso de reconocimiento.

Alfabeto de Pila: Un conjunto finito de símbolos que pueden ser almacenados en la pila.

Conjunto de Estados: Un conjunto finito de estados que el autómata puede tener. Incluye un estado inicial y estados de aceptación o rechazo.

Transiciones: Conjunto de reglas que dictan cómo el autómata cambia de estado basado en el símbolo de entrada actual, el símbolo en la parte superior de la pila y la transición se lleva a cabo.

Pila: Una estructura de datos tipo pila que almacena y recupera símbolos. La pila permite almacenar símbolos mientras el autómata procesa la entrada y puede ser consultada durante las transiciones.

El funcionamiento de un PDA se puede resumir en los siguientes pasos:

El PDA comienza en su estado inicial con una pila vacía.

Lee el símbolo de entrada actual.

Consulta la tabla de transiciones para determinar qué acción tomar. Las transiciones se basan en el estado actual, el símbolo de entrada actual y el símbolo en la parte superior de la pila.

Realiza la acción especificada por la transición, que puede incluir cambiar de estado, apilar un símbolo en la pila o desapilar un símbolo de la pila.

Repite los pasos 2 a 4 hasta que se haya leído toda la entrada o se alcance un estado de aceptación o rechazo.

Los PDAs son útiles para reconocer lenguajes que no pueden ser reconocidos por autómatas finitos. Se utilizan en la teoría de lenguajes formales, teoría de compiladores y en la modelización de gramáticas contextuales. Los lenguajes contextuales son más poderosos que los lenguajes regulares y permiten describir estructuras más complejas, como las expresiones matemáticas balanceadas, los lenguajes de programación y otros tipos de gramáticas con contextos más ricos.

Explicar la notación gráfica de PDA.

La notación gráfica de un Autómata de Pila (Push Down Automata, PDA) es una representación visual que ayuda a comprender y visualizar cómo el autómata procesa la entrada y las transiciones entre estados. Esta notación utiliza diagramas de estados y transiciones para ilustrar el funcionamiento del PDA de manera más intuitiva. Aquí te explico los elementos clave de la notación gráfica de PDA:

Nodos o Estados: Cada estado del PDA se representa como un nodo en el diagrama. Estos nodos pueden ser etiquetados con el nombre del estado y se pueden identificar como estado inicial, estado de aceptación o estado de rechazo.

Transiciones: Las transiciones entre estados se representan con flechas dirigidas. Cada transición incluye información sobre los símbolos de entrada, los símbolos en la parte superior de la pila y las acciones que se realizan durante la transición.

Etiquetas de Transición: Las etiquetas en las flechas representan las condiciones para que ocurra una transición. Pueden contener información sobre el símbolo actual de entrada, el símbolo actual en la parte superior de la pila y el símbolo a apilar o desapilar.

Pila: La pila se suele representar en el diagrama mediante una línea vertical junto al estado. Se muestra la parte superior de la pila y los símbolos almacenados en ella. Los símbolos se apilan o desapilan durante las transiciones.

Símbolo de Entrada: En las etiquetas de transición, se muestra el símbolo de entrada actual. Esto indica qué símbolo se está leyendo de la cadena de entrada en ese momento.

Símbolo en la Pila: En las etiquetas de transición, se muestra el símbolo actual en la parte superior de la pila. Esto indica qué símbolo se encuentra en la parte superior de la pila en ese momento.

Símbolo a Apilar/Desapilar: En las etiquetas de transición, se muestra el símbolo que se va a apilar o desapilar en la pila durante la transición.

Condiciones de Transición: En las etiquetas de transición, se describen las condiciones que deben cumplirse para que la transición se active. Esto incluye el estado actual, el símbolo actual de entrada y el símbolo actual en la parte superior de la pila.

Aceptación y Rechazo: Los estados de aceptación suelen estar resaltados de alguna manera, como con un doble círculo, y los estados de rechazo se pueden representar de manera similar.

