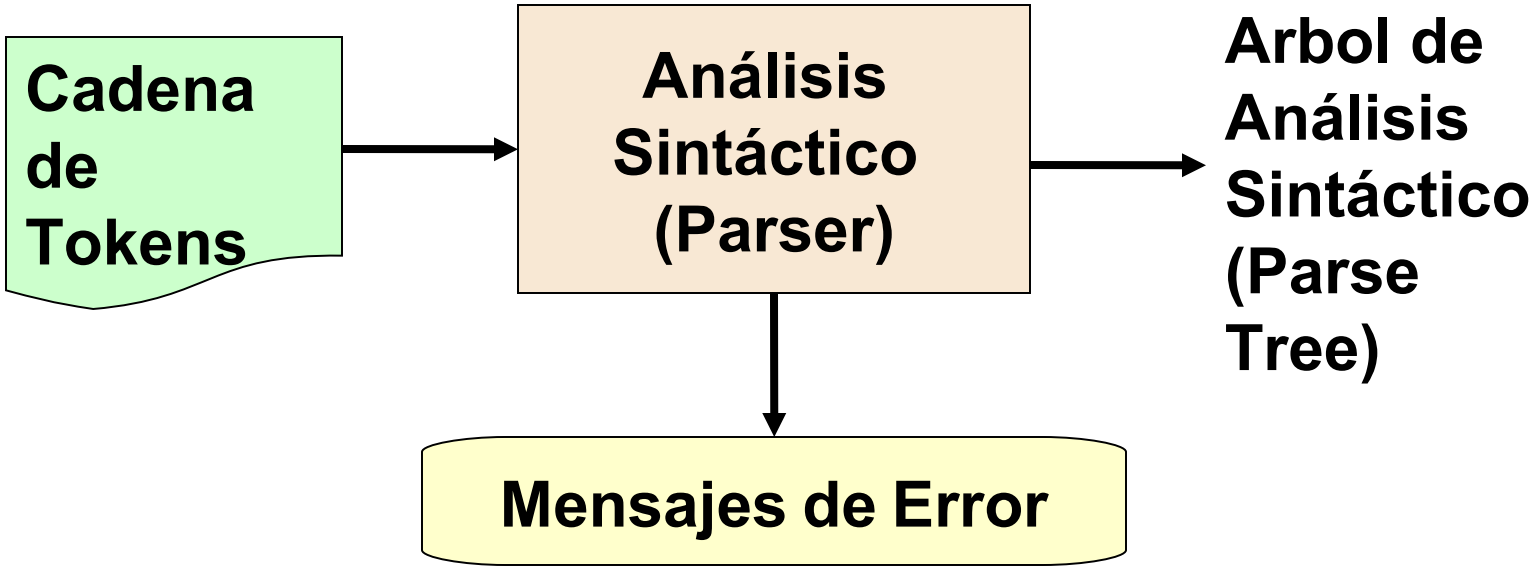
Compiladores:
Análisis Sintáctico

Pontificia Universidad Javeriana Cali
Ingenieria de Sistemas y Computación
Prof. Gloria Inés Alvarez V.

Sintaxis

- Define la estructura del lenguaje
- Ejemplo: Jerarquía en el Lenguaje C
  - Funciones
  - Bloques
  - Sentencias
  - Expresiones
- Las gramáticas permiten describir esta estructura, pero no incluyen información de la semántica

Análisis Sintáctico



Obtiene la cadena de tokens del analizador léxico y verifica que puede ser generada por la gramática que describe el lenguaje fuente

Gramática

Es una forma de describir un lenguaje formal.

