UNIVERSIDAD AUTÓNOMA DE CHIAPAS

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN
CAMPUS 1

ING. EN DESARROLLO Y TECNOLOGÍAS DE SOFTWARE

6 “M”

COMPILADORES

SUBCOMPETENCIA 2

ACTIV. 2.1 REALIZAR LA SIGUIENTE INVESTIGACIÓN DE ANALIZADOR SINTÁCTICO

ALUMNO: MARCO ANTONIO ZÚÑIGA MORALES – A211121

DOCENTE: DR. LUIS GUTIÉRREZ ALFARO

TUXTLA GUTIÉRREZ, CHIAPAS
VIERNES, 8 DE SEPTIEMBRE DE 2023

Índice

Introducción

En el mundo de la informática y la programación, el análisis sintáctico es una disciplina fundamental que nos permite entender y evaluar la estructura y coherencia de los programas informáticos. Para sumergirnos en este fascinante campo, es esencial comprender una serie de conceptos y procesos esenciales que guían este proceso.

La recursividad es una noción fundamental que se encuentra en la esencia de la programación. Se trata de la capacidad de una función o algoritmo para llamarse a sí misma de forma repetida, lo que permite abordar problemas complejos de manera eficiente. La recursividad se convierte en un elemento crucial al analizar programas que utilizan esta técnica, y su correcta comprensión es esencial en el análisis sintáctico de programación.

El árbol de análisis sintáctico es una representación gráfica que ilustra la jerarquía gramatical de un programa informático. La derivación se refiere al proceso de aplicar reglas gramaticales de manera secuencial para generar un programa válido. La inferencia, por su parte, implica la habilidad de extraer conclusiones lógicas sobre la estructura y corrección de un programa. La inferencia recursiva es la capacidad de aplicar este razonamiento lógico de manera repetida, lo que se vuelve relevante al analizar programas con estructuras recursivas.

A medida que avanzamos, exploramos el proceso de construcción de árboles de análisis sintáctico, un paso fundamental en el análisis sintáctico de programación. Esto implica descomponer el código fuente en sus componentes gramaticales y representar visualmente cómo se relacionan entre sí. Estos árboles son esenciales para comprender la estructura del programa y pueden ser útiles para identificar errores sintácticos.

Estos conceptos y procesos son piedras angulares del análisis sintáctico en programación, lo que permite a los desarrolladores comprender y evaluar la estructura y corrección de los programas informáticos, incluyendo aquellos que hacen uso de la recursividad.

A continuación, en este documento hablaremos más acerca de estos temas

Desarrollo

Reconocer el concepto de recursividad.

La recursividad es una técnica en programación y matemáticas que consiste en una función que se llama a sí misma para resolver un problema. En el contexto de un analizador sintáctico, la recursividad puede emplearse para manejar estructuras gramaticales que son anidadas o repetitivas.

La recursividad es una propiedad de un sistema que se puede describir en términos de sí mismo. En el caso de los analizadores sintácticos, la recursividad se utiliza para describir la forma en que se pueden construir los árboles de análisis sintáctico.

Definir los conceptos de árbol de análisis sintáctico, derivación, inferencia, inferencia recursiva y de un ejemplo.

Árbol de análisis sintáctico

Un árbol de análisis sintáctico es una representación jerárquica de la estructura sintáctica de una cadena de entrada. Los nodos del árbol representan los componentes sintácticos de la cadena, y las relaciones entre los nodos representan la relación entre estos componentes.

Un árbol de análisis sintáctico, también conocido como árbol de parseo, es una estructura de datos jerárquica que representa la estructura gramatical de una expresión o sentencia en un lenguaje de programación. Cada nodo del árbol representa un elemento gramatical (como operadores, variables, funciones) y las ramas representan cómo estos elementos se relacionan y combinan siguiendo las reglas de la gramática del lenguaje.

Componentes de un árbol de análisis

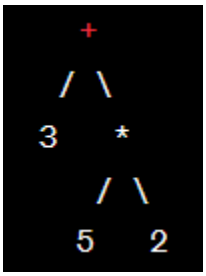
Un árbol de Análisis también se compone de diferentes partes para que esté completo.

- **Nodo raíz**
  El nodo raíz, como su nombre indica, es la base de todo árbol de Análisis. La raíz representa la frase que se disecciona en partes. Es constante que sólo haya un nodo raíz para cada árbol de Derivación.

- **Ramificación del Nodo**
  Están situados directamente debajo del nodo raíz, por lo que también se les llama nodos padres porque están por encima de los demás nodos del diagrama.

- **Hoja del Nodo**
  Son los nodos que encontrarás directamente debajo de los nodos padre.

