

Análisis Sintáctico (I)

Diseño de Lenguajes de Programación

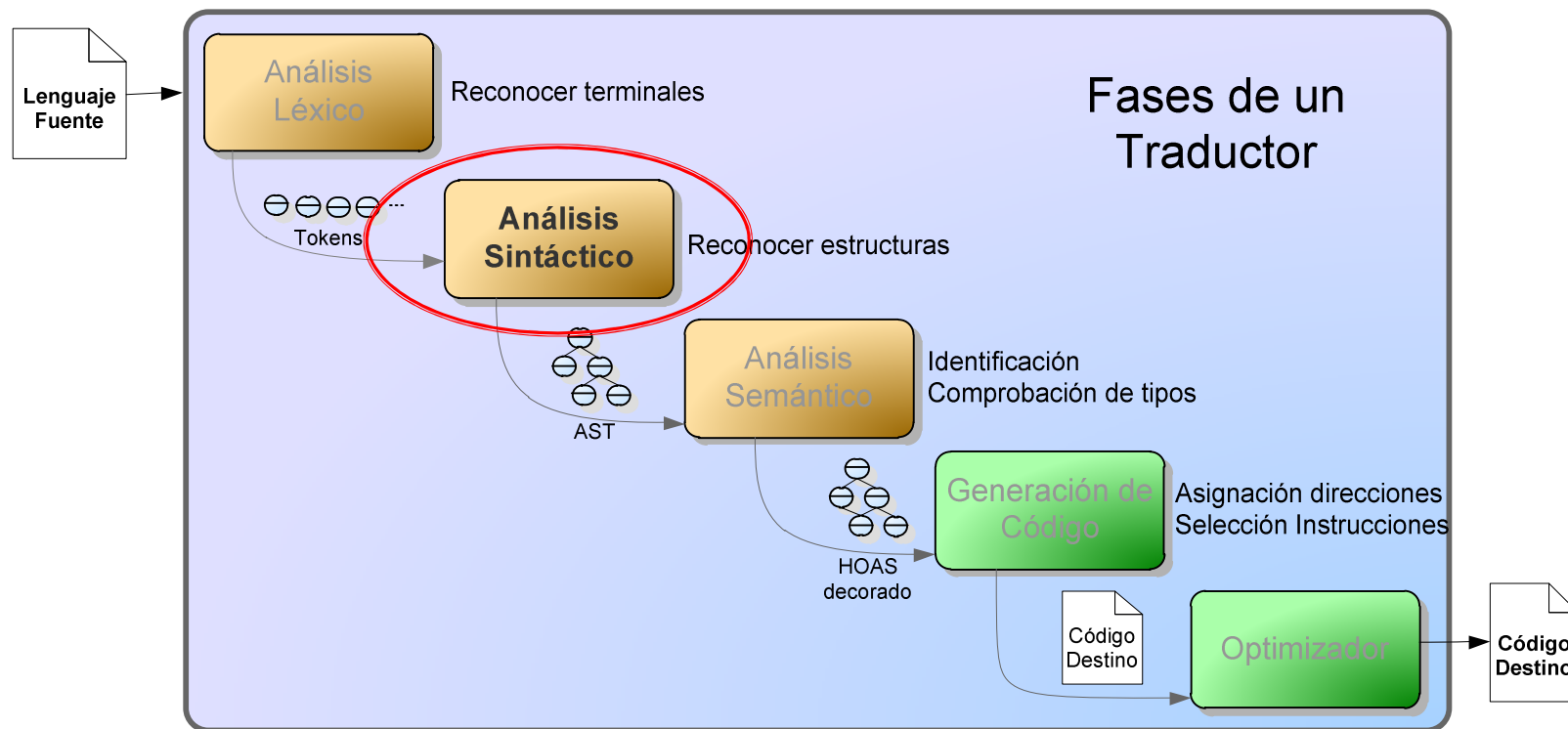
Ingeniería Informática

Universidad de Oviedo

(v2.0)

