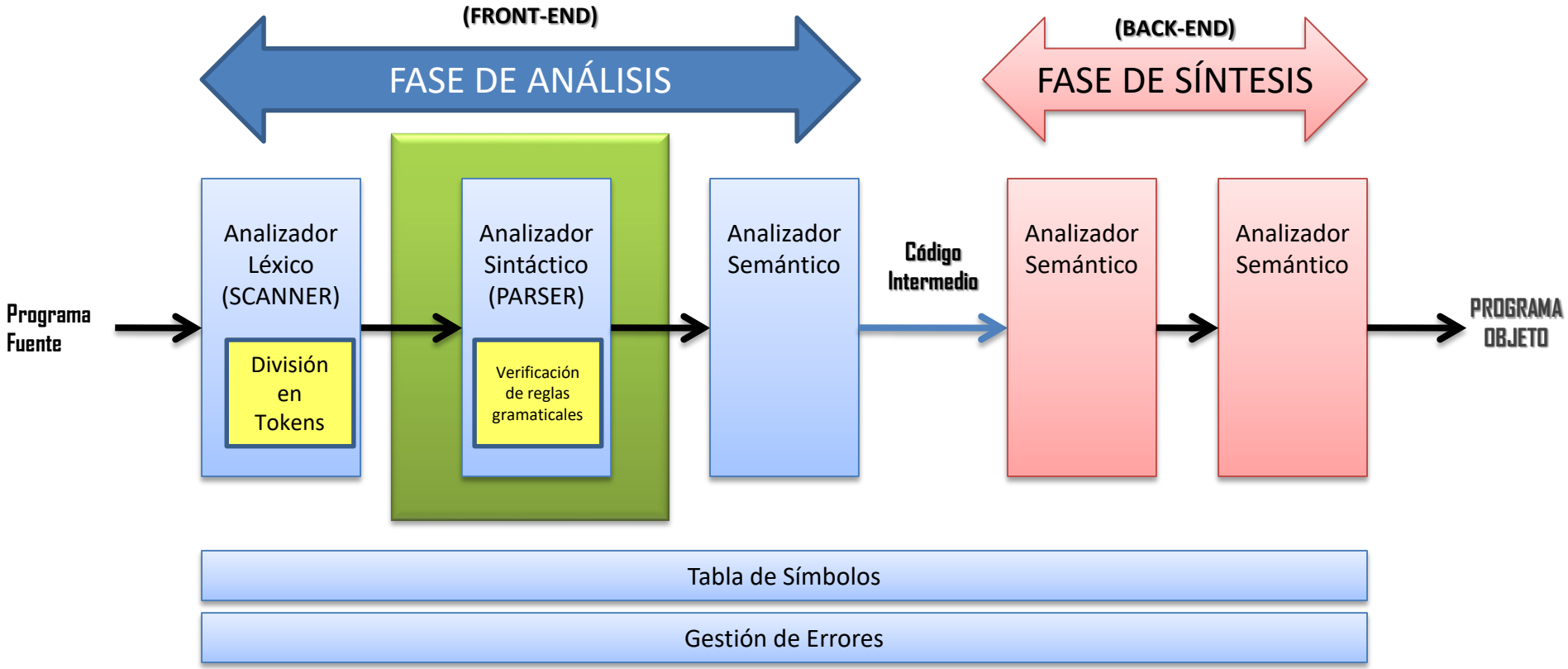
Sesión 4
Analizador Sintáctico
Introducción y Gramáticas

Antonio Moratilla Ocaña

Resumen del tema

- Objetivo:
  - Conocer las responsabilidades de un analizador sintáctico y las construcciones que necesita para poder ser utilizado.

Posición en el diagrama



Retomando el hilo...

