# Procesadores de Lenguajes

**Ingeniería Técnica superior de Ingeniería Informática**

Departamento de Lenguajes y Sistemas informáticos

3

# *Análisis sintáctico*

## *Gramáticas libres de contexto*

*Javier Vélez Reyes*
*jvelez@lsi.uned.es*
*Departamento de Lenguajes Y Sistemas Informáticos*
*UNED*

UNED

ETS de Ingeniería Informática

## Objetivos

- Conocer los lenguajes libres de contexto
- Conocer las responsabilidades del analizador sintáctico
- Aprender a utilizar gramáticas libres de contexto
  - Aprender a diseñar gramáticas libres de contexto
  - Aprender a reconocer y resolver fuentes de ambigüedad
  - Aprender a eliminar la recursividad por la izquierda
  - Aprender a factorizar por la izquierda
- Conocer el uso y tipos de derivaciones y árboles sintácticos
- Conocer la representación de lenguajes mediante diagramas de sintaxis
- Presentar los diferentes tipos de autómatas utilizados para construir analizadores sintácticos
- Discutir los diferentes tipos de analizadores sintácticos

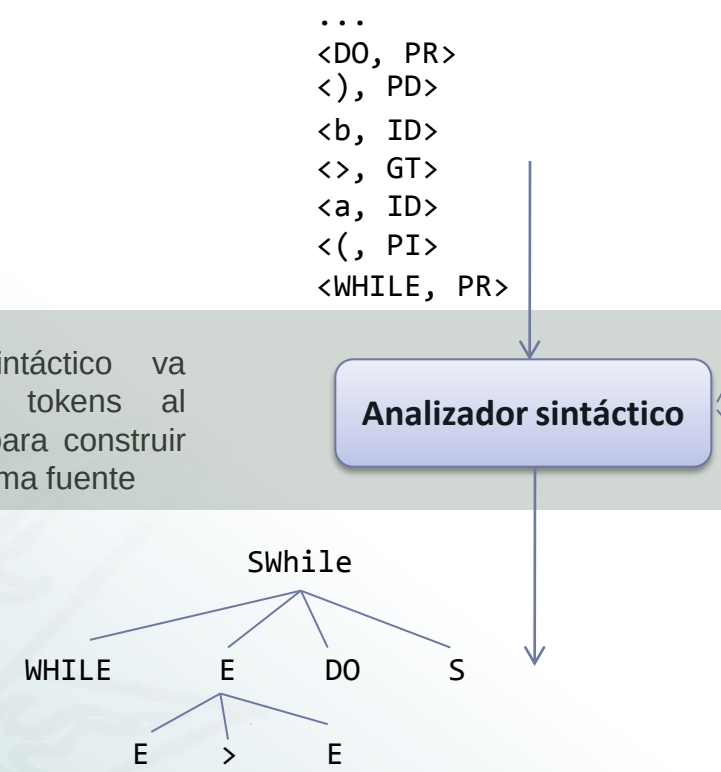
## Índice

## Análisis sintáctico

La fase de análisis sintáctico tiene por objetivo solicitar tokens al analizador léxico y construir una representación arborescente de todo el código fuente. Este proceso se encuentra dirigido por el conocimiento gramatical que del lenguaje tiene el analizador sintáctico

**Foco de atención**

El analizador sintáctico va pidiendo nuevos tokens al analizador léxico para construir un árbol del programa fuente




**Tema 3**

**¿Cómo se especifica formalmente un analizador sintáctico?**

- Formalismos
- Tratamiento de formalismos
- Conversión entre formalismos

**Temas 4 y 5**

**¿Cómo se implementa un analizador sintáctico?**

- Estrategias análisis sintáctico
- Tipos de gramáticas
- Tipos de analizadores

## Lenguajes libres de contexto

Desde una perspectiva sintáctica un lenguaje es una colección de construcciones sintácticas bien formadas desde un alfabeto de entrada y correctamente combinadas entre sí de acuerdo a una colección de reglas sintácticas

### Ejemplo

El lenguaje Pascal tiene unas normas de carácter gramatical que definen su naturaleza y expresividad. Visto de manera extensiva, el lenguaje Pascal sería el conjunto infinito de todos los posibles códigos fuente correctos escritos en sintaxis Pascal

**Pascal**

```
Uses crt;
var
 cantida
begin
 clrScr;
 s:=0;
 write('
 read(ca
 for con
  begin
   write
   Read(
   s:=s+
  end;
 write('
 readKey
end;.
```

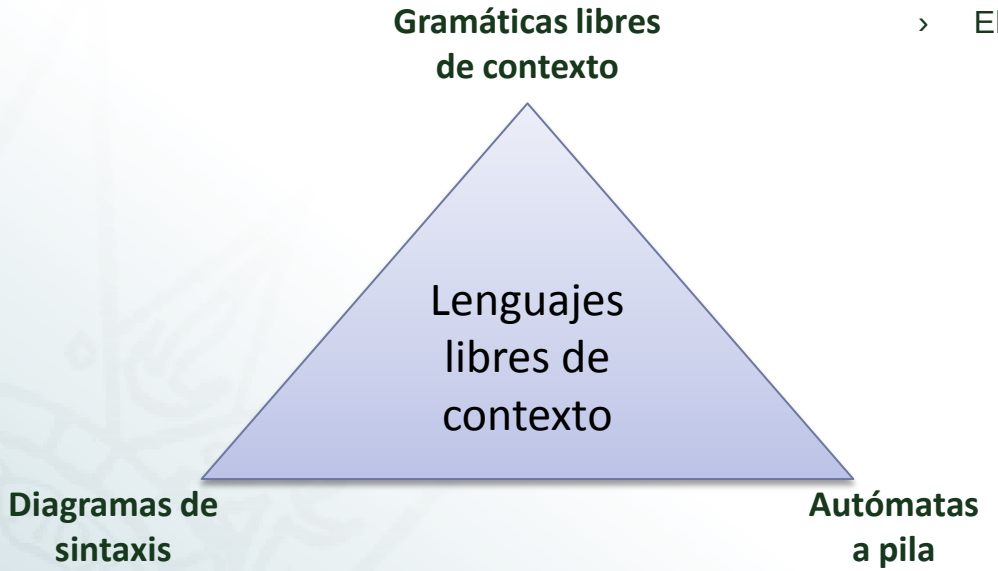
```
uses
 crt;
var
 contador
begin
 ClrScr;
 for conta
begin
   contado
   Write(c
   if cont
    begin
    end;
  end;
 ReadKey;
end.
```

