Universidad de Huelva

Universidad de Huelva

Grado en Ingeniería Informática

Tema 5. Análisis Semántico

Resumen

Autor: Alberto Fernández Merchán
Asignatura: Procesadores del Lenguaje

1. Características del Análisis Semántico

Un analizador semántico tiene como **objetivos**:

- Verificar la semántica de la cadena de entrada.
- Comprobación de tipos en las expresiones.
- Comprobación de los parámetros de una función.
- Elección de la función u operador en caso de sobrecarga o polimorfismo.
- Emitir informes de errores semánticos.
- Construir el Árbol de Sintaxis Abstracta.

1.1. Árbol de Sintaxis Abstracta

El árbol de sintaxis abstracta:

- Recoge toda la información sintáctica y semántica.
- Elimina los símbolos que no aportan significado.
- Los nodos reflejan los datos asociados a cada símbolo.
- El objetivo de las acciones semánticas es construir este árbol.

2. Gramáticas Atribuidas

Las gramáticas atribuidas son **gramáticas libres de contexto** ampliadas con **atributos**, elementos de los símbolos del lenguaje que almacenan información semántica, y **acciones semánticas**, expresiones que permiten calcular el valor de los atributos.
Existen dos tipos de atributos:

- **Atributos Sintetizados**: Su valor se calcula a partir de la información de los nodos hijos. Representa un flujo de información ascendente en el AST. (Se calcula **después** de reconocer sintácticamente el símbolo).
- **Atributos Heredados**: Su valor se calcula a partir de la información del nodo padre o de los hermanos. Representa un flujo de información descendente. (Se calcula **antes** de reconocer sintácticamente el símbolo.)

Los atributos de los tokens (símbolos no terminales) son siempre sintetizados.

3. Especificación de un traductor

Las **gramáticas con atributos por la izquierda** son subconjuntos de las gramáticas con atributos donde los atributos heredados se calculas solamente a partir de los símbolos de la izquierda del atributo. De esta forma, cualquier información heredada estará disponible en el momento de ejecutar las reglas de producción. En este tipo de gramáticas, los flujos de información nunca se producen de derecha a izquierda. Permiten definir traductores de una sola pasada.
Los **esquemas de traducción dirigidos por la sintaxis** (ETDS) son un formalismo para construir traductores de una sola pasada que emplean gramáticas con atributos por la izquierda. Incluyen acciones semánticas encerradas entre llaves en cualquier punto de la regla de producción. Las acciones semánticas pueden incluir fragmentos de código con instrucciones que no se refieran al cálculo de atributos

4. Traductores Descendentes

4.1. Desarrollo de un ETDS basado en análisis descendente:

Los atributos sintetizados de cada símbolo se almacenan en una estructura de datos que devuelve la función asociada al símbolo. Los atributos heredados de cada símbolo se describen como parámetros de la

función asociada al símbolo. Las acciones semánticas incluidas en las reglas de producción de un símbolo se añaden directamente al código de la función asociada.

5. Traductores Ascendentes

5.1. Desarrollo de un ETDS basado en análisis ascendente:

A cada estado de la tabla se le puede asociar un símbolo que corresponde al símbolo que provoca la transición a ese estado. Los atributos sintetizados se almacenan en una pila de atributos paralela a la pila de estados.

- Las acciones semánticas situadas al final de una regla se ejecutan al reducir la regla.
- Las acciones semánticas situadas en puntos intermedios se sustituyen por símbolos ficticios llamados marcadores.

6. Estructuras Básicas

6.1. Generación de Estructura de Lista

- Gramática LL(1) BNF
  - <Lista> → **id** <SigueLista>
    { e = new Element(), e.id = id, e.next = SigueLista.s, Lista.s = e }
  - <SigueLista> → **coma id** <SigueLista$_1$>
    { e = new Element(), e.id = id, e.next = SigueLista$_1$.s, SigueLista.s = e
  - <SigueLista> → λ { SigueLista.s = null }
- Gramática LL(1) EBNF
  - <Lista> → **id** { e = new Element(), e.id = id, e.next = null, Lista.s = e, last
    (**coma id** { e = new Element(), e.id = id, e.next = null ,
    last.next = e, last = last.next })*

- Gramática LR(1)
  - <Lista> → **id**
    { e = new Element(), e.id = id, e.next = null, Lista.s = e, Lista.last = e }
  - <Lista> → <Lista$_1$> **coma id**
    { e = new Element(), e.id = id, e.next = null, Lista.s = Lista$_1$.s,
    Lista$_1$.last.next = e, Lista.last = e }
- Gramática LR(1)
  - <Lista> → **id** { e = new Element(), e.id = id, e.next = null, Lista.s = e }
  - <Lista> → **id coma** <Lista$_1$>
    { e = new Element(), e.id = id, e.next = Lista$_1$.s, Lista.s = e }

6.2. Generación de Estructura de Árbol Binario

- Gramática LL(1) BNF
  - <Árbol> → **num** { SigueÁrbol.h = num }
    <SigueÁrbol> {Árbol.s = SigueÁrbol.s }
  - <SigueÁrbol> → **más num** { SigueÁrbol$_1$.h = new Suma(SigueÁrbol.h,nur
    <SigueÁrbol$_1$> { SigueÁrbol.s = SigueÁrbol$_1$.s }
  - <SigueÁrbol> → λ { SigueÁrbol.s = SigueÁrbol.h }

- Gramática LL(1) EBNF
  - <Árbol> → **num** { Árbol.s = num }
    (**más num** { Árbol.s = new Suma(Árbol.s,num) })*

- Gramática LR(1)
  - <Árbol> → **num** { Árbol.s = num }
  - <Árbol> → <Árbol$_1$> **más num** { Árbol.s = new Suma(Árbol$_1$.s, num) }

6.3. Generación de Estructura de Vector

class Vector { public Vector(); public add(Element e); }

- Gramática LL(1) BNF
  - <Lista> → **num** <SigueLista>
    { v = new Vector(), v.add(num), SigueLista.h = v, Lista.s = v }
  - <SigueLista> → **más num**
    { SigueLista.h.add(num), SigueLista$_1$.h = SigueLista.h }
    <SigueLista$_1$>
  - <SigueLista> → λ { }
- Gramática LL(1) EBNF
  - <Lista> → { v = new Vector() } **num** { v.add(num) }
    (**más num** { v.add(num) })* { Lista.s = v }

- Gramática LR(1)
  - <Lista> → **num** { v = new Vector(), v.add(num), Lista.s = v }
  - <Lista> → <Lista$_1$> **más num** { Lista.s = Lista$_1$.s, Lista.s.add(num) }