



**Universidad
Europea de Madrid**

LAUREATE INTERNATIONAL UNIVERSITIES

GENERACIÓN DE CÓDIGO INTERMEDIO

ÁRBOLES DE SINTAXIS ABSTRACTA (ASA)

Cartagena99

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

GENERACIÓN DE CÓDIGO INTERMEDIO
ÁRBOLES DE SINTAXIS ABSTRACTA (ASA)

© 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

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Índice

Presentación	4
Introducción	5
Tipos de estructuras	6
Árbol de Sintaxis Abstracta (ASA)	8
Árbol de Sintaxis Abstracta (ASA) II	9
ASA. Ejemplo	10
Tipos de nodos de un ASA	11
Secuencia de nodos en un ASA	13
Otras implementaciones de un ASA	14
Resumen	15

The logo for Cartagena99 features the text 'Cartagena99' in a stylized, blue, serif font. The '99' is significantly larger and more prominent than the word 'Cartagena'. The text is set against a light blue background with a white arrow pointing to the right, and a yellow shadow is cast beneath the text.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

GENERACIÓN DE CÓDIGO INTERMEDIO

ÁRBOLES DE SINTAXIS ABSTRACTA (ASA)

Presentación

El objetivo de este tema es comprender la necesidad de la generación de código intermedio, identificar sus posibles modalidades y entender el funcionamiento del Árbol de Sintaxis Abstracta (en adelante, ASA), así como su implementación.

Para todo ello se alcanzarán los siguientes objetivos:

- Comprender la necesidad de generar código intermedio, así como sus ventajas e inconvenientes.
- Conocer qué tipos de estructuras podemos utilizar.
- Entender qué es un ASA y como se construye.
- Entender cómo se construye el ASA para operaciones aritméticas mediante un ejemplo.
- Conocer qué tipos de nodos puede tener un ASA y su complejidad.
- Comprender mediante un ejemplo cuál sería la secuencia de nodos para crear un árbol.
- Conocer otras implementaciones.



**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

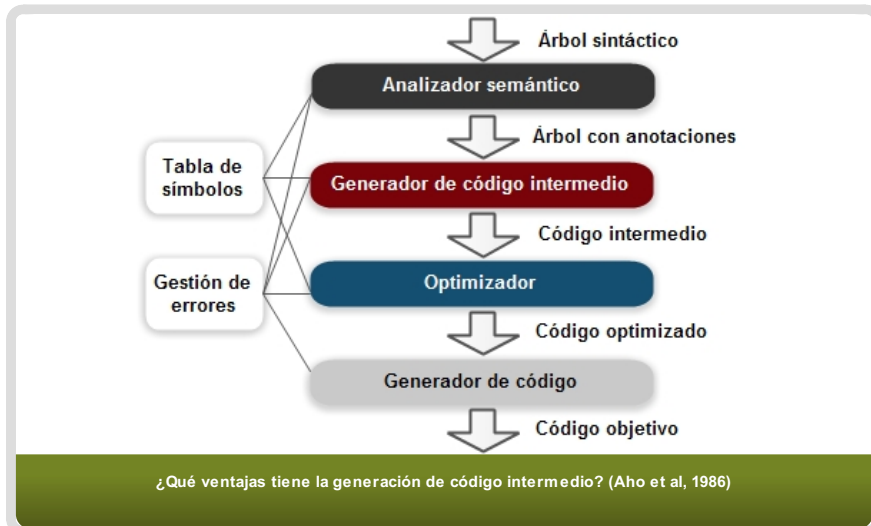
GENERACIÓN DE CÓDIGO INTERMEDIO

ÁRBOLES DE SINTAXIS ABSTRACTA (ASA)

Introducción

Una vez se han realizado en el compilador las tres fases de análisis (léxico, sintáctico y semántico) empiezan las fases de sintaxis, que se concretan en la generación de código para una máquina objetivo.

Véase que a partir de realizar el análisis semántico, nos podríamos saltar todos los pasos de la fase de sintaxis, generando código fuente directamente.



- Se facilita la redestinación, ya que solo habría que cambiar la parte que va asociada a la máquina objetivo.
- Se puede aplicar a la representación intermedia, un optimizador de código independiente de la máquina.

En este tema se definen cuáles son los métodos utilizados y por tanto, las estructuras de datos necesarias para llevar a cabo este proceso de generación de código intermedio.

Como nos indica Louden (2004), el código intermedio puede tener muchas formas, existiendo casi tantos estilos de código intermedio como compiladores.

