UNIVERSIDAD AUTÓNOMA DE CHIAPAS

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN
CAMPUS 1

Licenciatura en Ingeniería en Desarrollo y Tecnologías de Software

Act. 3.1. Realizar un Artex referente a Acciones semánticas de un analizador sintáctico

6 “M”

Materia: Compiladores

Docente: Luis Gutierrez Alfaro

ALUMNO:
A 211387
Steven de Dios Montoya Hernández

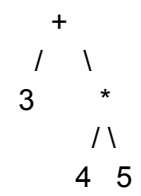
TUXTLA GUTIÉRREZ, CHIAPAS
Viernes, 27 De Octubre de 2023, 16:00

3.1 Árboles de Expresiones

Los árboles de expresiones son estructuras de datos utilizadas en la teoría de compiladores y en la programación en general para representar expresiones matemáticas o aritméticas de manera jerárquica. Estos árboles son especialmente útiles en la fase de análisis sintáctico de un compilador o intérprete, donde se descompone una expresión en sus componentes léxicos y se verifica su estructura gramatical.

Un árbol de expresiones se construye a partir de una expresión dada, descomponiéndola en sus partes constituyentes (operadores y operandos) y organizándolas de manera jerárquica de acuerdo con las reglas de precedencia y asociatividad de los operadores. Cada nodo en el árbol representa un operador o un operando, y las ramas del árbol conectan los nodos siguiendo la estructura de la expresión.

Por ejemplo, considera la expresión aritmética "3 + 4 * 5." Un árbol de expresiones para esta expresión se vería de la siguiente manera:



Los árboles de expresiones son útiles en la evaluación de expresiones, ya que permiten una evaluación recursiva de la expresión siguiendo un orden específico (generalmente el orden de un recorrido de árbol, como el recorrido en orden).

3.2 Acciones Semánticas de un Analizador Sintáctico

En el contexto de la construcción de un compilador, un analizador sintáctico es una parte fundamental del proceso de análisis de código fuente. Durante el análisis sintáctico, se verifica si la estructura del código fuente se ajusta a la gramática del lenguaje de programación. Las acciones semánticas son tareas o acciones que realiza el analizador sintáctico cuando se encuentra con ciertas construcciones sintácticas en el código fuente. Estas acciones pueden ser:

Generación de Árboles de Sintaxis Abstracta (AST): El analizador sintáctico puede construir un AST que representa la estructura del programa. El AST es una estructura de datos jerárquica que captura la semántica del programa de manera más abstracta que el árbol de análisis sintáctico. Este AST luego se utiliza en etapas posteriores del compilador, como la optimización y la generación de código.

Resolución de Nombres: Las acciones semánticas pueden ayudar a resolver nombres de variables y funciones. Esto implica asociar un nombre con su declaración correspondiente en el programa y garantizar que las referencias a esos nombres sean coherentes con sus definiciones.

Comprobación de Tipos: El analizador sintáctico también puede realizar comprobaciones de tipos para garantizar que las operaciones se realicen en tipos de datos compatibles. Por ejemplo, verificar que no se esté realizando una operación de suma entre una cadena y un número entero.

Control de Flujo: Las acciones semánticas pueden rastrear el flujo de control en el programa, como la verificación de que las sentencias return tengan un tipo de retorno válido en una función.

Optimizaciones Iniciales: En algunos casos, las acciones semánticas pueden realizar optimizaciones iniciales en el código, como la eliminación de código muerto o la simplificación de expresiones constantes.

3.3 Comprobaciones de Tipos en Expresiones

La comprobación de tipos en expresiones es una parte esencial de la fase de análisis semántico en un compilador o intérprete. Asegura que las operaciones y asignaciones en un programa se realicen entre tipos de datos compatibles, lo que evita errores de tiempo de ejecución y garantiza la coherencia del programa.

Aquí hay algunos conceptos clave relacionados con la comprobación de tipos en expresiones:

Inferencia de Tipos: En algunos lenguajes de programación, el tipo de una expresión puede inferirse en función de los tipos de sus operandos. Por ejemplo, si sumas dos enteros, el resultado también será un entero. La inferencia de tipos permite determinar el tipo de una expresión de manera automática.

Coerción de Tipos: En ciertos casos, los lenguajes de programación pueden realizar coerción de tipos implícita o explícita para hacer que los tipos sean compatibles en una operación. Por ejemplo, convertir un número entero en un número de punto flotante antes de realizar una operación aritmética.

Verificación de Tipos Estáticos vs. Dinámicos: Algunos lenguajes verifican los tipos en tiempo de compilación (verificación estática), mientras que otros lo hacen en tiempo de ejecución (verificación dinámica). La verificación estática es más estricta y puede detectar errores de tipos antes de que se ejecute el programa, lo que es deseable en términos de seguridad y rendimiento.

Reglas de Conversión de Tipos: En lenguajes que admiten conversión de tipos, existen reglas específicas para determinar cómo se realizará la conversión. Pueden ser reglas de promoción de tipos, conversión explícita, etc.