Raúl Izquierdo Castanedo

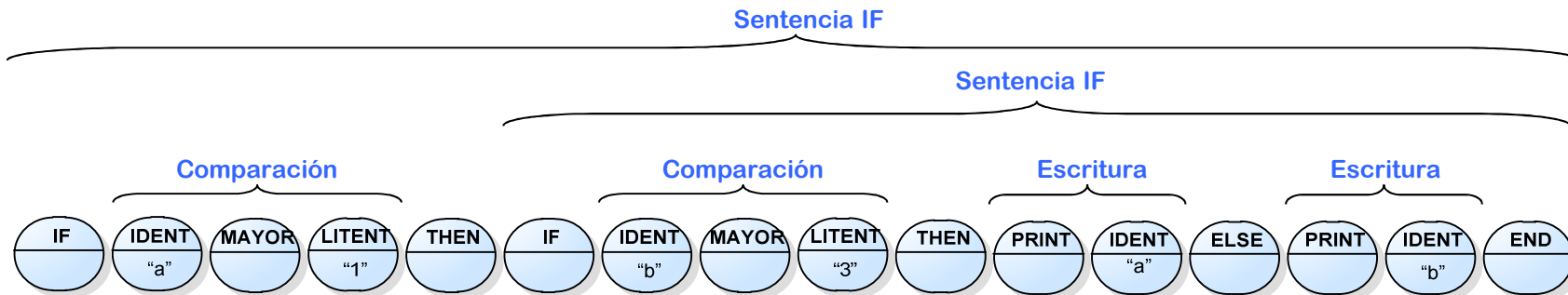
Análisis Sintáctico



Funciones

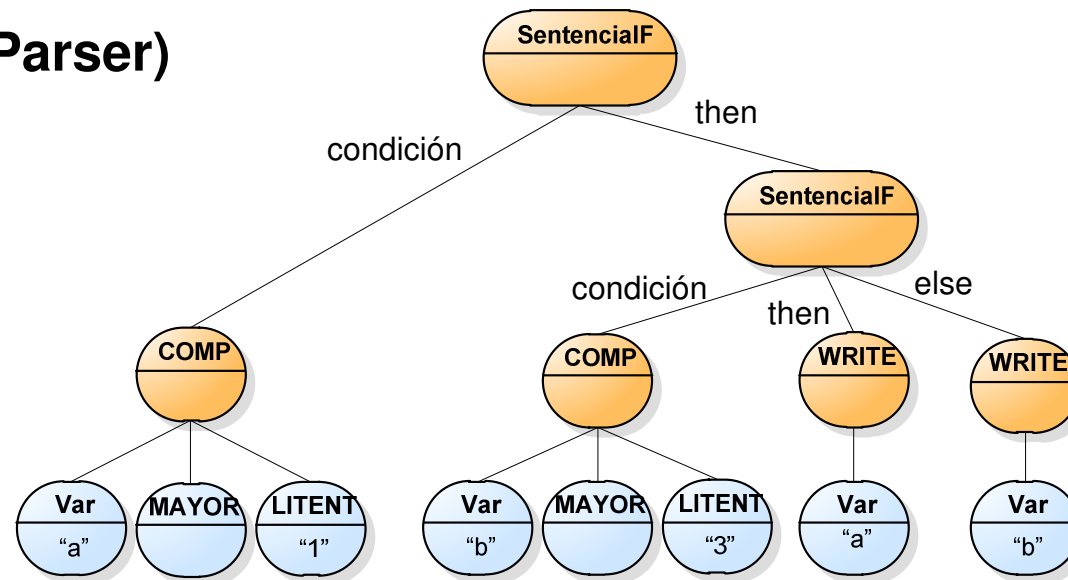
1. Reconocer Estructuras (Analizador)

Esta clase y la siguiente



2. Crear AST (Parser)

Clase 3 de Sintáctico



Metalinguaje Sintáctico

*Los "planos" del
analizador
sintáctico*

Metalinguajes

Lenguaje

$$3 * (4 + 5)$$

Definición en lenguaje natural

- Léxico: números, paréntesis, suma y producto
- Sintáctico
 - Las expresiones utilizan notación infija
 - El número de paréntesis abierto \geq cerrados
 - Deben ser iguales al finalizar

Gramáticas Libres de Contexto (GLC)

Gramática

$$G = \{VT, VN, s, P\}$$

VT = Símbolos terminales

- Tokens

VN = Símbolos no-terminales

- Estructuras del lenguaje

$s \in VN$

- Estructura que comprende a todas las demás

P = Reglas de producción

- $n \rightarrow \alpha$ / $n \in VN, \alpha \in (VN \cup VT)^*$
- Indican la composición de las estructuras

Ejemplo

$$G = \{VT, VN, programa, P\}$$

VT = { ident if = () constante + * }

VN = { programa, instrucciones, instr, expr }

P = {

programa \rightarrow instrucciones

instrucciones \rightarrow instr

instrucciones \rightarrow instrucciones instr

instr \rightarrow ident = expr

instr \rightarrow ident (expr)

instr \rightarrow if expr then instr else instr

expr \rightarrow constante

| expr + expr

| expr * expr

}

¿Válido "a = b = 0" ?
¿"else" opcional?
¿"if" anidados?

