Clase 2- semántica

sábado, 23 de marzo de 2024 22:38

Definición:

significado de los símbolos, palabras y frases de un lenguaje ya sea lenguaje natural o lenguaje informático que es sintácticamente válido

Estática → pre-ejecución → compilación

+ palabras con sintaxis (forma válida) → No significado

BNF no se mete con el contexto

Gramática con atributos (GyC)

- Las reglas sintácticas (producciones) son similares a BNF.
- Las reglas semánticas (ecuaciones) permiten detectar errores y obtener valores de atributos.
- Los atributos están directamente relacionados a los símbolos gramaticales (terminales y no terminales)
- Las GA se suelen expresar en forma tabular para obtener el valor del atributo

¿Cómo detectar errores de compatibilidad de Tipos (ej. C)?
¿Cómo detectar errores de declaración de variables duplicadas (ej. C)?
¿Como detectar errores de variables no declaradas antes de referenciarlas (ej. Python)?

Regla gramatical	Reglas semánticas (obtener el valor)
Regla 1	Ecuaciones de atributo asociadas
.	.
.	.
Regla n	Ecuaciones de atributo asociadas

15

Como funciona la gramática de atributos:

- mira la sintaxis de BNF/EBNF y busca atributos para **terminales y no terminales**, si encuentra el **atributo** deben llegar a tener su **valor**, para obtenerlo **genera ecuaciones**, **usa la tabla** donde machea si **encuentra la producción/regla** y **del otro lado están ecuaciones que me permiten llegar a los atributos.**

Regla gramatical	**Reglas semánticas**
decl → tipo lista-var	*lista-var.at = tipo.at*
tipo→ int	*tipo.at =* int
tipo→ float	*tipo.at =* float
lista-var→ id	*id.at = lista-var.at* *Añadetipo(id.entrada, lista-var.at)*
lista-var$_1$→ id, lista-var$_2$	*id.at = lista-var.at* *Añadetipo(id.entrada, lista-var$_1$.at)* *lista-var$_2$ .at = lista-var.at*

Similar a BNF pero no igual!
Cursiva No Terminal
Normal Terminal
-> se define como
, seguido
OR no existe, repetir fila

21

Dinámica → durante ejec.

→ Significado de lo q ejecutamos.

Semántica Dinámica - soluciones más utilizadas:

- **Formales y complejas:**
  - Semántica **axiomática**
  - Semántica **denotacional**

    La usan los diseñadores de compiladores
- **No formal:**
  - Semántica **operacional**

    La usan para manuales de lenguajes

Semántica Axiomática

- **Considera al programa** como "**una máquina de estados**" donde cada **instrucción** provoca un **cambio de estado**.
- Se parte de un **axioma (verdad)** que sirve **para verificar "estados y condiciones" a probar**
- **Los constructores de un lenguaje de programación se formalizan describiendo como su ejecución provoca un cambio de estado (cada vez que se ejecuta).**
- Se desarrolló para **probar la corrección de los programas**.
- La **notación** empleada es el "**cálculo de predicados**"

28

SEMÁNTICA

Semántica Denotacional

- Define una **correspondencia** entre los **constructores sintácticos** y sus **significados**

- Un **estado se describe con un predicado**
- **El predicado describe los valores de las variables en ese estado**

- Define una **correspondencia** entre los **constructores sintácticos** y sus **significados**
- Describe la **dependencia funcional** entre el **resultado** de la **ejecución** y sus **datos iniciales**
- **Lo que hace es buscar funciones que se aproximen a las producciones sintácticas.**

31

- **Veamos un ejemplo.....**

- **Un estado se describe con un predicado**
- **El predicado describe los valores de las variables en ese estado**
- Existe un **estado anterior** y un **estado posterior** a la ejecución del constructor.
- Cada sentencia se precede y se continúa con una ***expresión lógica*** que ***describe*** las ***restricciones y relaciones*** entre los datos.
  - **Precondición (condiciones de estado previo)**
  - **Poscondición (condiciones de estado posterior)**

Ejemplo: a/b a b
r c

Precondición: {b distinto de cero}
Sentencia: expresión que divide a por b
Postcondición: {a=b*c+r y r<b}

