Clase2-Semantica

Semantica: 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. Obivamente los lenguajes tienen distintas semánticas para definir un mismo concepto. Además se debe tener en cuenta y detectar otros errores semántica.

-Tipos de semántica:

-Estatica (antes de la ejecución): se la denomina asi porque el análisis para el chequeo se hace en compilación, no esta relacionada con el significado de la ejecución del programa, está más relacionado con las formas válidas (con la sintaxis).

-Debido a que BNF/EBNF son gramáticas libres de contexto no nos permiten detectar errores propios del lenguaje de programación.

Estatica – Gramatica de atributos: Son gramáticas sensibles al contexto (GSC). Si se relacionan con el significado. La usan los compiladores, antes de la ejecución y generalmente resuelven los aspectos de la semántica estática.

Estatica - Funcionamiento de la Gramática de atributos: A las construcciones del lenguaje se les asocia información a través de “atributos” asociados a los símbolos de la gramática (terminales o no terminales), que sirven para detectar errores

-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 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.

Funcionamiento: 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 y busca atributos para terminales y no terminales, si encuentra el atributo debe llegar a obtener su valor y para obtenerlo genera ecuaciones.

No logro entender el arbol de la pag 17.

En definitiva, se debe tratar de representar todo lo que necesitemos que salte antes de la ejecución y no sea detectado sintácticamente.

.Dinamica (durante la ejecución): describe el significado de ejecutar las diferentes construcciones del lenguaje de programacion, su efecto se ve durante la ejecucion del programa esta influira en la interaccion con el usuario y errores de la programacion.

-Los programas solo se pueden ejecutar si son correctos en la sintaxis y la semantica estatica. Luego viene la semantica dinamica.

-Es complejo describir la semantica dinamica dado que no existen herramientas estandar como en el caso de la sintaxis (diagramas sintacticos y BNF).

Soluciones mas utilizadas en la semantica dinamica:

Formales y complejos: semantica axiomatica y denotacional (la usan los diseñadores de compiladores).

No formal: semantica operacional (la usan para manuales de lenguajes)

Estas sirven para comprobar la ejecucion, la exactitud de un lenguaje y comparar funcionalidades de distintos programas. Estas se pueden usar combinadas, pero no todas sirven para todos los tipos de lenguajes.

Semantica Axiomatica: Esta considera al programa como “una maquina de estados” donde cada instrucción provoca un cambio de estado. Se parte de un axioma que sirve para verificar “estados y condiciones” a probar. Se desarrollo para probar la correcion de los programas.

-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 ejecucion del constructor. Cada sentencia se precede y se continua con una expresion logica que describe las restricciones y relaciones entre los datos.

Semantica denotacional: Esta se basa en la teoria de funciones recursivas y modelos matematicos, es mas exacto para obtener y verificar resultados, pero es mas dificil de leer. Se diferencia de la axiomatica por la forma que describe los estados ya que la axiomatica los describe a traves de los predicados y la denotacional los describe a traves de funciones.

-Define una correspondencia entre los constructores sintacticos y sus significados y describe la dependencia funcional entre el resultado de la ejecucion y sus datos iniciales.

Ejemplo en la pag 27.

Semantica operacional: Utiliza un lenguaje de bajo nivel implementado sobre una maquina abstracta para describir el significado de un programa. Cuando se ejecuta una sentencia del lenguaje los cambios de estado de la maquina abstracta definen su significado. El uso de un lenguaje de bajo nivel puede llevar a errores.

Procesamiento de un programa:

-Las computadoras ejecutan un lenguaje de bajo nivel llamado “lenguaje de maquina” (con 0’s y 1’s).

Historia.

-Los programas traductores traducen los lenguajes de alto nivel a los lenguajes de bajo nivel. Las alternativas son interpretacion, compilacion y la combinacion de estas dos, esto no es decision del programador sino del creador del lenguaje.

Interpretacion: Existe un programa que esta escrito en lenguaje de programacion interpretado y hay un programa llamado interprete que realiza la traduccion del lenguaje interpretado en el momento de ejecucion.

-El proceso que realiza cuando se ejecuta sobre cada una de la sentencias del programa es: leer – analizar – decodificar – ejecutar (una a una las sentencias del programa). Solo pasa por ciertas instrucciones no por todas y cada vez que vuelvo a ejecutar el programa se repite toda la secuencia.

-El interprete cuenta con una serie de herramientas para la traduccion a lenguaje a maquina. Por cada posible accion hay un subprograma en lenguaje de maquina que ejecuta esa accion. La interpretacion se realiza llamando a estos subprogramas en la secuencia adecuada hasta generar el resultado de la ejecucion. El interprete ejecuta rapidamente la siguiente secuencia de acciones:

Obtiene la proxima sentencia – Determina la accion a ejecutar – Ejecuta la accion.