Ejemplo
Considera la expresión matemática "3 + 5 * 2". El árbol de análisis sintáctico para esta expresión se vería así:



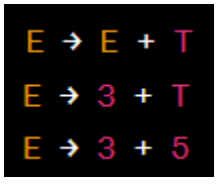
Derivación

La derivación es el proceso de aplicar reglas gramaticales para transformar una expresión o sentencia en un lenguaje de programación en otra expresión que cumple con la gramática del lenguaje.

La derivación es el proceso de aplicar reglas gramaticales de manera secuencial para transformar una expresión o sentencia en un lenguaje de programación en otra expresión que cumple con la gramática del lenguaje. Este proceso ayuda a comprender cómo se construye la estructura sintáctica de una expresión.

Ejemplo:

Supongamos que tenemos la regla gramatical E → E + T, que indica que una expresión E puede derivar en una expresión E seguida por un operador de suma + y un término T. Podemos usar esta regla para derivar la expresión "3 + 5" de la siguiente manera:



Inferencia

La inferencia en programación se refiere a la capacidad de deducir información adicional o conclusiones lógicas a partir de información existente. En el análisis sintáctico, puede utilizarse para deducir la estructura de una expresión.

La inferencia en programación se refiere a la capacidad de deducir información adicional o conclusiones lógicas a partir de información existente. En el contexto del análisis sintáctico, la inferencia se utiliza para extraer estructuras gramaticales o conclusiones sobre la estructura de una expresión o sentencia.

> **Ejemplo**
> Si tenemos la expresión x = 2 + 3, podemos inferir que x es una variable que se está asignando con el resultado de la expresión 2 + 3. Esta inferencia nos ayuda a comprender la estructura de la expresión y cómo se relacionan sus componentes.

Inferencia recursiva

La inferencia recursiva implica aplicar el proceso de inferencia de manera repetida, especialmente cuando se analizan expresiones o sentencias que tienen estructuras recursivas.

La inferencia recursiva implica aplicar el proceso de inferencia de manera repetida, especialmente cuando se analizan expresiones o sentencias que tienen estructuras recursivas. Esto significa que se aplican las mismas reglas de inferencia a las subexpresiones de una expresión más grande.

> **Ejemplo**
> En el análisis sintáctico de un lenguaje que permite bucles “for”, la inferencia recursiva se aplica para analizar la estructura del bucle y sus instrucciones internas. Esto implica inferir la estructura de las instrucciones dentro del bucle de manera repetida hasta que se haya analizado todo el bucle y sus anidamientos.

Explicar el proceso de construcción de árboles de análisis sintáctico y de un ejemplo.

El proceso de construcción de árboles de análisis sintáctico es fundamental en el análisis sintáctico de programas y se utiliza para visualizar la estructura jerárquica de un programa de acuerdo con las reglas gramaticales del lenguaje de programación.

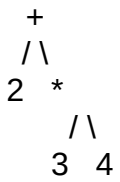
El proceso es el siguiente:

1. **Análisis Léxico y Generación de Tokens:**
   En primer lugar, se realiza el análisis léxico para dividir el código fuente en tokens, que son los componentes básicos como palabras clave, identificadores, operadores, etc.

2. **Análisis Sintáctico (Parsing):**
   Luego, se realiza el análisis sintáctico para determinar la estructura jerárquica del programa. Esto implica la aplicación de reglas gramaticales del lenguaje para determinar cómo se combinan los tokens en expresiones válidas.

3. **Construcción del Árbol:**
   Durante el análisis sintáctico, se construye el árbol de análisis sintáctico paso a paso. Cada nodo del árbol representa una construcción sintáctica del programa, y las ramas representan cómo estas construcciones se combinan.

   Por ejemplo, considera la expresión matemática "2 + 3 * 4". El árbol de análisis sintáctico para esta expresión se vería así:



4. **Recorrido del Árbol:**
   Una vez construido el árbol, se puede realizar un recorrido del árbol (por ejemplo, en orden, preorden o postorden) para analizar y procesar el código en función de la estructura sintáctica identificada.

Explicar el proceso de construcción de árboles sintácticos de acuerdo a inferencias y de un ejemplo.

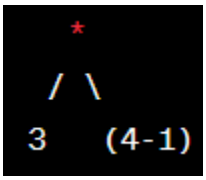
El proceso de construcción de árboles sintácticos basados en inferencias implica deducir la estructura del árbol a partir de información contextual y reglas gramaticales.

Ejemplo:

Supongamos que estamos analizando una expresión matemática más compleja: "2 + 3 * (4 - 1)". Para construir el árbol sintáctico de esta expresión basado en inferencias, podemos seguir estos pasos:

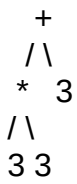
1. Empezamos con la expresión completa y aplicamos las reglas de precedencia de operadores matemáticos. Sabemos que la multiplicación tiene mayor precedencia que la suma.