GLC en notación BNF

Notación de una GLC

$G = \{VT, VN, \text{programa}, P\}$

$VT = \{ \text{ident if } = () \text{ constante } + * \}$

$VN = \{ \text{programa, instrucciones, instr, expr} \}$

$P = \{$
 $\text{programa} \rightarrow \text{instrucciones}$

 $\text{instrucciones} \rightarrow \text{instr}$
 $\text{instrucciones} \rightarrow \text{instrucciones instr}$

 $\text{instr} \rightarrow \text{ident} = \text{expr}$
 $\text{instr} \rightarrow \text{ident} (\text{expr})$
 $\text{instr} \rightarrow \text{if expr then instr else instr}$

 $\text{expr} \rightarrow \text{constante}$
 $| \text{expr} + \text{expr}$
 $| \text{expr} * \text{expr}$
 $\}$

Notación BNF

- Añade el operador '|'.

Solo se indicará P. El resto se deduce:

- VT serán los símbolos en mayúsculas y los caracteres no alfanuméricos.
- VN serán los símbolos en minúsculas.
- s es el antecedente de la primera regla.

programa: instrucciones

instrucciones: instr
 | instrucciones instr

instr: IDENT = expr
 | IDENT (expr)
 | IF expr THEN instr ELSE instr

expr: CONSTANTE
 | expr + expr
 | expr * expr

Notación EBNF

Notación BNF

- Añade el operador '|'.

programa: instrucciones

instrucciones: instr
| instrucciones instr

instr: IDENT = expr
| IDENT (expr)
| IF expr THEN instr
ELSE instr

expr: CONSTANTE
| expr + expr
| expr * expr

Notación EBNF

- Añade los operadores '+', '*' y '?'.

programa: instr⁺

instr: IDENT = expr
| IDENT (expr)
| IF expr THEN instr
ELSE instr

expr: CONSTANTE
| expr + expr
| expr * expr

Validación de Entradas

Aplicación de una Gramática



Definiciones (I)

Transformación (o Paso de Derivación)

- Sea

$$G = \{V_T, V_N, s, P\}$$

- Se dice que $\beta\alpha\gamma$ es una transformación de $\beta n \gamma$ si existe $(n \rightarrow \alpha) \in P$

- Y se escribe

$$\beta \mathbf{n} \gamma \Rightarrow \beta \mathbf{\alpha} \gamma$$

Ejemplo

$$X a b Z \Rightarrow X Y a b Z$$

GLC

$$s \rightarrow a b Z$$

$$a \rightarrow X a$$

$$\mathbf{a} \rightarrow \mathbf{Y} \mathbf{a}$$

$$a \rightarrow$$

$$b \rightarrow W b$$

$$b \rightarrow$$

Definiciones (II)

Derivación

- α_n es una derivación de α_1 si se obtiene aplicando una o más transformaciones

$$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n$$
$$\alpha_1 \xRightarrow{*} \alpha_n$$

- ¿Es "X Y W b Z" una derivación de "X a b Z"?

□ Por tanto

$$X a b Z \xRightarrow{*} X Y W b Z$$

GLC

$s \rightarrow a b Z$

$a \rightarrow X a$

$a \rightarrow Y a$

$a \rightarrow$

$b \rightarrow W b$

$b \rightarrow$

Definiciones (III)

Sentencia

- Sea s el símbolo inicial de la gramática
- α será una sentencia si cumple que
 1. $s \xRightarrow{*} \alpha$ (α es una derivación de s)
 2. $\alpha \in VT^*$ (α está formada únicamente por terminales)
- Ejemplos
 - ¿Es una sentencia "X W Z"?
 - ¿Es una sentencia "X b Z"?

GLC

$s \rightarrow a b Z$

$a \rightarrow X a$

$a \rightarrow Y a$

$a \rightarrow$

$b \rightarrow W b$

$b \rightarrow$

Definiciones (IV)

Lenguaje

- Sea $G = \{VT, VN, s, P\}$
- El lenguaje que genera G ($L(G)$) es el conjunto de todas sus sentencias

$$L(G) = \{t \in VT^* / s \Rightarrow^* t\}$$

GLC

$$s \rightarrow X a$$

$$a \rightarrow Y b$$

$$a \rightarrow \lambda$$

$$b \rightarrow W$$

$$b \rightarrow \lambda$$

$$L(\text{GLC}) = \{ \quad \quad \quad \}$$

Objetivo

Nuestro objetivo no es generar el lenguaje

Nuestro objetivo

$X Y W$

□ ¿Pertenece?

$s \rightarrow X a$

$a \rightarrow Y b$

$a \rightarrow \lambda$

$b \rightarrow W$

$b \rightarrow \lambda$

▪ ¿Qué hay que hacer?

