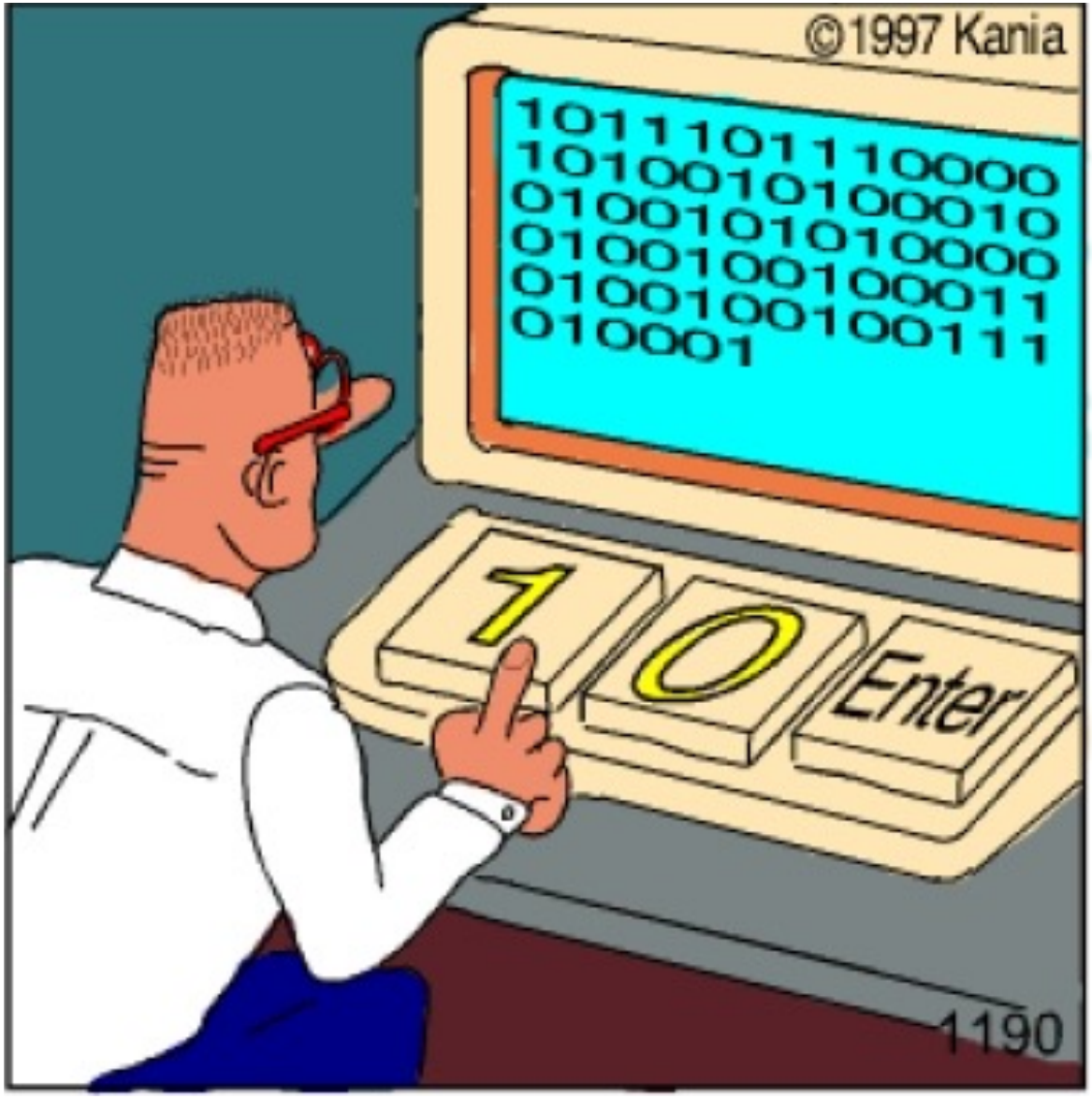
Tema 1. Lenguajes de programación y procesadores de lenguajes

1. Lenguajes de programación y traductores.
    - *1.1. Compiladores e intérpretes.*
    - *1.2. La máquina virtual Java*
2. Especificación de un lenguaje de programación
3. Modelo conceptual de un compilador
4. Modelos de programación.
5. Entornos de Compilación.