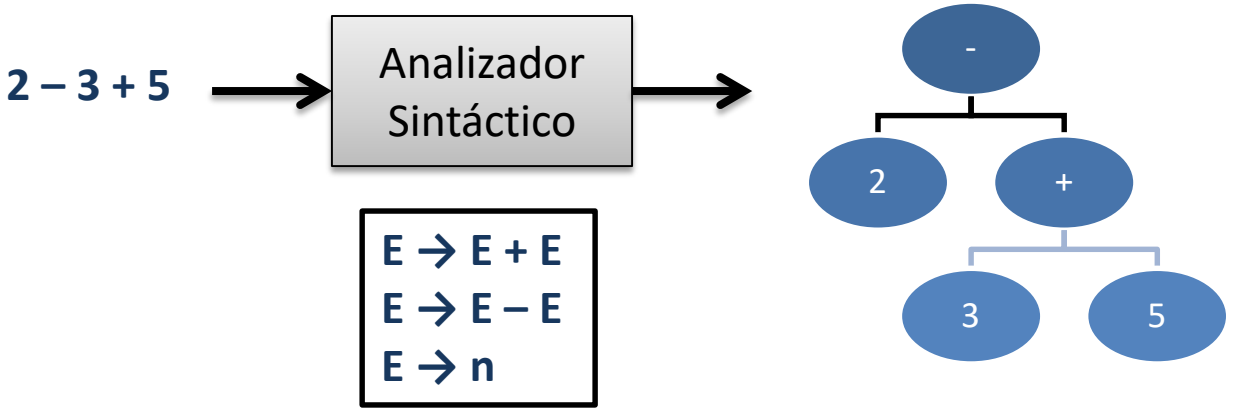
- Ya sabemos
  - Que los analizadores léxicos leen un fichero de entrada, y lo convierten en TOKENS.
- Lo que queremos saber
  - Y ahora… ¿qué hacemos con los tokens? ¿cómo lo hacemos?

Función del analizador sintáctico

- El **analizador sintáctico** construye una representación intermedia del programa analizado.
  - Construye un árbol de análisis a partir de los componentes léxicos que recibe, aplicando las producciones de la gramática con el objeto de comprobar la corrección sintáctica de las frases.
- Comprobar que el orden en que el analizador léxico le va entregando los *tokens* es válido:
  - Para ello verifica que la cadena pueda ser generada por la **gramática** del lenguaje fuente.
- Informar acerca de los errores de sintaxis, recuperándose de los mismos (si es posible) para continuar procesando la entrada.

Ejemplo de Analizador sintáctico

- Dada la expresión: “2 –3 + 5”
- El analizador sintáctico utiliza las reglas de producción de la gramática para construir el árbol sintáctico.



Gramática Independiente del Contexto

- La sintaxis de un lenguaje se especifica mediante las denominadas **Gramáticas Independientes del Contexto.**
  - Una Gramática Independientes del Contexto (GIC) es una gramática formal en la que cada regla de producción es de la forma: **Exp → x**

    Donde *Exp* es un símbolo no terminal y x es una cadena de terminales y/o no terminales. El término independiente del contexto se refiere al hecho de que el no terminal *Exp* puede siempre ser sustituido por x sin tener en cuenta el contexto en el que ocurra.

    Un lenguaje formal es independiente de contexto si hay una gramática libre de contexto que lo genera.

UTILIZACIÓN

- Describen de forma natural la estructura jerárquica de las construcciones de los lenguajes de programación.
- Facilitan la construcción de analizadores sintácticos eficientes.
- Impone una estructura al lenguaje que posteriormente resulta útil para su traducción a código objeto y para la detección de errores.
- Facilitan la extensión (ampliación con nuevas construcciones) del lenguaje.

Axioma: Para cualquier Gramática Independiente del Contexto se puede construir un analizador sintáctico.

Gramática Independiente del Contexto

Componentes de una gramática: $G = (V_t, V_n, S, P)$

- Símbolos terminales(componentes léxicos [tokens]), $\mathbf{V_t}$
- Símbolos no-terminales, $\mathbf{V_n}$
- Símbolo inicial o axioma, **S**
- Producciones, formadas por no-terminales y terminales, **P**

Una gramática se describe

– Mostrando una lista de sus producciones.