Compilacion: El programa llamado compilador realiza la traduccion del lenguaje de alto nivel a lenguaje maquina. Se compila antes de ejecucion, este pasa por todas las instrucciones antes de la ejecucion y el codigo generado se guarda y se puede reusar ya compilado. La compilacion implica varias etapas.

Traduccion: Comparacion interprete – compilador (pag 44).

Combinacion de tecnicas de traduccion: La interpretacion y compilacion pura son dos exteremos, muchos lenguajes combinan ambas tecnicas para sacar provecho a cada una. Los compiladores e interpretes se diferencian en como reportan los errores de ejecucion.

1. Primero interpreto y luego compilo: Utiliza el interprete en la etapa de desarrollo para facilitar el diagnositco de errores y con el programa validado se compila para generar un codigo objeto mas eficiente.

2. Primero compilo y luego interpreto: Se hace traduccion a un codigo intermedio a bajo nivel que luego se interpretara. Sirve para generar codigo portable, es decir, codigo facil de transferir a diferentes maquinas y con diferntes arquietecturas.

Compiladores – Funcionamiento: Traducen todo el programa, estos pueden generar un “codigo ejecutable” o un “codigo intermedio”. La compilacion puede ejecutarse en 1 o 2 etapas, en ambos casos se cumplen varias sub-etapas, las principales son:

1. Etapa de analisis: Analisis lexico (programa scanner), sintactico (programa parser) y semantico (programa de semantica estatica), en esta estapa puede generarse codigo intermedio y optimizarse este codigo.

2. Etapa de sintesis: Se realiza la optimizacion del codigo y la generacion del codigo final (maquina).

-La etapa 1 esta mas viculado al codigo fuente y la 2 mas vinculado a caracteristicas del codigo objetos y del hw y arquitectura.

1. Etapa de analisis del programa fuente:

Analisis lexico (scanner): lleva una tabla para la especificacion del analizador lexico. Incluye cada categoria, el conjunto de atributos y acciones asociadas. Pone los identificadores en la tabla de simbolos y reemplaza cada simbolo por su entrada en la tabla de simbolos. Genera errores si la entrada no coincide con ninguna categoria lexica. Su resultado sera el descubrimiento de los items lexicos o tokens y deteccion de errores.

Analisis sintactico (parser): el analisis se realiza a nivel de sentencia/estructura, este usa los tokens del analizador lexico, tiene como objetivo encontrar las estructuras presentesn en su entrada.

Estas estructuras se pueden representar mediante el arbol de analisis sintactico, que explica como se puede derivar la cadena de entrada en la gramatica que especifica el lenguaje.

El analizador sintactico(parser) se alterna/interactua con el analisis lexico y analisis semantico. Se usa una gramatica para construir el “arbol sintactico” del programa. El objetivo principal del arbol es representar una sentencia del lenguaje y validar de esta forma que pertenece o no a la gramatica para ver que lo que entra es correcto.

Analsis semantica (semantica estatica): debe pasar bien scanner y parser, esta es una fase de las mas importantes. Procesa las estructuras sintacticas, agrega otro tipo de info implicita y la estructura del codigo ejecutable continua tomando forma, ademas realiza la comprobacion de tipos, agrega a la tabla de simbolos los descriptores de tipos, realiza comprobaciones de duplicados, problemas de tipo, etc. Realiza comprobaciones de nombres (toda var debe estar declarada en su entorno). Este funciona como nexo entre etapas inicial y final del compilador.

Generacion de codigo intermedio: se transforma el codigo fuente a codigo intermedio para una maquina abstracta, queda una representacion independiente en la que se va a ejecutar el programa, para realizar esto hay distintas tecnicas, hay que recordar que el codigo objeto es dependiente de la maquina.

-El codigo intermedio mas habitual es el codigo de 3-direcciones. Pasa todo el codigo del programa a secuencia de proposiciones de la forma: x := y op z.

2. Etapa de sintesis: Parte casi final, esta construye el programa ejecutable y genera el codigo necesario, si hay traduccion separada de otros modulos interviene el linkeditor (programa) y se enlazan los distintos modulos objeto del programa. Se genera un modulo de carga (programa objeto completo), se realiza el proceso de optimizacion (optativo) y el cargador loader (programa) lo carga en memoria.

Optimizacion: no se hace siempre y no lo hacen todos los compiladores. Los optimizadores de codigo (programas) pueden ser herramientas independientes, o estar incluidas en los compiladores e invocarse por medio de opciones de compilacion.

Hay diversas formas y cosas a optimizar: elegir la velocidad de ejecucion y tamaño del codigo ejecutable, generar coedigo para un microprocesador especifico dentro de una familia de microprocesadores, eliminacion de codigo muerto o no utilizado, etc.