La notación gráfica de PDA es una herramienta visual poderosa para comprender cómo un autómata de pila procesa la entrada y realiza las transiciones. Ayuda a representar de manera clara y concisa las reglas y condiciones de las transiciones, lo que facilita la comprensión de la dinámica del PDA en un nivel más intuitivo.

Explicar las características de los lenguajes de PDA.
Los lenguajes reconocidos por un Autómata de Pila (PDA, por sus siglas en inglés Push Down Automata) son conocidos como "lenguajes de PDA". Estos lenguajes se encuentran en la categoría de lenguajes contextuales en la jerarquía de Chomsky, lo que significa que son más poderosos en términos de capacidad de reconocimiento que los lenguajes regulares pero menos poderosos que los lenguajes sensibles al contexto. Aquí están las características clave de los lenguajes de PDA:

Contexto Libre: Los lenguajes de PDA son lenguajes contextuales, lo que significa que pueden describir patrones más complejos que los lenguajes regulares. Pueden capturar estructuras anidadas y relaciones de contexto en la gramática, lo que los hace adecuados para describir muchas construcciones en lenguajes de programación y otros contextos.

Expresividad: Los lenguajes de PDA pueden reconocer lenguajes que los autómatas finitos (AF) no pueden, como el conjunto de todas las cadenas en el lenguaje de programación C, que incluyen expresiones anidadas y constructos de control de flujo.

Memoria Limitada: Aunque los PDA tienen la capacidad de almacenar información en su pila, esta memoria es limitada. A diferencia de las máquinas de Turing, los PDA no tienen memoria infinita y pueden operar con una cantidad finita de almacenamiento en la pila.

Operaciones de Pila: Una característica distintiva de los PDA es su pila, que les permite almacenar y recuperar símbolos mientras procesan la entrada. Pueden apilar (push) símbolos en la parte superior de la pila y desapilar (pop) símbolos de la parte superior de la pila.

Transiciones Basadas en Contexto: Las transiciones en un PDA se realizan en función del símbolo de entrada actual, el símbolo en la parte superior de la pila y el estado actual. Esto permite que el autómata tome decisiones basadas en el contexto en lugar de simplemente examinar la entrada actual.

Gramáticas Contextuales: Los lenguajes de PDA están asociados con gramáticas contextuales que generan esos lenguajes. Las producciones en estas gramáticas pueden modificar el contenido de la pila durante el proceso de derivación.

Capacidad de Reconocimiento: Los PDA pueden reconocer lenguajes generados por gramáticas contextuales. Pueden aceptar cadenas validando que existe una secuencia de transiciones que lleva desde el estado inicial hasta un estado de aceptación mientras procesan la cadena y la pila.

No Todos los Lenguajes Contextuales: Aunque los lenguajes de PDA son una superconjunto de los lenguajes regulares, no todos los lenguajes contextuales son reconocidos por PDA. Existen lenguajes contextuales que requieren una mayor capacidad de reconocimiento, como las gramáticas sensibles al contexto.

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

La determinación de equivalencia entre un Autómata de Pila (PDA, por sus siglas en inglés Push Down Automata) y una Gramática Libre de Contexto (CFG, por sus siglas en inglés Context-Free Grammar) implica verificar si representan el mismo lenguaje, es decir, si generan exactamente las mismas cadenas. Aquí está el proceso general para determinar la equivalencia entre un PDA y una CFG:

Pasos para Determinar la Equivalencia entre un PDA y una CFG:

Dado: Tienes un PDA y una CFG.

Conversión PDA a CFG (opcional): Si el PDA no está dado en forma de CFG, es posible que necesites realizar una conversión. Esto puede ser un proceso complicado y depende del PDA en cuestión. Una vez que tengas una CFG equivalente al PDA, puedes continuar con los siguientes pasos.

Construcción de PDA a CFG: Si has realizado la conversión, tu PDA ahora tiene una CFG equivalente.

