INGENIERÍA EN SISTEMAS

Creamos, Transformamos y Simplificamos

Tabla de Símbolos

Ing. Alex Moncada

UNAH

UNIVERSIDAD NACIONAL
AUTÓNOMA DE HONDURAS

LU
CEM
ASPI
CIO

Tabla de Simbolos

Las tablas de símbolos son estructuras de datos que utilizan los compiladores para guardar información acerca de las construcciones de un programa fuente. La información se recolecta en forma incremental mediante las fases de análisis de un compilador, y las fases de síntesis la utilizan para generar el código destino.

También llamada «tabla de nombres» o «tabla de identificadores», se trata sencillamente de una estructura de datos de alto rendimiento que almacena toda la información necesaria sobre los identificadores de usuario. Tiene dos funciones principales:

- Efectuar chequeos semánticos.
- Generar código.

¿Quién crea las entradas en la tabla de símbolos?

El analizador léxico, el analizador sintáctico y el analizador semántico son los que crean y utilizan las entradas en la tabla de símbolos durante la fase de análisis. Con su conocimiento de la estructura sintáctica de un programa, por lo general, un analizador sintáctico está en una mejor posición que el analizador léxico para diferenciar entre las distintas declaraciones de un identificador.

En algunos casos, un analizador léxico puede crear una entrada en la tabla de símbolos, tan pronto como ve los caracteres que conforman un lexema.

Sin embargo, sólo el analizador sintáctico puede decidir si debe utilizar una entrada en la tabla de símbolos que se haya creado antes, o si debe crear una entrada nueva para el identificador.

Tabla de Simbolos

La tabla de símbolos almacena la información que en cada momento que se necesita sobre las variables del programa; información tal como: nombre, tipo, dirección de localización en memoria, tamaño, etc. Una adecuada y eficaz gestión de la tabla de símbolos es muy importante, ya que su manipulación consume gran parte del tiempo de compilación.

La tabla de símbolos también sirve para guardar información referente a los tipos de datos creados por el usuario, los tipos enumerados y, en general, cualquier identificador creado por el usuario.

Información sobre los identificadores de usuario

La información que el desarrollador decida almacenar en esta tabla dependerá de las características concretas del traductor que esté desarrollando. Entre esta información puede incluirse:

- Nombre del elemento. El nombre o identificador puede almacenarse limitando o no la longitud del mismo.
- Tipo del elemento. Cuando se almacenan variables, resulta fundamental conocer el tipo de datos a que pertenece cada una de ellas, tanto si es primitivo como si no.
- Dirección de memoria en que se almacenará su valor en tiempo de ejecución.
- Valor del elemento.
- Número de dimensiones. Si la variable a almacenar es un array, también pueden almacenarse sus dimensiones.

Información sobre los identificadores de usuario

- Tipos de los parámetros formales.

Otra información. Con objeto de obtener resúmenes estadísticos e información varia, puede resultar interesante almacenar otros datos: números de línea en los que se ha usado un identificador, número de línea en que se declaró, tamaño del registro de activación, si es una variable global o local, en qué función fue declarada si es local, etc

Consideraciones de la Tabla de Simbolos

La tabla de símbolos puede inicilizarse con cierta información útil, que puede almacenarse en una única estructura o en varias.

Constantes: PI, E, NUMERO_AVOGADRO, etc.
Funciones de librería: EXP, LOG, SQRT, etc.

Palabras reservadas. Algunos analizadores lexicográficos no reconocen directamente las palabras reservadas, sino que sólo reconocen identificadores de usuario.

Una vez tomado uno de la entrada, lo buscan en la tabla de palabras reservadas por si coincide con alguna; si se encuentra, devuelven al analizador sintáctico el token asociado en la tabla; si no, lo devuelven como identificador de verdad. Esto facilita el trabajo al lexicográfico, es más, dado que esta tabla es invariable, puede almacenarse en una tabla de dispersión perfecta (aquélla en la que todas las búsquedas son exactamente de O(1))