- Una producción consta de un símbolo no-terminal (parte izquierda), una flecha, y una secuencia de símbolos terminales y no-terminales (parte derecha).

  E → E + E

  E → E – E

  E → n

– Usando la notación Backus-Naur Form (BNF) o Extended BNF (EBNF)

BNF -Backus-Naur Form

- Es una notación alternativa para la especificación de producciones (BNF –Backus-Naur Form).

  Representación.

  ::= significa “se define como”

  | significa "or lógico"

  < > encierran los no-terminales

  Los terminales se escriben tal y como son.

Ejemplos:

1. <identificador>::=<letra>|<identificador>[<letra>|<dígito>]
2. <nombre_completo> ::= [<trato>] <nombre> <apellidos> | <apellidos> , <nombre>
3. *<validchar>* ::= <letter> | <digit> | _
   <letter> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
   |A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
   <digit> ::= 1|2|3|4|5|6|7|8|9|0

Amplía sobre BNF en http://www.garshol.priv.no/download/text/bnf.html

Lenguaje definido por una gramática

- Un lenguaje definido por una gramática es el conjunto de cadenas de componentes léxicos derivadas del símbolo inicial de la gramática.

$$L(G) = \{s \mid exp \rightarrow * s\}$$

Ejemplos:

1. $E \rightarrow (E) \mid a$ $\quad\quad$ $L(G) = \{(a), ((a)), (((a))), \ldots\}$

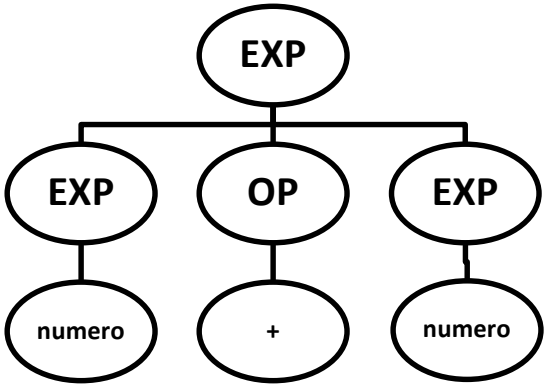
2. $E \rightarrow E+ a \mid a$ $\quad\quad$ $L(G) = \{a, a+a, a+a+a, \ldots\}$

Árbol gramatical

Un árbol gramatical correspondiente a una derivación es un árbol etiquetado en el cual los nodos interiores están etiquetados por no terminales, los nodos hoja están etiquetados por terminales, y los hijos de cada nodo interno representan el reemplazo del no terminal asociado en un paso de la derivación

Ejemplo:

```
EXP → EXP OP EXP
EXP → numero
OP → +
OP → -
```



Derivaciones

Se denomina derivación a la sucesión de una o más producciones:

$$A_1 \Rightarrow A_2 \Rightarrow A_3 \Rightarrow \ldots \Rightarrow A_n \text{ o también } A_1 \Rightarrow A_n$$

- Una cadena de componentes léxicos es considerada válida si existe una derivación en la gramática del lenguaje fuente que parta del símbolo inicial y tras aplicar las producciones a los no terminales, genere la frase a reconocer.
- Las derivaciones pueden ser por la izquierda o por la derecha (canónicas).

  –Derivación por la izquierda: sólo el no-terminal de más a la izquierda de cualquier forma de frase se sustituye en cada paso:

  E →E + E | E * E | (E) | -E | id

  E => -E => -(E) => -(E+E) => -(id+E) => -(id+id)

Reglas de producción - Recursividad

Permite expresar iteración utilizando un número pequeño de reglas de producción.

La estructura de las producciones recursivas es:

- Una o más reglas no recursivas que se definen como caso base.
- Una o más reglas recursivas que permiten el crecimiento a partir del caso base.