29

Semántica Operacional

- El **significado** de un **programa** se describe **mediante otro lenguaje** de **bajo nivel** implementado **sobre** una **máquina abstracta**
- **Cuando se ejecuta una sentencia del lenguaje de programación los cambios de estado de la máquina abstracta definen su significado**
- Es un **método informal** porque **se basa en otro lenguaje de bajo nivel y puede llevar a errores**
- Es el **más utilizado** en los **libros** de texto para **explicar el significado de los lenguajes**
- **PL/1** fue el primero que lo utilizó

→ Un otro lenguaje p. explicar.

Semántica Operacional

Ejemplo: en Pascal

Lenguajes	Máquina abstracta
for i := pri to ul do	*i := pri (inicializo i)*
begin	*lazo if i > ul goto sal*
.....	.....
end	*i := i +1*
	goto lazo
	sal

Procesamiento de lenguaje(traducción)
Computadoras ejecutan lenguaje de maquina, pero es un bajón codear con 0s y 1s.
Entonces se usaba código mnemotécnico sum()

Lenguaje ensamblador--> programa ensamblador
Necesito un programa que me lo traduzca a lenguaje de maquina

Intérprete y Compilador

Interpretación--> escribo lenguaje interpretado, interprete traduce.
Se hace cuando yo ejecuto. Lo va haciendo a medida que ejecuto.

Hay partes del programa ser interpretadas.

- El proceso se realiza cuando se **ejecuta**
  - **Lee**
  - **Analiza**
  - **Decodifica** y
  - **Ejecuta** una a una las sentencias de un programa
- Sólo pasa por ciertas instrucciones según sea la ejecución (ventajas y desventajas)

Tiene subprogreamas. Por cada acción hay un subprograma en lenguaje de máquina que la ejecuta.
La interpretación se hace llamando a esos suprogramas.

E → intérprete → S
Datos

- Un intérprete ejecuta **repetidamente** la siguiente secuencia de acciones:
  - **Obtiene** la próxima sentencia
  - **Determina** la acción a ejecutar
  - **Ejecuta** la acción

Compilación--> Antes de ser ejecutado. Pasa por todas las instrucciones. El código generado se guarda y se puede reusar ya compilado

lenguaje fuente → lenguaje objeto.

(ejecuta blp lenguaje intermedio (ensamblador)

lenguaje fuente → compilación → lenguaje objeto. lenguaje intermedio (bytecode)

Compilador VS intérprete.

Intérprete: ejecuta en ejecucución
Compi:lo hace antes de ejecutar

Intérprete:sigue orden del programa
Compi:ve todo

Intérprete: cada vez que pasa por un loop lo revisa, si tiene muchas iteraciones es medio un bajón
Compi:una sola vez. Capaz tarda más en analizar todo pero una vez hecho ya está } velocidad.

Intérprete: más lento, - eficiente
Compi: más rápido porque ya fue compilado } eficiencia

Intérprete: ocupa menos espacio
Compilador: pasa por todas las sentencias y escribe en código de máquina, se hacen muchas tablas que capaz no se van a usar. } espacio.

Intérprete: errores se ven directamente, se detectan los errores por donde pasa la ejecución
Compi: referencia al código fuente se pierde en el cod objeto, es mucho más difícil identificar errores } identificación de errores.

Primero interpreto y después compilo → Combinan está bueno pq veo errores + fácil.

Primero compilo y después interpreto → Traduce a cod intermedio q' después interpreto. código portable.
→ Como Java, me llevo programa → se interpreta en cualquier.

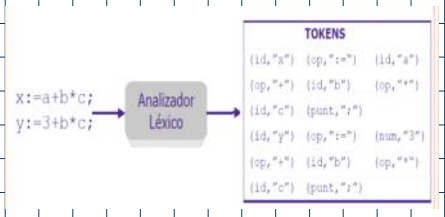
Compilar hace q' corra + rápido pq ya pasó a un lenguaje + cercano al de la máquina donde.