...

```
Uses crt;
var
 suma,numero,contador:integer;
begin
 clrScr;
 write('Numero: ');
 readln(numero);
 suma:=0;
 for contador:=1 to numero do
  begin
   suma:=suma + contador;
  end;
  write('Suma: ',suma);
 readKey;
end.
```

## Especificación de lenguajes libres de contexto

Existen 3 diferentes maneras de definir formalmente un lenguaje de contexto libre. A lo largo de esta sección estudiaremos cada una de ellas en detalle y veremos cómo se puede pasar de cada una a las otras 2




- Elementos
  - Gramáticas libres de contexto
  - Diagramas de sintaxis
  - Autómatas a pila

## Especificación mediante gramáticas libres de contexto

De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

### Definición de gramática libre de contexto

Una gramática libre de contexto es un conjunto de 4 elementos G = (T, N, S, P) donde:

- T es un conjunto de símbolos terminales
- N es un conjunto de símbolos no terminales
- S Є N axioma gramatical
- P un conjunto de reglas de producción de la forma
  1. A ::= X donde X es una secuencia de elementos en N U T
  2. A ::= x donde x es una secuencia de elementos en T
  3. A :: = siendo la cadena vacía

A veces T se elide, N se deduce de los antecedentes de las reglas de P y se asume que el antecedente de la primera regla es S con lo G sólo a través de P

### Ejemplo

L = Lenguaje de operadores

Sea G = (T, N A, P) con

T = {+, -, *, /, n}

N = {E}

P = {

E ::= E + E

E ::= E - E

E ::= E * E

E ::= E / E

E ::= n

}

## Especificación mediante gramáticas libres de contexto

De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

### Notación

**¿Cómo se escriben las reglas?**

**Notación estándar**

Las reglas se escriben como A → X. Cada regla se escribe en una linea distinta

```
E → E + E
E → E – E
E → id
```

**Notación BNF**

Las reglas se escriben como A ::= X. Las reglas de un mismo antecedente se escriben como A ::= X | Y | Z… Cada regla con antecedente distinto se escribe en una nueva línea

```
D ::= T L
T ::= int | bool | char
L ::= id , L | id
```

**Foco de atención**

**Notación EBNF**

Se usa la notación BNF. Las reglas recursivas se pueden esquematizar con los meta-caracteres { y }

```
D ::= T L
T ::= int | bool | char
L ::= { id , } id
```

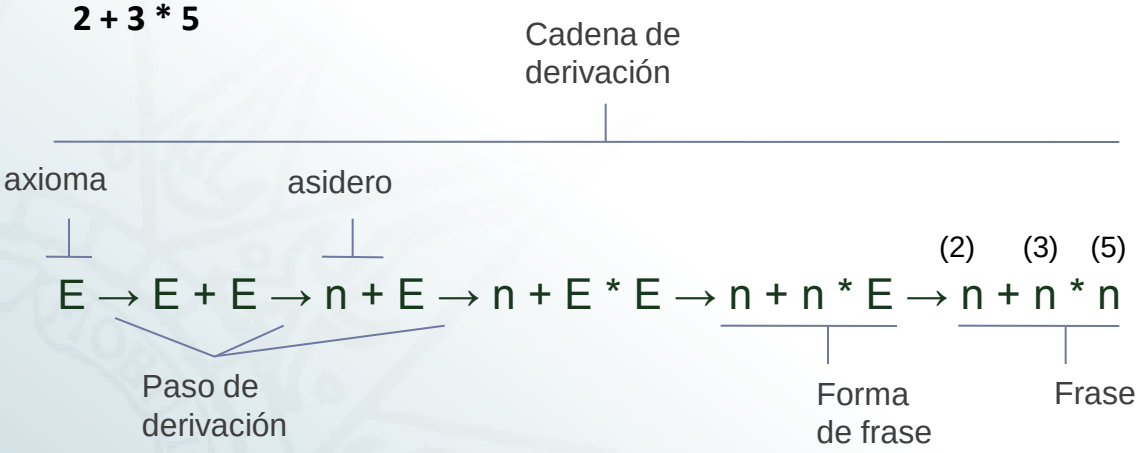
## Especificación mediante gramáticas libres de contexto

De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

### Derivación gramatical

Se plantea el siguiente problema de decisión: Dado un lenguaje descrito a través de una gramática G, L(G) determinar si la colección de terminales x pertenece o no al lenguaje al mismo. Esto es ¿x Є L (G)?

**2 + 3 * 5**




L = Lenguaje de operadores

Sea G = (T, N A, P) con

T = {+, -, *, /, n}

N = {E}

P = {

E ::= E + E

E ::= E - E

E ::= E * E

E ::= E / E

E ::= n

}

## Especificación mediante gramáticas libres de contexto

De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

### Derivación gramatical

Se plantea el siguiente problema de decisión: Dado un lenguaje descrito a través de una gramática G, L(G) determinar si la colección de terminales x pertenece o no al lenguaje al mismo. Esto es ¿x Є L (G)?

**Frase** Secuencia de terminales que pertenece al lenguaje definido por la gramática. Si x Є L (G), x es frase de L

**Forma de frase** Secuencia de terminales y no terminales que representa una colección de frases del lenguaje. Existe una cadena de derivación que llega hasta ella

**Axioma** Forma de frase más abstracta que representa a L (cualquier derivación de Si x Є L (G) comienza por el axioma)

**Paso de derivación** Aplicación de una regla de producción que transforma una forma de frase más general en otra más especifica para x

**Cadena de derivación** Secuencia de pasos de derivación que parten del axioma gramatical para intentar alcanzar la frase analizada x. Demostración de x Є L (G)

**Asidero** Parte izquierda de cualquier forma de frase de x Є L (G) formada únicamente por símbolos terminales

## Especificación mediante gramáticas libres de contexto

De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

### Derivación gramatical

Se plantea el siguiente problema de decisión: Dado un lenguaje descrito a través de una gramática G, L(G) determinar si la colección de terminales x pertenece o no al lenguaje al mismo. Esto es ¿x Є L (G)?

