



**Universidad  
Europea de Madrid**

**LAUREATE** INTERNATIONAL UNIVERSITIES

## **ANÁLISIS SINTÁCTICO I**

### **ANALIZADORES SINTÁCTICOS**

© Todos los derechos de propiedad intelectual de esta obra pertenecen en exclusiva a la Universidad Europea de Madrid, S.L.U. Queda terminantemente prohibida la reproducción, puesta a disposición del público y en general cualquier otra forma de explotación de toda o parte de la misma.

La utilización no autorizada de esta obra, así como los perjuicios ocasionados en los derechos de propiedad intelectual e industrial de la Universidad Europea de Madrid, S.L.U., darán lugar al ejercicio de las acciones que legalmente le correspondan y, en su caso, a las responsabilidades que de dicho ejercicio se deriven.

## Índice

Presentación	4
El analizador sintáctico	5
Autómata a pila	6
Tipos de analizadores sintácticos	8
Analizadores sintácticos descendentes	10
Analizadores sintácticos por descenso recursivo	10
Analizador Sintáctico Predictivo LL(1)	10
Árbol de análisis descendente	11
Analizadores sintácticos ascendentes	14
Árbol de análisis ascendente	15
Árbol de sintaxis abstracta (ASA)	18
Propiedades de un analizador sintáctico	19
Resumen	21

## Presentación

El objetivo de este tema es conocer los tipos de analizadores sintácticos que podemos encontrarnos y entender su proceso de análisis sintáctico dependiendo del tipo al que pertenezcan.

Para ello se va a describir el autómata a pila como una evolución del AFD que, aunque se basa en él, sintetiza las transiciones en una tabla de análisis y utiliza una pila que debe tenerse en cuenta en cada transición.

Se verán en este tema los tipos de analizadores sintácticos, así como la forma de construir el árbol de análisis sintáctico dependiendo de si el método de análisis es descendente o ascendente.

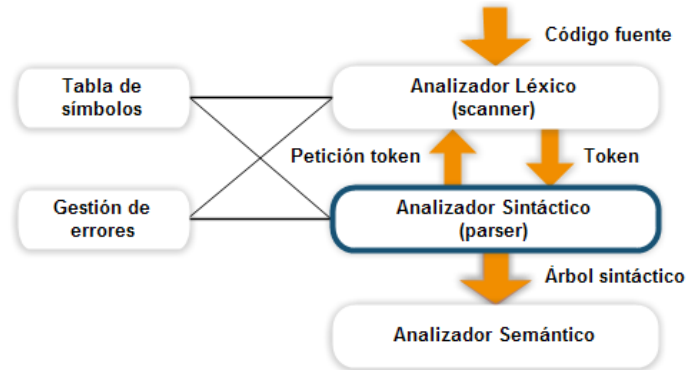
Para cumplir los objetivos del tema, estudiaremos los siguientes contenidos:

- El analizador sintáctico y sus tipos.
- Autómata a pila.
- Analizadores sintácticos descendentes y construcción de su árbol de análisis.
- Analizadores sintácticos ascendentes y construcción de su árbol de análisis.
- Árbol de sintaxis abstracta.
- Propiedades de un analizador sintáctico.



### El analizador sintáctico

Las fases de la compilación, entre las que se encontraba la fase de análisis sintáctico, son las siguientes:



Como puede apreciarse en la figura, el analizador sintáctico obtiene una cadena de *tokens* del analizador léxico y genera un árbol sintáctico, siendo el objetivo de esta fase la de comprobar que la secuencia de componentes léxicos que le entrega el analizador léxico cumple las reglas de la gramática que se han definido previamente.

Los dos tipos generales más usados de analizadores sintácticos son los descendentes y los ascendentes. En Aho et al (2004), se cita un tercer tipo, denominado método universal, basado en el algoritmo de Cocke-Younger-Kasami o el de Earley, que en teoría pueden analizar cualquier gramática pero que son demasiado ineficientes para usarlos en la producción de compiladores.