Equivalencia (I)

Sean las siguientes GLC

$$\begin{array}{l} s \rightarrow s + s \\ \quad | X \end{array}$$

$$s \rightarrow X \text{ mt}$$

$$\text{mt} \rightarrow + X \text{ mt}$$

$$\text{mt} \rightarrow \lambda$$

Equivalencia (II)

Dado un lenguaje hay INFINITAS gramáticas *equivalentes*

- *Pero no todas tienen las mismas propiedades*

Árbol de Análisis Sintáctico (árbol concreto)

Árbol de análisis gramatical o sintáctico

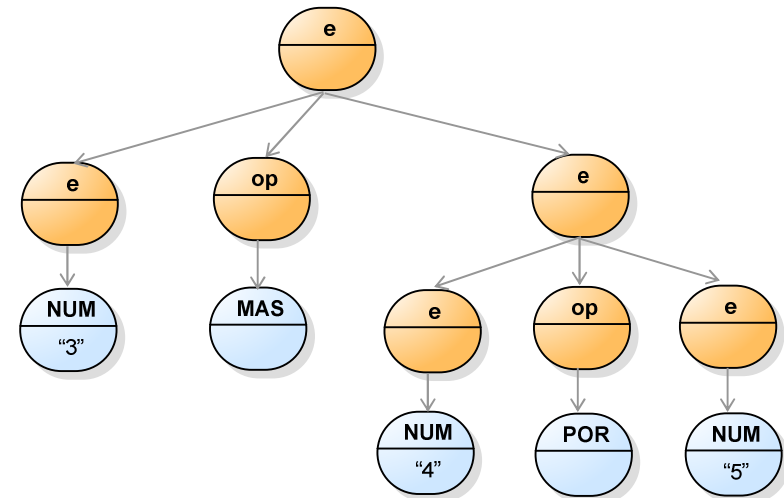
- Registro de las transformaciones realizadas en una derivación
- Muestra las estructuras encontradas en la entrada

Se construye a la vez que se realiza la derivación

- Ejemplo. Derivación de '3 + 4 * 5'

$$\begin{aligned} & \Rightarrow \overbrace{e \quad op \quad e}^e \\ & \Rightarrow \mathbf{NUM} \quad op \quad e \\ & \Rightarrow NUM \quad + \quad e \\ & \Rightarrow NUM \quad + \quad \overbrace{e \quad op \quad e}^e \\ & \Rightarrow NUM \quad + \quad \mathbf{NUM} \quad op \quad e \\ & \Rightarrow NUM \quad + \quad NUM \quad * \quad e \\ & \Rightarrow NUM \quad + \quad NUM \quad * \quad \mathbf{NUM} \end{aligned}$$

$$\begin{aligned} e &\rightarrow e \quad op \quad e \\ e &\rightarrow \mathbf{NUM} \\ op &\rightarrow + \\ op &\rightarrow * \end{aligned}$$



Formato

- El nodo raíz es s
- Nodos hoja $\subset VT$
- Nodos internos $\subset VN$
- Si $hijos(n) = \{x, y, z\}$ entonces ' $n \rightarrow x \ y \ z$ ' $\in P$

