Apuntes de la asignatura Procesadores de Lenguajes

Javier Monescillo Buitrón

6 de diciembre de 2018

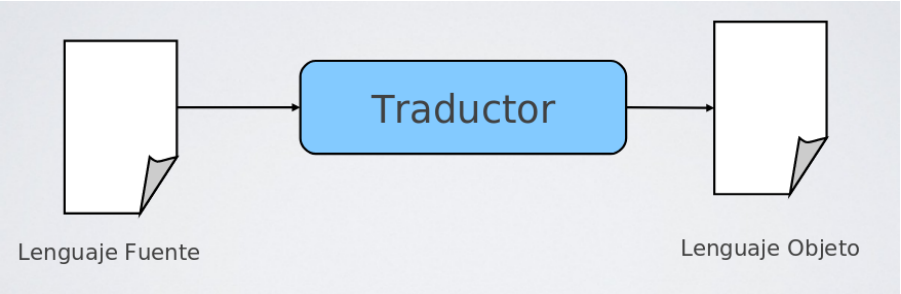
Índice general

Capítulo 1

Introducción

1.1. Definiciones introductorias

- **Procesador**: Sistema cuyo objetivo es el procesamiento de una entrada con el fin de producir una salida.
- **Lenguaje**: Conjunto de cadenas formadas por elementos tomados de un alfabeto.
- **Procesador de lenguajes**: Toma como entrada cadenas de un lenguaje, que son procesadas para producir una salida.
- **Traductor**: Entrada texto en L_1 (lenguaje fuente) y salida texto en L_2 (lenguaje objeto, preservando el significado).
- **Intérprete**: procesa las instrucciones del lenguaje fuente y las ejecuta, NO las traduce.



Tipos de traductores:

	Lenguaje fuente (L1)	**Lenguaje objeto (L2)**
Compilador	Lenguaje alto nivel	Lenguaje máquina
Ensamblador	Lenguaje ensamblador	Lenguaje máquina
Preprocesador	L. Alto nivel con extensiones	L. Alto Nivel sin extensiones
Conversor fuente-fuente	Lenguaje de Alto nivel	Lenguaje de Alto nivel (distinto)
Decompilador	Lenguaje de Bajo nivel	lenguaje de Alto nivel

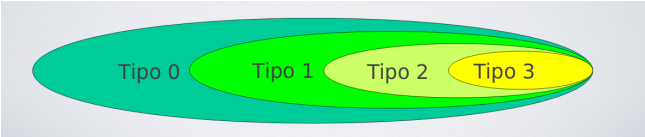
1.2. Lenguajes, gramáticas y autómatas

¿Qué es una gramática?
Las gramáticas son un ente formal para especificar de manera finita, el conjunto de cadena de símbolos potencialmente infinitos que constituyen un lenguaje. Una gramática es una cuádrupla(V,T,P,S):

- V: símbolos no terminales.
- T: símbolos terminales.
- P: reglas de producción.
- S: símbolo inicial de la gramática.

Podemos definir las cadenas que pertenecen al lenguaje por **extensión** o por **derivación** (a partir de su gramática y símbolo inicial). Existen varios tipos de gramáticas:

- **Gramática tipo 0**: Sin restricciones o de estructuras de frases.
- **Gramática tipo 1**: Sensibles al contexto.
- **Gramática tipo 2**: Independientes del contexto.
- **Gramática tipo 3**: Regulares.



El lenguaje generado por una gramática G=(V,T,P,S), será notado como L(G), y se define como el conjunto de cadenas formadas por símbolos terminales que son derivables a partir del símbolo inicial de la gramática.

$$L(G) = \{u \epsilon T | S \Rightarrow u\}$$

Gramáticas	Lenguajes	Máquinas
GLC	Libre de contexto	Autómatas a Pila
GR	Regular	Autómatas Finitos

1.3. Derivación de una gramática

Dada una gramática G(V,T,P,S) y dos palabras α,$\beta \epsilon$ (V∪T)*, solo decimos que β es derivable a partir de α en un paso ($\alpha \Rightarrow \beta$) si y solo si existen palabras δ_1, $\delta_2 \epsilon$ (V∪T)* y una producción Y $\longrightarrow \emptyset$ tales que $\alpha = \delta_1$ y δ_2 y $\beta = \delta_1 \emptyset \delta_2$

1.4. Definición de un lenguaje

La definición de un lenguaje implica:

Léxico: vocabulario del lenguaje junto con sus categorías. Por ejemplo, en un lenguaje de programación, dentro de la categoría tipos, podríamos encontrar el vocabulario (int, float,double, etc).