#### ¿Por qué se denominan analizadores sintácticos descendentes?

Por la forma en la que construyen el árbol de análisis sintáctico, empezando en la raíz y terminando en las hojas.

## Autómata a pila

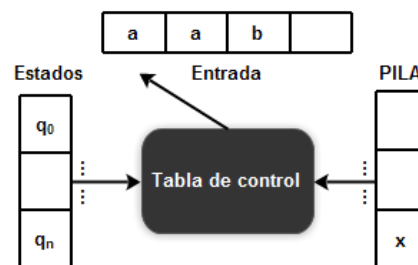
Independientemente del tipo de analizador sintáctico, todos utilizan un autómata a pila (se denomina también como autómata con pila o basado en pila) para realizar su trabajo de análisis, en el que analizan una entrada para verificar si esta pertenece a la gramática o no. Tanto los analizadores sintácticos descendentes como los ascendentes utilizan este mecanismo o modelo para verificar la entrada.

Los autómatas a pila tienen la misma estructura que los autómatas finitos, pero se les añade una pila, a modo de memoria auxiliar, para poder guardar la información que se utilizará a lo largo del proceso de análisis (Isasi et al, 1997).

La manera formal de definir un autómata a pila es la siguiente:

$AP = (\Sigma, \Gamma, Q, A_0, q_0, f, F)$ , donde:

- $\Sigma$  = alfabeto de símbolos de entrada.
- $\Gamma$  = alfabeto de símbolos de la pila.
- $Q$  = es el conjunto de estados.
- $A_0$  = es el símbolo inicial de la pila.
- $q_0$  = es el estado inicial y forma parte de  $Q$ .
- $f$  = es la función de transición entre estados.
- $F$  = es el conjunto de estados finales.



Un autómata a pila es la forma de representar un lenguaje basado en una **gramática independiente del contexto**, del mismo modo que las expresiones regulares utilizan un autómata finito determinista.

 [Comportamiento del autómata a pila](#)  
En detalle

Dependiendo del tipo de analizador la tabla de control funciona de una manera o de otra, así como los símbolos que se introducen en la pila.

**En detalle****Comportamiento del autómata a pila**

El autómata a pila lee un símbolo de la entrada, que debe **pertenecer al alfabeto ( $\Sigma$ )**. Aparte del **estado  $q$**  en el que se encuentre el autómata, hay **un símbolo  $X$ , que debe pertenecer a  $\Gamma$** , en la cima de la pila, por lo que se producirá una transición del autómata para esta terna representada por  $a$ ,  $X$  y  $q$  hacia el siguiente estado, se introducen en la cima de la pila los símbolos que formen parte de la gramática, a la vez que cambia el estado, y se avanza en la lectura del siguiente símbolo de la entrada, siendo la tabla de control quien decide si la terna (estado, entrada y pila) es coherente y por tanto se ha reconocido el carácter de la entrada.

### Tipos de analizadores sintácticos

Hay dos categorías generales de algoritmos para construir analizadores sintácticos: los analizadores sintácticos **descendentes** y los analizadores sintácticos **ascendentes**.

Estos a su vez se subdividen en varios métodos, cada uno de los cuales tiene distintas capacidades y propiedades.

<p>Tipos de analizadores sintácticos descendentes (top-down)</p>	<p>Construyen el árbol de análisis sintáctico desde arriba (raíz o axioma de la gramática) hacia abajo (hojas con los terminales):</p> <ul style="list-style-type: none"> <li>● <b>Análisis sintáctico por descenso recursivo (o con retroceso):</b> este analizador hace una búsqueda en profundidad retroceso, es decir, que puede hacer varios exámenes de la entrada. Presenta el inconveniente de que no es eficiente.</li> <li>● <b>Análisis sintáctico predictivo:</b> es un analizador que no necesita retroceso. Para poder utilizarlos la gramática no puede ser ambigua y se determina qué regla aplicar a partir de un análisis previo de los <i>tokens</i> de la entrada (símbolo de preanálisis, que es el componente léxico de la cadena de entrada situado más a la izquierda). Un ejemplo de este tipo de analizadores sintácticos descendentes es el método LL(1).</li> </ul>
--	---



Tipos de analizadores  
sintácticos ascendentes  
(bottom-up)

Construye el árbol de análisis sintáctico desde abajo hacia arriba.