**¿ En qué orden se aplican las reglas?**

**Left Most Derivation**

En cada paso de derivación se escoge el no terminal más a la izquierda de la forma de frase a derivar

**Right Most Derivation**

En cada paso de derivación se escoge el no terminal más a la derecha de la forma de frase a derivar

$E \rightarrow E + E \rightarrow$

$n + E \rightarrow$

$n + E * E \rightarrow$

$n + n * E \rightarrow$

$n + n * n$

$E \rightarrow E + E \rightarrow$

$E + E * E \rightarrow$

$E + E * n \rightarrow$

$E + n * n \rightarrow$

$n + n + n$

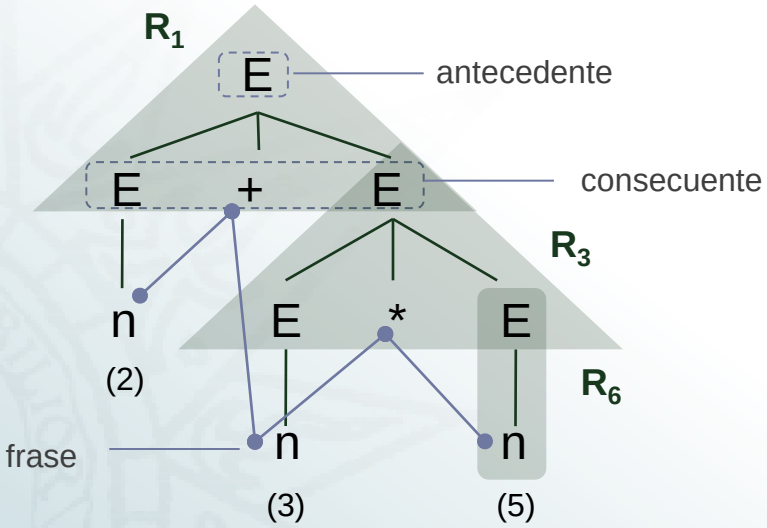
## Especificación mediante gramáticas libres de contexto

De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

### Árboles de análisis sintáctico y arboles sintácticos abstractos

Frecuentemente no es preciso reflejar el orden de aplicación de las derivaciones sino solamente el proceso de derivación desde el axioma a la frase final. Para ello se interpreta cada regla como un árbol con raíz en el antecedente e hijos los símbolos terminales y no terminales del consecuente de la regla

**2 + 3 * 5**




L = Lenguaje de operadores

Sea G = (T, N A, P) con

T = {+, -, *, /, n}

N = {E}

P = {

$R_1$: E ::= E + E

$R_2$: E ::= E - E

$R_3$: E ::= E * E

$R_4$: E ::= E / E

$R_5$: E ::= ( E )

$R_6$: E ::= n

}

## Especificación mediante gramáticas libres de contexto

De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

### Árboles de análisis sintáctico y arboles sintácticos abstractos

Existen dos tipos de representaciones arborescentes que se utilizan frecuentemente para representar las derivaciones de una frase de un lenguaje.

**Tipos de árboles sintácticos**

**Arboles de análisis sintáctico**

Los hijos de cada regla son una representación exacta de la colección de terminales y no terminales que aparecen en el consecuente de la misma

**Foco de atención de la asignatura**

**Arboles sintácticos abstractos**

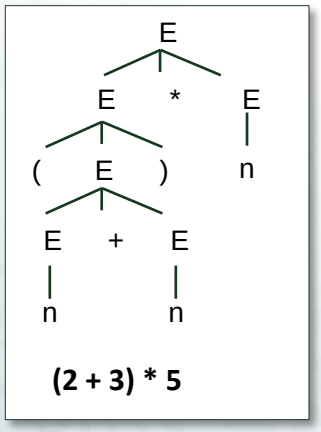
Se hace una representación abstracta de la gramática de tal manera que en los hijos del árbol solo aparecen los símbolos semánticamente relevantes

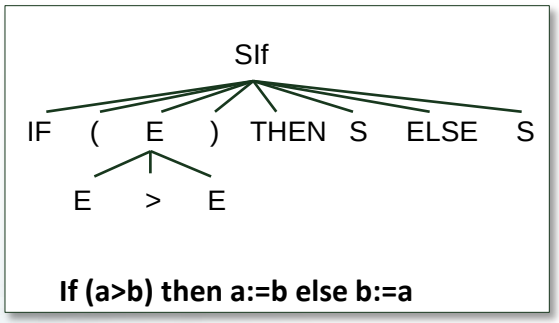
## Especificación mediante gramáticas libres de contexto

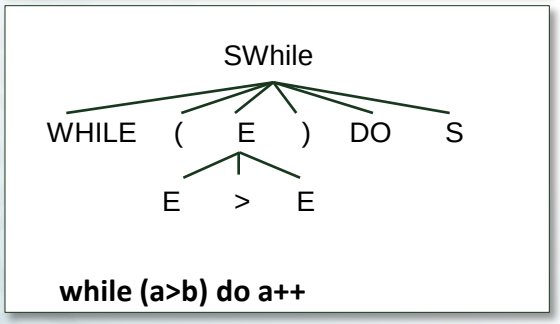
De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