Ejemplo:

Estructura de un tren formado por una locomotora y un número de vagones cualquiera detrás.

Solución 1 (no recursiva):

tren → locomotora
tren → locomotora vagón
tren → locomotora vagón vagón ...

Solución 2 (recursiva):

Regla base: tren → locomotora
Regla recursiva : tren → tren vagón

La regla recursiva permite el crecimiento ilimitado

Reglas de producción - Recursividad

Una gramática se dice que es recursiva si en una derivación de un símbolo no-terminal aparece dicho símbolo en la parte derecha: A *=> aAb

Tipos de recursividad:

- Por la izquierda: Problemática para el análisis descendente, funciona bien en los analizadores ascendentes. A *=> Ab
- Por la derecha: Utilizada para el análisis descendente. A *=> aA
- Por ambos lados: No se utiliza porque produce gramáticas ambiguas.

- Transformar la recursividad: A=>Aa|b →

  A=>bC

  C=aC|Ɛ

Reglas de producción - Ambigüedad

Una gramática es ambigüa si el lenguaje que define contiene alguna cadena que pueda ser generada por más de un árbol sintáctico distinto aplicando las producciones de la gramática.

- Para la mayoría analizadores sintácticos es preferible que la gramática no sea ambigua
  - Es complicado conseguir en todos los casos la misma representación intermedia.
  - El analizador resultante puede no ser tan eficiente
- Es posible, mediante ciertas restricciones, garantizar la no ambigüedad de una gramática.

- Algunos generadores automáticos son capaces de manejar gramáticas ambiguas, no obstante se deben proporcionar reglas para evitar la ambigüedad y generar un único árbol sintáctico.

Reglas de producción - Ambigüedad

- Evitar producciones recursivas en las que las variables no recursivas de la producción puedan derivar a la cadena vacía

  S → **HR**S S → s H → h | ε R → r | ε

- No permitir ciclos : S →A S →a **A →S**
- Suprimir reglas que ofrezcan caminos alternativos

  S → A S → B **A → B**

La ambigüedad implica que a una misma sentencia se le pueden asignar significados (semánticas) diferentes.

- El algoritmo para poder crear un árbol sintáctico para una gramática ambigua necesita de prueba y retroceso
- Si un lenguaje es ambiguo existirán varios significados posibles para el mismo programa, y por tanto el compilador podría generar varios códigos máquina diferentes para un mismo código fuente.
- Un análisis sintáctico determinista (sin posibilidad de elegir entre varias opciones) es más eficiente

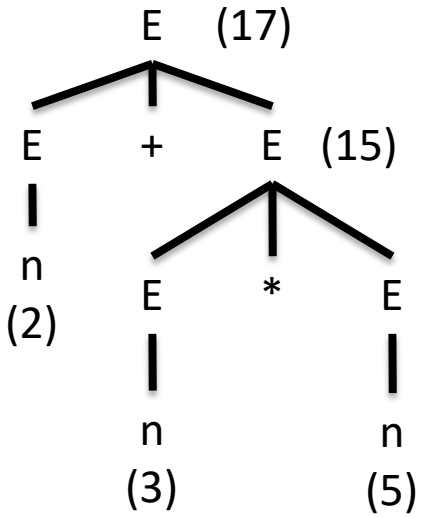
Las gramáticas de los lenguajes de programación no deben ser ambiguas.