- **Analizadores de precedencia de operador:** Se utiliza para un pequeño conjunto de gramáticas, denominadas gramáticas de operadores (una propiedad que deben cumplir estas gramáticas es que no pueden tener dos terminales seguidos). Se definen relaciones de precedencia disjuntos entre los terminales, que son los operadores, y estas relaciones guían la selección de las producciones.
- **Analizadores LR:** Hacen un examen de la entrada de izquierda a derecha (*left-to right*, L) y construyen las derivaciones por la producción más a la derecha (*rightmost derivation*, R). Estos analizadores también son denominados analizadores por desplazamiento y reducción. Hay varios autómatas LR y todos llevan un símbolo de análisis por anticipado de la entrada (símbolo de *lookahead*).

¿Cuál elegir? Depende de lo que queramos hacer: los analizadores sintácticos descendentes del tipo predictivo permiten construir analizadores sintácticos eficientes con mayor facilidad, mientras que los analizadores sintácticos ascendentes pueden manejar una mayor clase de gramáticas, siendo estos últimos los más utilizados por los compiladores actuales.

#### Categorías generales

Además de estas dos categorías generales para construir analizadores sintácticos, existen otros métodos, denominados **universales**, basados en los algoritmos de Cocke-Younger-Kasami y Early que pueden servir para cualquier gramática, pero que resultan muy ineficaces en los compiladores que se usan habitualmente.

### Analizadores sintácticos descendentes

Los analizadores sintácticos descendentes pueden considerarse como una manera de encontrar una derivación por la izquierda para una cadena de entrada. Como se ha indicado, los hay de dos tipos: recursivos (también denominados con retroceso) y predictivos.

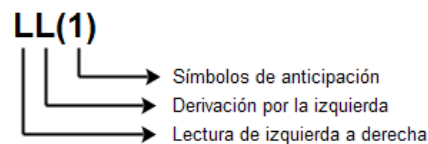
#### Analizadores sintácticos por descenso recursivo

Contiene un conjunto de procedimientos recursivos, donde cada no terminal de la gramática tiene asociado un procedimiento. Cuando se construye el árbol de análisis, al llegar al terminal comprueba si el procedimiento escogido es el adecuado y si lo es avanza el siguiente *token*, si no lo es emite mensaje de error y aplica una estrategia de recuperación que puede consistir en deshacer la operación, retroceder y llamar a otro procedimiento.

#### Analizador Sintáctico Predictivo LL(1)

Es un analizador sintáctico descendente que no necesita retroceder, y para ello es necesario que a partir del *token* a reconocer, sea posible determinar qué regla se debe aplicar. El método LL(1) es efectivo porque está basado en un autómata a pila que contiene en la misma terminales y no terminales (aunque en la cima de la pila siempre hay un no terminal), y en una tabla de análisis, donde se especifica la producción a aplicar para cada *token* que se tenga que reconocer de la entrada y el no terminal que inicia la producción.

El significado de las siglas LL(1) es el siguiente:



Las características del método LL(1) son:

1. No necesita retroceso.
2. Se sabe qué producción hay que aplicar para cada *token* de la entrada.
3. No puede haber ambigüedad en la gramática.


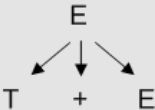
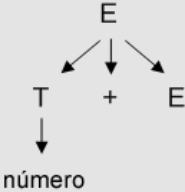
## Árbol de análisis descendente

El algoritmo de construcción del árbol de análisis sintáctico descendente es el siguiente:

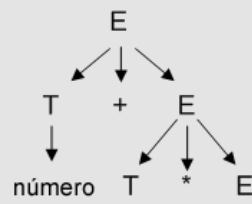
Algoritmo de construcción del árbol de análisis sintáctico descendente:

1. Colocar el axioma como raíz del árbol de derivación.
2. Hasta que solo haya terminales en las hojas derivar por el símbolo que está más a la izquierda y hacia la derecha.