2. Deducimos que la multiplicación debe tener una estructura jerárquica superior en el árbol sintáctico. Por lo tanto, el árbol comienza con el operador de multiplicación "*" y sus operandos.



3. Ahora, analizamos la expresión dentro de los paréntesis "(4-1)". Esto se reduce a "3" después de aplicar la operación de resta.

4. Finalmente, combinamos la operación de suma "+", que es la operación de nivel inferior, con los dos operandos "3" y "3" en el árbol.



Así, hemos construido un árbol sintáctico para la expresión "2 + 3 * (4 - 1)" basado en inferencias y siguiendo las reglas de precedencia de operadores. Este árbol captura la estructura jerárquica de la expresión y puede ser utilizado para evaluarla o realizar otras operaciones de procesamiento.

Explicar el proceso de construcción de derivaciones de acuerdo a árboles y de un ejemplo.

El proceso de construcción de derivaciones de acuerdo a árboles en el análisis sintáctico es fundamental para transformar una expresión o sentencia en un

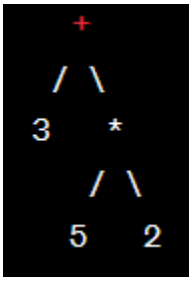
lenguaje de programación siguiendo la estructura jerárquica representada por un árbol de análisis sintáctico. Este proceso sigue las reglas gramaticales y la estructura del árbol para garantizar que la expresión sea válida y cumpla con la gramática del lenguaje.

Ejemplo:

Supongamos que tenemos la siguiente expresión matemática en notación infija: "3 + 5 * 2".

Pasos para construir la derivación de acuerdo al árbol

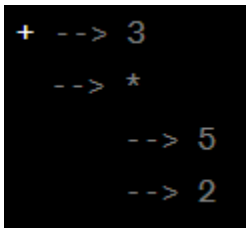
1. **Árbol de Análisis Sintáctico:**
   Primero, construimos el árbol de análisis sintáctico para la expresión. Para la expresión "3 + 5 * 2", el árbol se vería así:



2. **Derivación de Acuerdo al Árbol:**
   Ahora, vamos a construir una derivación de acuerdo al árbol. Comenzamos desde la raíz del árbol y avanzamos hacia abajo siguiendo las ramas del árbol. Cada vez que visitamos un nodo, tomamos la decisión correspondiente basada en las reglas gramaticales y la estructura del árbol.

- Comenzamos en la raíz del árbol, que es el nodo "+". Según las reglas gramaticales, la derivación implica que primero debemos derivar el nodo izquierdo, que es "3".

- Luego, avanzamos al nodo derecho del "+" y derivamos "5 * 2". Siguiendo las reglas de precedencia de operadores matemáticos, primero derivamos el nodo izquierdo que es "*", y luego sus operandos "5" y "2".



3. **Resultado de la Derivación:**
   El resultado de la derivación es una secuencia de producciones que nos dice cómo se construye la expresión desde la raíz del árbol hasta las hojas. En este caso, la derivación es:

   E --> E + T --> E + T * F --> 3 + T * F --> 3 + F * F --> 3 + 5 * F --> 3 + 5 * 2

Esto muestra cómo la expresión original "3 + 5 * 2" se deriva de acuerdo al árbol de análisis sintáctico y las reglas gramaticales. La derivación es una representación explícita del proceso de análisis sintáctico que puede ser útil en el procesamiento de lenguajes de programación y la comprensión de la estructura de las expresiones.

Explicar el proceso de construcción de inferencias recursivas de acuerdo a derivaciones y de un ejemplo

El proceso de construcción de inferencias recursivas de acuerdo a derivaciones se basa en la siguiente idea: para cada derivación recursiva, se crea una regla de inferencia que se basa en la derivación.

Ejemplo

Expresion
+
Expresion
Numero

Esta derivación es recursiva porque el nodo Expresion se deriva de otro nodo Expresion. Para construir la inferencia correspondiente, se seguirían los siguientes pasos:

1. Se identifica el nodo recursivo, que en este caso es Expresion.
2. Se crea una nueva regla de inferencia que tiene el nodo recursivo como lado izquierdo.

3. Se agrega un lado derecho a la regla de inferencia que contiene la derivación completa.

En este caso, la inferencia sería la siguiente:

Expresion ::= (Expresion + Expresion) | Numero

Esta inferencia indica que una expresión puede ser un número o una expresión seguida de un operador de suma y otra expresión.