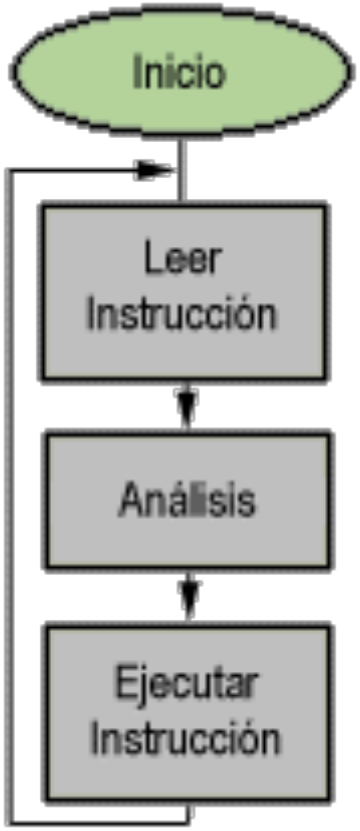
1. Lenguajes de programación y traductores

Programamando sin traductores

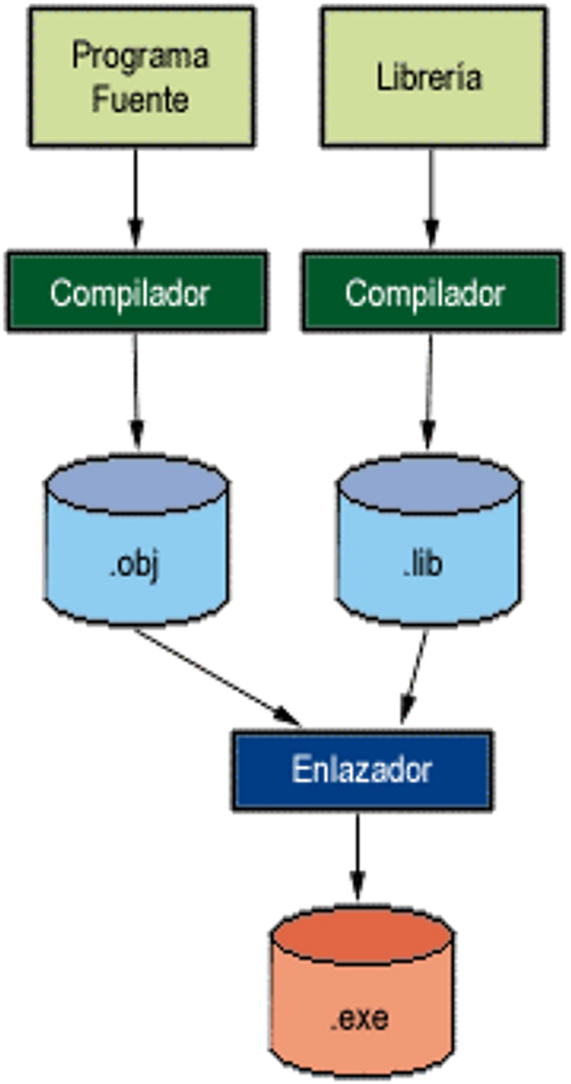


Intérprete vs Compilador

Intérprete



Compilador



Intérprete vs Compilador

Compilador

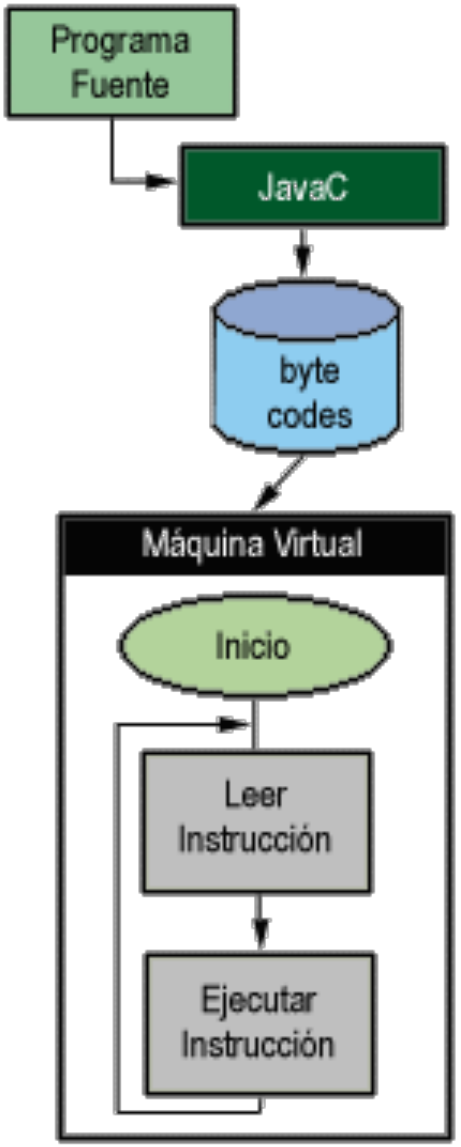
- Genera un ejecutable
- El código fuente no es necesario para la ejecución
- Solo se realiza la traducción 1 vez
- Traducción en tiempo de compilación
- Comprobación de errores en tiempo de compilación