Algunos de estos métodos tratan de linealizar el árbol sintáctico, es decir, realizan una representación secuencial del árbol sintáctico. Esto es así cuando después se va a aplicar un optimizador y se quiere obtener código muy eficiente.

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Tipos de estructuras

La estructura de datos que representa el programa fuente durante la traducción se denomina **representación intermedia** (IR, de *Intermediate Representation*) y a la representación intermedia que se parece al código objetivo se le denomina **código intermedio** (Louden, 2004).

Hay que indicar que la representación del código intermedio depende de la máquina para la que se está generando el código (máquina objeto) y esta puede ser de la siguiente forma:

Máquinas de pila (0 direcciones)	Se refiere a una máquina (física o virtual) con un conjunto de instrucciones de 0 direcciones (todas las direcciones son implícitas), llamada así porque opera con valores que se leen o insertan en el tope de la pila (operaciones de <i>push</i> y <i>pop</i>).
Máquinas de registro (2 direcciones)	Cuando se utilizan máquinas de registro una de las direcciones se usa para especificar uno de los operando fuente el resultado y la otra dirección para el otro operando fuente ($a = a + b$).
Máquinas RISC (3 direcciones)	Cuando se cuenta con instrucciones de tres direcciones (lo habitual en el entorno PC) se utilizan dos direcciones para los operando fuente y una dirección para el resultado. ($a = b + c$).

Las tres estructuras que vamos a ver para generar **código intermedio** son:

1. **Árbol de Sintaxis Abstracta (ASA)**: tiene sentido utilizarlos cuando no se va a realizar una optimización de código y directamente se va a generar código. Tiene una versión mejorada formada por los grafos dirigidos acíclicos (GDA)
2. **Notación postfija**: no necesita paréntesis, porque la posición y el número de argumentos de los operadores permiten solo una descodificación por ejemplo: $(5-3)+8$. Esto en notación postfija es: $53-8+$.
3. **Código de tres direcciones**: este código está pensado para las operaciones aritméticas y tiene la siguiente forma: $x = y \text{ op } z$.

En este tema veremos el ASA y más adelante el código de tres direcciones. En cualquier caso y se use la notación que se use, para realizar la traducción de código intermedio a código objeto hay que volver a recorrer el árbol para generar el código final.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

GENERACIÓN DE CÓDIGO INTERMEDIO
ÁRBOLES DE SINTAXIS ABSTRACTA (ASA)

Notación postfija

A la notación postfija también se la denomina notación polaca inversa (RPN).

The logo for Cartagena99 features the text 'Cartagena99' in a stylized, blue, serif font. The '99' is significantly larger and more prominent than the 'Cartagena' part. The text is set against a light blue, abstract background that resembles a stylized 'C' or a wave. Below the text, there is a horizontal orange bar with a slight gradient and a shadow effect.

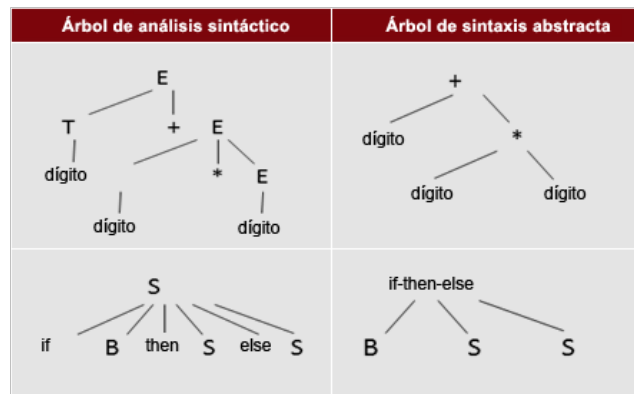
**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Árbol de Sintaxis Abstracta (ASA)

Como nos indica Aho et al (1986), el uso de árboles sintácticos como representación intermedia permite que la traducción se separe del análisis sintáctico. Es importante indicar que los tipos de árboles que nos interesan son los árboles de sintaxis abstracta (ASA) que son árboles de análisis sintácticos a los que se les han eliminado los símbolos superfluos y es muy útil para representar las construcciones del lenguaje.

Veamos los siguientes ejemplos:



Como podemos ver, en un ASA nos interesan las operaciones y los operadores y es por eso que eliminamos los símbolos superfluos (terminales de la gramática), generándose un árbol condensado.

[¿Cómo se construye el ASA?](#)
En detalle

En detalle

¿Cómo se construye el ASA?

- Se genera un nodo para cada operador y cada operando.
- Los hijos de un nodo operador son las raíces de los nodos que representan las subexpresiones que constituyen los operandos de dicho operador.