Ambigüedad

Gramática Ambigua

GLC

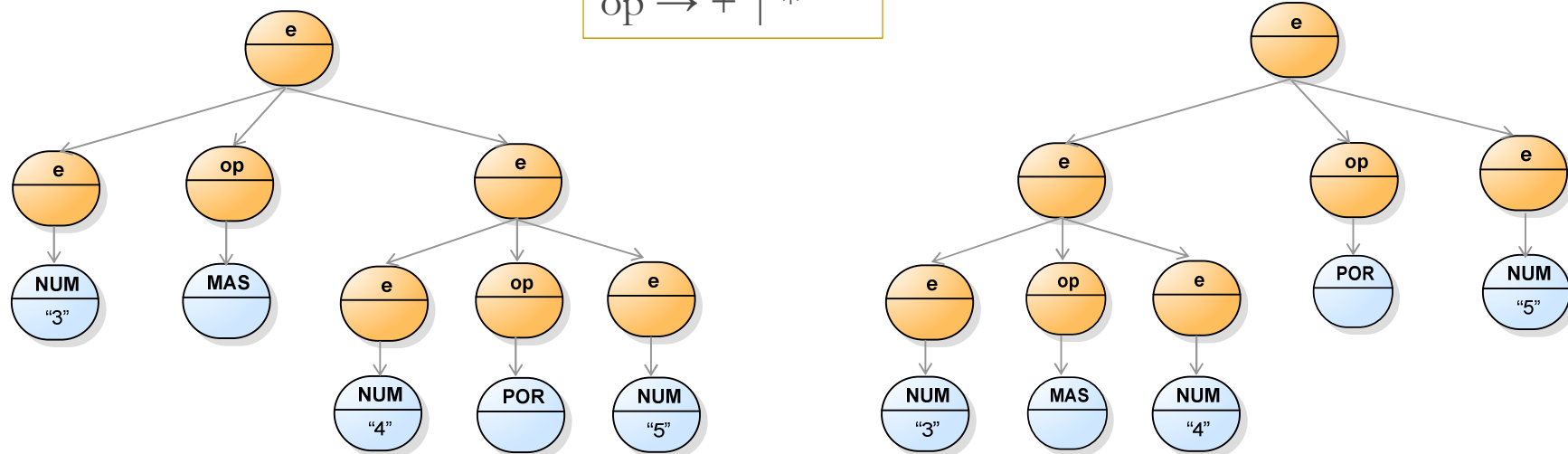
$e \rightarrow \text{NUM}$

$e \rightarrow e \text{ op } e$

$\text{op} \rightarrow + \mid *$

Entrada

$3 + 4 * 5$



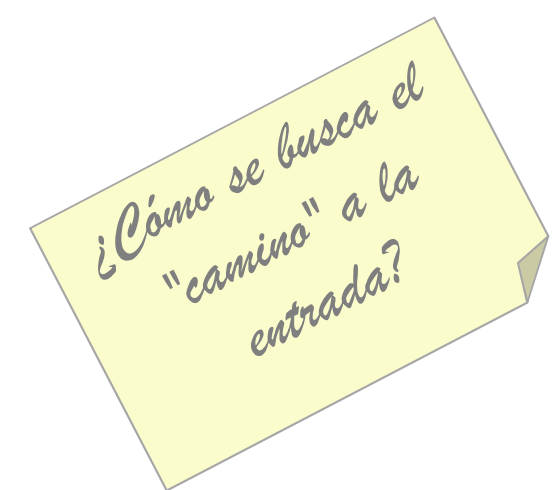
- ❑ Cada sentencia debe tener un solo árbol concreto
 - ¿Por qué?
- ❑ ¿Se pueden detectar?
- ❑ Lenguajes intrínsecamente ambiguos

Resumen

Dada una cadena se quiere saber si es válida (si pertenece al lenguaje)

- Hay que buscar una derivación
 - Si no se encuentra
 - Cadena no válida
 - Si se encuentra
 - En cada paso de derivación se enlaza el no-terminal con los símbolos que lo han sustituido
 - Se obtiene así la estructura de la cadena (árbol concreto)
 - La GLC debe ser no-ambigua para que sólo exista un árbol

Técnicas de Reconocimiento



Técnicas de Reconocimiento

Tarea fundamental de un Analizador Sintáctico

- Encontrar una derivación desde el símbolo inicial hasta la cadena de entrada

Hay dos técnicas fundamentales para hacer esto:

- Top-Down (descendente)
 - Parte del símbolo inicial e intenta llegar a la cadena
- Bottom-Up (ascendente)
 - Parte de la cadena e intenta llegar al símbolo inicial
 - Realiza transformaciones a la inversa

$s \rightarrow A a b E$

$a \rightarrow a B C$

$a \rightarrow B$

$b \rightarrow D$

$b \rightarrow a B C$

¿Pertenece A B B C D E ?

Análisis Descendente. Características

Limitación

- No permite Recursividad a Izquierda

$a \rightarrow a \dots$

Consideraciones para Implementarla

- ¿Cómo elige la regla correcta?

$s \rightarrow b c D$

$b \rightarrow X b$

$b \rightarrow X$

$c \rightarrow \lambda$

$c \rightarrow Y c$

Opción A

s
 $\Rightarrow b c D$
 $\Rightarrow X b c D$
 $\Rightarrow X X b c D$
 $\Rightarrow \dots$

Opción B

s
 $\Rightarrow b c D$
 $\Rightarrow X b c D$
 $\Rightarrow X X c D$
 $\Rightarrow \dots$

Análisis Descendente. Implementación

Alternativas más comunes

- Análisis con Retroceso (Backtracking)
- Análisis Predictivo
 - Usa los k siguientes tokens para determinar la regla correcta

a: b | c;
b: IDENT IDENT
c: CTE CTE

Entrada: 24 25

Con Retroceso

a
⇒ b
⇒ ERROR (retroceder)
⇒ c
...

Predictivo

a (token == CTE)
⇒ c
...

- Se denomina gramática LL(k) a aquella para la cual se puede implementar un analizador que, ante cualquier entrada, puede determinar la regla a aplicar mirando k tokens como máximo
 - Las LL(1) son las más comunes a la hora de implementar
 - Hay gramáticas que no pueden ser reconocidas de esta manera (no existe un k)
- Herramientas que implementan esta técnica
 - ANTLR, JavaCC, Coco/R...

