Sintaxis y Semántica del Lenguaje

Wilo Carpio Cáceres

2013

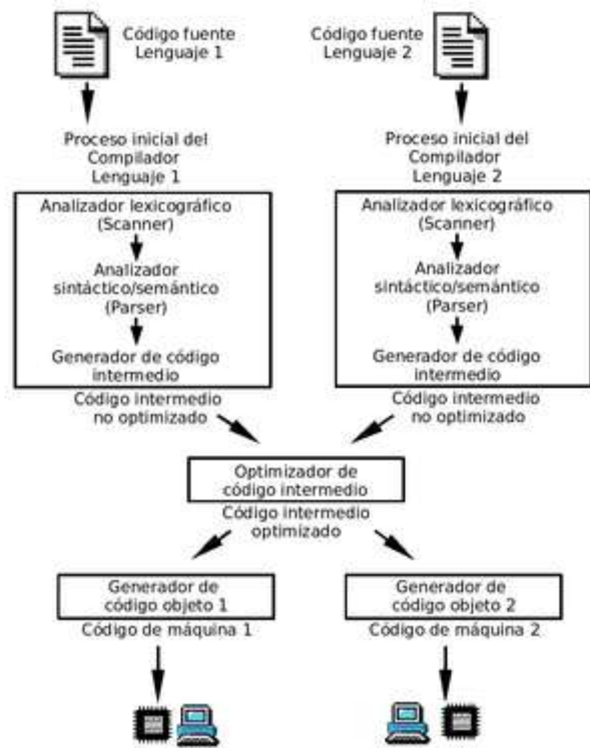
INTRODUCCION A LOS COMPILADORES

Wilo Carpio Cáceres

2012

COMPILACION

Genéricamente el compilador es un software traductor un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina puede interpretar.
Tal procedimiento de traducción es la compilación, que traduce el código fuente de un programa en lenguaje de alto nivel, a otro lenguaje de nivel inferior, de manera que se puede diseñar un programa en lenguaje cercano al humano, para luego compilarlo a un programa procesable por la computadora.
Una aplicación de las teorías de las gramáticas y los autómatas es el desarrollo de estos softwares denominados compiladores, cuyo estudio introductivo, requiere del conocimiento de los siguientes conceptos:



- **TRADUCTOR**: Programa que toma como entrada un texto escrito en un lenguaje, llamado fuente y da como salida otro texto en un lenguaje, denominado objeto.

  Texto Lenguaje Fuente → TRADUCTOR → Texto Lenguaje Objeto

- **COMPILADOR**: Traductor cuyo lenguaje fuente es un lenguaje de programación de alto nivel y la salida es un lenguaje de bajo nivel o ensamblador o código de máquina.

  Lenguaje Fuente → COMPILADOR → Lenguaje Objeto

  Sus ventajas son:
  - Se compila una vez, se ejecuta varias veces.
  - La compilación de bucles genera código equivalente al bucle, pero interpretándolo se traduce tantas veces una línea como veces se repite el bucle.
  - Cada mensaje de error es más detallada, por que el compilador tiene vista global del programa.

- **INTERPRETE**: A diferencia del compilador que genera un programa equivalente, el interprete toma una sentencia del programa fuente en lenguaje de alto nivel y luego la traduce al código equivalente y al mismo tiempo lo ejecuta.

  Sus ventajas son:
  - El intérprete requiere menos memoria que un compilador.
  - En tiempo de desarrollo, permite más interactividad con el código.

- **LINKER**: Construye un fichero ejecutable añadiendo al fichero objeto generado por el compilador las cabeceras necesarias y las funciones de librería utilizadas por el programa fuente.

- **DEPURADOR**: Permite seguir paso a paso la ejecución de un programa, siempre que el compilador haya generado adecuadamente el programa objeto.

- **ENSAMBLADOR**: Compilador cuyo lenguaje fuente es el lenguaje ensamblador. Algunos compiladores, en vez de generar código objeto, generan un programa en lenguaje ensamblador que debe después convertirse en un ejecutable mediante un programa ensamblador.

- **COMPILADOR CRUZADO**: Genera código en lenguaje objeto para una máquina diferente de la que se está utilizando para compilar. Se puede construir un compilador de Pascal que genere código MS- DOS y que el compilador funcione en Linux y se haya escrito en C++.