### Árboles de análisis sintáctico y arboles sintácticos abstractos

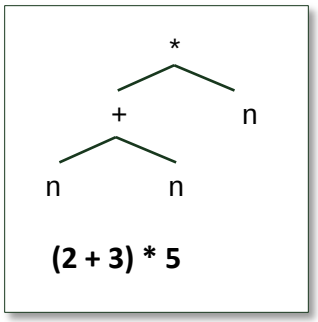
**Arboles de análisis sintáctico**

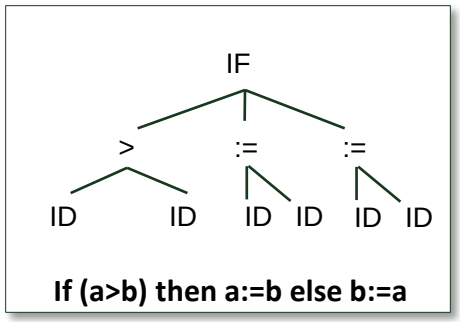


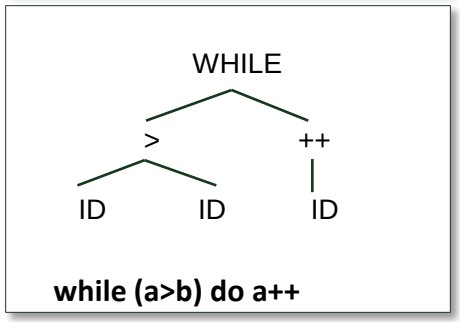




**Arboles sintácticos abstractos**



## Especificación mediante gramáticas libres de contexto

De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

**Ejercicios**

- L1. Declaración de variables en Pascal
- L2. Declaración de estructuras en C
- L3. Declaración de procedimientos en Pascal
- L4. Declaración de procedimientos en C
- L5. Declaración de sentencias en C
- L6. Declaración de sentencias en Pascal
- L7. Sentencia for en C
- L8. Sentencia Repeat – Until en Pascal

L5

```
Sentencia ::= BloqueSentencias |
              SentenciaIf |
              Sentencia While
BloqueSentencias ::= { listaSentencias }
listaSentencias ::= sentencia ; ListaSentencias |

SentenciaIf ::= …
SentenciaWhile ::= …
```

Construya arboles de análisis sintáctico para frases en cada uno de los lenguajes anteriores

## Especificación mediante gramáticas libres de contexto

De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

### Limpieza gramatical

Para no tener problemas en la construcción de analizadores sintácticos es conveniente aplicar ciertas operaciones de trasformación gramatical. Estos problemas son:

**Limpieza gramatical**

**Eliminación de la ambigüedad**

Es necesario eliminar las expresiones gramaticales que generen una interpretación potencialmente gramatical ambigua

**Eliminación de la recursividad a izquierdas**

En algunos casos las reglas con recursividad a izquierdas de la forma A ::= A α pueden acarrear problemas en la construcción de analizadores

**Factorización por la izquierda**

Las reglas con partes comunes a la izquierda de los consecuentes pueden también generar problemas, en función del analizador que se construya

## Especificación mediante gramáticas libres de contexto

De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

$R_1$: E ::= E + E

$R_2$: E ::= E - E

$R_3$: E ::= E * E

$R_4$: E ::= E / E

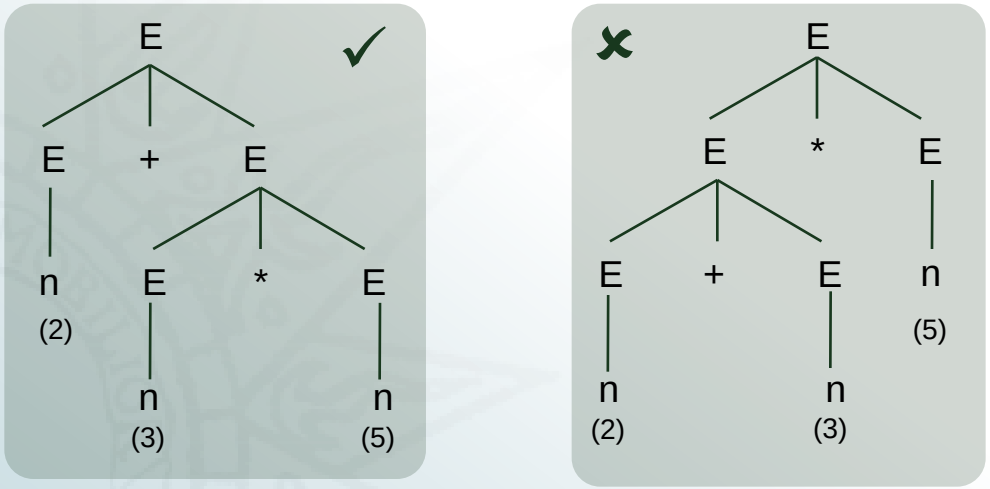
$R_5$: E ::= ( E )

$R_6$: E ::= n

### Ambigüedad gramatical

Una gramática es inherentemente ambigua cuando es posible construir dos arboles de derivación para al menos una frase del lenguaje

*Interpretación 1* **2 + 3 * 5** *Interpretación 2*




**¿Cuál es el problema?**

Ambos árboles son sintácticamente correctos pero responden a interpretaciones semánticamente diferentes, una válida y la otra no, con respecto a la semántica operacional matemática

## Especificación mediante gramáticas libres de contexto

De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

### Fuentes de ambigüedad gramatical

Para demostrar que una gramática es ambigua es suficiente con encontrar dos árboles de derivación diferentes para una misma frase. Sin embargo no existe ninguna manera de demostrar que una gramática no es ambigua. Existen criterios heurísticos que identifican fuentes de potencial ambigüedad

Gramáticas con ciclos

```
S ::= A
S ::= a
A ::= S
```

Caminos alternativos

```
S := A
S := B
A := B
```

Reglas recursivas con ε en casos base

```
S ::= HRS
S ::= s
H ::= h|ε
R ::= r|ε
```

Reglas con igual principio y fin

```
E ::= E…E
```

No terminales que derivan ε

```
S ::= HR
H ::= h|ε
H ::= h|ε
```

## Especificación mediante gramáticas libres de contexto

De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

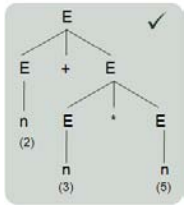
### Corrección de la ambigüedad gramatical

La forma gramatical puede inducir una interpretación semánticamente invalidada por ello es necesario introducir correcciones para evitar la ambigüedad

**Corrección de la ambigüedad gramatical**

**Corrección explicita**

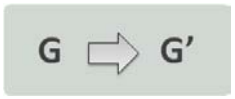
Se mantiene la gramática ambigua pero se añaden criterios para indicar al compilador el árbol sintáctico que debe construir en cada caso




La resolución explicita hace escoger este árbol gramatical

**Alteración gramatical**

Se modifica la gramática para conseguir otra equivalente que no presente ambigüedad.