- La gramática permite generar cadenas a partir de un simbolo inicial y aplicando reglas que indican como ciertas combinaciones de símbolos pueden ser reemplazadas usando otras combinaciones de símbolos

- Una Gramática Formal **G** se compone de:
- Un conjunto finito de **Símbolos No Terminales (N)**
- Un conjunto finito de **Símbolos Terminales (Σ)**
- Un conjunto finito de **Reglas de Producción (P)**
  - Cada regla tiene la forma $\alpha \rightarrow \beta$, donde **α** y **β** pertenecen a **(Σ U N)***, la parte izquierda (**α**) debe contener al menos un símbolo no terminal
- Un símbolo de Inicio (**S**), que pertenece a **N**

Gramática

- El lenguaje de una gramática **G = (*N*, Σ, *P*, *S*)**, se denota como **L(*G*)**, y se define como todas aquellas cadenas sobre **Σ** que pueden ser generadas inicando en el simbolo **S**, y aplicando las reglas de producción **P**, hasta reemplazar todos los símbolos no terminales

Gramática: Ejemplo

G:

- N = { S }
- Σ = { 0, 1 }
- P = { $S \rightarrow 0\ S$

  $S \rightarrow 1\ S$

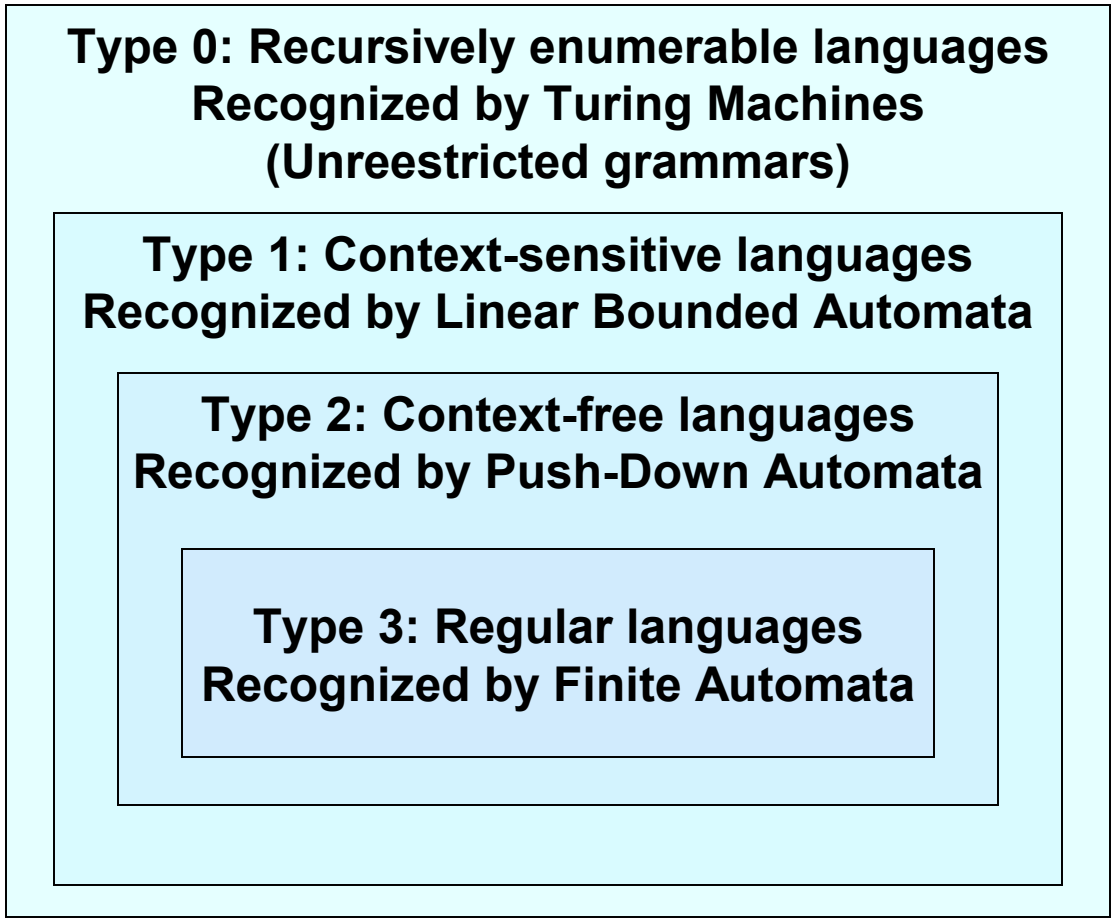
  $S \rightarrow Є$ }

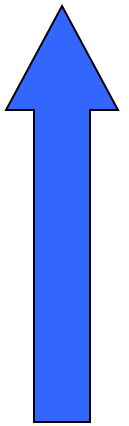
Cadenas generadas por G:

- 00000
- 010101
- 0001

Como quitar los 0´s al inicio de las cadenas?