Comparación de las Gramáticas: Ahora que tienes ambas CFG (la original y la derivada del PDA), debes compararlas para verificar si son equivalentes. Para ello, puedes llevar a cabo los siguientes pasos:

> Comprueba si las producciones de ambas CFG son idénticas. Esto significa que las reglas de derivación de ambas gramáticas deben ser las mismas.
>
> Asegúrate de que las variables iniciales de ambas CFG sean las mismas. En otras palabras, las variables que se usan para comenzar la derivación deben ser las mismas en ambas gramáticas.
>
> Verifica que las producciones de ambas CFG generen las mismas cadenas. Esto implica que ambas gramáticas deben ser capaces de generar exactamente las mismas cadenas a partir de sus derivaciones.

Construcción de CFG a PDA (opcional): Si deseas verificar la equivalencia al revés (CFG a PDA), puedes construir un PDA a partir de la CFG dada. Nuevamente, esto puede ser un proceso complejo y depende de la CFG en cuestión.

Comparación de PDAs: Si has construido el PDA a partir de la CFG, ahora tienes dos PDAs: el original y el derivado de la CFG. Debes compararlos para verificar si son equivalentes.

Asegúrate de que los estados, símbolos de entrada y de pila, transiciones y condiciones de aceptación/rechazo sean idénticos en ambos PDAs.

Comprueba que ambos PDAs acepten y rechacen exactamente las mismas cadenas.

Resultado: Si los PDAs y las CFG son equivalentes, esto significa que representan el mismo lenguaje. Si no son equivalentes, entonces representan lenguajes distintos.

La construcción de un Autómata de Pila (PDA) a partir de una Gramática Libre de Contexto (CFG) es un proceso que permite visualizar cómo una CFG genera cadenas y cómo un PDA puede reconocer esas mismas cadenas. Aquí está el proceso general para construir un PDA a partir de una CFG:

Pasos para Construir un PDA a partir de una CFG:

Dado: Tienes una Gramática Libre de Contexto (CFG).

Definir el PDA: Define el PDA con los siguientes componentes:

Alfabeto de Entrada: Consiste en los símbolos terminales de la CFG, es decir, los símbolos que aparecen en las cadenas generadas por la gramática.

Alfabeto de Pila: Consiste en los símbolos de pila utilizados por el PDA. Puede ser igual al alfabeto de entrada o puede incluir símbolos adicionales.

Conjunto de Estados: Define los estados del PDA. Debes tener al menos un estado inicial y, opcionalmente, estados de aceptación y rechazo.

Transiciones: Cada producción en la CFG se traduce en una o más transiciones en el PDA. Las transiciones se basan en la forma de las producciones y se definen para cambios de estado, símbolos de entrada y símbolos en la parte superior de la pila.

Estado de Aceptación: Define los estados de aceptación si el PDA debe aceptar cadenas. Puedes definir uno o varios estados de aceptación.

Estado de Rechazo: Define los estados de rechazo si el PDA debe rechazar cadenas. Esto puede ser útil para indicar errores o cadenas no válidas.

Construcción de Transiciones: Las transiciones se basan en las producciones de la CFG. Cada producción se descompone en una serie de pasos de acuerdo a la forma de la producción. Por ejemplo, si tienes una producción A -> aB, las transiciones pueden involucrar la lectura de a en la entrada, empujar B en la pila y cambiar de estado.

Estado Inicial y de Aceptación: Define el estado inicial y los estados de aceptación/rechazo del PDA.

Verificación: Verifica que las transiciones se construyan correctamente y que el PDA pueda generar las mismas cadenas que la CFG.

Prueba: Prueba el PDA con diversas cadenas generadas por la CFG para asegurarte de que el PDA las acepta.