Consideraciones de la Tabla de Simbolos

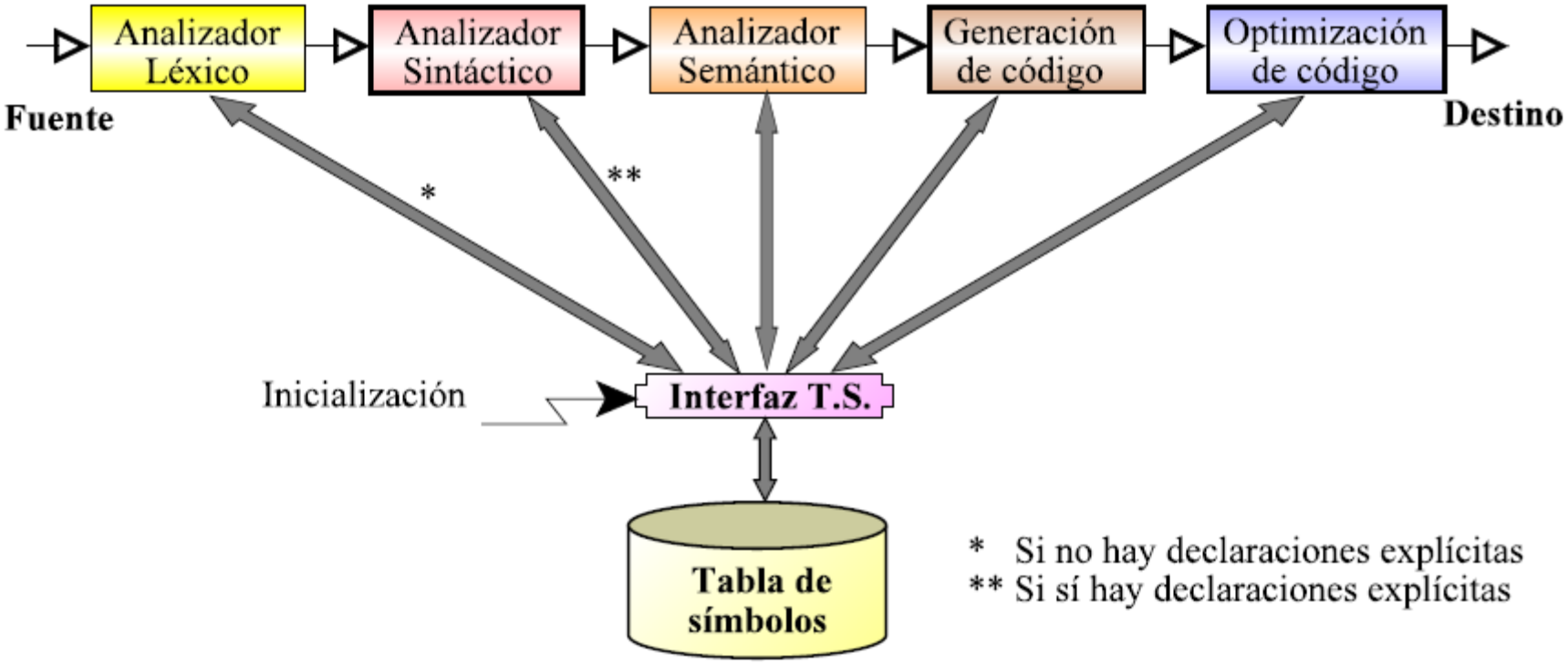


Figura 6.1. Accesos a la tabla de símbolos por parte de las distintas fases de un compilador

Ejemplo: Calculadora de Variables

Llegados a este punto, para simplificar el diseño no incluiremos un área de declaraciones de variables, sino que éstas se podrán utilizar directamente inicializándose a 0, puesto que tan sólo disponemos del tipo entero.

Para introducir las variables en la calculadora necesitamos una nueva construcción sintáctica con la que podemos darle valores: se trata de la asignación. Además, continuaremos con la sentencia de evaluación y visualización de expresiones a la que antepondremos la palabra reservada PRINT. En cualquier expresión podrá intervenir una variable que se evaluará al valor que tenga en ese momento. Así, ante la entrada:

```
1 a := 7 * 3;
2 b := 3 * a;
3 a := a + b;
```

Ejemplo: Calculadora de Variables

se desea almacenar en la variable **a** el valor 21, en la **b** el valor 63 y luego modificar **a** para que contenga el valor 84. Para conseguir nuestros propósitos utilizaremos una tabla de símbolos en la que almacenaremos tan sólo el nombre de cada variable, así como su valor ya que estamos tratando con un intérprete. La tabla de símbolos tendrá una estructura de lista no ordenada simplemente encadenada