Dos gramáticas son equivalentes si reconocen el mismo lenguaje

**Ambigüedad no esencial**

Hay situaciones en las que la ambigüedad no presenta problemas para construir analizadores sintácticos

```
listaSent ::= listaSent ; listaSent |
              sentencia
```

# Especificación mediante gramáticas libres de contexto

De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

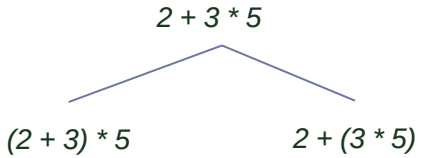
## Casos típicos de ambigüedad gramatical

Dado que resulta imposible de resulta imposible ofrecer un algoritmo para descubrir y remediar la ambigüedad gramatical analizaremos casos arquetípicos de ambigüedad que suelen aparecen en los lenguajes de programación

**Estudio de casos típicos de ambigüedad**

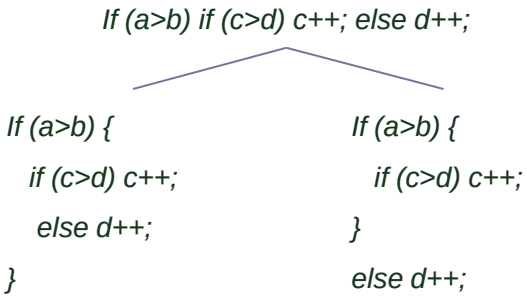
### Gramática de operadores

La gramática canónica de operadores es inherentemente ambigua ya que no se sabe el orden en que deben aplicarse los operadores




### El problema del else ambiguo

Cuando se anidan una sentencia if con otra if–else es imposible saber a qué if emparejar el else.



## Especificación mediante gramáticas libres de contexto

De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

### Corrección de la ambigüedad en gramáticas de operadores

Las gramáticas canónicas de operadores son inherentemente ambiguas. Para corregir este hecho es necesario inyectar información sobre la precedencia y la asociatividad de los operadores

Gramática canónica de operadores sobre operadores {@, #, $, %}

E ::= E @ E

E ::= E # E

E ::= E & E

E ::= E % E

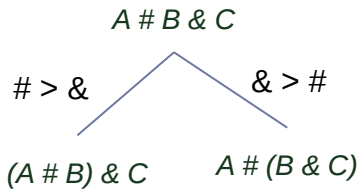
E ::= ( E )

E ::= n

**Alteraciones gramaticales**

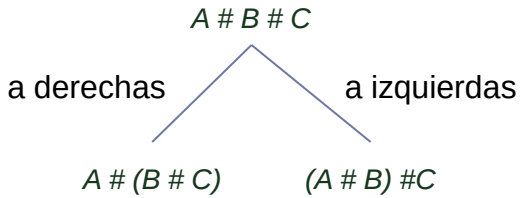
**Precedencia de operadores**

La precedencia de operadores indica la prioridad de aplicación de los operadores para combinar subexpresiones en una mayor




**Asociatividad de operadores**

Cuando tres o mas subexpresiones se combinan a través de un mismo operador tiene sentido definir la asociatividad del mismo



## Especificación mediante gramáticas libres de contexto

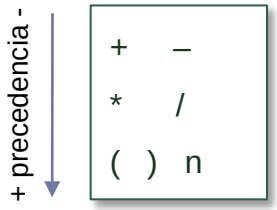
De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

### Corrección de la ambigüedad en gramáticas de operadores

Las gramáticas canónicas de operadores son inherentemente ambiguas. Para corregir este hecho es necesario inyectar información sobre la precedencia y la asociatividad de los operadores

**Precedencia de operadores**

Se ordena en una tabla los niveles de prioridad. Para cada nivel se define un nuevo no terminal. Cada nivel incrementa en 1 la distancia al axioma

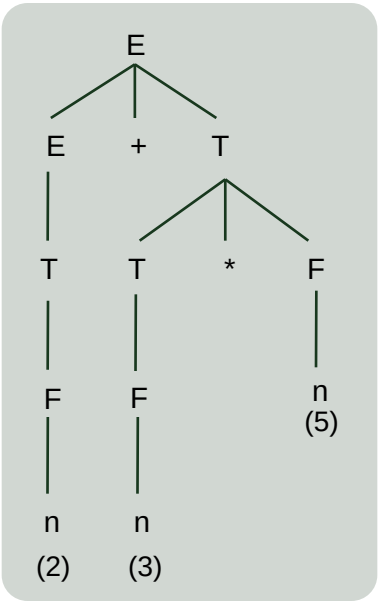



E ::= E + E
E ::= E – E
E ::= E * E
E ::= E / E
E ::= ( E )
E ::= n

Gramática de operadores no ambigua

E ::= E + T | E – T | T
T ::= T * F | T / F | F
F ::= ( E ) | n

**2 + 3 * 5**



# Especificación mediante gramáticas libres de contexto

De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

## Corrección de la ambigüedad en gramáticas de operadores

Las gramáticas canónicas de operadores son inherentemente ambiguas. Para corregir este hecho es necesario inyectar información sobre la precedencia y la asociatividad de los operadores