- Más posibilidades de optimización: El ejecutable consume menos recursos
- El compilador no tiene que estar presente en tiempo de ejecución
- …

Intérprete

- Más fácil depurar errores (se puede ejecutar parcialmente un programa con errores)
- El código no es dependiente de la plataforma: fácil portabilidad
- Mayor control sobre el código: se ejecuta sobre el intérprete
- …

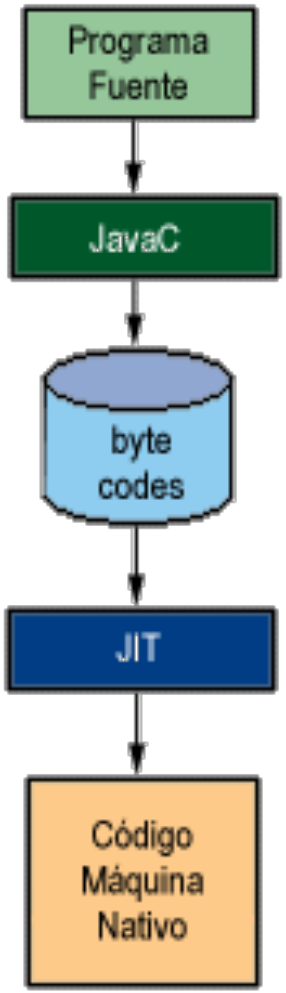
Máquina virtual Java (JVM)