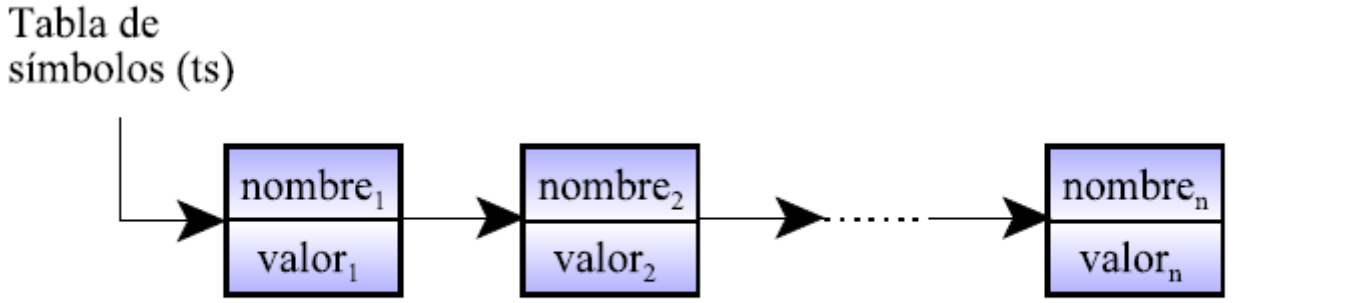


Figura 6.2. Una de las tablas de símbolos más simples (y más ineficientes) que se pueden construir

Ejemplo: Calculadora de Variables

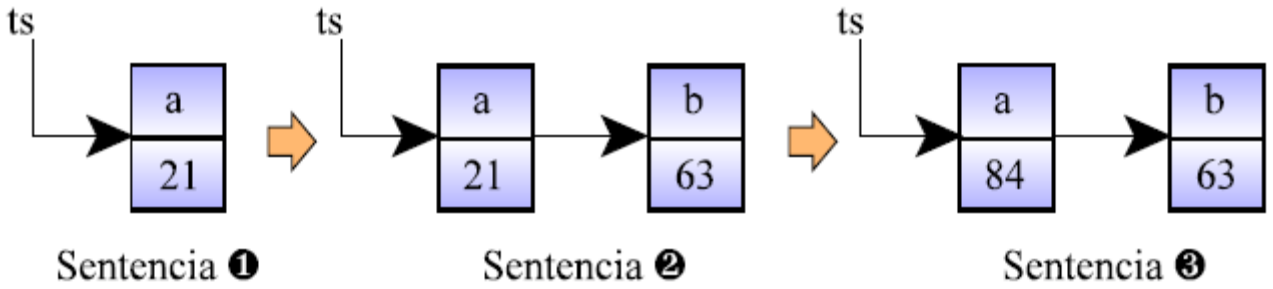


Figura 6.3. Situación de la tabla de símbolos tras ejecutar algunas sentencias de asignación

INGENIERÍA EN SISTEMAS

Creamos, Transformamos y Simplificamos

Analizador Sintáctico

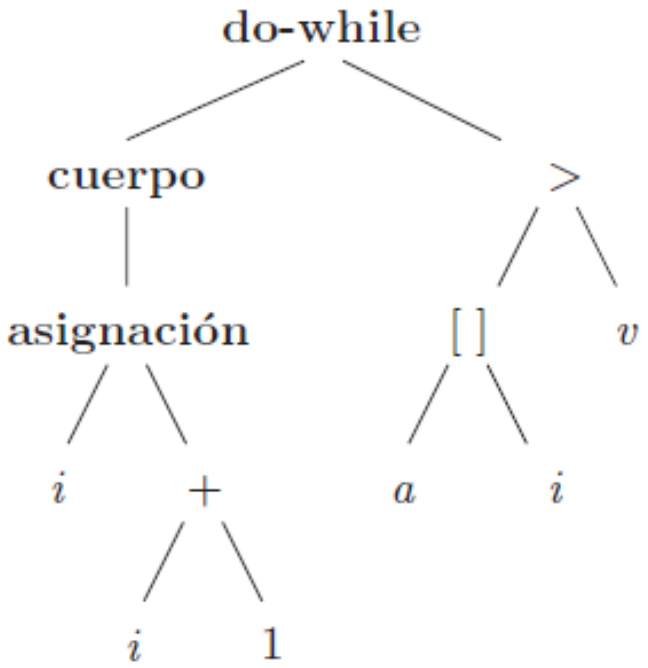
Ing. Alex Moncada

UNAH

UNIVERSIDAD NACIONAL
AUTÓNOMA DE HONDURAS

LU
CEM
ASPI
CIO

Analizador Sintáctico



(a)

```
1:  i = i + 1
2:  t1 = a [ i ]
3:  if t1 < v goto 1
```

(b)

Figura 2.4: Código intermedio para "`do i=i+1; while (a[i] <v);`"

Concepto de analizador sintáctico

Es la fase del analizador que se encarga de chequear la secuencia de tokens que representa al texto de entrada, en base a una gramática dada. En caso de que el programa de entrada sea válido, suministra el árbol sintáctico que lo reconoce en base a una representación computacional. Este árbol es el punto de partida de la fase posterior de la etapa de análisis: el analizador semántico.