### Asociatividad de operadores

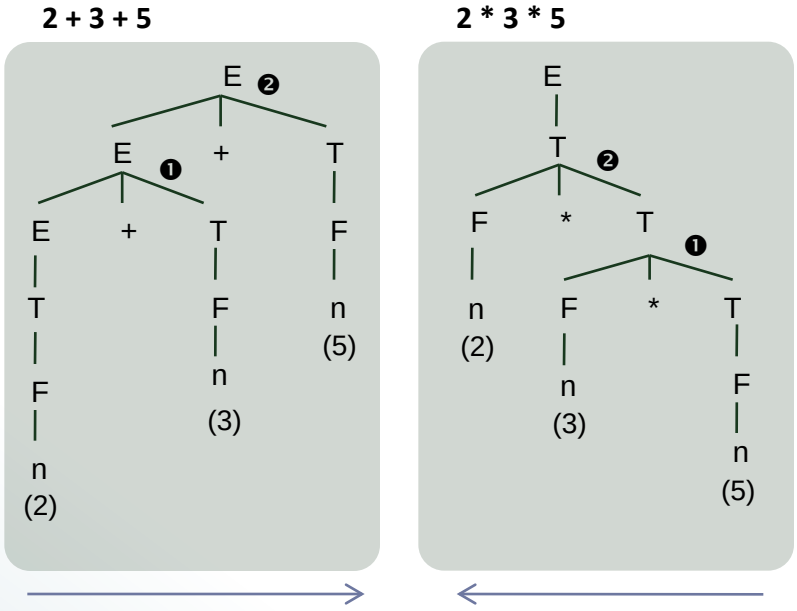
La recursividad a izquierdas marca asociatividad a izquierdas. La recursividad a derecha marca asociatividad a derechas

Gramática de
operadores no ambigua

E ::= E + T | E – T | T

T ::= F * T | T / F | F

F ::= ( E ) | n



## Especificación mediante gramáticas libres de contexto

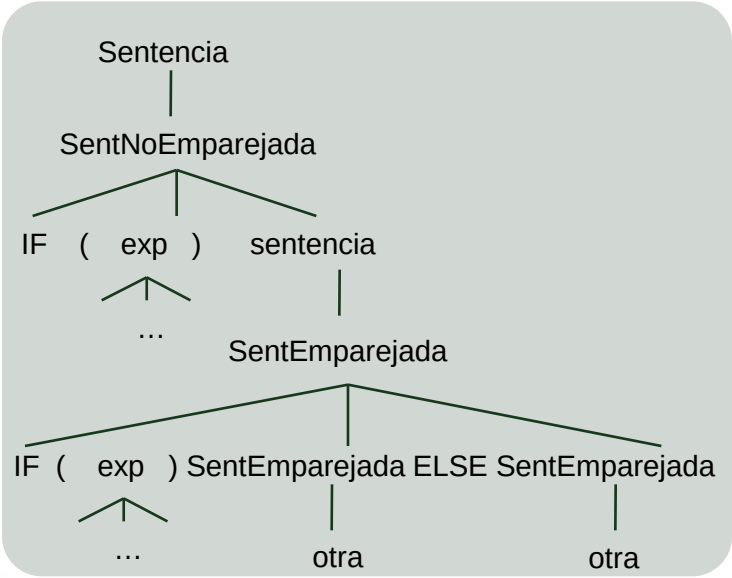
De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

### El problema del else ambiguo

De las dos interpretaciones que presentábamos antes la correcta es la que asocia el else al if mas cercano (esta regla se llama regla de anidamiento más cercano. Veamos cómo es la gramática original y cómo debe transformarse

```
sentencia ::= sentenciaIf | otra
sentenciaIf ::= IF ( exp ) sentencia |
                IF ( exp ) sentencia ELSE sentencia
```

```
sentencia ::= sentEmparejada | sentNoEmparejada
sentEmparejada ::=  IF ( exp ) sentEmparejada ELSE sentEmparejada |
                    otra
sentNoEmparejada ::=  IF ( exp ) sentencia |
                      IF ( exp ) sentEmparejada ELSE sentNoEmparejada
```



## Especificación mediante gramáticas libres de contexto

De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

### Eliminación de la recursividad por la izquierda

Las reglas con recursividad por la izquierda de la forma A ::= Aα | β pueden presentar problemas a la hora de construir analizadores sintácticos por tanto es conveniente reformularlas equivalentemente

**Recursividad directa**

A ::= Aα | β

A ::= αA'

A' ::= β A' |

**Ejemplo**

*Gramática de operadores no ambigua*

E ::= E + T | E – T | T

T ::= T * F | T / F | F

F ::= ( E ) | n

*Gramática de operadores LL (1)*

E ::= TE'

E' ::= + TE' | – TE' |

T ::= FT'

T' ::= * FT' | / FT' |

F ::= ( E ) | n

## Especificación mediante gramáticas libres de contexto

De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

### Eliminación de la recursividad por la izquierda

Las reglas con recursividad por la izquierda de la forma A ::= Aα | β pueden presentar problemas a la hora de construir analizadores sintácticos por tanto es conveniente reformularlas equivalentemente

**Recursividad indirecta**

$A \rightarrow^{+} A$

```
Numerar los no terminales A1, A2, … An
for i := 1 to n do
  for j := 1 to i – 1 do
     Sustituir cada Ai ::= Aj γ por
     Ai ::= δ1 γ | δ2 γ | ... | δk γ donde
     Aj ::= δ1 | δ2 | ... | δk son todas las
     reglas actuales de Aj
  eliminar recursividad por la izquierda de Ai
```

## Especificación mediante gramáticas libres de contexto

De manera similar a como ocurre en los lenguajes regulares, los lenguajes libres de contexto se pueden expresar mediante el uso de gramáticas libres de contexto. Su definición es una extensión de las gramáticas regulares con reglas de producción potencialmente más complejas