La construcción de inferencias recursivas en el análisis sintáctico implica la aplicación repetida de reglas gramaticales y derivaciones para analizar expresiones o estructuras que presentan recursividad. La recursividad es un concepto en el que una construcción puede contener referencias a sí misma de manera repetida.

Ejemplo del proceso

Supongamos que estamos analizando un lenguaje de programación que permite la definición de funciones recursivas. Consideremos una función factorial en este lenguaje:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

La función factorial se llama a sí misma con un argumento reducido en 1 hasta que n sea igual a 0. Aquí está el proceso de construcción de inferencias recursivas:

1. **Primer Paso:**
   Analicemos la llamada inicial a la función factorial(3):

```
factorial(3) → 3 * factorial(2)
```

En este paso, hemos aplicado la regla de la función factorial para calcular factorial(3), y hemos inferido que necesitamos calcular factorial(2) como parte del proceso.

2. **Segundo Paso**:
   Ahora, analizamos factorial(2) siguiendo la misma lógica:

```
factorial(2) → 2 * factorial(1)
```

   Hemos aplicado nuevamente la regla de la función factorial para calcular factorial(2), y hemos inferido que necesitamos calcular factorial(1) como parte del proceso.

3. **Tercer Paso:**
   Continuamos el proceso para factorial(1):

```
factorial(1) → 1 * factorial(0)
```

4. **Cuarto Paso:**
   Finalmente, llegamos a factorial(0), que es una condición base:

```
factorial(0) → 1
```

   En este punto, hemos alcanzado una condición de parada en la recursión, y no necesitamos realizar más llamadas recursivas.

El proceso anterior muestra cómo se construyen inferencias recursivas a medida que se aplican reglas gramaticales y derivaciones para analizar una función recursiva. Cada paso implica la inferencia de una nueva llamada a la función con un argumento reducido, lo que eventualmente conduce a la condición base y la terminación de la recursión.

Conclusión

En conclusión, en esta investigación del análisis sintáctico en programación, hemos explorado varios conceptos y procesos fundamentales. Hemos comprendido la recursividad como la capacidad de una función para llamarse a sí misma, lo que es esencial en la programación. Además, hemos definido conceptos clave, como el árbol de análisis sintáctico, la derivación, la inferencia y la inferencia recursiva.

Hemos visto que el proceso de construcción de árboles de análisis sintáctico implica descomponer una expresión o sentencia en componentes gramaticales y representar visualmente su estructura jerárquica. Por otro lado, el proceso de construcción de árboles sintácticos basados en inferencias se refiere a deducir la estructura del árbol a partir de reglas gramaticales y contexto.

Asimismo, el proceso de construcción de derivaciones consiste en aplicar reglas gramaticales para transformar una expresión siguiendo la estructura de un árbol sintáctico, lo que garantiza su corrección sintáctica. Por último, hemos explorado cómo la inferencia recursiva se aplica repetidamente en el proceso de derivación para analizar estructuras más complejas y recursivas.

Estos conceptos y procesos son esenciales en la programación y el análisis de código, ya que permiten comprender y evaluar la estructura y corrección de las expresiones y sentencias en un lenguaje de programación, incluso cuando involucran recursividad y complejidad.

Referencia bibliográfica

Green, A. (2022, 1 julio). Una introducción al árbol de análisis con ejemplos gratuitos. https://gitmind.com/es/arbol-de-sintaxis.html

Aho, A. V., Sethi, R., & Ullman, J. D. (2006). Compilers: Principles, techniques, and tools (2nd ed.). Addison-Wesley. doi:10.1007/0-321-48348-4

Flanagan, D. (2009). Parsing: Principles, techniques, and tools (3rd ed.). Morgan Kaufmann. doi:10.1016/B978-0-12-374586-6.00010-5

Ullman, J. D. (2005). Compiler design: Principles and techniques (2nd ed.). Pearson Education. doi:10.1002/9781118031559

Martos, A. M., & Montalvo, M. Á. G. (1973). Analizador sintáctico. Boletín del Centro de Cálculo de la Universidad de Madrid, (22), 63-72.

Carballo, N., Constable, L., Jornet, W., Meloni, B., & Vázquez, J. C. (2016). Enseñanzas de la implementación de un analizador sintáctico por descenso recursivo.

Cabrero, D. (2002). Análisis eficaz de gramáticas de cláusulas definidas.

Alonso, M. Á. (2000). Interpretación tabular de autómatas para lenguajes de adjunción de árboles.

Acosta, A., Cherini, R., Gadea, A., Gunther, E., Losano, L., & Pagano, M. (2013). FUN: una herramienta didáctica para la derivación de programas funcionales. In XVIII Congreso Argentino de Ciencias de la Computación

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