# Análisis Sintáctico

Veronica Paulina – Cchimbo Coronelale Pilco

## 1. Análisis Sintáctico

Recibe como entrada los tokens que le pasa el analizador léxico y comprueba si estos van llegando en el orden correcto. Su salida "teórica" sería un árbol sintáctico.

Sus funciones son:

•Aceptar lo que es válido sintácticamente y rechazar lo que no lo es

•Hacer explícito el orden jerárquico que tienen los operadores en el lenguaje de que se trate

•Guiar el proceso de traducción (traducción dirigida por sintaxis)

Un factor de división entre el análisis léxico y sintáctico es la recursión. Las construcciones léxicas no requieren recursión, mientras que las construcciones sintácticas suelen requerirla. Las GLC son una formalización de reglas recursivas que se pueden usar para guiar el análisis sintáctico.

Se puede describir la sintaxis de las construcciones de los LP por medio de gramáticas independientes del contexto o notación BNF.

•Una gramática da una especificación sintáctica precisa y fácil de entender de un LP

•A partir de algunas clases de gramáticas se puede construir automáticamente un analizador sintáctico eficiente que determine si un programa fuente está sintácticamente bien formado

•Una gramática diseñada adecuadamente imparte una estructura a un lenguaje de programación útil para la traducción de programas fuente a código objeto correcto y para la detección de errores

•Los lenguajes evolucionan con el tiempo, adquiriendo nuevas construcciones y realizando tareas adicionales

## 2. Tipos de Analizadores Sintácticos

*Ascendentes*. Construyen árboles sintácticos a partir de las hojas y suben a la raíz

*Descendentes*. Construyen árboles sintácticos de la raíz a las hojas

En ambos casos se examina la entrada al A.S. de izquierda a derecha, un símbolo a la vez

## 3. Manejo de errores sintácticos

A menudo, gran parte de la detección y recuperación de errores en un compilador se centra en la fase de análisis sintáctico

Razones

- La cadena de componentes léxicos no obedece las reglas gramaticales que definen al L.P.
- Precisión en los métodos modernos de A.S.

El manejador de errores en un A.S. tiene objetivos fáciles de establecer:

- Debe informar de la presencia de errores con claridad y exactitud
- Se debe recuperar de cada error con la suficiente rapidez como para detectar errores posteriores
- No debe retrasar de manera significativa el procesamiento de programas correctos

El manejador de errores debe informar de la presencia de un error, indicando el lugar preciso en el programa, y si sabe cuál es el error, se incluye un mensaje.

## Opciones para implementar un parser

1."a mano"

2.Utilizando un generador de analizadores sintácticos, por ejemplo YACC

## 4. Notación EBNF

Extended Backus-Naur Form. Objetivo, reducir el número de producciones en las gramáticas. Notaciones adicionales.

*1.Alternativas de una regla.* Se utiliza el símbolo "|" para separar las distintas posibilidades que definen al no terminal de la izquierda.

Ejemplo: Si S→a y S→aSb entonces se escribe como S→a | aSb

2. *Llaves {}.* Lo que aparece dentro de ellos se repite de cero a n veces.

Ejemplo: Linea_dec→Tipo iden { , iden} ;

3. *Llaves con repetición especificada: { }$y_x$ .* Lo que aparece dentro de ellas se repite un número de veces comprendido entre x e y.

Ejemplo: iden→letra {digito | letra}$^{7}_{0}$

4. *Corchetes:[].* Es un caso particular de 3 ($\{\}^1_0$). Lo que esta dentro puede o no aparecer.

Ejemplo: prop_if→if condicion then bloque [else bloque]

**5. Diseño de gramáticas para lenguajes de programación**

### 1. Recursividad

Problema: Un compilador debe procesar correctamente un número infinito de programas, pero por otra parte la especificación sintáctica de un lenguaje debe ser finita.