La Jerarquía de Chomsky de los Lenguajes Formales





Menos Reestrictivas

Type 3: Regular Grammars

Una Gramática **G = (*N*, Σ, *P*, *S*)**, es una gramática regular si sus reglas de producción son de la forma:

$$A \rightarrow a$$

$$A \rightarrow a\ B$$

Donde **A**, **B** pertenecen a **N**, y **a** pertenece a **Σ**

El primer símbolo en el lado derecho de las reglas de producción debe ser un terminal y puede estar seguido por un no terminal

Type 3: Regular Grammars
Ejemplo

- El lenguaje { $a^n b^m$ | $m,n > 0$ } , sobre el Σ={a,b}

Type 3: Regular Grammars
Ejemplo

- El lenguaje { $a^n b^m$ | $m,n > 0$ } , sobre el Σ={a,b}

- P:
  - S -> aA
  - A -> aA
  - A -> bB
  - B -> bB
  - B -> Є

Type 2: Context-Free Grammars

Una Gramática **G = (N, Σ, P, S)**, es una gramática libre de contexto, si sus reglas de producción son de la forma:

$A \rightarrow \alpha$

Donde **A** pertenece a **N**, y **α** es cualquier cadena de **(Σ U N)***

Type 2: Context-Free Grammars
Ejemplo

- El lenguaje { $a^n b^n$ | $n > 0$ } , sobre el Σ={a,b}

Type 2: Context-Free Grammars
Ejemplo

- El lenguaje { $a^n b^n$ | $n > 0$ } , sobre el Σ={a,b}

- P:
  - S -> aSb
  - S -> ab

Type 1: Context-Sensitive Grammars

Una Gramática **G = (N, Σ, P, S)**, es una gramática sensitiva al contexto, si sus reglas de producción son de la forma:

$$\alpha \rightarrow \beta$$

Donde **α** y **β** pertenecen a **(Σ U N)***, α contiene al menos un símbolo no terminal, y la longitud de α es menor o igual que la longitud de β

Son usadas para procesamiento de lenguaje natural

Type 1: Context-Sensitive Grammars
Ejemplo

- El lenguaje { $a^n b^n c^n \mid n > 0$ } , sobre el Σ={a,b}

Type 1: Context-Sensitive Grammars Ejemplo

- El lenguaje { $a^n b^n c^n \mid n > 0$ } , sobre el Σ={a,b}

- P:
  - S -> aBSc
  - S -> abc
  - Ba -> aB
  - Bb -> bb

Backus-Naur Form (BNF)

- Notación mas comúnmente usada para representar Gramáticas Libres de Contexto (Context Free Grammars – CFG)
- Jhon Backus y Peter Naur pioneros en diseño de compiladores
- La notación BNF es un conjunto de **reglas de derivación** escritas así:

  <símbolo> ::= <expresión con símbolos>

  Donde <símbolo> es un No Terminal, y <expresión> consta de secuencias de símbolos separados por |

Gramáticas Limpias y Bien Formadas

- Sin **Reglas Innecesarias**. A :: = b, es una regla innecesaria si A no hace parte del lado derecho de otra regla. A es un **símbolo innaccesible**.

- Sin **Reglas Superfluas**. Dada la gramática G = ({a, b}, { S, A, B}, S, {S::=AB, A::=Aa|a, B:.=Bb}), la regla B::=Bb es superflua porque no puede derivar una cadena que solo contenga símbolos terminales.