Para llevarlo a la práctica necesitamos unas funciones auxiliares:

- hazNodo (operador, izquierda, derecha).
- hazHoja (id, entrada).



**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Árbol de Sintaxis Abstracta (ASA) II

Vamos a ver con un ejemplo práctico la implementación usando la notación de definición dirigida por la sintaxis para la gramática de operadores aritméticos. Se utiliza como atributo sintetizado un puntero que apunta a las funciones **hazNodo** y **hazHoja** que acabamos de definir.

Partimos de la siguiente gramática, donde op puede representar a cualquier operador aritmético:

- $E_1 \rightarrow E_2 \text{ op } T$
- $E \rightarrow T$
- $T \rightarrow \text{id}$
- $T \rightarrow \text{dígito}$

Generándose las siguientes reglas semánticas:

Producción	Regla semántica
$E_1 \rightarrow E_2 \text{ op } T$	$E_1.\text{puntero} = \text{hazNodo}(\text{op}, E_2.\text{puntero}, T.\text{puntero})$
$E \rightarrow T$	$E.\text{puntero} = T.\text{puntero}$
$T \rightarrow \text{id}$	$T.\text{puntero} = \text{hazHoja}(\text{id}, \text{id.nombre})$
$T \rightarrow \text{dígito}$	$T.\text{puntero} = \text{hazHoja}(\text{dígito}, \text{dígito.val})$



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Tipos de nodos de un ASA

En el ejemplo que hemos visto el tipo de nodo utilizado es para una operación aritmética, donde lo que se utiliza es una expresión binaria (dos operadores). Se debe tener en cuenta que hay una gran tipología de expresiones a la hora de diseñar el ASA, no solo las binarias.

 [Tipos de expresiones](#)
En detalle

Comprobamos que el nodo que creíamos típico, `hazNodo` (operador, `operando1.puntero`, `operando2.puntero`), no lo es tanto, y debemos definir los nodos de una manera más amplia si queremos que sirva para la mayoría de los casos



Una estructura genérica en C para un nodo podría ser la siguiente:

```
1. struct nodo {  
2.     int TipoNodo;  
3.     struct nodo *campo1;  
4.     struct nodo *campo2;  
5.     .....  
6. }
```

Donde el `TipoNodo` contiene un entero para identificar el tipo de nodo, mientras que los otros nodos apuntan a otros nodos del árbol.

Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

En detalle

Tipos de expresiones

En la mayoría de los lenguajes encontramos las siguientes (en este caso utilizamos C como ejemplo):

- Declaraciones: `int a`.
- Constantes: `#define PI 3,141592`.
- Asignaciones: `a := b`.
- Bloques.
- Operaciones unarias: `a := op b` (donde `op`, por ejemplo, puede ser `+`, `-`, `!` -negación-).
- Operaciones binarias: `a := b op c` (dentro de las operaciones binarias, además de las aritméticas, entran las operaciones lógicas y las relacionales).
- Bifurcaciones: `goto A1`.
- Bifurcaciones condicionales: `if (a == 5) goto A1` (que surgen cuando se desglosan los bucles).
- Bucles. Por ejemplo, `while` y `for`.
- `while (expr_condicional) {Sentencias }`.
- `for (a:=0; a < b, a++) {Sentencias }`.
- Control de flujo: `if condición then S1, if condición then S1 else S2, switch`.
- Operaciones con arrays: asignación y carga (`a := b[i]` o `b[i] = a`).
- Operaciones con punteros: `a := *`, `a := *b`, `*a:=b`.
- Llamadas a funciones: `x= f(arg1, arg2,..argN)`.

The logo for Cartagena99 features the text 'Cartagena99' in a stylized, blue, serif font. The '99' is significantly larger and more prominent than the 'Cartagena' part. The text is set against a light blue background with a white shadow effect, and a yellow arrow-like shape points upwards from behind the text.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