- **COMPILADOR CON MONTADOR**: Compila distintos módulos de forma independiente y después es capaz de enlazarlos.

- **AUTOCOMPILADOR**: Compilador escrito en el mismo lenguaje que se va a compilar, asi no puede ejecutarse la primera vez. Sirve para ampliar lenguajes, mejorar el código generado, etc.

- **METACOMPILADOR**: Compilador de compiladores, es un programa que recibe como entrada las especificaciones del lenguaje para el que se desea obtener un compilador y genera como salida el compilador para ese lenguaje.

- **DESCOMPILADOR**: Programa que acepta como entrada código máquina y lo traduce a un lenguaje de alto nivel, realizando el proceso inverso a la compilación.

Lenguaje Fuente ← DESCOMPILADOR ← **Lenguaje Objeto**

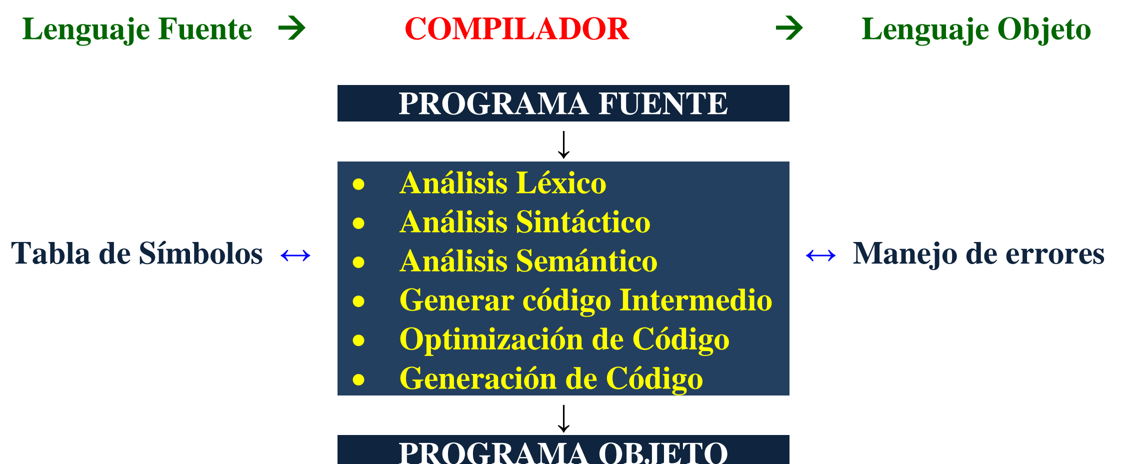
ETAPAS DE COMPILACIÓN

Generar un programa ejecutable suele abarcar dos pasos.

- **Compilación** Traduce el código fuente escrito en un lenguaje de programación almacenado en un archivo a código en bajo nivel (normalmente en código objeto, no directamente a lenguaje máquina).

- **Enlazado** Enlaza el código de bajo nivel generado de todos los ficheros y subprogramas que se han mandado compilar y se añade el código de las funciones que hay en las bibliotecas del compilador para que el ejecutable pueda comunicarse directamente con el sistema operativo, traduciendo así finalmente el código objeto a código máquina, y generando un módulo ejecutable.

ACCION DE COMPILACIÓN



TAREAS DE COMPILACIÓN

- **ANÁLISIS**: Comprueba la corrección del programa fuente, abarca las fases de análisis:
  - **Léxico**: En la descomposición del programa fuente en componentes léxicos
  - **Sintáctico**: O agrupación de los componentes léxicos en frases gramaticales
  - **Semántico**: Que comprueba la validez semántica de las sentencias aceptadas.

- **SÍNTESIS**: Genera la salida en el lenguaje objeto y suele estar formado por una o varias combinaciones de fases de Generación de Código (código intermedio o de código objeto) y de Optimización de Código (código más eficiente).