Sintáctico: Reglas que establecen como construir frases válidas del lenguaje usando los elementos del vocabulario.

Semántico: El significado de las frases, es la información de la cadena, por ejemplo al leer la cadena, ïnt v = 0", el significado es üna variable numérica v se asigna el valor 0".

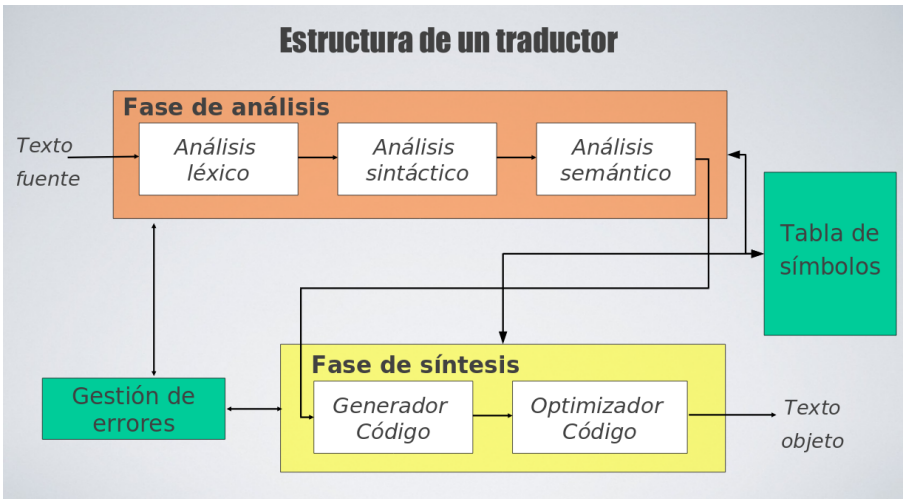
El léxico corresponde a un lenguaje regular, por lo que se expresa con una GR. Por otro lado, el sintáctico se corresponde con una GLC (aunque no tiene por qué ser al completo).

Para distinguir una GR de un GLC hay que fijarse en la parte derecha de la producción y ver si tiene uno o mas no terminales.

1.5. Estructura de un traductor

El proceso de la traducción implica dos fases:

- **Fase de Análisis**(front end): Comprueba que el texto de entrada esta escrito conforme a las reglas (léxicas y sintácticas del lenguaje). Esta fase es dependiente del lenguaje.
- **Fase de Síntesis** (back end): Genera texto equivalente optimizado en el lenguaje objeto. Esta fase es dependiente de la máquina.

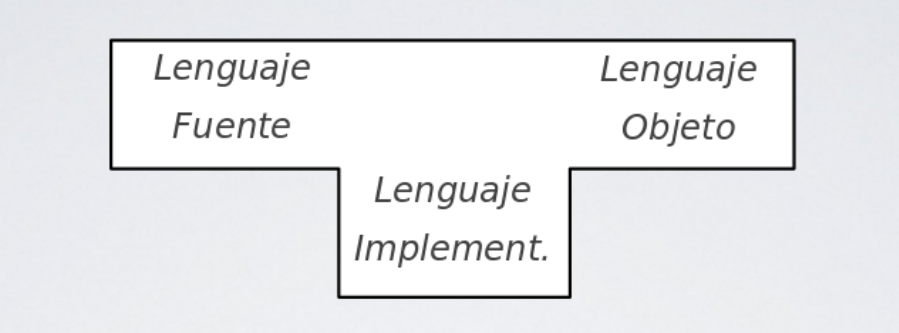


En el análisis léxico se va iterando carácter a carácter, reconociendo si son palabras válidas del lenguaje. Es decir lee la palabra “int” y deduce la categoría “tipos”. Elimina e ignora espacios, tabuladores, saltos de línea, etc. Después en el análisis sintáctico se leen las palabras recibidas (una vez de sabe que pertenecen al lenguaje), y se itera desde el símbolo inicial de la gramática esa secuencia de palabras, así se comprueba que la estructura es correcta.