- **Etapa de Análisis**
  - **Análisis léxico (Programa Scanner)**
  - **Análisis sintáctico (Programa Parser)**
  - **Análisis semántico (Semántica estática)**
- **Etapa de Síntesis**
  - **Optimización del código**
  - **Generación del código**

Debe haber quedado todo en el medio

llevan + tiempo → toman cada palabra, ve q' es y clasifican

- Genera **errores** si la entrada no coincide con ninguna categoría léxica
- **Convierte** a representación interna los **números** en punto fijo o punto flotante
- Pone los **identificadores** en la **tabla de símbolos**
- Reemplaza cada **símbolo** por su entrada en la **tabla de símbolos**
- El resultado de este paso será el descubrimiento de los items léxicos o tokens.



parser → verificar si el programa está bien.

El **analizador sintáctico** se alterna con el **análisis semántico.** Usualmente se utilizan técnicas basadas en **gramáticas formales**.

Se aplica una **gramática** para **construir** el **árbol sintáctico del programa.** Toma una sentencia y después **compara con los árboles derivación** para ver si lo que entra es correcto

51

Semántica estática → de lo + importante. Significado pero antes de ejecutar

Semántica estática → de lo + importante.
→ Significado pero antes de ejecutar
→ comprobación de tipos.

Variables no declaradas, repetidas, etc.

- Se **agrega** la **información implícita (variables no declaradas)**
- Se **agrega a** la **tabla** de **símbolos** los **descriptores de tipos**, etc. a la vez que se hacen consultas para realizar **comprobaciones (duplicados, problema de tipos, etc.)**
- Se hacen las **comprobaciones de nombres**. Ej: toda variable debe estar declarada en su entorno.
- Es el **nexo** entre **etapas de análisis** y la **síntesis**

53

Nexo el análisis y síntesis.

Generación de código intermedio:

- Es la **transformación** del **código fuente** en una representación de **código intermedio para una máquina abstracta.**
- A esta **representación intermedia,** que se parece al código objeto pero que sigue siendo **independiente de la máquina**, se le llama **código intermedio.**

- Fácil de producir
- Fácil de traducir
- Punto medio
- Código de 3 direcciones x = y op z

Síntesis

- **Se construye** el **programa ejecutable**.
- Se **genera** el **código** necesario y se **optimiza** el **programa** generado.
- Si hay **traducción separada** de **módulos**, es en esta etapa cuando se **enlaza**. (módulos, units, librerías, procedimientos, funciones, subrutinas, macros, etc.) **(Programa Linkeditor)**
- Se realiza el proceso de **optimización**. **Optativo**

58

→ llama a módulos externos y los enlaza.

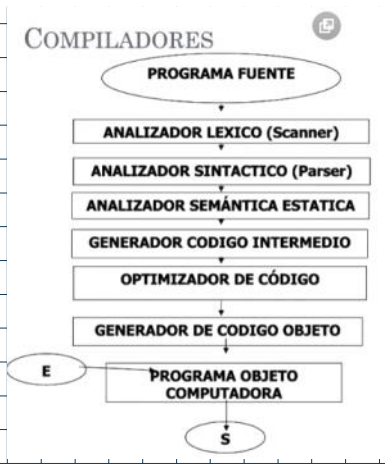
→ (puede optimizarse o no)

Optimización → Puede ser independiente o parte de los compiladores.

- elegir entre **velocidad de ejecución y tamaño del código ejecutable.**
- generar código **para un microprocesador** específico dentro de una familia de microprocesadores,
- eliminar la **comprobación de rangos o desbordamientos de pila**
- evaluación para **expresiones booleanas,**
- **eliminación de código muerto** o no utilizado,
- **eliminación de funciones no utilizadas**
- Etc...

59

En resumen...


COMPILADORES
PROGRAMA FUENTE
ANALIZADOR LEXICO (Scanner)
ANALIZADOR SINTACTICO (Parser)
ANALIZADOR SEMÁNTICA ESTATICA
GENERADOR CODIGO INTERMEDIO
OPTIMIZADOR DE CÓDIGO
GENERADOR DE CODIGO OBJETO
E
PROGRAMA OBJETO COMPUTADORA
S