Clase 2

Semántica

Describe el significado de los símbolos, palabras y frases de un lenguaje.
Hay dos tipos:

- Semántica estática Relacionado con la sintaxis. Se realiza el análisis durante la compilación por ello se llama estática. Para describirla se usa la *gramática de atributos* ya que BNF u otra cosa no cumbre ciertos aspectos necesarios. Resuelve la gramática sensible al contexto.
    - Gramática de atributos: A las construcciones del lenguaje se le asocia información a través de los llamados "atributos "asociados a los símbolos gramaticales correspondiente. Los valores de los atributos se calculan mediante llamadas "ecuaciones o reglas semánticas". Lo atributos están directamente relacionados con los símbolos gramaticales (termínales y no terminales). La forma, generalmente de expresar la gramática con atributos se escribe de forma tabular.
      La evaluación de las reglas semánticas puede generar código, insertar información en la tabla de símbolos, realizar el chequeo semántico, car mensajes de error, etc.
      Ejemplo:

Regla gramatical
decl→ tipo lista-var
tipo→ int
tipo→ float
lista-var→ id
lista-var1→ id, lista-var2

      Con cursiva se escribe los nodos no terminales, los demás son nodos terminales. Para definir se una "→". Para tener opcional tenemos entradas duplicadas y la "," se usa para decir seguido de.
- Semántica dinámica: Se aplica luego de comprobar que la sintaxis y que la semántica estática sea correcta. La semántica dinámica aparece durante la ejecución de un programa. Describe el efecto de ejecutar las diferentes construcciones en un lenguaje de programación, ósea, describe el proceso que una computadora sigue cuando se ejecuta un programa en un lenguaje en específico y su efecto se describe durante la ejecución.
  ¿Cómo se describe? No es tan fácil describirla como en el caso de la sintaxis, es más compleja. Alguna de las soluciones formales compleja y se usan en diferentes momentos (mucha matemática y complejo las primeras dos, usada normalmente por los compiladores):
    - (compleja)Semántica axiomática: Trabaja con axiomas, ve al programa como "una máquina de estados". Un estado se describe como un predica que 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 continua con una expresión lógica que describe las restricciones y relación entre los datos.
    - (compleja)Semántica denotaciones: Se basa en funciones recursivas y modelos matemáticos. Basada en la gramática sintética. Se busca encontrar

una función que pueda escribir la producción. Se define una correspondencia entre los constructores sintéticos y sus significados.

- Semántica operacional (lenguaje más acotado de bajo nivel-informal): El significado de un programa se describe mediante otro lenguaje de bajo nivel implementado sobre una maquina abstracta. Usa los cambios de estados para darle un significado. Es más simple, más fácil de entender, pero puede llegar a ser ambiguo, menos exacto.

Lenguaje maquina

Antes se programaba con 1 y 0. Después nació el "Lenguaje Ensamblado" el cual al momento de compilarlo se transforma en 1 y 0. El lenguaje ensamblado es un lenguaje nemotécnico, el cual me permitía programar de una forma más simple. Tenía el problema de que cada máquina tenía su propia instrucción y no había compatibilidad. Para ello nace el "lenguaje de alto nivel".

¿Cómo se traduce un programa?

- Interpretación: Programa que interpreta p traducir nuestro programa, realizado en el momento de ejecutar. Realiza una tarea repetitiva la cual consiste en: *obtener* la próxima sentencia, *determinar* la acción a ejecutar y *ejecutar* la acción. Por cada acción hay un subprograma en lenguaje máquina que realiza la acción. El intérprete ira llamando a los diferentes subprogramas en la secuencia indicada. (traduce en el momento.)
- Compilación: Al igual que el intérprete el compilador es un programa que traduce nuestro programa, pero el mismo se realiza previo a la ejecución. Prepara el programa en el lenguaje máquina para que luego pueda ser ejecutado.
  ¿Cómo funciona?
  Los compiladores pueden ejecutarse en un solo paso o en dos. Dichos pasos cumplen con varias etapas, principalmente se pueden dividir en dos:
  - 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. De esta forma verifica todas las palabras, sino que también se ve sin son válidas o no. Con esto descubrimos los tokens. Es el que lleva más tiempo.
    - 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, se agrega información implícita (variables no declaradas), se agregan tablas de símbolos de los descriptores de tipos, etc. a la vez que se hacen consultas para realizar comprobaciones. Se hace comprobaciones de nombres.
  - Síntesis: Se construye el programa ejecutable, genera el código necesario y se optimiza el programa generado. Si hay traducción separada de módulos, en esta etapa es cuando se linkedita (en enlazan)
    - Optimización del código (opcional)
    - Generación del código:

Entre en análisis y la síntesis se puede generar un código intermedio. Para acelerar el código se pueden aplicar varios métodos. Se debe poder producir fácil con estas técnicas y traducir de una manera fácil. Se toma el código, se analiza y lo pasa a un código intermedio. Para aplicarlo hay varias técnicas Ejemplo es el código de tres dimensiones.

- Compilación e interpretación (se usa mucho)
  Se puede combinar de dos maneras
    - Primero interpreto y lego compilo: Utilizan el intérprete en la etapa de desarrollo, facilitando el diagnostico de errores y luego que el programa ha sido validado se compila para generar código más eficiente.
    - Primero compilo y luego interpreto: Sirve para generar código portable, es decir, código fácil de trasferir a diferentes maquinas. Traducción a un código intermediario que luego se interpretara.

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.
    - Compilador: Ocurre antes de ejecutar. Compila todo y genera un código objeto de un lenguaje de un modelo más bajo.
- 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 de realiza el proceso de decodificación para determinas las operaciones a ejecutar y sus operandos. Si la sentencia esta 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.
    - Compilador: Mas rápido desde el punto de vista del hardware, pero tarda más en compilar.
- 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 0 de sentencias de máquina.
- Detección de errores
    - Interprete: Las sentencias del código fuente puede ser relacionadas directamente con la que se está ejecutando.
    - Compilador: Le cuesta más determinar los errores ya que cualquier referencia al código fuente se pierden en el código objeto.