Pero esto es la teoría; en la práctica, el analizador sintáctico dirige el proceso de compilación, de manera que el resto de fases evolucionan a medida que el sintáctico va reconociendo la secuencia de entrada por lo que, a menudo, el árbol ni siquiera se genera realmente.

Concepto de analizador sintáctico

En la práctica, el analizador sintáctico también:

• Incorpora acciones semánticas en las que colocar el resto de fases del compilador (excepto el analizador léxico): desde el análisis semántico hasta la generación de código.

• Informa de la naturaleza de los errores sintácticos que encuentra e intenta recuperarse de ellos para continuar la compilación.

• Controla el flujo de tokens reconocidos por parte del analizador léxico.

En definitiva, realiza casi todas las operaciones de la compilación, dando lugar a un método de trabajo denominado compilación dirigida por sintaxis.

La función del analizador sintáctico

En nuestro modelo de compilador, el analizador sintáctico obtiene una cadena de tokens del analizador léxico, como se muestra en la figura, y verifica que la cadena de nombres de los tokens pueda generarse mediante la gramática para el lenguaje fuente.

Esperamos que el analizador sintáctico reporte cualquier error sintáctico en forma inteligible y que se recupere de los errores que ocurren con frecuencia para seguir procesando el resto del programa. De manera conceptual, para los programas bien formados, el analizador sintáctico construye un árbol de análisis sintáctico y lo pasa al resto del compilador para que lo siga procesando.

La función del analizador sintáctico

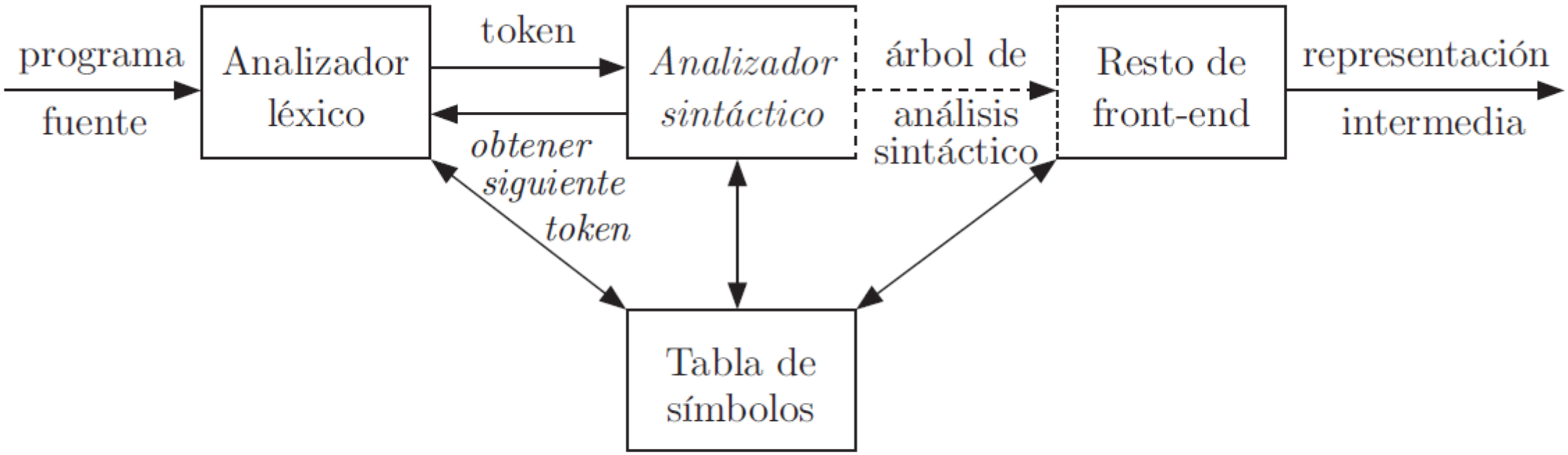


Figura 4.1: Posición del analizador sintáctico en el modelo del compilador

La función del analizador sintáctico

Existen dos tipos generales de analizadores para las gramáticas: descendentes y ascendentes.