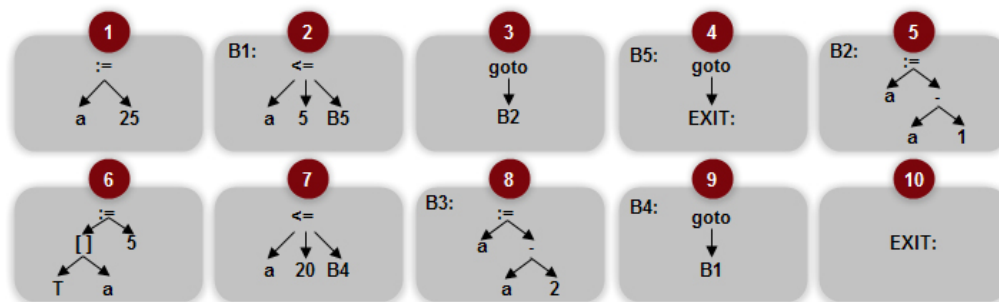
Secuencia de nodos en un ASA

Vamos a suponer el siguiente ejemplo:

```
1. a := 25;  
2. while a > 5 {  
3. a:= a -1;  
4. T[a] := 5;  
5. if (a > 20) {  
6. a:= a -2;  
7. }  
8. }
```

Vemos que el bucle while, se descompone en varios bloques: B1, goto B2 (si se cumple la condición), B5 (si no se cumple la condición). Con la sentencia if, también pasa lo mismo, se descompone en un salto si $a \leq 20$, a B4 en caso de no cumplir la condición y a B3 en caso de cumplir la condición.

Por tanto el AST debe poder unir todos los nodos, para construir el árbol y probablemente deba apoyarse en alguna estructura, tipo vector para ordenar los nodos dentro del struct que forma el árbol, además de apoyarse en la tabla de símbolos para controlar los ámbitos de las variables.

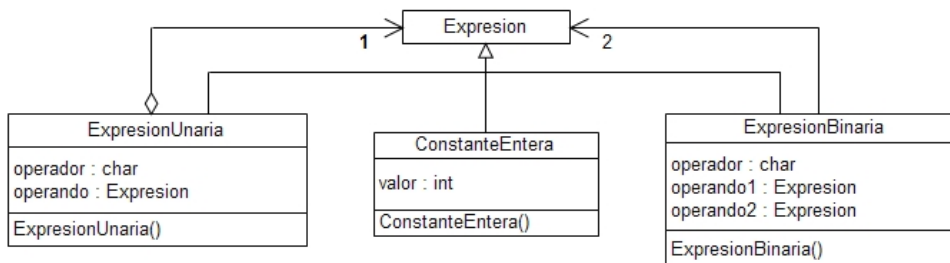


CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Otras implementaciones de un ASA

Ortín Soler et al, (2004), utilizan el siguiente diagrama de clases, donde todos los nodos son instancias derivadas de la clase abstracta expresión.



Su implementación en Bison sería la siguiente:

```
expresión: expresión '+' termino {$$ = new ExpresionBinaria('+', $1, $3); }
          | expresión '-' termino {$$ = new ExpresionBinaria('-', $1, $3); }
          | termino {$$ = $1; }
          ;
termino: termino '*' factor {$$ = new ExpresionBinaria('*', $1, $3); }
        | termino '/' factor {$$ = new ExpresionBinaria('/', $1, $3); }
        | termino {$$ = $1; }
        ;
factor: '-' factor {$$ = new ExpresionBinaria('-', $2); }
       | '(' expresión ')' {$$ = $2; }
       | CTE_ENTERA {$$ = new ConstanteEntera($1); }
       ;
```

Clase abstracta expresión

Veremos que todas las expresiones binarias, serán instancias de la clase ExpresionBinaria, y mediante un atributo tipo char se define el operador.

Lo mismo se puede hacer con las expresiones unarias y las declaraciones de constantes.

Implementación

En el documento citado de Ortín Soler et al, se describe como mediante el patrón de diseño Visitor se puede implementar.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

GENERACIÓN DE CÓDIGO INTERMEDIO

ÁRBOLES DE SINTAXIS ABSTRACTA (ASA)

Resumen

En este tema hemos comprendido porqué es necesario realizar la generación de código intermedio y qué ventajas tiene, como el facilitar la redestinación o poder aplicar la optimización de forma independiente de la máquina.

También se ha visto que hay distintos tipos de máquinas para las que generar código, en función de las direcciones que podemos utilizar a la vez en una operación (0, 2 o 3 direcciones).

Aunque hay varias modalidades de generación de código intermedio, las más ampliamente usadas son los ASA (cuando no se va a realizar la fase de optimización) o su variante mejorada los grafos dirigidos acíclicos (GDA) y el código de tres direcciones.

Se ha podido ver con varios ejemplos, primero con operaciones aritméticas y posteriormente con uno más completo, como se construye el ASA y que tipos de nodos podemos encontrarnos en un lenguaje genérico. Asimismo se han comentado otras implementaciones que pueden resultar útiles a la hora de diseñar y construir un compilador.

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99