Ejemplo:

Supongamos que tenemos la siguiente CFG:
S -> aSb | ε

Podemos construir un PDA de la siguiente manera:

Alfabeto de Entrada: {a, b}

Alfabeto de Pila: {Z0}

Conjunto de Estados: {q0, q1}

Transiciones:
q0, a, Z0 -> q0, aZ0 (Empujar a en la pila)
q0, a, a -> q0, aa (Empujar a en la pila)
q0, b, a -> q1, ε (Desapilar a de la pila)
q1, b, a -> q1, ε (Desapilar a de la pila)
q1, ε, Z0 -> q1, ε (Aceptación)

Bibliografía

Introduction to the Theory of Computation por Michael Sipser

Sipser, M. (Año de publicación). Introduction to the Theory of Computation. Editorial.
Parsing Techniques: A Practical Guide por Dick Grune y Ceriel J.H. Jacobs

Grune, D., & Jacobs, C. J. H. (Año de publicación). Parsing Techniques: A Practical Guide. Editorial.
Programming Language Pragmatics por Michael L. Scott

Scott, M. L. (Año de publicación). Programming Language Pragmatics. Editorial.
Introduction to Automata Theory, Languages, and Computation por John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman

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

Unidad I

En Fundamentos generales de programación, de Luis Joyanes Aguilar, 33-37. México: Mc Graw-Hill, 2013.

Aquilar, Luis joyanes. En Metodologia de la programación Diagramas de flujo, Algoritmos y programación Estructurada, 66-88. Mexico: Mc Graw-Hill, 1987.

En Fundamentos de Programación , de Manuel Santos. Ismael Patiño. Raul Carrasco, 37-40. México : AlfaOmega-Rama, 2006.

Chorda, Gloria de Antonio. Ramon. «Metodologia de la Programación.» 39-41. España: Ra-ma, s.f.

En Lenguaje Ensamblador para Mircrocomputadoras IBM, de J. Terry Godfrey, 73-76. México: Prentice Hall, 1991.

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.

Unidad II

En Compiladores, Principios, Técnicas y Herramientas , de Sethi, R, Ullman, J Aho A, 86-89,94-108,115-131,164-187,190-193,200-201,209-212,221-226,264-274,443-444. U.S.A: Adisson-Wesley Iberoamericana , 1990.

En Compiladores e Interpretes. Teoria y Practica, de M Moreno, M de la cruz, A Ortega y E Pulido, 78-85, 191-194, 199-204, 221-223. España: Pearson- Prentice Hall, 2006.

Unidad III

En Compiladores, Principios, Técnicas y Herramientas , de Sethi, R, Ullman, J Aho A, 86-89,94-108,115-131,164-187,190-193,200-201,209-212,221-226,264-274,443-444. U.S.A: Adisson-Wesley Iberoamericana , 1990.

En Compiladores e Interpretes. Teoria y Practica, de M Moreno, M de la cruz, A Ortega y E Pulido, 78-85, 191-194, 199-204, 221-223. España: Pearson- Prentice Hall, 2006.

Unidad IV

En Compiladores, Principios, Técnicas y Herramientas , de Sethi, R, Ullman, J Aho A, 86-89,94-108,115-131,164-187,190-193,200-201,209-212,221-226,264-274,443-444. U.S.A: Adisson-Wesley Iberoamericana , 1990.

En Compiladores e Interpretes. Teoria y Practica, de M Moreno, M de la cruz, A Ortega y E Pulido, 78-85, 191-194, 199-204, 221-223. España: Pearson- Prentice Hall, 2006.

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

Sipser, M. (Year). Introduction to the Theory of Computation. Publisher.

Hopcroft, J. E., Motwani, R., & Ullman, J. D. (Year). Formal Languages and Their Relation to Automata. Publisher.

Cooper, K. D., & Torczon, L. (Year). Engineering a Compiler. Publisher.

Grune, D., & Jacobs, C. J. H. (Year). Parsing Techniques: A Practical Guide. Publisher.

Kozen, D. C. (Year). Automata and Computability. Publisher.

Tremblay, J. P., & Harrington, R. (Year). Introduction to Automata Theory, Languages, and Computation. Publisher.

Appel, A. W. (Year). Modern Compiler Implementation in C/Java/ML. Publisher.

Scott, M. L. (Year). Programming Language Pragmatics. Publisher.