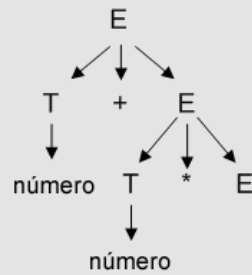
Veamos un ejemplo: en la siguiente tabla, puedes comprobar los pasos y árboles correspondientes a la aplicación del algoritmo (ver [supuestos iniciales](#)):

Paso 1: E	 <pre>graph TD; E[E];</pre>
Paso 2: $E \rightarrow T + E$	 <pre>graph TD; E[E] --&gt; T[T]; E --&gt; plus["+"]; E --&gt; E2[E];</pre>
Paso 3: $T \rightarrow \text{número}$	 <pre>graph TD; E[E] --&gt; T[T]; E --&gt; plus["+"]; E --&gt; E2[E]; T --&gt; numero[número];</pre>

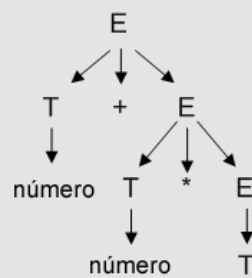
Paso 4:  $E \rightarrow T * E$



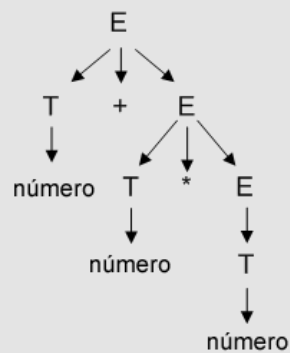
Paso 5:  $T \rightarrow \text{número}$



Paso 6:  $E \rightarrow T$



Paso 7:  $T \rightarrow \text{número}$



Dado que la gramática es ambigua, estos pasos pueden ser aplicados de diferente forma. En este caso, la ambigüedad, que se resuelve factorizando, no hubiera cambiado el árbol de análisis sintáctico.

**Supuestos iniciales**

- Cadena de entrada = número + número \* número

- Gramática:

$$E \rightarrow T \mid T + E \mid T * E$$

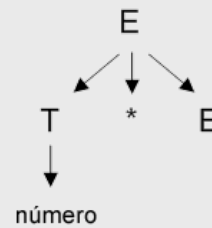
$$T \rightarrow \text{identificador} \mid \text{número}$$

**Gramática ambigua**

La gramática es ambigua, puesto que las tres producciones de la gramática comienzan por T.

**Gramática ambigua**

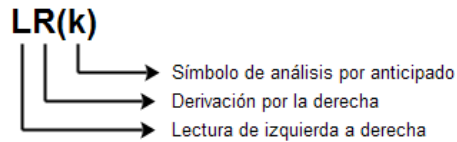
Si estuviéramos usando un método recursivo (o con retroceso), puesto que es uno de los tipos de análisis sintáctico descendente (y hubiéramos aplicado la primera producción como paso 2):  $E \rightarrow T * E$ , al derivar el terminal a partir de T,  $T \rightarrow \text{número}$ , el árbol quedaría como se puede comprobar en la imagen y, con la producción que se ha aplicado, no llegaríamos a reconocer la sentencia.



Por tanto se hubiera tenido que retroceder para aplicar la otra producción válida:  $E \rightarrow T + E$ . Por eso estos métodos son tan ineficientes mientras que los métodos LL(1) son capaces de detectar la siguiente producción a aplicar basándose en una tabla y viendo el carácter de la entrada que toca reconocer en cada momento sin necesidad de recursividad.