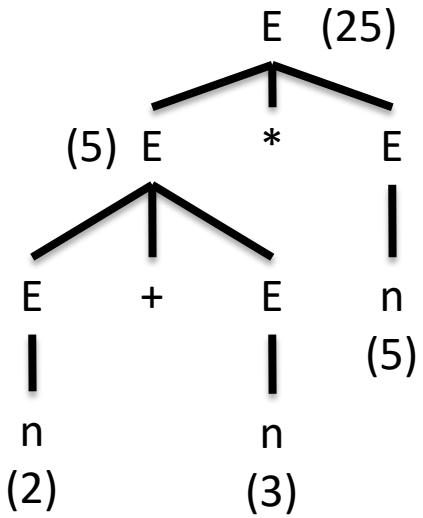
Reglas de producción - Ambigüedad

EJEMPLO DE AMBIGÜEDAD

E → E + E
E → E * E
E → (E)
E → n

Expresión
2+3*5





Producciones
E → E + E
→ n + E
→ n + E * E
→ n + n * E
→ n + n * n

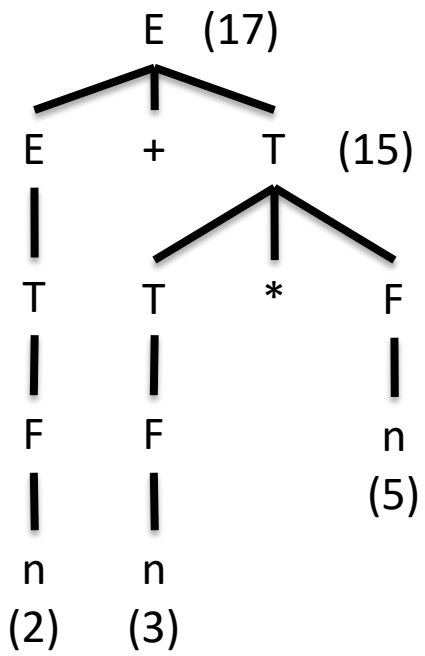
Producciones
E → E * E
→ E + E * E
→ n + E * E
→ n + n * E
→ n + n * n

Reglas de producción - Ambigüedad

SOLUCIÓN A LA AMBIGÜEDAD

E → E + T | T
T → T * F | F
F → (E) | n

Expresión
2+3*5



Producciones

E → E + E
→ n + E
→ n + E * E
→ n + n * E
→ n + n * n

Tipos de analizador sintáctico

- Para comprobar si una cadena pertenece al lenguaje generado por una gramática, los analizadores sintácticos construyen una representación en forma de árbol de 2 posibles métodos:
  - **Analizadores sintácticos descendentes** (Top-down)
    - Construyen el árbol sintáctico de la raíz (arriba) a las hojas (abajo).
    - Parten del símbolo inicial de la gramática (*axioma*) y van expandiendo producciones hasta llegar a la cadena de entrada.
  - **Analizadores sintácticos ascendentes** (Bottom-up)
    - Construyen el árbol sintáctico comenzando por las hojas.
    - Parten de los terminales de la entrada y mediante reducciones llegan hasta el símbolo inicial.
  - En ambos casos se examina la entrada de izquierda a derecha, analizando los testigos o tokens de entrada de uno en uno.

- Notas:
  1. Los métodos descendentes se pueden implementar más fácilmente sin necesidad de utilizar generadores automáticos.
  2. Los métodos ascendentes pueden manejar una mayor gama de gramáticas por lo que los generadores automáticos suelen utilizarlos.
  3. Para cualquier gramática independiente de contexto hay un analizador sintáctico "general" que toma como máximo un tiempo de $O(n^3)$ para realizar el análisis de una cadena de n componentes léxicos. Se puede conseguir un análisis lineal O(n)para la mayoría de lenguajes de programación.

Ejercicios

- Muestre el ADF equivalente a las siguientes ER (continuamos los del otro día):
  1. a|(ab)
  2. (a|b)+
  3. (ab)|(b*)
  4. (a*|b*)c+|d?
  5. (a|b*)+
  6. [a-zA-Z]([0-9]|[a-zA-Z])*

Fuentes

- Para la elaboración de estas transparencias se han utilizado:
  - Transparencias de cursos previos (elaboradas por los profesores Dr. D. Salvador Sánchez, Dr. D. José Luis Cuadrado).
  - Libros de referencia (en especial capítulos 2 y 4 de Aho).