- Según sus nombres, los métodos descendentes construyen árboles de análisis sintáctico de la parte superior (raíz) a la parte inferior (hojas), mientras que los métodos ascendentes empiezan de las hojas y avanzan hasta la raíz.
- En cualquier caso, la entrada al analizador se explora de izquierda a derecha, un símbolo a la vez.
- Los métodos descendentes y ascendentes más eficientes sólo funcionan para subclases de gramáticas, pero varias de estas clases, en especial las gramáticas LL (método de análisis sintáctico predictivo) y LR (herramientas automatizadas), son lo bastante expresivas como para describir la mayoría de las construcciones sintácticas en los lenguajes de programación modernos.

Gramáticas libres de contexto

Una gramática describe en forma natural la estructura jerárquica de la mayoría de las instrucciones de un lenguaje de programación.

if (expr) instr **else** instr

Esta regla de estructuración puede expresarse de la siguiente manera:

instr → **if** (expr) instr **else** instr

En donde la flecha se lee como “puede tener la forma”. A dicha regla se le llama *producción*.

- En una producción, los elementos léxicos como la palabra clave **if** y los paréntesis se llaman *terminales*.
- Las variables como *expr* e *instr* representan secuencias de terminales, y se llaman *no terminales*.

Gramáticas libres de contexto

Una gramática libre de contexto tiene cuatro componentes:

1. Un conjunto de símbolos terminales, a los que algunas veces se les conoce como “tokens”.
2. Un conjunto de no terminales, a las que algunas veces se les conoce como “variables sintácticas”.
3. Un conjunto de producciones, en donde cada producción consiste en un no terminal, llamada encabezado o lado izquierdo de la producción, una flecha y una secuencia de terminales y no terminales, llamada cuerpo o lado derecho de la producción.
4. Una designación de una de los no terminales como el símbolo *inicial*.

Revisar la sección 4.2.2 del libro principal pag. 198

Gramáticas libres de contexto

Ejemplo 2.1: Varios ejemplos en este capítulo utilizan expresiones que consisten en dígitos y signos positivos y negativos; por ejemplo, las cadenas como 9-5+2, 3-1 o 7. Debido a que debe aparecer un signo positivo o negativo entre dos dígitos, nos referimos a tales expresiones como "listas de dígitos separados por signos positivos o negativos". La siguiente gramática describe la sintaxis de estas expresiones. Las producciones son:

$$\begin{aligned}
\textit{lista} &\rightarrow \textit{lista} + \textit{dígito} && (2.1)\\
\textit{lista} &\rightarrow \textit{lista} - \textit{dígito} && (2.2)\\
\textit{lista} &\rightarrow \textit{dígito} && (2.3)\\
\textit{dígito} &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 && (2.4)
\end{aligned}$$

Gramáticas libres de contexto

Ejemplo 2.2: El lenguaje definido por la gramática del ejemplo 2.1 consiste en listas de dígitos separadas por signos positivos y negativos. Las diez producciones para el *dígito* no terminal le permiten representar a cualquiera de los terminales `0`, `1`, . . . , `9`. De la producción (2.3), un dígito por sí solo es una lista. Las producciones (2.1) y (2.2) expresan la regla que establece que cualquier lista seguida de un signo positivo o negativo, y después de otro dígito, forma una nueva lista.

Las producciones (2.1) a (2.4) son todo lo que necesitamos para definir el lenguaje deseado. Por ejemplo, podemos deducir que `9-5+2` es una *lista* de la siguiente manera:

a) `9` es una *lista* por la producción (2.3), ya que `9` es un *dígito*.

b) `9-5` es una *lista* por la producción (2.2), ya que `9` es una *lista* y `5` es un *dígito*.

c) `9-5+2` es una *lista* por la producción (2.1), ya que `9-5` es una *lista* y `2` es un *dígito*. □

Gramáticas libres de contexto

Ejemplo 4.5: La gramática en la figura 4.2 define expresiones aritméticas simples. En esta gramática, los símbolos de los terminales son:

id + - * / ( )

Los símbolos de los no terminales son *expresión*, *term* y *factor*, y *expresión* es el símbolo inicial. □

$$
\begin{aligned}
\textit{expresión} &\rightarrow \textit{expresión} + \textit{term} \\
\textit{expresión} &\rightarrow \textit{expresión} - \textit{term} \\
\textit{expresión} &\rightarrow \textit{term} \\
\textit{term} &\rightarrow \textit{term} * \textit{factor} \\
\textit{term} &\rightarrow \textit{term} / \textit{factor} \\
\textit{term} &\rightarrow \textit{factor} \\
\textit{factor} &\rightarrow (\textit{expresión}) \\
\textit{factor} &\rightarrow \mathbf{id}
\end{aligned}
$$