Finalmente, en el análisis semántico se recibe "el árbol de derivación" (o un equivalente) y se determina que función hace esa cadena introducida. Cuando sabe su funcionamiento, lo transmite al generador de código, para que el lenguaje objeto tenga la misma función.

1.6. Técnicas de construcción de traductores con T-Diagramas

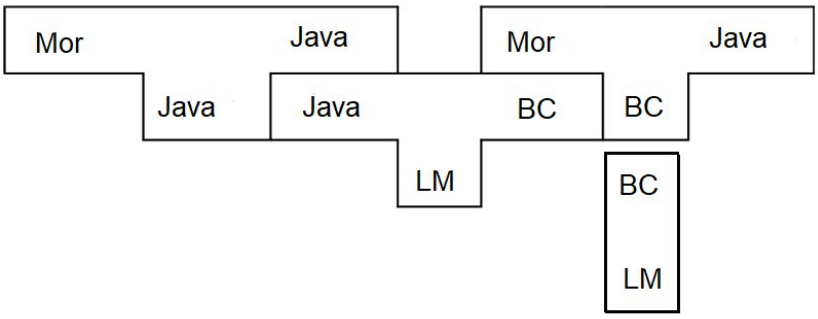
Una notación muy útil para describir programas en general y traductores en particular, son los llamados T-Diagramas.



En el siguiente diagrama tipo T se explica el funcionamiento del procesador que se pretende implementar, tomamos **Mor** como lenguaje fuente, y Java como lenguaje objeto, y además el lenguaje que implementa es Java, es decir, nuestro compilador compila a Java, y esta escrito a Java.

Para utilizar ese compilador, utilizamos un compilador auxiliar, que está escrito en código máquina para dar lugar a bytecode, el típico archivo **.class** que generamos al compilar un **.java**.

Por último tenemos que compilar el bytecode, con un compilador que acepta bytecode escrito en código máquina.



Capítulo 2

Análisis léxico

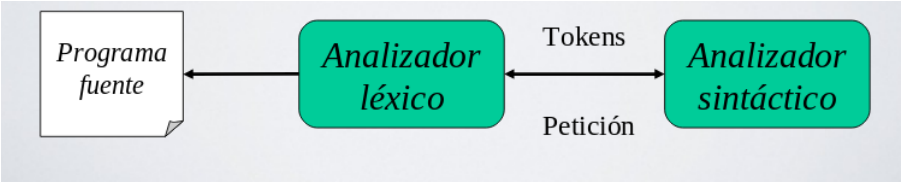
2.1. ¿Cuando interesa utilizar un Procesador de Lenguajes?

Cuando quiero hacer la comunicación en un sistema y dicha comunicación es compleja, hay ocasiones donde me interesa definir un lenguaje con un dominio físico.

2.2. ¿Cuál es la función del analizador léxico?

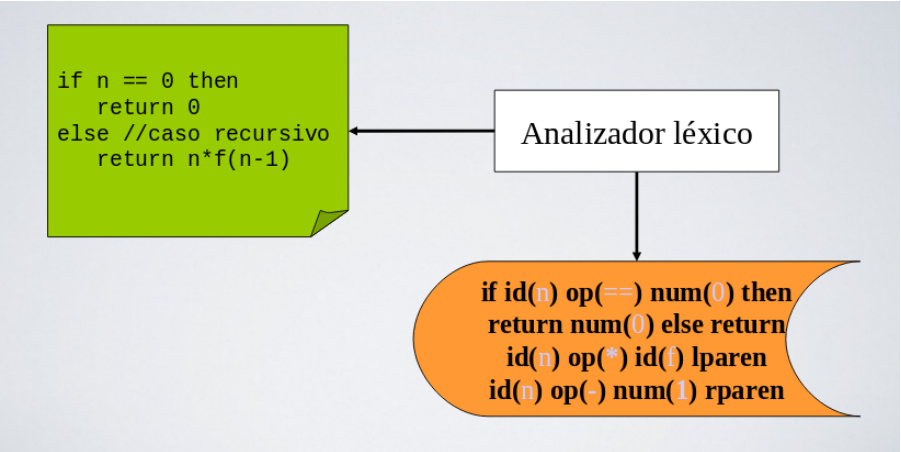
Es función del analizador léxico:

- Reconocer los símbolos o tokens que componen el texto fuente.
- Eliminar comentarios del texto fuente.
- Eliminar los espacios en blanco, saltos de línea y página, tabuladores
- Avisar de los errores léxicos detectados.



El analizador léxico o parser es el único que interactúa con la entrada, no se detiene al encontrar los primeros errores.

Un ejemplo de funcionamiento del analizador léxico.



2.3. Función del analizador léxico

- **Token**: un token es uno o varios elementos básicos del lenguaje fuente que tienen un significado propio. Ejemplos de tokens, presentes en todos los lenguajes de programación, identificadores, palabras reservadas,...

- **Lexema**: es *if* por ejemplo, una combinación concreta de símbolos que forman un token.

- **Patrón**: regla asociada al token que describe a un conjunto de lexemas

Son símbolos terminales de la GLC que se van a encargar de comprobar que la estructura del lenguaje es válida, al analizador léxico no le importan los espacios, ni los tabuladores, solo le interesa que la secuencia sea correcta.

Es útil estudiar análisis léxico para el reconocimiento de patrones.

El problema es encontrar lexemas y disparar patrones, mediante una gramatica regular especificamos los lexemas.

Token	**Lexema**	**Patrón**
moore	moore	m·o·o·r·e
estados	estados	e·s·t·a·d·o·s
estado_in	estado_in	e·s·t·a·d·o·_·i·n
alf_in	alf_in	a·l·f·_i·n
alf_out	alf_out	a·l·f·_o·u·t
transicion	transicion	t·r·a·n·s·i·c·i·o·n
comportamiento	comportamiento	c·o·m·p·o·r·t·a·m·i·e·n·t·o
Paréntesis abierto	(	(
Paréntesis cerrado	)	)
Llave abierta	{	{
Llave cerrada	}	}
Punto y coma	;	;
Coma	,	,
Asterisco barra	∗/	∗·/
Barra asterisco	/∗	/·∗
ID	hola	[A-Za-z][A-Zaz0-9_]*
CMP	c1	c[1-9][0-9]*
CODIGO	#codigo aquí #	#· codigo aquí ·#

Capítulo 3

Análisis sintáctico

Capítulo 4

Análisis sintáctico descendente

4.1. Introducción al análisis sintáctico descendente

Un *Analizador Sintáctico Descendente* es un tipo de Analizador (ASD) que construye un árbol sintáctico de una Gramática Libre de Contexto (GLC) desde la raíz hasta las hojas. [1]

Existen dos tipos de Analizadores ASD:

- Con retroceso
- Sin retroceso

Básicamente un **ASD con retroceso** utiliza un método de fuerza bruta en el que trata de expandir siempre por la primera producción disponible para el símbolo no terminal más a la izquierda. Si son todos los símbolos terminales, entonces finaliza, si no, toca repetir el paso uno hasta finalizar.

Este método es muy ineficiente debido a que es probable que no se alcance la cadena del lenguaje a buscar, por lo tanto, habrá muchas vueltas atrás y producciones recursivas a la izquierda que no se pueden parar.

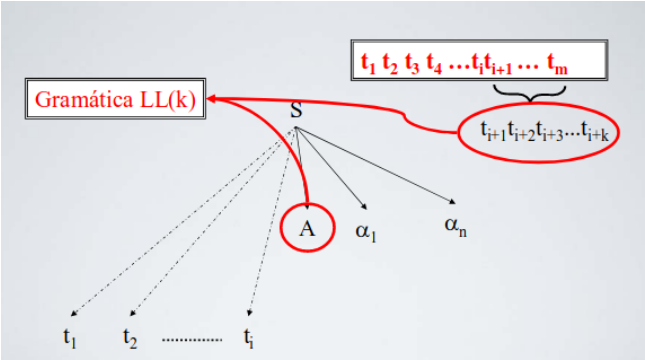
La **recursividad** más a la **izquierda** es un problema general de cualquier técnica de análisis sintáctico descendente por lo que tenemos que solucionar este problema, se explica más detalladamente en la sección (5.1)