- Sin **Símbolos No Generativos**. Dada la gramática G = (N, Σ, S, P), para cada símbolo A de N se construye la gramática $G(A)=(N_A, \Sigma_A, A, P_A)$, si L(G(A)) es vacío, entonces A es un símbolo no generativo.

Gramáticas Limpias y Bien Formadas

- Sin **Reglas No Generativas**. U :: = Є, es una regla no generativa. Si el lenguaje representado por la grámática no contiene la palabra vacía es posible eliminar todas las reglas no generativas, de lo contrario se debe admitir la regla S::= Є

- Sin **Reglas de Redenominación**. A ::= B es una regla de redenominación.

Gramáticas Limpias y Bien Formadas

- **Ejemplo**: G = ({0,1}, {S,A,B,C}, S,
  { S::=AB | 0S1 | A | C ,
    A::= 0AB | Є ,
    B::= B1 | Є,
    C::=01C })

Gramáticas Limpias y Bien Formadas

Ejemplo:

- C es superflua

- Se eliminan las reglas no generativas: G = ({0,1}, {S,A,B,C}, S,
  { S::=AB | 0S1 | A | B | Є,
    A::= 0AB | 0B | 0A | 0,
    B::= B1 | 1 })

Gramáticas Limpias y Bien Formadas

Ejemplo:

- Se eliminan las reglas de redenominación:

G = ({0,1}, {S,A,B,C}, S,

{ S::=AB | 0S1 | A | B | Є,

A::= 0AB | 0B | 0A | 0,

B::= B1 | 1 })

Derivaciones

- Tratando las reglas de producción como una regla de reescritura, en la que el no terminal del lado izquierdo es reemplazado por la cadena del lado derecho de la producción, se obtiene una derivación

- Ejemplo:

  Lista_id $\rightarrow$ TID TCOMA lista_id
  | TID

  Lista_id $\Rightarrow$ TID TCOMA lista_id
  $\Rightarrow$ TID TCOMA TID TCOMA Lista_id
  $\Rightarrow$ TID TCOMA TID TCOMA TID

  Lista_id $*\Rightarrow$ TID TCOMA TID TCOMA TID

Derivaciones

- $\alpha A \beta \Rightarrow \alpha\gamma\beta$ ($\alpha A \beta$ deriva $\alpha\gamma\beta$), si $A \rightarrow \gamma$ es una regla de producción, α, β, γ son cadenas de símbolos gramaticales

- Si $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n$, decimos que α_1 deriva α_n
- El símbolo "$\overset{*}{\Rightarrow}$" significa que deriva en un paso
- El símbolo "$\overset{*}{\Rightarrow}$" significa que deriva en cero o mas pasos
- $\alpha \overset{*}{\Rightarrow} \alpha$
- Si $\alpha \overset{*}{\Rightarrow} \beta$ y $\beta \overset{*}{\Rightarrow} \gamma$ entonces $\alpha \overset{*}{\Rightarrow} \gamma$
- Dada una gramática **G**, con símbolo inicial **S**, si $S \overset{*}{\Rightarrow} \alpha$, decimos que α es una Forma Sentencial de **G**