Figura 4.2: Gramática para las expresiones aritméticas simples

Gramáticas libres de contexto

Árbol de análisis sintáctico para $-(\mathbf{id} + \mathbf{id})$

$$
\begin{aligned}
E &\rightarrow E + T \mid E - T \mid T \\
T &\rightarrow T * F \mid T / F \mid F \\
F &\rightarrow (E) \mid \mathbf{id}
\end{aligned}
$$

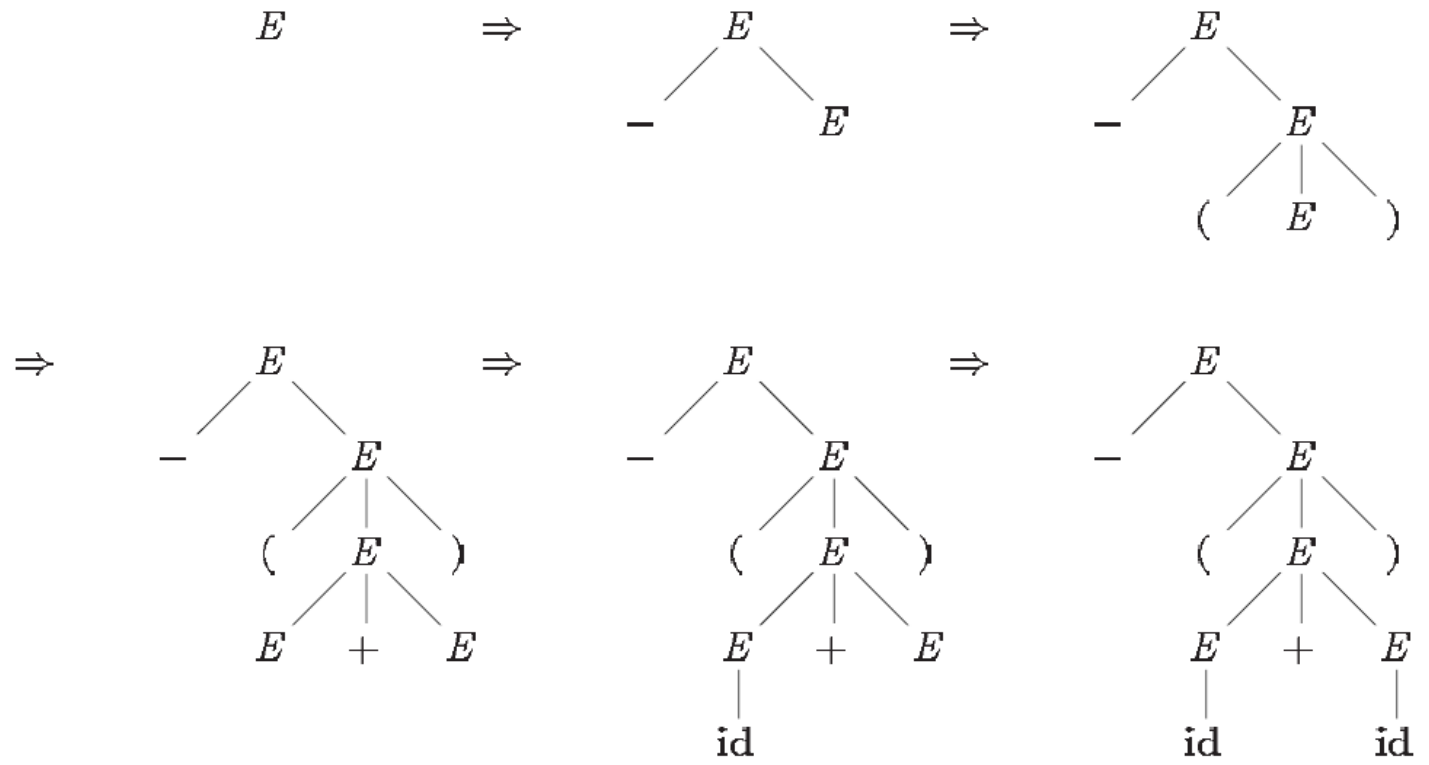


Figura 4.4: Secuencia de árboles de análisis sintáctico para la derivación (4.8)