Instituto Tecnológico Nacional De México

Instituto Tecnológico De Pachuca

Ingeniería en sistemas computacionales

Semestre: 6 **Grupo:** A

Materia:

Lenguajes y Autómatas 1

Trabajo:

5.1 Requerimiento de la Gramatica

Docente: Ing. Rodolfo Baume Lazcano

Alumnos: Melanie Domínguez Figueroa
No. Control: 22200181

Ángel Daniel Samperio Gardini

No. Control: 20200940

Fecha: 30 de Mayo de 2024

DESCRIPCIÓN DETALLADA DE LA GRAMÁTICA DEL LENGUAJE

Objetivo:

Proveer una descripción clara y completa de la gramática del lenguaje que se analizara, especificando todas las producciones y reglas necesarias para la correcta implementación del analizador sintáctico.

Contenido

INTRODUCCIÓN

Este programa es un analizador léxico (también llamado tokenizador) que lee caracteres de un archivo de texto y los grupos en tokens. El analizador léxico está representado por la clase 'AnalizadorLexico', que tiene un objeto 'BufferedReader' llamado lector para leer caracteres del archivo de entrada, una variable 'caracterActual' para almacenar el carácter actual y una variable 'tokenActual' para almacenar el token actual.

El constructor de la clase 'AnalizadorLexico' inicializa el objeto lector con el archivo de entrada especificado y lee el primer carácter del archivo. La función 'hasNextToken' devuelve 'true' si hay más tokens para leer y falseen caso contrario. La función 'nextToken' lee y devuelve el siguiente token del archivo de entrada. Si el carácter actual es un dígito, 'leerNumeros' llama al método para leer el número y crear un token de tipo 'NUMERO'. Si el carácter actual es +, se crea un token de tipo 'MAS'. De lo contrario, se crea un token de tipo "CARACTER_DESCONOCIDO" con el carácter actual.

La función 'leerEspacios' salta los espacios en blanco, 'leerNumero' lee un número y 'leerCaracterDesconocido' lee y crea un token de tipo "CARACTER_DESCONOCIDO" con el carácter actual.

El programa principal crea un objeto 'AnalizadorLexico' con el archivo de entrada especificado y llama a las funciones 'hasNextToken' y 'nextToken' en un bucle hasta que se haya leído todo el archivo. Los tokens leídos se imprimen en la salida estándar.

La clase 'Token' representa un token y tiene dos atributos: tipo de tipo 'TipoToken' para almacenar el tipo de token y valor de tipo 'String' para almacenar el valor del token. La clase 'TipoTokenes' una enumeración que define los tipos de tokens posibles.

El propósito de este código es implementar un analizador léxico simple en Java que lea un archivo de texto y genere tokens a partir de los caracteres del archivo.

El analizador léxico es una etapa importante en el análisis de un programa de computadora, ya que divide el código fuente en tokens, que son las unidades más pequeñas de significado en un lenguaje de programación.

En este caso, el analizador léxico implementa reconoce cuatro tipos de tokens:' NUMERO ', ' MAS ', ' CARACTER_DESCONOCIDO ' y ' FIN_ARCHIVO '. Los números son secuencias de dígitos, ' MAS ' representan el símbolo +, ' CARACTER_DESCONOCIDO ' representan cualquier otro carácter e ' FIN_ARCHIVO 'indican el final del archivo de entrada. El código lee caracteres del archivo de entrada y construye tokens a partir de ellos. Cuando encuentre un espacio en blanco, salte al siguiente carácter. Si el carácter actual es un dígito, lee una secuencia de dígitos y crea un token de tipo 'NUMERO'. Si el carácter actual es +, crea un token de tipo MAS. De lo contrario, crea un token de tipo ' CARACTER_DESCONOCIDO ' con el carácter actual. El bucle principal del programa crea un objeto ' AnalizadorLexico 'con el archivo de entrada especificado y llama a las funciones ' hasNextToken ' y ' nextToken ' en un bucle hasta que se haya leído todo el archivo. Los tokens leídos se imprimen en la salida estándar.

SÍMBOLOS TERMINALES

Lista completa de los símbolos terminales del lenguaje

Palabras clave (private in, new, throws, public boolean, return, if, return new, null, else if, else, while, private void, char, enum, public, private)

Operadores (+, =, ==, !, -, // \\,)

Delimitadores (, (), {}, ;,)

Literales (“entrada.txt”,”+”, ‘null’, -1, ‘+’, ‘NUMERO’, ‘MAS’, CARÁCTER_DESCONOCIDO’, ‘FIN_ARCHIVO’)

Identificadores (BufferedReader lector, carácter Actual, token Actual, lector, lector.read, hasNextToken, Token, leerEspacios, leerNumero, leerCaracterDesconocido)

SÍMBOLOS NO TERMINALES

Definir los símbolos no terminales que representan categorías o estructuras sintácticas del lenguaje

‘Token’: representa un token individual, que es la unidad básica de análisis léxico.

‘AnalizadorLexico’: representa el analizador léxico en sí, que es responsable de leer el código de entrada y generar tokens.