Comprobaciones de Tipos en Estructuras de Control: Las comprobaciones de tipos también se aplican en estructuras de control como las sentencias if y switch para asegurarse de que las expresiones condicionales sean de tipo booleano.

Sobrecarga de Operadores: Algunos lenguajes permiten la sobrecarga de operadores, lo que significa que los operadores pueden tener comportamientos diferentes según los tipos de datos involucrados. La comprobación de tipos debe manejar estas situaciones.

Síntesis

Árboles de Expresiones (3.1): Los árboles de expresiones son estructuras jerárquicas que representan expresiones matemáticas de manera organizada. Estos árboles descomponen una expresión en operadores y operandos, siguiendo reglas de precedencia y asociatividad. Son útiles para evaluar expresiones y comprender su estructura.

Acciones Semánticas de un Analizador Sintáctico (3.2): Las acciones semánticas son tareas realizadas por el analizador sintáctico al analizar el código fuente. Estas acciones incluyen la construcción de árboles de sintaxis abstracta, resolución de nombres, comprobación de tipos y control de flujo. Se utilizan para verificar la coherencia y la semántica del programa.

Comprobaciones de Tipos en Expresiones (3.3): La comprobación de tipos es esencial para garantizar que las operaciones y asignaciones en un programa se realicen entre tipos de datos compatibles. Esto evita errores de tiempo de ejecución y garantiza la coherencia del programa. Involucra la inferencia de tipos, la coerción de tipos y reglas de conversión, y puede realizarse en tiempo de compilación o ejecución.

Conceptos Importantes:

3.1 Árboles de Expresiones:

Árbol de Expresiones: Una estructura jerárquica utilizada para representar expresiones matemáticas o aritméticas de manera organizada.

Operadores y Operandos: Los nodos del árbol representan operadores y operandos, y las ramas conectan estos nodos siguiendo las reglas de precedencia y asociatividad.

Jerarquía de Operaciones: Los operadores se organizan jerárquicamente en el árbol, reflejando la forma en que deben evaluarse las expresiones.

Evaluación de Expresiones: Los árboles de expresiones son útiles para evaluar expresiones de manera recursiva, siguiendo un orden específico de recorrido en el árbol.

3.2 Acciones Semánticas de un Analizador Sintáctico:

Analizador Sintáctico: Es una parte del proceso de compilación que verifica si la estructura del código fuente cumple con la gramática del lenguaje de programación.

Acciones Semánticas: Son tareas realizadas por el analizador sintáctico para garantizar la coherencia y semántica del programa.

Construcción de AST: Una acción semántica importante es la construcción del Árbol de Sintaxis Abstracta (AST), que captura la estructura y semántica del programa de manera más abstracta.

Resolución de Nombres: Las acciones semánticas ayudan a asociar nombres con sus declaraciones correspondientes y garantizar que las referencias sean coherentes.

Comprobación de Tipos: Verificar que las operaciones se realicen entre tipos de datos compatibles es una tarea crucial realizada por acciones semánticas.

3.3 Comprobaciones de Tipos en Expresiones:

Comprobación de Tipos: Es un proceso que garantiza que las operaciones y asignaciones en un programa se realicen entre tipos de datos compatibles, evitando errores de tiempo de ejecución.

Inferencia de Tipos: En algunos casos, se puede inferir el tipo de una expresión en función de los tipos de sus operandos.

Coerción de Tipos: Puede implicar la conversión implícita o explícita de tipos para hacer que sean compatibles en una operación.

Verificación Estática vs. Dinámica: Puede realizarse en tiempo de compilación (verificación estática) o en tiempo de ejecución (verificación dinámica).

Sobrecarga de Operadores: Algunos lenguajes permiten la sobrecarga de operadores, lo que requiere reglas específicas de conversión de tipos.

Ideas principales:

3.1 Árboles de Expresiones:

Los árboles de expresiones son estructuras jerárquicas que representan expresiones matemáticas de manera organizada.

Están compuestos por nodos que representan operadores y operandos, y las conexiones entre los nodos siguen las reglas de precedencia y asociatividad.

Los árboles de expresiones son útiles para evaluar expresiones de manera recursiva y siguiendo un orden específico en el árbol.

3.2 Acciones Semánticas de un Analizador Sintáctico:

Las acciones semánticas son tareas realizadas por el analizador sintáctico durante el análisis del código fuente para garantizar la coherencia y semántica del programa.

Estas acciones incluyen la construcción de Árboles de Sintaxis Abstracta (AST), resolución de nombres, comprobación de tipos, control de flujo y optimizaciones iniciales.

El analizador sintáctico es una parte esencial del proceso de compilación que verifica si el código fuente cumple con la gramática del lenguaje de programación.