Sin embargo, los **ASD sin retroceso** tratan de leer la cadena de izquierda a derecha, predecimos qué producción aplicar en función del símbolo o símbolos actuales de la cadena, así eliminamos las vueltas atrás, por lo que es un Análisis predictivo, tiene la peculiaridad de que solo es aplicable a un tipo especial de gramáticas, denominadas **LL(1)** y **LL(k)**.

4.2. Gramáticas LL(1) y LL(k)

Las gramáticas **LL(k)** son un tipo especial de gramáticas que permiten un análisis descendente sin retroceso. [2]

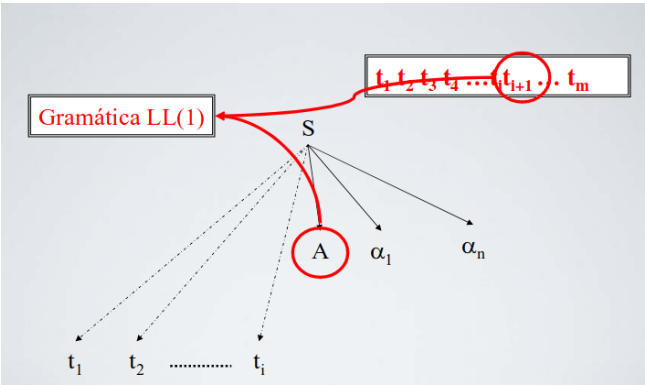
- L: leen la cadena de izquierda a derecha *(Left to right)*
- L: para generar la cadena utilizan derivaciones más a la izquierda *(Leftmost derivations)*
- k. utilizan los *k* primeros símbolos de la cadena para determinar cual es la producción que se debe de aplicar.



Gramática LL(k)

Las gramáticas **LL(1)** son un caso particular de las gramáticas **LL(k)** en el que **k=1**

- Esto quiere decir que utiliza el símbolo actual de la cadena para determinar cual es la producción que se debe de aplicar. [2]



Gramática LL(1)

4.3. Cálculo de Iniciales, Seguidores y Símbolos de predicción

Para poder realizar el cálculo de los símbolos de predicción de nuestra gramática y poder generar nuestro **ASD** predictivo tendremos que definir la operación **Iniciales(ϒ)**.

4.3.1. Iniciales

Iniciales(ϒ) es el conjunto de símbolos terminales que pueden comenzar cualquier cadena generada por ϒ (ϒϵ(V∪T)+)

¿Qué ocurre cuando la cadena vacía ϵ pertenece al conjunto Iniciales de alguna cadena? Tenemos que seguir leyendo de la entrada y obtener el terminal de esa producción, o el terminal que derive del siguiente No-terminal. [2]

```
for todos terminales T do Iniciales(T) := {T};

for cada produccion X → s1 … sk do
 { Iniciales(X) := Iniciales(X) ∪ Iniciales(s1);
  for i := 2 to k do
   if anulable(si-1) then
     Iniciales(X) := Iniciales(X) ∪ Iniciales(si);
   else exit; }
```

Cálculo iniciales

Necesitamos entonces el concepto de **Anulables(X)**.

Anulables

Anulables(X) será igual a **True** si el No-terminal X puede generar ϵ.

```
Anulable(X) := false;
for cada produccion X → s1 … sk do
   if Anulable(X)=false and
     s1 … sk son todos anulables then
        Anulable(X) := true;
```