Análisis Ascendente. Características

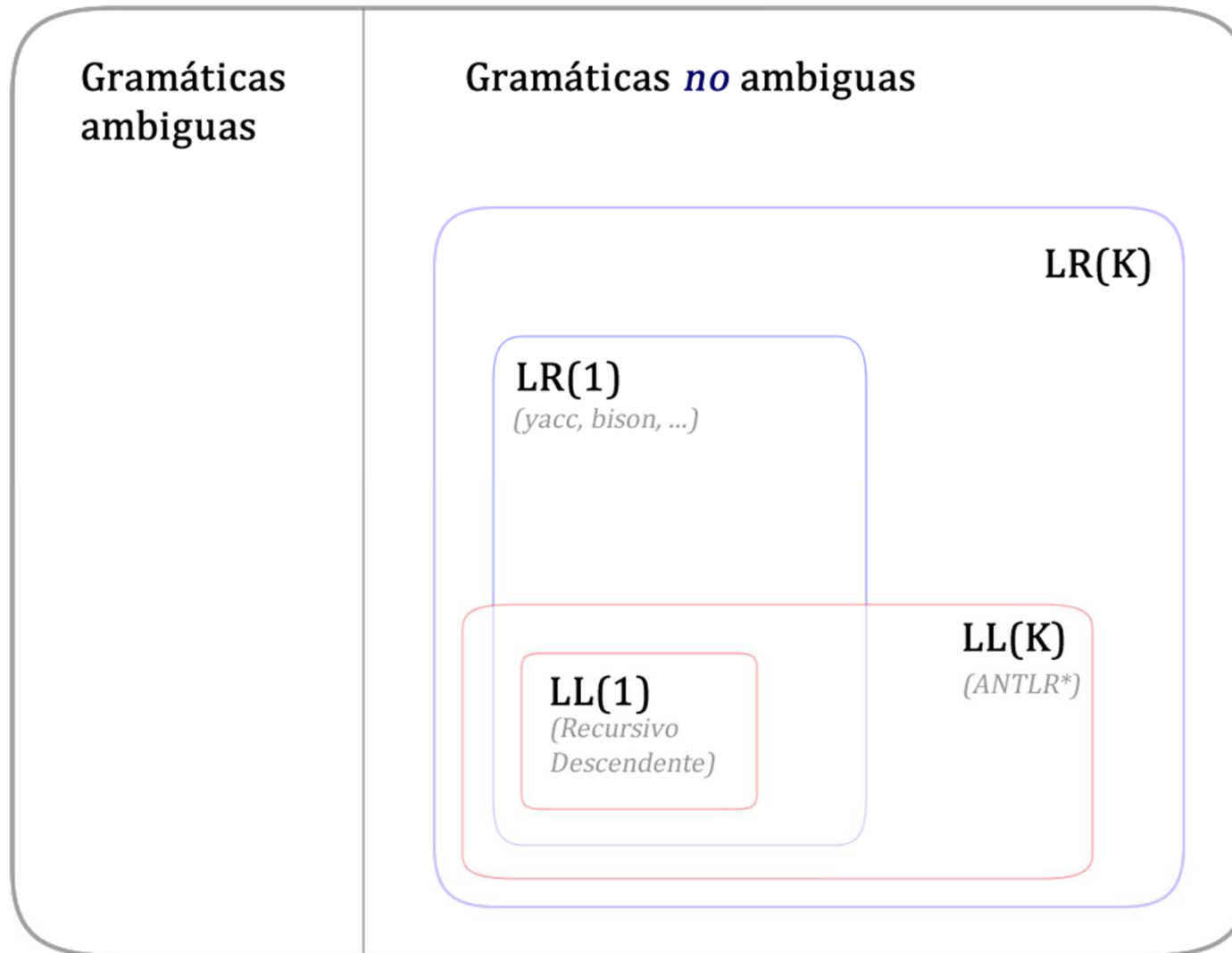
Permite Recursividad a Izquierda

- De hecho, es más eficiente que a derecha

Formas de Implementarla

- Retroceso (Backtracking)
 - Mismos problemas que en ascendente
- Parser LR(k)
 - Usa los k siguientes tokens para determinar la regla correcta a ...
 - Se denomina gramática LR(k) a aquella para la cual se puede implementar un analizador que, ante cualquier entrada, puede determinar la regla a aplicar mirando k tokens como máximo
 - Las LR(1) son las más comunes a la hora de implementar
 - Hay gramáticas que no pueden ser reconocidas de esta manera (no existe un k)
 - Herramientas que implementan esta técnica
 - Yacc, BYaccJ, Bison, Cup, SableCC, ...
 - Permite implementar un rango mayor de Gramáticas
 - Toda gramática LL(k) es LR(k)
 - Por tanto es una técnica más potente que la descendente

Resumen de las Técnicas de Reconocimiento



Análisis Descendente Predictivo (I)

Formas de Implementación

Análisis Descendente Predictivo

Repaso

- Partiendo del símbolo *inicial* (descendente) intenta llegar a la cadena *sin hacer backtracking* (predictivo)
 - Para ello mira los k siguientes tokens
- Lo más sencillo (y común) es implementar un Analizador LL(1)

Formas de Implementar un Analizador Descendente Predictivo

1. Autómata de Pila
2. Técnica Recursiva Descendente
 - Cada regla se implementa mediante una *subrutina*
 - Se encarga de retirar de la entrada los tokens que correspondan a sus derivaciones
 - Se puede obtener la implementación de las reglas:
 - De forma manual
 - Sencillo y práctico para gramáticas pequeñas
 - Con herramienta
 - ANTLR, JavaCC, Coco/R...