### Analizadores sintácticos ascendentes

Hay varios tipos de analizadores sintácticos LR, dependiendo de cómo se construya el AFD y la tabla de análisis en el que están basados. El significado del acrónimo LR(k) es el siguiente:



Al igual que el método LL(1), está basado en una pila y en una tabla de análisis sintáctico que tiene dos partes: acción e ir\_a.

A este método se le conoce también como análisis sintáctico por desplazamiento y reducción (Aho, 1986).

Hay dos tipos, pero puesto que los analizadores por precedencia de operador se dedican a unas gramáticas muy restringidas, como las gramáticas de operador, nos vamos a centrar en los analizadores LR que son los más extendidos por la amplitud de gramáticas que manejan. Los analizadores LR se subdividen a su vez en los siguientes tipos:

SLR	LR sencillo (Simple LR). Es el más sencillo de implementar.
LR(1)	Es el más costoso de implementar, puesto que incorpora el cálculo de los símbolos de <i>lookahead</i> a cada estado del AFD. También es el que más gramáticas reconoce.
LALR	Se trata de conseguir los beneficios del método LR(1) pero con el coste del método SLR.

El método óptimo, si soporta la gramática, es el LALR, debido a que necesita el número mínimo de estados lo que lo hace eficiente a la hora de reconocer una cadena de entrada.

#### Análisis sintáctico por desplazamiento y reducción

- **Análisis sintáctico por desplazamiento:** porque va desplazando el símbolo de la entrada a la pila de análisis.
- **Análisis sintáctico por reducción:** porque se trata de sustituir una producción de la gramática por la frase que la ha generado. Esta sustitución se produce cuando el autómata ha reconocido la parte derecha de una producción, y se sustituye por la parte izquierda, reduciéndose el número de términos en la pila.