Cálculo anulables

4.3.2. Seguidores

La operación **Seguidores(X)** calcula el conjunto de símbolos terminales t que pueden aparecer inmediatamente después del símbolo no terminal X, es decir hay una derivación desde el símbolo inicial, a una cadena con la forma α**X**tβ. [2]

```
for todos los símbolos X∈V do SEGUIDORES(X):={};
SEGUIDORES(S):={$}
repeat
 for cada producción X → s_i … s_k do
   for i := 1 to k-1 do
    if (S_i∈V) then
     {SEGUIDORES(s_i) := SEGUIDORES(s_i)∪INICIALES(s_i+1);
      j := i+2;
      while (Anulable(s_j-1) and j<=k) do
       {SEGUIDORES(s_i) := SEGUIDORES(s_i)∪INICIALES(s_j);
        j := j+1; }
      if (Anulable(S_k) and j>k) then
        SEGUIDORES(s_i) := SEGUIDORES(s_i)∪SEGUIDORES(X);
     }
    if (S_k∈V) then
      SEGUIDORES(s_k) := SEGUIDORES(s_k)∪SEGUIDORES(X);
until no más cambios
```

Calculo seguidores

Esta operación que parece más compleja, realmente es sencilla, simplemente al calcular los seguidores de un No-terminal, tenemos que buscar ese No-terminal en las partes derechas de las producciones, y buscar los símbolos terminales que estén más a su derecha, y si estos fueran anulables (los No-terminales) tendríamos que calcular los iniciales del siguiente terminal o No-terminal.

4.3.3. Símbolos de predicción

Los símbolos de predicción es lo que realmente necesitamos para construir nuestro **ASD** predictivo pero para llegar hasta aquí necesitamos las operaciones anteriores.

Para cada producción $X \rightarrow \alpha_i$

- Símbolos de Predicción = INICIALES(α_i)
- Si esa producción hace **Anulable(X)=True** entonces:

  Simbolos de Predicción serán los elementos en: INICIALES(α_i) ∪ SEGUIDORES(X)

Cálculo de símbolos de predicción

4.4. Condición LL(1)

¿Qué propiedades ha de tener una gramática para que esta en particular sea LL(1)?

- No puede ser recursiva a la izquierda
- Para cada producción X $\longrightarrow \alpha_1 \mid ... \mid \alpha_k$ se tiene que cumplir que sus símbolos de predicción sean distintos, o lo que es lo mismo:
  - **Iniciales**(α_i) $\cap$ **Iniciales**(α_j) = $\emptyset$
  - Si es **Anulable(X)** entonces: **Iniciales**(α_i) $\cap$ **Seguidores**(X) = $\emptyset$

4.5. Gramática con recursividad por la izquierda

Presentamos una gramática con el siguiente conflicto a resolver, es recursiva por la izquierda, para una mejor comprensión se utilizará la sintaxis **Proletool** [4] al indicar todas las gramáticas en el trabajo, pero antes ¿Qué es la recursividad por la izquierda?

4.5.1. Recursividad por la izquierda

Una gramática es **recursiva por la izquierda** cuando existe una derivación del tipo $A \overset{*}{\Longrightarrow} A\alpha$

En particular, una gramática es recursiva por la izquierda si contiene una regla de producción de esa forma, en este caso la recursividad sería indirecta.

Ya sabemos que cuando la gramática es **recursiva por la izquierda** el análisis descendente predictivo no funciona, por ello tenemos que eliminarla.

4.5.2. Gramática

Listing 4.1: Gramática recursiva a la izquierda

```
1
2  grammar ejercicio_1{
3  analysis LL1;
4  nonterminal programa, metodo, cuerpo;
5  terminal class, end, def, ops, other_exp;
6
7  programa := class metodo end programa |;
8  metodo := metodo def | cuerpo ':';
9  cuerpo := ops | other_exp |;
10 }
```

Capítulo 5

Análisis sintáctico ascendente

Bibliografía

[1] Apuntes de la asignatura de Procesadores de lenguajes, Tema 3 Análisis Sintáctico

[2] Apuntes de la asignatura de Procesadores de lenguajes, Tema 4 Análisis Sintáctico Descendente

[3] Factorización de una gramática libre de contexto y eliminación de recursividad por la izquierda, Universidad de Costa Rica, Escuela de ciencias de la computación e informática :http://www.di-mare.com/adolfo/cursos/2009-2/pp-A73280-A73372-A76757.pdf

[4] Proletool, Software para la enseñanza y aprendizaje de Procesadores de Lenguajes