FASES DE COMPILACIÓN

Fase	Acción
1. **FRONT END**	**Análiza** el código fuente, comprueba su validez, genera el árbol de derivación y rellena los valores de la tabla de símbolos. Suele ser independiente de la plataforma o sistema para el cual se vaya a compilar, y está compuesta por las fases comprendidas entre el Análisis Léxico y la Generación de Código Intermedio
2. **BACK END**	**Generación y optimización de código** que depende del lenguaje objeto y debe ser independiente del lenguaje fuente Genera el código máquina, específico de una plataforma, a partir de los resultados de la fase de análisis, realizada por el Front End. El código que genera BackEnd no suele ejecutarse directamente, sino que necesita ser enlazado por un programa enlazador (linker)

DISEÑO DEL COMPILADOR

Requiere:

Especificación	Describe
LÉXICA:	Con expresiones regulares componentes léxicos o tokens o palabras del lenguaje. A partir de estas, se construye el analizador léxico del compilador.
SINTÁCTICA:	La sintaxis o forma o estructura de los programas en lenguaje fuente, usando una G_{IC} en notación BNF o un diagrama sintáctico. A partir de esta, se construye el analizador sintáctico del compilador
SEMÁNTICA:	El significado de cada construcción sintáctica y las reglas semánticas que deben cumplirse

GENERACION DE CÓDIGO INTERMEDIO
El lenguaje intermedio permite construir en menor tiempo un compilador para otra máquina y compiladores para otros lenguajes fuente, generando códigos para la misma máquina.

OPTIMIZACIÓN DE CÓDIGO
Abarca los procesos del compilador para generar programas objetos más eficientes, rápido en ejecución, que insuman menos memoria.

GENERACIÓN DE CÓDIGO
El código intermedio optimizado se traduce a una secuencia de instrucciones en ensamblador o en el código de máquina del procesador, así la sentencia

Z : = X + Y se convertirá en:

```
LOAD    X
ADD     Y
STORE   Z
```

TABLA DE SIMBOLOS
Es una estructura de datos internos donde el compilador almacena información de objetos que encuentra en el texto fuente, como variables, etiquetas, declaraciones de tipos, etc. En esta tabla, el compilador puede insertar un nuevo elemento en ella, consultar información, borrar, etc.

MANEJO DE ERRORES
Cada error puede ocultar otros o provocar otros errores que se solucionan con el primero. Frente a esto se plantean dos criterios para manejar errores:

- Pararse al detectar el primer error
- Detectar todos los errores de una pasada.

ANÁLISIS LÉXICO O SCANNER (AL)
Desde la entrada lee los caracteres uno a uno, para formar **TOKENS** o secuencia de caracteres con alguna relación entre sí, que serán tratados como entidad única, que será la entrada para la siguiente etapa del compilador. En esta fase se manejan los siguientes conceptos:

- **TOKENS**: Símbolos terminales de cada gramática como palabras reservadas, identificadores, signos de puntuación, constantes numéricas, operadores, cadenas, etc. Existen tokens con varios signos, como: **:** =, = =, + =, | |, etc. Los tokens que tiene dos componentes: **Tipo** y **Valor**, pueden estar constituidas por:
  - **TIRAS ESPECÍFICAS**: Son las palabras reservadas que solo tienen tipo, pueden ser: **if then else, function, while, ;**, los operadores aritméticos o lógicos **::** =, = =, **+,** =, | |, etc.
  - **TIRAS NO ESPECÍFICAS**: Constituidas por los identificadores, constantes o etiquetas. Estas tiras tienen tipo y valor. Por ejemplo, si "contador" es un identificador, el tipo de token será identificador y su valor será la cadena "contador".

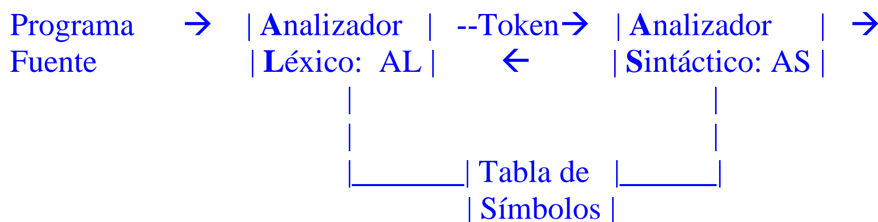
- **PATRÓN:** Expresión regular que define el conjunto de cadenas que puede representar a cada uno de los tokens.

- **LEXEMA:** Cadena que se produce al analizar el texto fuente y encontrar una cadena de caracteres que representan un token determinado. Así, el lexema es una secuencia de caracteres del código fuente que concuerda con el patrón de un token.

- **ATRIBUTOS:** Información adicional sobre los tokens en forma de atributos asociados, cuyo número de depende de cada token; aunque en la práctica el único atributo es el registro que contiene la información propia como, lexema, tipo de token y línea y columna en la que fue encontrado.