3.3 Comprobaciones de Tipos en Expresiones:

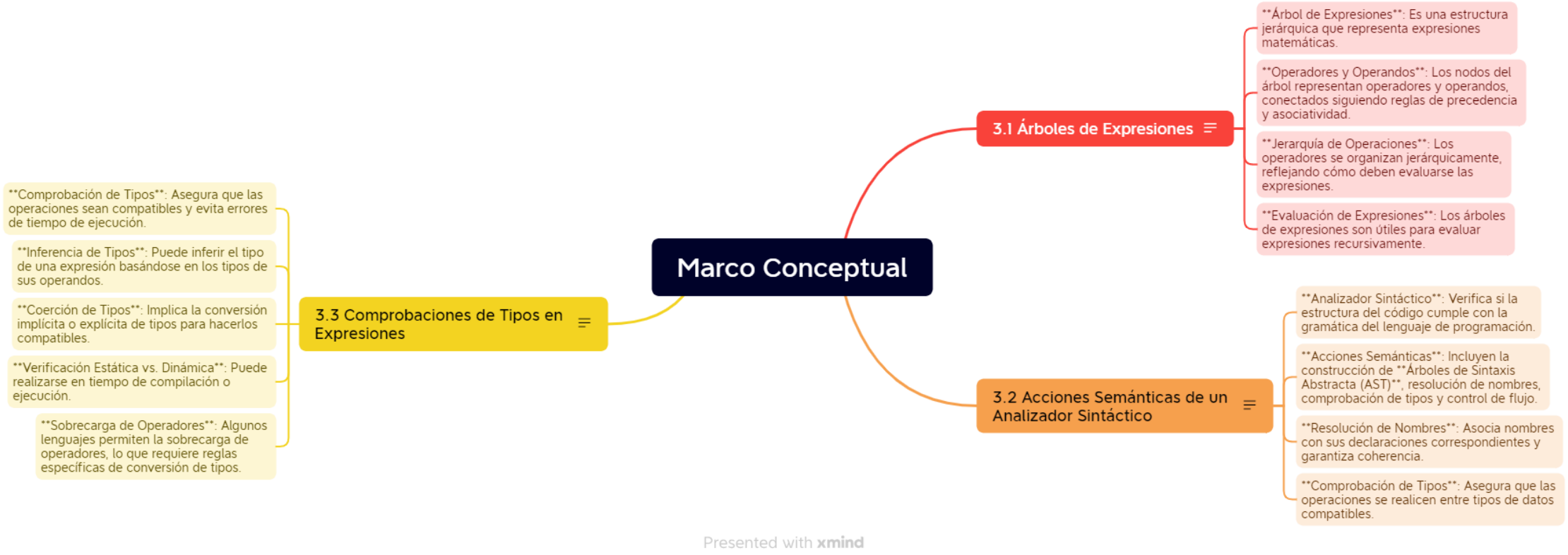
La comprobación de tipos es un proceso fundamental que asegura que las operaciones y asignaciones en un programa se realicen entre tipos de datos compatibles.

Puede implicar la inferencia de tipos, la coerción de tipos, y la verificación estática o dinámica, según el lenguaje de programación.

También se aplica en estructuras de control, como las sentencias condicionales, para garantizar que las expresiones sean de tipo correcto.

La comprobación de tipos es esencial para evitar errores de tiempo de ejecución y garantizar la coherencia del programa.

Esquema:


Marco Conceptual
3.1 Árboles de Expresiones
Árbol de Expresiones: Es una estructura jerárquica que representa expresiones matemáticas.
Operadores y Operandos: Los nodos del árbol representan operadores y operandos, conectados siguiendo reglas de precedencia y asociatividad.
Jerarquía de Operaciones: Los operadores se organizan jerárquicamente, reflejando cómo deben evaluarse las expresiones.
Evaluación de Expresiones: Los árboles de expresiones son útiles para evaluar expresiones recursivamente.
3.2 Acciones Semánticas de un Analizador Sintáctico
Analizador Sintáctico: Verifica si la estructura del código cumple con la gramática del lenguaje de programación.
Acciones Semánticas: Incluyen la construcción de **Árboles de Sintaxis Abstracta (AST)**, resolución de nombres, comprobación de tipos y control de flujo.
Resolución de Nombres: Asocia nombres con sus declaraciones correspondientes y garantiza coherencia.
Comprobación de Tipos: Asegura que las operaciones se realicen entre tipos de datos compatibles.
3.3 Comprobaciones de Tipos en Expresiones
Comprobación de Tipos: Asegura que las operaciones sean compatibles y evita errores de tiempo de ejecución.
Inferencia de Tipos: Puede inferir el tipo de una expresión basándose en los tipos de sus operandos.
Coerción de Tipos: Implica la conversión implícita o explícita de tipos para hacerlos compatibles.
Verificación Estática vs. Dinámica: Puede realizarse en tiempo de compilación o ejecución.
Sobrecarga de Operadores: Algunos lenguajes permiten la sobrecarga de operadores, lo que requiere reglas específicas de conversión de tipos.
Presented with xmind