Análisis Descendente Predictivo (II)

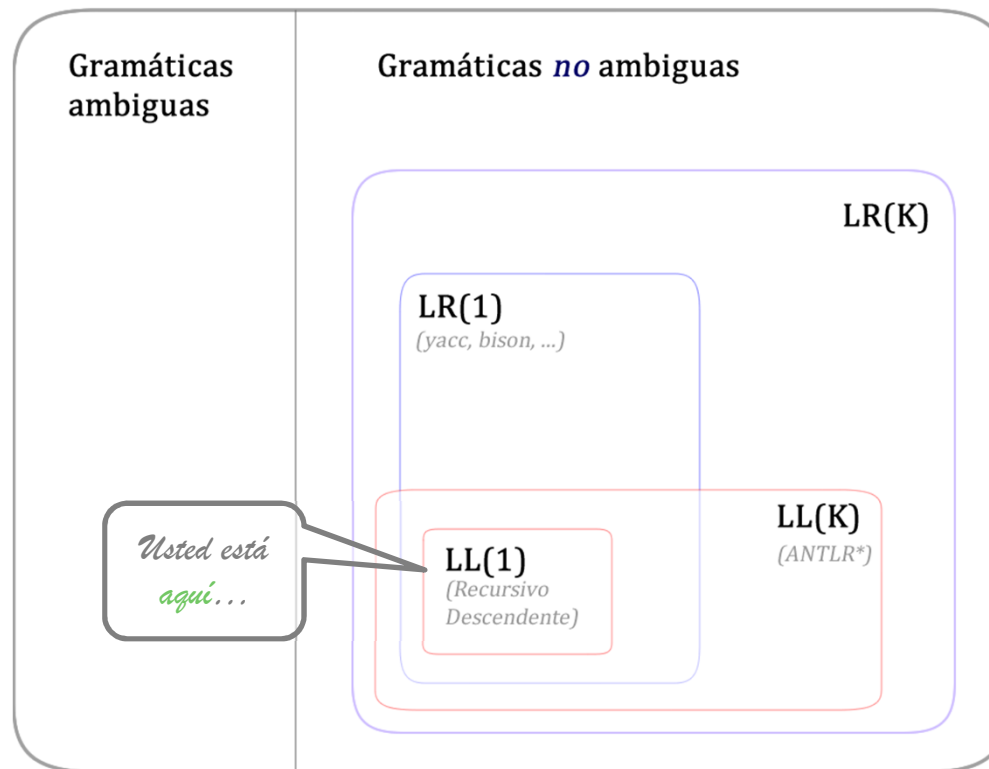
Implementación de un Analizador
Sintáctico de manera Descendente y
Predictiva mediante la Técnica
Recursiva Descendente para
Gramáticas LL(1)



Implementación Recursiva Descendente LL(1)

Característica

- Es la técnica más sencilla de reconocer una entrada...
 - ... si la gramática lo permite.



Estructura Básica

```
public class RecursiveParser {  
    private Lexicon lex;  
    private Token token;  
  
    public RecursiveParser(Lexicon lexico) throws ParseException {  
        lex = lexico;  
        advance();  
    }  
  
    public void start() throws ParseException {  
        // ...  
    }  
  
    private void advance() {  
        token = lex.nextToken();  
    }  
  
    private void error() throws ParseException {  
        throw new ParseException("Error sintáctico");  
    }  
}
```

SM/1

Análisis Descendente Predictivo (III)

Implementación de un Analizador
Sintáctico de manera Descendente y
Predictiva mediante una Herramienta
de generación de código (ANTLR4)