### Factorización por la izquierda

Las reglas con factores comunes por la izquierda en sus consecuentes de la forma $A ::= \alpha \beta_1 \mid \alpha \beta_2$ pueden generar problemas para construir analizadores sintácticos y conviene transformarlas

$$A ::= \alpha \beta_1 \mid \alpha \beta_2 \mid \ldots \mid \alpha \beta_n \mid \gamma$$

$$A ::= \alpha A' \mid \gamma$$

$$A' ::= \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

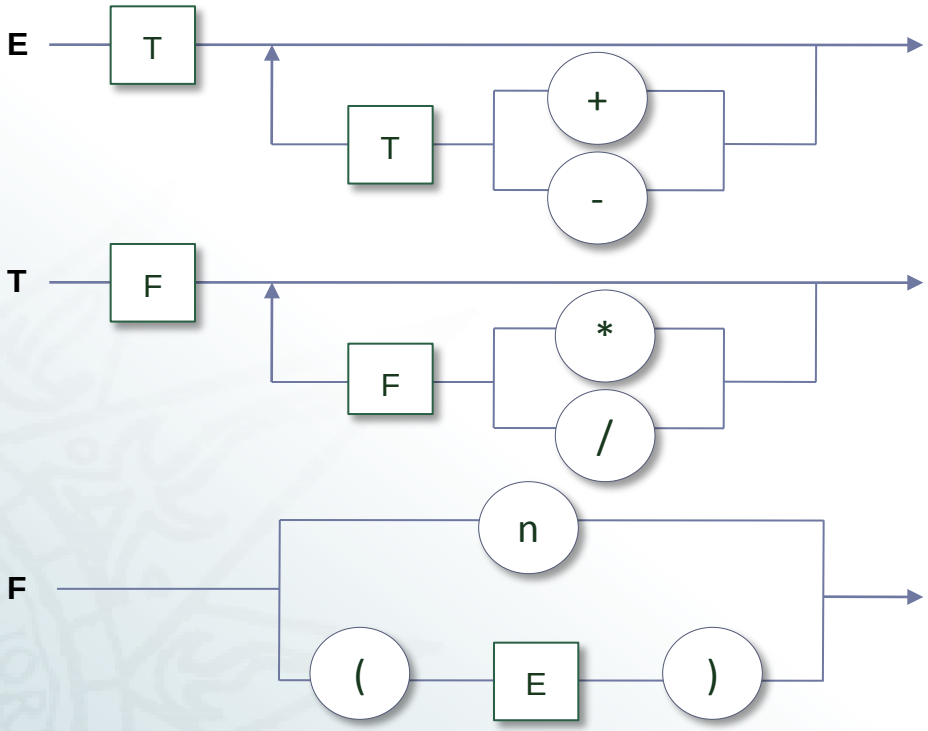
**Ejemplo**

SentenciaIf ::= IF ( exp ) THEN sentencia |

IF ( exp ) THEN sentencia ELSE sentencia

## Especificación mediante diagramas de sintaxis

Los diagramas de transición son una forma gráfica de representar las restricciones sintácticas de los lenguajes libres de contexto. Cada diagrama de sintaxis está etiquetada con el nombre de un no terminal al que representa y se compone de cajas y esferas que representan terminales y no terminales unidas por flechas que indican secuencias y selecciones .




```
E ::= E + T  |
      E - T  |
      T
T ::= T * F  |
      T / F  |
      F
F ::= ( E )  |
      n
```

La equivalencia entre gramáticas y diagramas de sintaxis resulta evidente

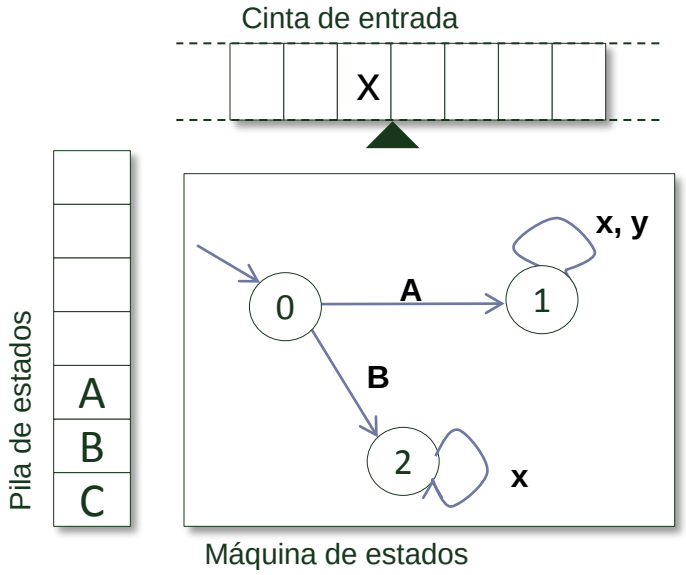
## Especificación mediante autómatas a pila deterministas

Los autómatas a pila son una extensión con memoria de los autómatas finitos deterministas que se utilizan para realizar analizadores sintácticos. Constan de una cinta de entra donde se leen los tokens del programa fuente, una pila de estados (la memoria) y una máquina de estados. Las transiciones de estados dependen ahora de la entrada y de la historia del proceso de análisis almacenada en la pila

### Definición de autómata a pila determinista

Un autómata a pila es una tupla AP = (T, P, Q, $A_0$, $q_0$, f, F) donde

- T es un alfabeto de símbolos terminales de entrada
- P es un alfabeto de estados de la pila
- Q es un conjunto de estados finito no vacío
- $A_0$ Є P es el simbolo inicial en la pila
- $q_0$ Є Q es un estado inicial o estado de arranque
- f: Q x T U { } x P → Q x P una función de transición
- F C Q es un subconjunto de estados finales de aceptación