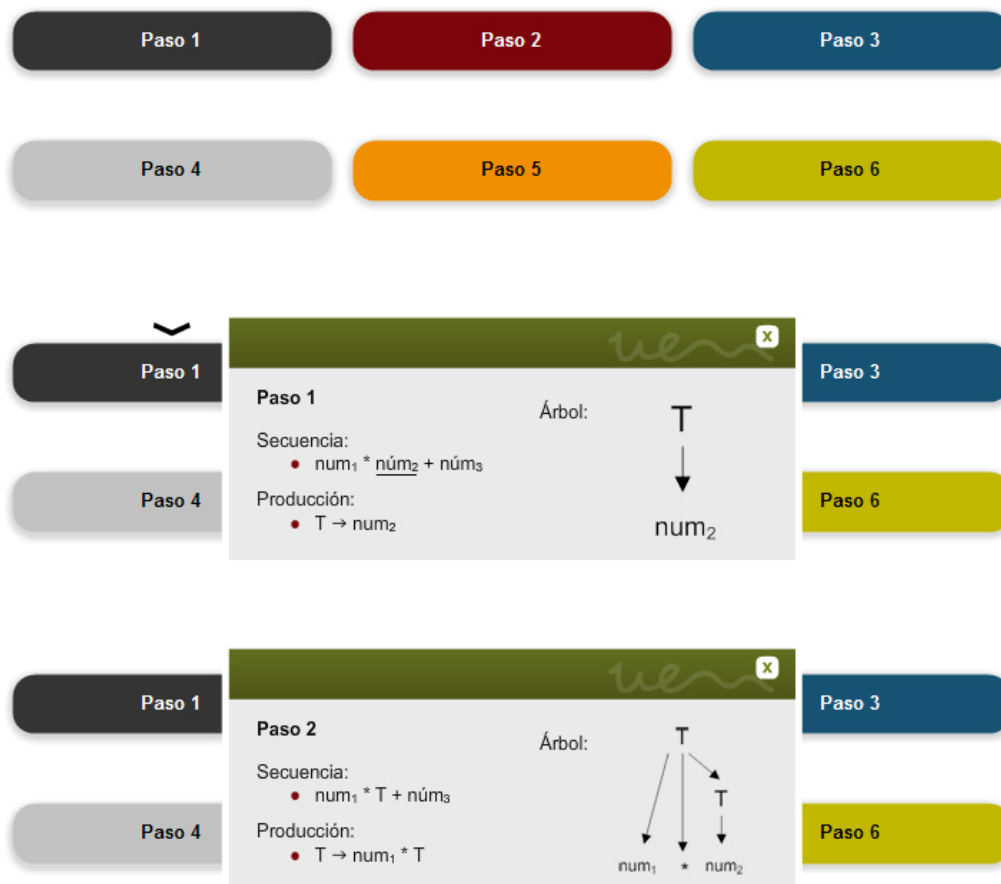
### Árbol de análisis ascendente

Para entender cómo se construye el árbol de análisis sintáctico ascendente necesitamos definir algunos conceptos, para saber cómo se seleccionan las producciones que generan reducciones y permiten, por tanto, avanzar en la construcción del árbol.


Uno de estos conceptos es el de mango (Aho, et al, 2008) que "es una subcadena que coincide con el cuerpo de una producción, y cuya reducción representa un paso a lo largo del inverso de una derivación por la derecha". Es importante destacar que si una gramática no tiene ambigüedad, entonces cada forma de frase derecha de la gramática solo tiene un mango.

Supongamos la gramática de [este enlace](#).

**¿Como se realizaría el proceso de reconocimiento por pasos?** El método LR nos indica que hay que leer la cadena de entrada de izquierda a derecha y derivar por el símbolo que esté más a la derecha, una vez se ha reconocido el mango.





 [Descarga todos los pasos del proceso en pdf](#)

Documentos

### Mango

Mango en inglés se define como *handle*. Hay traducciones en las que el autor llama a este concepto pivote.



### Gramática

Supongamos una gramática en la que la cadena de entrada es:  $\text{num} * \text{num} + \text{num}$ :

$$E \rightarrow T + E \mid T$$

$$T \rightarrow \text{num} * T \mid \text{num}$$

### Árbol de sintaxis abstracta (ASA)

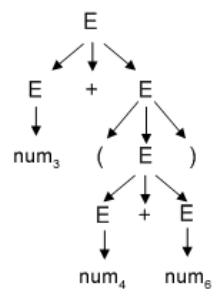
Los árboles de análisis, tanto ascendentes como descendentes, son una forma de representación útil de una cadena de entrada. Esta cadena de entrada se convierte en *tokens* que constituyen las hojas de este árbol de análisis, mientras que los nodos internos y la raíz del mismo representan los pasos y la estructura para llegar al reconocimiento de esa cadena de entrada.

Sin embargo, un árbol de análisis contiene más información de la necesaria para generar código ejecutable, por lo que necesitamos otra estructura que simplifique el árbol de análisis sintáctico pero que tenga toda la información necesaria para el procesamiento eficiente del mismo. Esta estructura recibe el nombre de **Árbol de sintaxis abstracta (ASA)**.

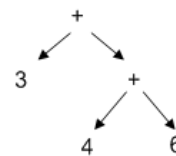
Veamos un ejemplo, a partir de la siguiente gramática con esta cadena de entrada  $3 + (4 + 6)$ :

$$E \rightarrow E + E \mid ( E ) \mid \text{num}$$

El árbol de análisis sintáctico necesario sería:



Árbol de análisis sintáctico



Árbol de sintaxis abstracta

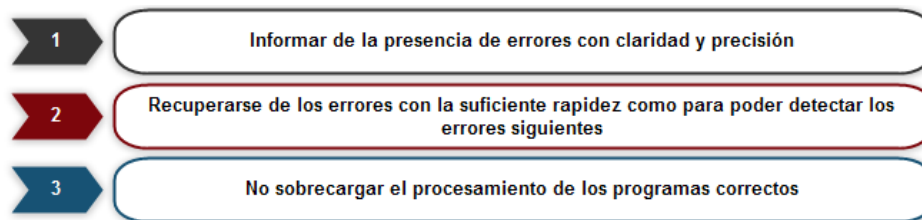
Podemos observar que nos sobran los nodos intermedios donde hay no terminales, así como terminales como los paréntesis. Seguimos teniendo en cuenta la estructura, pero obtenemos un árbol más fácil de usar y de implementar utilizando, en el caso de las operaciones, estructuras que tengan la operación, el operando izquierdo y el operando derecho.