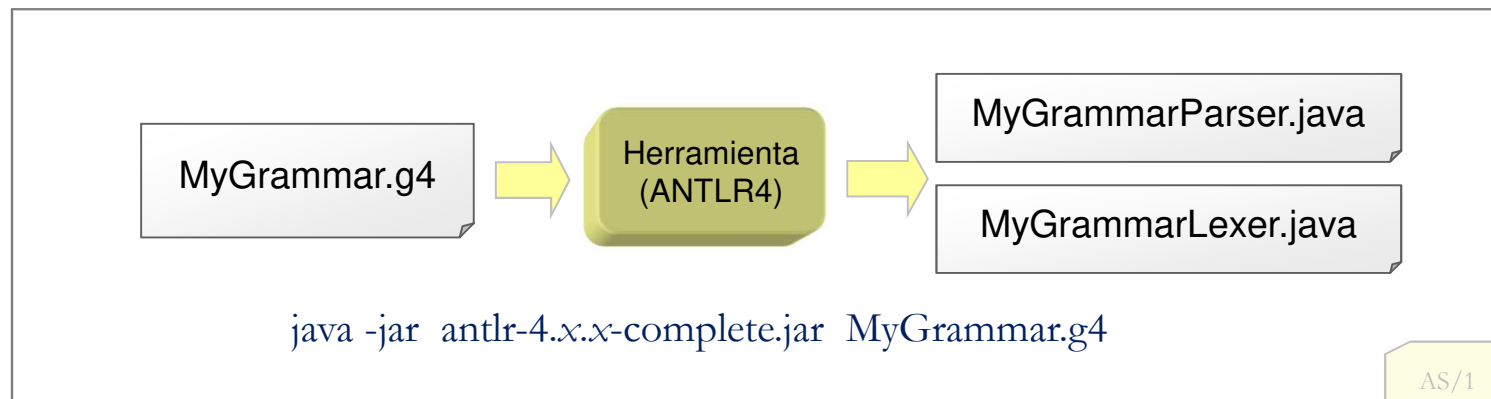
ANTLR

Características

- Herramienta que genera código utilizando la Técnica Recursiva Descendente
 - Con algunas mejoras
 - Transformación de ciertas estructuras habituales
 - Recuperación de errores

<https://www.antlr.org/>

<https://www.antlr.org/download/antlr-4.7.2-complete.jar>



ANTLR. Sintaxis

```
grammar Grammar;  
import Lexicon;  
  
start  
    : INT_CONSTANT EOF;
```

```
lexer grammar Lexicon;  
  
INT_CONSTANT  
    : [0-9]+;  
  
WHITESPACE  
    : [ \t\r\n]+ -> skip;
```

```
public static void main(String[] args) throws Exception {  
    GrammarLexer lexer = new GrammarLexer(CharStreams.fromFileName(program));  
    GrammarParser parser = new GrammarParser(new CommonTokenStream(lexer));  
    parser.start();  
}
```

Creación de Gramáticas

Creación de Gramáticas

Análisis Sintáctico

- Para saber si una sentencia pertenece al lenguaje hace falta una gramática
- ¿Cómo se crea una gramática?
 - Identificando las construcciones básicas en el lenguaje

Creación de Gramáticas

Construcciones básicas

- Secuencias
 - Indican el orden el que deben aparecer los componentes de la estructura

```
asignación: IDENT = expr ';' 
```

- Listas
 - Indican que la estructura se forma repitiendo otra estructura

```
instrucciones: instrucción  
              | instrucciones instrucción
```

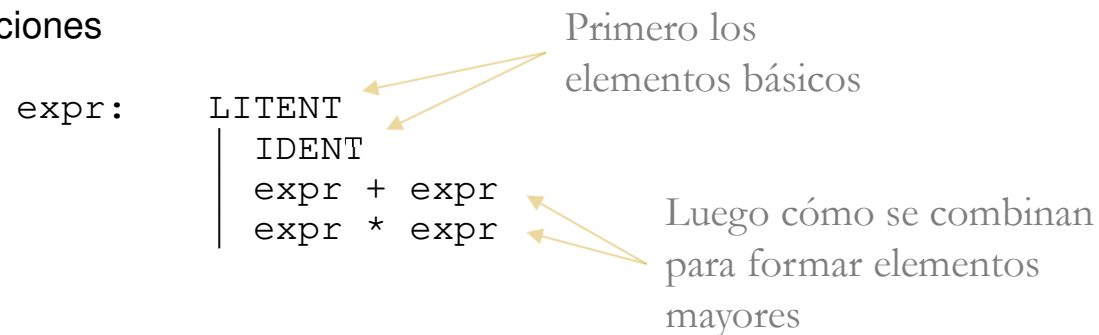
- Forman un árbol desequilibrado a izquierda o a derecha

- Composiciones

```
expr: LITENT  
      | IDENT  
      | expr + expr  
      | expr * expr
```

Primero los elementos básicos

Luego cómo se combinan para formar elementos mayores



Ejercicio E1

Hacer una GLC que genere el siguiente lenguaje

- Un conjunto está formado por uno o más elementos entre paréntesis
- Cada elemento puede ser un número u otro conjunto
- Los números están formados por dígitos de 1 al 3
- Los números están formados por un número impar de dígitos y son palíndromos

(131)

(3 222 12321)

(12121 2 (333) (2 3 1111111))

(2132312 ((1 2) 2 131) 1 2) 3322233)