Clase 2

SEMÁNTICA

La semántica describe el significado de los símbolos, palabras y frases de un lenguaje ya sea lenguaje natural o lenguaje informático que es sintácticamente válido Para luego poder darle significado a una construcción del lenguaje

Tipos de semántica que analizaremos:

Semántica estática

* No está relacionada con el significado de la ejecución del programa, está más relacionado con las formas válidas.
* El análisis está ubicado entre el análisis sintáctico y el análisis de semántica dinámica, pero más cercano a la sintaxis.
* Se las llama así porque el análisis para el chequeo se hace en compilación (antes de la ejecución).
* BNF/EBNF no sirve para esto. Estas son gramáticas libres de contexto no se meten con el significado si con las formas)
* Gramática de atributos
  + A las construcciones del lenguaje se les asocia información a través de “atributos” asociados a los símbolos (terminales o no terminales) de la gramática
  + Un atributo puede ser el valor de una variable, el tipo de una variable o expresión, lugar que ocupa una variable en la memoria, dígitos significativos de un número, etc.
  + Los valores de los atributos se obtienen mediante las llamadas “ecuaciones o reglas semánticas” asociadas a las producciones gramaticales.
  + Las reglas sintácticas (producciones) son similares a BNF.
  + Las ecuaciones (reglas semánticas) 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

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

* + ¿Cómo funciona?
    - Usa la tabla y machea si encuentra la producción/regla y del otro lado la ecuación que me permite llegar a los atributos.
    - Mira las Reglas (símil BNF/EBNF) y busca atributos para terminales y no terminales
    - Si encuentra el atributo debe llegar a obtener su valor.
    - Para obtenerlo genera ecuaciones.
  + De la ejecución de las ecuaciones:
    - se ingresan símbolos a la tabla de símbolos,
    - Detectar y dar mensajes de error
    - detecta dos variables iguales,
    - controla tipo y variables de igual tipo,
    - ciertas combinaciones no permitidas (reglas específicas del lenguaje)
    - otras cosas no permitidas por el lenguaje
    - Es decir, el chequeo de semántica estática
    - Generar un código para el siguiente paso

Semántica dinámica

* Es la que describe el significado de ejecutar las diferentes construcciones del lenguaje de programación.
* Su efecto se ve durante la ejecución del programa.
* Influirá la interacción con el usuario y errores de la programación
* Recordemos: Los programas sólo se pueden ejecutar si son correctos en la sintáxis y la semántica estática.
* ¿Cómo se describe la semántica estática?
  + No es fácil escribirla
  + No existen herramientas estándar
  + Es complejo describir relación entre entrada y salida del programa
  + Es complejo describir cómo se ejecutará en cierta plataforma.
  + Soluciones más utilizadas:
    - Formales y complejas:
      * Semántica axiomática: Considera al programa como “una máquina de estados” donde cada instrucción provoca un cambio de estado. Parte de un axioma (verdad) que sirve para verificar "estados y condiciones" a probar. Describe estados a través de los PREDICADOS (con variables de estado). Existe un estado anterior y un estado posterior a la ejecución del constructor.
      * Semántica denotacional: Se basa en la teoría de funciones recursivas y modelos matemáticos. Describe el estado a través de FUNCIONES (funciones recursivas). Define una correspondencia entre los constructores sintácticos y sus significados. Describe la dependencia funcional entre el resultado de la ejecución y sus datos iniciales
    - No formal:
      * Semántica operacional: El significado de un programa se describe mediante otro lenguaje de bajo nivel implementado sobre una máquina abstracta. Los cambios que se producen en el estado de la máquina abstracta, cuando se ejecuta una sentencia del lenguaje de programación, definen su significado

Texto

Descripción generada automáticamente

* + Sirven para comprobar la ejecución, la exactitud de un lenguaje, comparar funcionalidades de distintos programas.
  + Se pueden usar combinados, no sirven todos para todos los tipos de lenguajes de programación

**¿Cómo los programas escritos en lenguajes de alto nivel pueden ser ejecutados sobre una computadora cuyo lenguaje es muy diferente y de muy bajo nivel que entiende 0s y 1s?**

Con Programas Traductores del lenguaje Alternativas (cual se usa es decisión del que crea el lenguaje):

* Interpretación.
* Compilación.
* Interpretación y Compilación (combinación)

INTERPRETACIÓN

Hay un Programa escrito en lenguaje de programación interpretado. Hay un Programa llamado Intérprete que realiza la traducción de ese lenguaje interpretado en el momento de ejecución

El proceso que realiza cuando se ejecuta sobre cada una de las sentencias del programa es:

1. Leer
2. Analizar
3. Decodificar
4. Ejecutar una a una las sentencias de programa (vuelvo a 1.)

Sólo pasa por ciertas instrucciones no por todas

En el Intérprete por cada posible acción hay un subprograma en lenguaje de máquina que ejecuta esa acción. La interpretación se realiza llamando a estos subprogramas en la secuencia adecuada hasta generar el resultado de la ejecución

Interfaz de usuario gráfica

Descripción generada automáticamente

COMPILADOR

El compilador es un programa que traduce nuestro programa previo a la ejecución. Prepara el programa en el lenguaje máquina para que luego pueda ser ejecutado. El compilador toma todo el programa escrito en un lenguaje de alto nivel que llamamos lenguaje fuente antes de su ejecución. Luego de la compilación va a generar un lenguaje objeto que es generalmente el ejecutable (escrito en lenguaje de máquina .exe) o un lenguaje de nivel intermedio (o lenguaje ensamblador .obj).