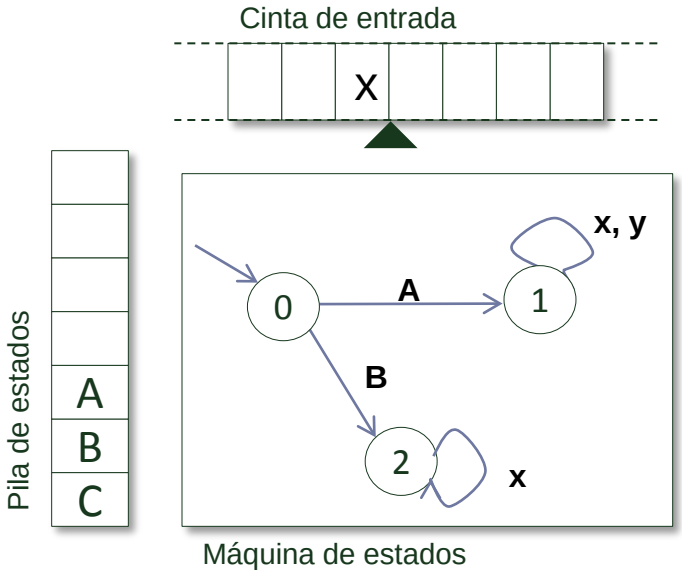


## Especificación mediante autómatas a pila deterministas

Los autómatas a pila son una extensión con memoria de los autómatas finitos deterministas que se utilizan para realizar analizadores sintácticos. Constan de una cinta de entra donde se leen los tokens del programa fuente, una pila de estados (la memoria) y una máquina de estados. Las transiciones de estados dependen ahora de la entrada y de la historia del proceso de análisis almacenada en la pila

### Movimientos en un autómata a pila determinista

Un movimiento de un autómata a pila consiste en consumir un elemento de la entrada (x), la cima de la pila (A) y transitar desde el estado en curso (0) a un nuevo estado (1) apilando un nuevo estado en la cima de la pila




### Reconocimiento de lenguajes

**Reconocimiento de lenguajes**

**Reconocimiento por estados finales**

Se reconoce una frase si tras una secuencia de movimientos se llega a un estado final de aceptación. El estado final de la pila no importa

**Reconocimiento por vaciado de pila**

Se reconoce una frase si tras una secuencia de movimientos se llega a vaciar completamente la pila. El estado del autómata no importa

## Especificación mediante autómatas a pila deterministas

Los autómatas a pila son una extensión con memoria de los autómatas finitos deterministas que se utilizan para realizar analizadores sintácticos. Constan de una cinta de entra donde se leen los tokens del programa fuente, una pila de estados (la memoria) y una máquina de estados. Las transiciones de estados dependen ahora de la entrada y de la historia del proceso de análisis almacenada en la pila

### Clasificación de analizadores sintácticos

**Analizadores sintácticos**

**Analizadores no deterministas**

Son analizadores generales que imponen pocas restricciones sobre la naturaleza sintáctica del lenguaje. Suelen tener complejidad asintótica O ($n^3$)

- Cualquier tipo de gramática
- Analizadores en backtracking

**Analizadores deterministas**

Son analizadores que imponen ciertas restricciones sobre la naturaleza de los lenguajes que soportan

**Analizadores sintácticos descendentes**

Se parte del axioma y se aplica una cadena de derivaciones para construir un árbol sintáctico

- Gramáticas LL
- Reconocimiento por vaciado de pila

**Analizadores sintácticos ascendentes**

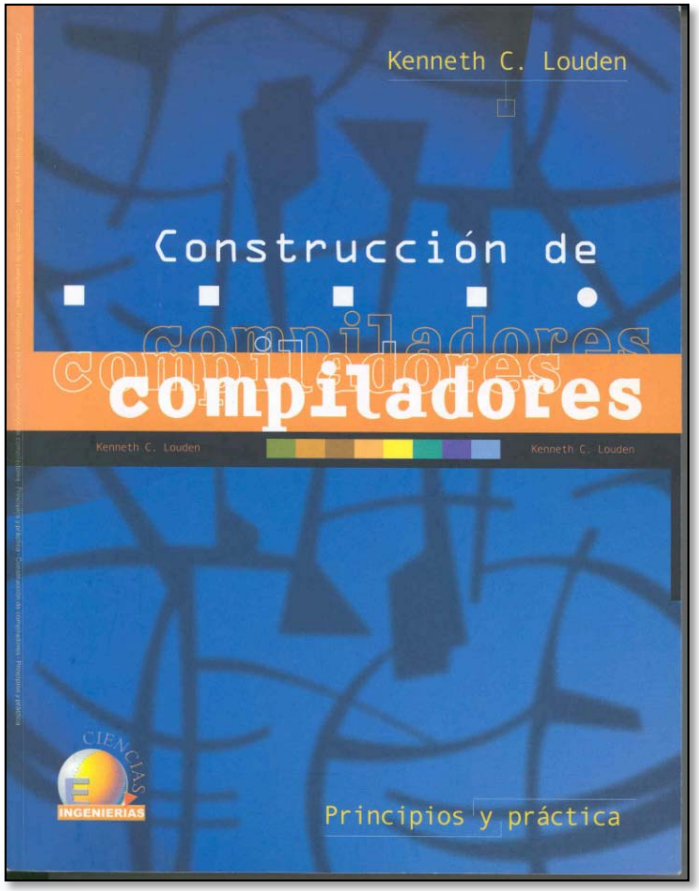
Se parte de los terminales y se construye la inversa de una derivación para intentar alcanzar el axioma

- Gramáticas LR
- Reconocimiento por estados finales

## Material de estudio

### Bibliografía básica

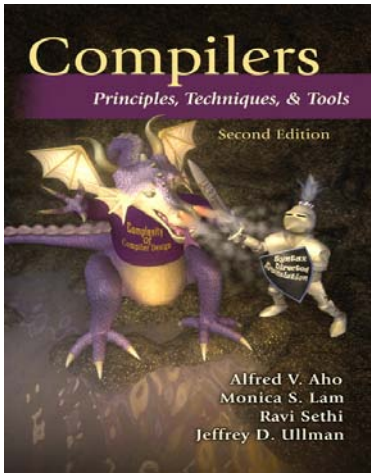
Construcción de compiladores: principios y práctica
Kenneth C. Louden International Thomson Editores, 2004 ISBN 970-686-299-4



## Material de estudio

### Bibliografía complementaria

Compiladores: Principios, técnicas y herramientas.
Segunda Edición Aho, Lam, Sethi, Ullman
Addison – Wesley, Pearson Educación, México 2008




Diseño de compiladores. A. Garrido, J. Iñesta, F. Moreno y J. Pérez. 2002. Edita Universidad de Alicante