‘leerCaracter()’: una función que lee el siguiente carácter del archivo de entrada.

‘obtenerSiguienteToken()’: una función que genera el siguiente token en el archivo de entrada.

‘estadoActual’: una variable que representa el estado actual del analizador léxico.

‘caracterActual’: una variable que representa el carácter actual que se está analizando.

‘tipoToken’: una variable que representa el tipo del token actual.

‘valorToken’: una variable que representa el valor del token actual.

PRODUCCIONES Y REGLAS

1. Programa → Ficha { Ficha }
   (Un programa se compone de uno o más tokens.)
2. Ficha → Número | Operador | CaracterDesconocido | FinArchivo
   (Un token puede ser un número, un operador, un carácter desconocido)
3. Numero → **ExcavarDigito { **Excavardígito } *
   (Un número se compone de uno o más dígitos.)
4. Operador → '+'
   (Un operador es el símbolo '+'.)
5. CaracterDesconocido → **CaracterCaracterNoTerminal
   (Un carácter desconocido es cualquier carácter que no sea un dígito, un operador)
6. FinArchivo→ ε
   (El fin de archivo se representa con una cadena vacía.)
7. Dígito → '0'
   (Un dígito es cualquier carácter entre '0' y ‘9’)
8. CaracterNoTerminal → cualquier carácter que no sea terminal

Notas:

Los símbolos en las terminales son: **Programa , **TokenFicha , Número , **Operador , **CocheCaracterDesconocido , FinArchivo , Digito** y **CaracterNoTerminal** .

Los símbolos terminales son: los dígitos (0-9), el operador '+', los caracteres no terminales y el fin de cadena (representado por una vacía).

- La producción 1 indica que un programa se compone de uno o más tokens, lo que se logra mediante la repetición de la producción 2.
- La producción 2 indica que un token puede ser un número, un operador, un carácter desconocido o el fin de archivo.
- La producción 3 indica que un número se compone de uno o más dígitos.
- La producción 4 indica que un operador es el símbolo '+'.
- La producción 5 indica que un carácter desconocido es cualquier carácter que no sea un dígito, un operador o un espacio en blanco.
- La producción 6 indica que el fin de archivo se representa con una cadena vacía.
- La producción 7 indica que un dígito es cualquier carácter entre '0' y '9'.
- La producción 8 indica que un carácter no terminal es cualquier carácter que no sea un dígito, un operador o un espacio en blanco.

SÍMBOLO INICIAL

El símbolo inicial de este programa es **Programa** .

PRECEDENCIA Y ASOCIATIVIDAD

En este código, solo hay un operador, que es el operador de suma (+). Por lo tanto, no hay necesidad de definir reglas de precedencia y asociatividad, ya que no hay conflictos entre operadores.

Sin embargo, si se agregan más operadores en el futuro, se podrían definir las reglas de precedencia y asociatividad de la siguiente manera:

Precedencia :

El operador de suma (+) tiene una precedencia baja.

Asociatividad :

El operador de suma (+) es asociativo por la izquierda.

Esto significa que cuando hay varias operaciones de suma en una expresión, se evalúan de izquierda a derecha. Por ejemplo, la expresión a + b + cse evalúa como (a + b) + c.

Es importante destacar que, en este código, no hay operadores con precedencia alta, como exponentes o multiplicación, por lo que no hay necesidad de definir reglas de precedencia y asociatividad más complejas.

COMENTARIOS Y ANOTACIONES

Producción de Programa

Programa → Ficha { Ficha }

Esta producción indica que un programa está compuesto por uno o más tokens. Un token es una unidad básica de entrada, como un número o un símbolo. La llave {} indica que el token puede repetirse cero o más veces.

Por ejemplo, si el input es *1 + 2 + 3,* el programa se descompone en los siguientes tokens: *1, +, 2, +, 3*. Cada token es una unidad separada que se procesa de manera independiente.

Producción de tokens

Ficha → Número | Operador

Esta producción indica que un token puede ser un número o un operador. La barra vertical |se utiliza para indicar la opción entre dos o más producciones.

Producción de Números

Número → [0-9]+

Esta producción indica que un número es una secuencia de uno o más dígitos (del 0 al 9). El símbolo +indica que la secuencia debe tener al menos un dígito.

Por ejemplo, los siguientes son números válidos: 1, 12, 123, etc.

Producción de Operador

Operador → +

Esta producción indica que el único operador permitido es el símbolo +.

Reglas de precedencia y asociatividad

Como mencioné anteriormente, en este código solo hay un operador, que es el operador de suma (+). Por lo tanto, no hay necesidad de definir reglas de precedencia y asociatividad, ya que no hay conflictos entre operadores.

Sin embargo, si se agregan más operadores en el futuro, se podrían definir las reglas de precedencia y asociatividad de la siguiente manera:

La precedencia del operador de suma (+) es baja.

El operador de suma (+) es asociativo por la izquierda.

Esto significa que cuando hay varias operaciones de suma en una expresión, se evalúan de izquierda a derecha. Por ejemplo, la expresión *a* + *b* + *c* se evalúa como *(a* + *b)* + *c.*