¿Cómo funciona un compilador?

Los compiladores pueden ejecutarse en un solo paso o en dos. Dichos pasos cumplen con varias etapas, principalmente se pueden dividir en dos (1) más vinculado al código fuente 2) más vinculado a características del código objeto y del hardware y arquitectura):

* Análisis
  + Análisis léxico (Scanner): Hay un análisis a nivel de palabra (LEXEMA).
    - Se fija que cada elemento del programa se corresponda con un operador, un identificador, etc., básicamente divide el programa en sus elementos: identificadores, delimitadores, símbolos especiales, números, palabras clave, palabras reservadas, comentarios, etc. De esta forma ve si todas las palabras son válidas o no.
    - Es el que lleva más tiempo.
    - Filtra comentarios y separadores como: espacios en blanco, tabulaciones, etc.
    - Genera errores si la entrada no coincide con ninguna categoría léxica.
    - Con esto descubrimos los tokens
    - 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.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

* + Análisis semántico (Parser):
    - Se realiza luego de análisis léxico.
    - Se analiza a nivel de cada sentencia.
    - Busca estructuras, sentencia, declaración, expresiones, variable, ayudándose de los tokens.
    - Se alterna el analizador sintáctico con el análisis semántico.
    - Construye el árbol sintáctico del programa.
  + Análisis semántico (semántica estática):
    - Es la fase más importante.
    - Todas las estructuras sintácticas reconocidas son analizadas.
    - Se realiza una comprobación de tipos y se agrega información implícita (variables no declaradas).
    - Se agregan tablas de símbolos de los descriptores de tipos, etc.
    - Se hace comprobaciones de nombres y duplicados.
    - Nexo entre etapas inicial y final del compilador

Luego de esta etapa se puede realizarla la transformación del código fuente en una representación de código intermedio. Esta representación intermedia llamada código intermedio se parece al código objeto, pero sigue siendo independiente de la máquina. El código objeto es dependiente de la máquina. Este código intermedio debe ser fácil de producir y debe ser fácil de traducir al programa objeto. Hay varias técnicas, la más común es el código de tres dimensiones

* Síntesis: Se construye el programa ejecutable, genera el código necesario y se optimiza el programa generado. Si hay traducción separada de módulos se enlazan los distintos módulos objeto del programa (módulos, unidades, librerías, procedimientos, funciones, subrutinas, macros, etc.) mediante el linkeditor
  + Se genera el módulo de carga. Programa objeto completo.
  + Se realiza el proceso de optimización. (Optativo)

La semántica interviene en el análisis y es de sumamente importancia ya que es el nexo entre las etapas inicial y final del compilador

Captura de pantalla de computadora

Descripción generada automáticamente

El intérprete y el compilador se puede comparar de diferentes formas

* En como ejecuta
  + Interprete: Lo hace durante la ejecución. Ejecuta sentencia a sentencia, lo traduce y lo ejecuta. Pasa un montón de veces. Para ser ejecutado en otra máquina se necesita tener si o si el intérprete instalado.
  + Compilador: Ocurre antes de ejecutar. Compila todo y genera un código objeto de un lenguaje de un modelo más bajo. El programa fuente no será publico
* Orden que lo ejecuta
  + Interprete: Sigue el orden lógico de ejecución (ya que va sentencia por sentencia del código)
  + Compilador: Sigue el orden físico de las sentencias.
* Tiempo de ejecución
  + Interprete: Por cada sentencia se realiza el proceso de decodificación para determinas las operaciones a ejecutar y sus operandos. Si la sentencia está en un proceso iterativo, se realiza la tarea tantas veces como sea requerido. Puede afectar la velocidad de proceso.
  + Compilador: Genera código de máquina para cada sentencia. No repite lazos, se decodifica una sola vez.
* Eficiencia
  + Interprete: Mas lento en ejecución. Se repite el proceso cada vez que se ejecuta el programa.
  + Compilador: Mas rápido desde el punto de vista del hardware, pero tarda más en compilar. Detectó más errores al pasar por todas las sentencias. Está listo para ser ejecutado. Ya compilado es más eficiente.
* Espacio ocupado
  + Interprete: Ocupa menos espacio de memoria ya que cada sentencia se deja en la forma original y las instrucciones necesarias para ejecutarlas se almacenan en los subprogramas del interprete en memoria.
  + Compilador: Una sentencia puede ocupar decenas o centenas de sentencias de máquina al pasar a código objeto
* Detección de errores
  + Interprete: Las sentencias del código fuente puede ser relacionadas directamente con la que se está ejecutando. Se puede ubicar el error, es más fácil detectarlos por donde pasa la ejecución y es más fácil corregirlos
  + Compilador: Le cuesta más determinar los errores ya que cualquier referencia al código fuente se pierden en el código objeto. Se pierde la referencia entre el código fuente y el código objeto. Es casi imposible ubicar el error, pobres en significado para el programador.

Combinaciones técnicas:

1. Primero Interpreto y luego Compilo
   * Se utiliza el intérprete en la etapa de desarrollo para facilitar el diagnóstico de errores.
   * Con el programa validado se compila para generar un código objeto más eficiente.
2. Primero Compilo y luego Interpreto
   * Se hace traducción a un código intermedio a bajo nivel que luego se interpretará
   * Sirve para generar código portable, es decir, código fácil de transferir a diferentes máquinas y con diferentes arquitecturas.