  Ejemplo: Sea el número 2003
  - TOKEN Constante entera
  - LEXEMA 2003
  - PATRON: ER $(+ \mid - \mid \varepsilon) \cdot \text{Dígito} \cdot \text{Dígito}^*$

Para procesar cadenas de caracteres y devolver pares (token, lexema), el analizador léxico funciona así:



En la misma pasada el AL puede funcionar como subrutina del analizador sintáctico, es la etapa del compilador que identifica el formato del lenguaje, leyendo para ello, los caracteres del programa e ignorando sus elementos innecesarios para la siguiente fase, como tabuladores, comentarios, espacios en blanco, etc.

El AL agrupa por categorías léxicas los caracteres de entrada, estableciendo el alfabeto con el que se escriben los programas válidos en el lenguaje fuente y rechaza cadenas con símbolos ilegales del alfabeto, desarrollando para ello las siguientes operaciones:

- PROCESO LÉXICO DEL PROGRAMA FUENTE: identificador de tokens y de sus lexemas que deberán entregar al analizador sintáctico e interaccionar con la tabla de símbolos.
- MANEJA EL FICHERO DEL PROGRAMA FUENTE; Abre, lee sus caracteres y cierra.
- IGNORA COMENTARIOS: Como separadores, espacios blancos, tabuladores, retornos de carro, etc.
- LOCALIZA ERRORES: Cuando se produce un error sitúa la línea y la posición en el programa fuente y registra las líneas procesadas.
- PROCESO DE MACROS, definiciones, constantes y órdenes de inclusión de otros ficheros.

Para reconocer un tokens el AL lee los caracteres hasta llegar a uno que no pertenece a la categoría del token leído, el último carácter es devuelto al buffer de entrada para ser leído en primer lugar en la próxima llamada al analizador léxico. Cuando el AS vuelve a llamar al AL, éste

lee los caracteres desde donde quedó en la llamada anterior, hasta completar un nuevo token y devolver el par Token - Lexema.

CREACION DE ANALIZADORES LEXICOS: Las alternativas son:

- Usar un generador automático de analizadores léxicos, como el LEX: su entrada es un código fuente con la especificación de las expresiones regulares de los patrones que representan a los token del lenguaje, y las acciones a tomar cuando los detecte.
- Escribir el AL en lenguaje de alto nivel de uso general utilizando sus funciones de E/S.
- Usar lenguaje ensamblador.

ERRORES LEXICOS

En esta etapa, donde el compilador tiene solo una visión muy local del programa los pocos errores característicos detectados por el analizador léxico son:

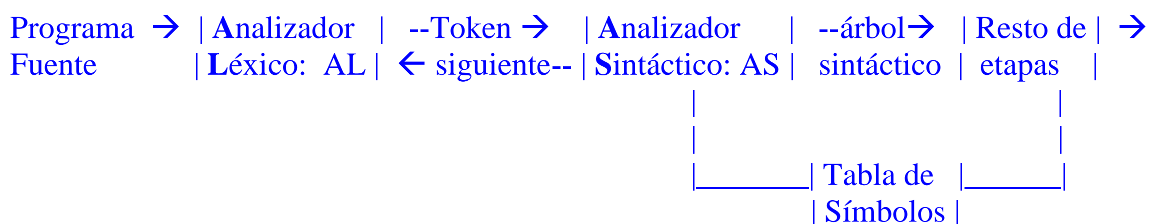
- Uso de caracteres que no pertenecen al alfabeto del lenguaje (**ñ** o **±**).
- Palabras que no coinciden con ninguno de los patrones de los tokens posibles

La detección de un error puede implicar:

- Ignorar los caracteres no válidos hasta formar un token según los patrones dados;
- Borrar los caracteres extraños;
- Insertar un carácter que pudiera faltar;
- Reemplazar un carácter presuntamente incorrecto por uno correcto;
- Conmutar las posiciones de dos caracteres adyacentes.

ANÁLISIS SINTÁCTICO O PARSER: (AS)

El AS además de comprobar la corrección de la sintaxis del programa fuente, construye una representación interna del programa o caso contrario enviar un mensaje de error. Comprueba si los token van llegando en el orden definido, para luego generar la salida con formato de árbol sintáctico.