Solución: Recursividad

### Estructura de la recursividad

1.Regla no recursiva que se define como caso base

2.Una o mas reglas recursivas que permiten el crecimiento a partir del caso base

**Ejemplo:** Gramática para definir un número entero

Digito→0|1|2|…|9
Entero→Digito Entero *Regla recursiva*

Entero→Digito *Caso base*

**Definición.** Una gramática es recursiva, si podemos derivar una tira en la que nos vuelve a aparecer el símbolo no terminal que aparece en la parte izquierda de la regla de derivación. A => αAβ o A=> Aβ Recursividad izquierda

A=>αARecursividad derecha

### 2. Ambigüedad

Una gramática es ambigua si el lenguaje que define contiene alguna sentencia que tenga más de un único árbol de análisis sintáctico, es no ambigua cuando cualquier tira del lenguaje que representa, tiene un único árbol sintáctico.

No es posible construir analizadores sintácticos eficientes para gramáticas ambiguas.

No se disponen de técnicas para saber si una gramática es ambigua o no. La única forma de saberlo es encontrando una cadena con dos o más árboles sintácticos distintosAlgunas de las características que tiene las gramáticas ambiguas son las siguientes:

• Gramáticas con ciclos simples o menos simples

S→A | a
A→S

• Alguna regla con una forma

E→E … E

• Un conjunto de reglas de forma parecida a:

S→A | B
A→B

• Producciones recursivas en las que las variables no recursivas de la producción puedan derivar a la cadena vacía:

S→HRS | s
H→h | λ
R→r | λ

• Variables que puedan derivar a la cadena vacía y a la misma cadena de terminales, y que aparezcan juntas en la parte derecha de una regla o en alguna forma sentencial:

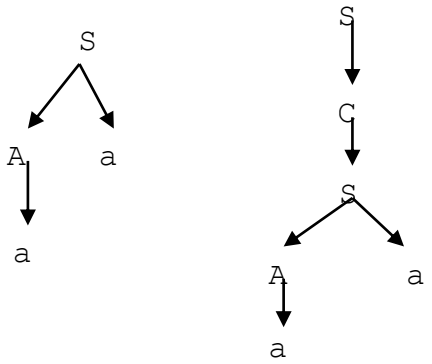
S→HR
H→h | λ
R→r | h | λ

**Ejemplo:** Sea la gramática

S→Aa | C
C→S | Ac
A→a

La tira 'aa' tiene dos árboles sintácticos.

Para solucionar la ambigüedad se deben modificar las reglas de producción de la gramática




## 3. Asociatividad y precedencia de operadores

### Asociatividad

La asociatividad de un operador se define cómo se operan tres o más operandos.

Tipos de asociatividad:

• *Asociatividad izquierda* (se evalúa de izquierda a derecha)

•*Asociatividad derecha* (se evalúa de derecha a izquierda)

La asociatividad en una gramática se refleja en el tipo de recursividad que se emplea. Si la *asociatividad del operador es por la izquierda*, la regla sintáctica en la que interviene dicho operador debe ser *recursiva por la izquierda*; en el caso de *asociatividad por la derecha* se utiliza *recursión por la derecha.*

### Precedencia

La precedencia de un operador especifica el orden relativo de cada operador con respecto a los demás operadores.

La precedencia en una gramática se refleja de la siguiente manera: cuánto más *cerca* esté la producción de la del *símbolo inicial*, *menor será la precedencia* del operador.

### Parentización

Los paréntesis son operadores especiales que tiene la máxima precedencia.

Para incluirlos en la gramática, se añade una variable que produzca expresiones entre paréntesis y los operandos (números, variables, etc.) a la mayor distancia posible del símbolo inicial. En esta producción se colocan los operadores unarios a no ser que tengan una precedencia menor.

## 6. Bibliografía


1. Viñuela, P. I., Viñuela, P. I., MillÔan, D. B., FernÔandez, P. M., Martínez, P., Borrajo, D., ... & Millán, D. B. (1997). Lenguajes, gramáticas y autómatas: un enfoque práctico. Pearson Educación.
2. Aho, A. V., Sethi, R., & Ullman, J. D. (1998). Compiladores: principios, técnicas y herramientas. Pearson Educación.
3. Sanchis Llorca, F. J., & Pascual, C. G. (1986). Compiladores: teoría y construcción. Paraninfo.