Derivaciones

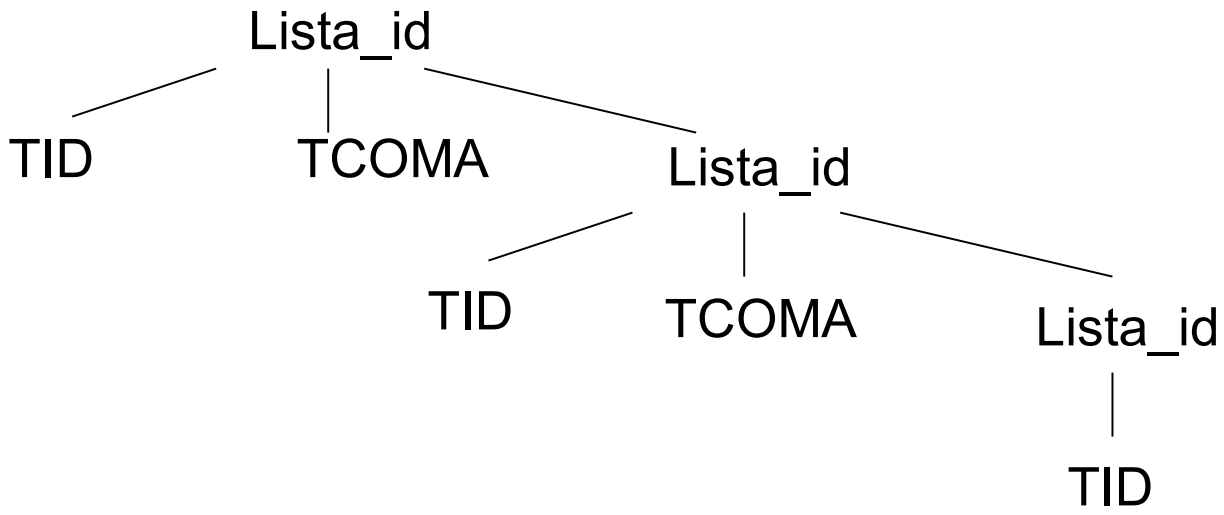
- Por la Izquierda (leftmost derivation): en cada paso de la derivación se reemplaza el no terminal que esta mas a la izquierda

- Por la Derecha (rightmost derivation): en cada paso de la derivación se reemplaza el no terminal que esta mas a la derecha. A estas derivaciones se les llama Derivaciones Canónicas

Arbol de Análisis Sintáctico (Parse Tree)

- Representación gráfica de una derivación
- Los nodos interiores del arbol corresponden a un no Terminal A, y sus hijos corresponden a los símbolos del lado derecho de la regla de producción que se usó para reemplazar A en la derivación.
- Las hojas corresponden a No Terminales o Terminales, y al recorrerlas de izquierda a derecha se tiene una forma sentencial

Arbol de Análisis Sintáctico (Parse Tree): Ejemplo



Ambiguedad

- Una Gramática Ambigua produce mas de un árbol de parser para una sentencia
- Produce mas de una derivación por la derecha o derivación por la izquierda para la misma cadena
- Algunas veces las gramáticas ambiguas deben ser reescritas para eliminar la ambiguedad

Ambiguedad

- Sentencia IF ambigua:

  Instrucción → Tif expr Tthen Instrucción
  | Tif expr Tthen Instrucción Telse Instrucción
  | OtrasInstrucciones

- Eliminando la ambiguedad:

  Instrucción → InstrCompleta
  | InstIncompleta

  InstrCompleta → Tif expr Tthen InstrCompleta Telse InstrCompleta
  | OtrasInstrucciones

  InstrIncompleta → Tif expr Tthen Instrucción
  | Tif expr Tthen InstrCompleta Telse InstrIncompleta

Recursión Izquierda

- Una gramática es Recursiva por la Izquierda si tiene un no terminal **A** tal que $A \overset{+}{\Rightarrow} A\alpha$ para alguna cadena α

- Los métodos de parsing top-down **no** manejan gramáticas recursivas por la izquierda

- Eliminar la recursión izquierda:
  - Si $A \rightarrow A\alpha \mid \beta$

    Reemplazar por: $A \rightarrow \beta A'$

    $A' \rightarrow \alpha A' \mid \epsilon$
  - Generalizando: $A \rightarrow A\alpha_1 \mid A\alpha_2 \mid ... \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid ... \mid \beta_n$

    Reemplazar por: $A \rightarrow \beta_1 A' \mid \beta_2 A' \mid ... \mid \beta_n A'$

    $A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid ... \mid \alpha_n A' \mid \epsilon$

Factorización Izquierda

- Una transformación de una gramática necesaria para realizar parsing predictivo

- Factorizando por la izquierda:
  - Si $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n \mid \gamma$

    Donde α es un prefijo comun, y $\alpha \neq \epsilon$, y γ representa todas las alternativas que no empiezan con α

    Reemplazar por: $A \rightarrow \alpha A' \mid \gamma$

    $A' \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$