De modo que sus funciones serán:

- Aceptar solo lo que es válido sintácticamente.
- Explicitar el orden jerárquico que tienen los operadores en el lenguaje de que se trate.
- Guiar el proceso de traducción dirigida por la sintaxis.

A partir del **árbol sintáctico** que representa la sintaxis de un programa, el AS construye una derivación con recorridos por la izquierda o por la derecha del programa fuente, y partir de ellos construye una representación intermedia de ese programa fuente, que puede ser un **árbol sintáctico abstracto** o bien un programa en un **lenguaje intermedio**.

Así para la sentencia T = X – Y + Z

Árbol Sintáctico	Árbol Sintáctico Abstracto	Lenguaje Intermedio
=		
/ \	| ASIGNAR | T | o |	restar X Y t_1
T +		
/ \	| SUMAR | o | Z |	sumar t_1 Z t_2
- Z		
/ \	| RESTAR | X | Y |	asignar t_2 T
X Y		

TIPOS DE ANÁLISIS SINTÁCTICO:

Según la teoría de Análisis Sintáctico, hay dos estrategias para construir el árbol sintáctico:

- **ANÁLISIS DESCENDENTE**: Opera desde la raíz de árbol y va aplicando reglas por la izquierda, de forma de obtener una derivación por izquierda de la cadena de entrada. Recorriendo el árbol en profundidad de izquierda a derecha, encontraremos en las hojas los tokens, que nos devuelve el A.L. en ese mismo orden.

- **ANÁLISIS ASCENDENTE:** Desde la cadena de entrada se construye el árbol empezando por las hojas, luego se crean nodos intermedios hasta llegar a la raíz; construyendo así al árbol de abajo hacia arriba. El recorrido se hará desde las hojas a la raíz. El orden en el que se van encontrando las producciones corresponde a la inversa de una derivación por la derecha.

ANÁLISIS SEMÁNTICO

Determina el tipo de los resultados intermedios, comprueba que los argumentos que tiene un operador pertenecen al conjunto de los operadores posibles, y si son compatibles entre sí, comprueba que el significado de lo que va leyendo sea válido, de manera de generar como salida teórica un árbol semántico, que es un árbol sintáctico cuyas ramas han adquirido el significado concerniente.

En caso del símbolo único con varios significados u **operador polimórfico**, el análisis semántico determina cuál es aplicable; así, para el signo “+”, que permite sumar números, concatenar cadenas de caracteres y unir conjuntos, este análisis comprobará que B y C sean de tipo compatible y que se les pueda aplicar dicho operador y si B y C son números los sumará, si son cadenas las concatenará y si son conjuntos calculara su unión.

Los compiladores analizan semánticamente los programas para verificar su compatibilidad con las especificaciones semánticas del lenguaje al cual pertenecen y así poder comprenderlos, para permitir la recepción por parte del procesador de las órdenes cuando ese programa se ejecuta. Luego de verificar la coherencia semántica, traduce al lenguaje de la máquina o a una representación portable entre distintos entornos.

Esta fase efectúa la traducción asociada al significado que cada símbolo de la gramática adquiere por aparecer en una producción, asociándole información o atributos, y acciones semánticas a las reglas gramaticales, que serán código en un lenguaje de programación; para evaluar los atributos y manipular dicha información de las tareas de traducción, mediante los siguientes pasos:

- Análisis léxico del texto fuente.
- Análisis sintáctico de la secuencia de tokens producida por el analizador léxico.
- Construcción del árbol de análisis sintáctico.
- Recorrido del árbol por el analizador semántico ejecutando las acciones semánticas.

Al actuar la semántica y la traducción, los árboles de análisis sintáctico reciben en cada nodo información o atributos, dando lugar a los **árboles con adornos**, de manera que cada símbolo, se comporta como un registro, cuyos campos son cada uno de sus propios atributos con información semántica, dando lugar al concepto de **gramática atribuida**.

Bibliográfica

- **Teoría de autómatas y lenguajes formales**.
  Autómatas y complejidad. Kelly Dean Editorial Prentice Hall

- **Introducción a la teoría de autómatas, lenguajes y computación**
  John E. Hopcroft; Jeffrey D. Ullman Editorial Cecsa

- **Teoría de la computación**
  J. Gleuu Brokshear Editorial Addison Wesley Iberoamericana