## Propiedades de un analizador sintáctico

Las propiedades que debe cumplir un analizador sintáctico son las siguientes:

1. Ser eficiente
2. Determinar la producción a aplicar reconociendo unos pocos tokens de la entrada.
3. Hacer una adecuada gestión de los errores que se produzcan

La gestión de errores tiene que cumplir con los siguientes objetivos (Aho et al, 2008):



Para realizar adecuadamente esta gestión de errores hay varias estrategias:

Recuperación en modo pánico	<p>El analizador sintáctico descarta símbolos de la entrada, uno por uno, hasta encontrar un <i>token</i> que pertenezca al conjunto de <i>tokens</i> de sincronización (delimitadores -punto y coma-, palabras clave <i>-end-</i>, conjunto PRIMERO, u otros que determine el diseñador).</p> <p>Es el método más sencillo que puede ser utilizado por la mayoría de analizadores sintácticos y no entra en bucles infinitos, aunque en su contra está que se omite una gran cantidad de símbolos (que no son analizados) para recuperarse de un error.</p>
Recuperación a nivel de frase	<p>Una vez se descubre un error, se corrige añadiendo algún símbolo que permita continuar el análisis, como por ejemplo un punto y coma.</p> <p>Presenta el inconveniente de que el error se detecta después de que se haya producido, lo que puede dar lugar a entrar en bucles infinitos.</p>
Producciones de error	<p>Conocidos los errores más comunes, se extiende la gramática con producciones de error, dándose diagnósticos precisos del error que se ha producido.</p>

Corrección global

Se basa en algoritmos que eligen una secuencia mínima de cambios para obtener una corrección global del programa con el menor coste.

Presenta el inconveniente del coste en tiempo y recursos.

**Reconociendo unos pocos *tokens* de la entrada**

Normalmente con uno es suficiente.

## Resumen

En este tema hemos repasado el funcionamiento del analizador sintáctico y hemos entendido cómo funciona un autómatas a pila.

A partir de dicho conocimiento se han identificado los tipos de analizadores sintácticos, tanto descendentes como ascendentes. En el caso de los descendentes el método más utilizado es el predictivo (puesto que no necesita retroceso), que está representado por el método LL(1) y en el caso de los analizadores sintácticos ascendentes hemos visto los dos tipos generales, que son los de precedencia de operador (solo para gramáticas de operadores) y los analizadores LR.

Estos últimos, a su vez, se subdividen en SLR (LR sencillo), LR(1) y LALR, que trabajan con más gramáticas que los métodos descendentes LL(1). Todos utilizan un símbolo de anticipación para detectar la producción a aplicar, siendo el LR(1) el que más gramáticas reconoce, seguido del LALR y finalmente del SLR, aunque el más eficiente es el LALR.

También hemos aprendido a construir el árbol de análisis sintáctico para métodos descendentes y ascendentes. Los métodos descendentes comienzan por la raíz con el axioma de la gramática y realizan derivaciones para llegar hasta los terminales que constituyen la cadena de entrada. En el caso de los ascendentes se parte de las hojas, representadas por los terminales que forman la cadena de entrada y realizan derivaciones por la derecha, para llegar hasta la raíz a partir del mango.

Además del concepto de mango o pivote, otro concepto importante es el de los árboles de sintaxis abstracta (ASA) que permiten eliminar los no terminales que constituyen derivaciones superfluas y los terminales que una vez se ha reconocido la cadena de entrada no son necesarios para el código que se va a generar.

Para finalizar hemos visto las propiedades que debe tener un buen analizador sintáctico, así como las estrategias más comunes de recuperación de errores: modo pánico, a nivel de frase, producciones de error y corrección global.