Intérprete de ByteCodes

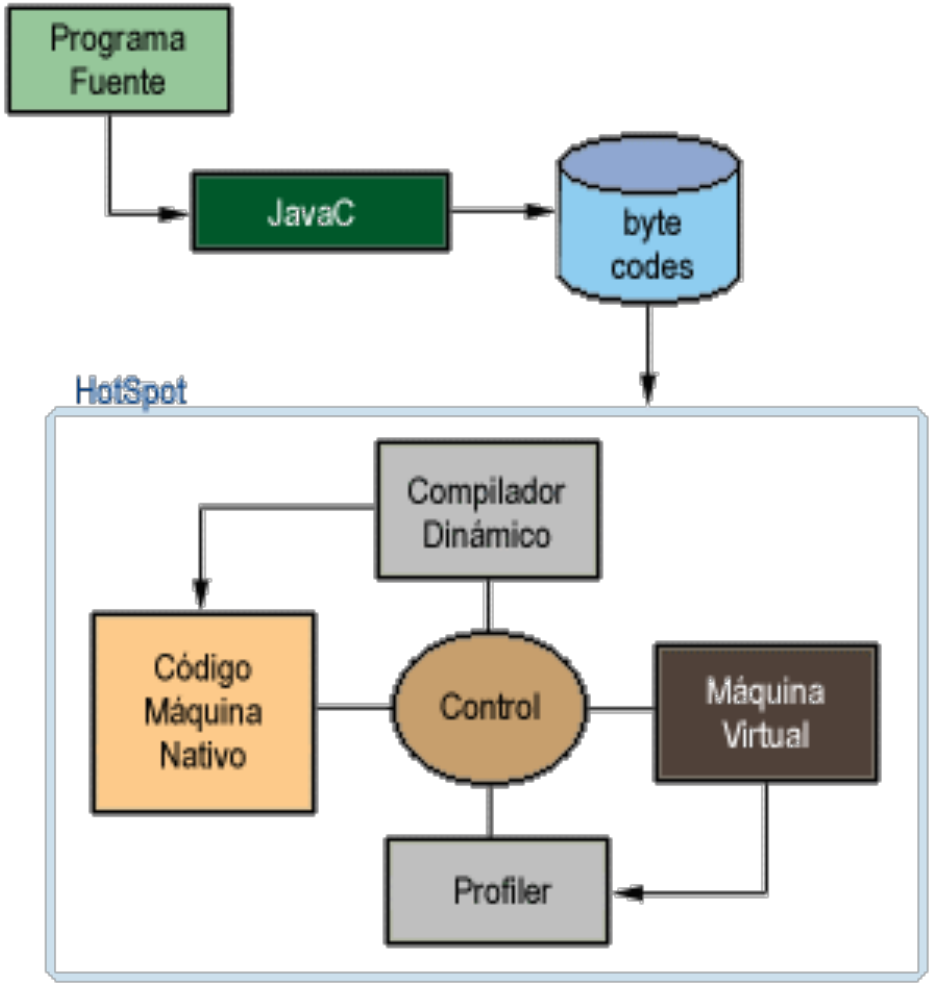


Compilador JIT vs Hot-Spot

Compilador Just-In-Time



Hot Spot



Inconvenientes del compilador JIT

- El compilador se ejecuta en tiempo de ejecución: el usuario percibe retardos en la ejecución. No se implementan optimizaciones costosas

- Las optimizaciones típicas de los lenguajes estructurados son menos eficaces en Java:
  - Java es dinámicamente seguro: Comprobaciones de tipo en tiempo de ejecución
  - Muchos objetos se almacenan en el montículo (heap) → Recolección de basura menos eficiente
  - Muchas llamadas virtuales a métodos (posiblemente polimórficos) -> Más difícil optimizar las llamadas.
  - Los programas pueden cambiar en tiempo de ejecución→ Más difícil aplicar optimizaciones globales

Java HotSpot Virtual Machine

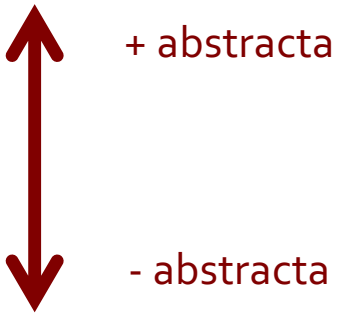
- Dos versiones:
  - Cliente
  - Servidor
- Biblioteca compartida del entorno de ejecución de Java.
- Proporciona cargador de clases, hilos de ejecución, sincronización de objetos, recolector de basura, monitorización, herramientas de depuración,…

Java HotSpot Virtual Machine

- HotSpot VM ejecuta el programa en su intérprete y analiza la ejecución del código detectando porciones críticas ("Hot spots").
- Incluye un compilador dinámico que compila las porciones críticas de bytecodes en código máquina optimizado
- Compilador de 3 fases:
  1. Front-end que convierte los bytecodes a un formato intermedio de alto nivel
  2. Back-end que genera una representación de bajo nivel
  3. Optimización por método de la mirilla, asignación de registros y generación de código máquina

2. Especificación de un lenguaje de programación

- Léxica
  - *Expresiones regulares*
- Sintáctica
  - *Gramáticas independientes del contexto*
- Semántica
  - Para comprobaciones semánticas
  - Para definir la dinámica de la ejecución
    - *Semántica axiomática*
    - *Semántica denotacional*
    - *Semántica operacional*



Semántica axiomática

El significado se describe mediante sentencias lógicas sobre el efecto de la ejecución de un programa

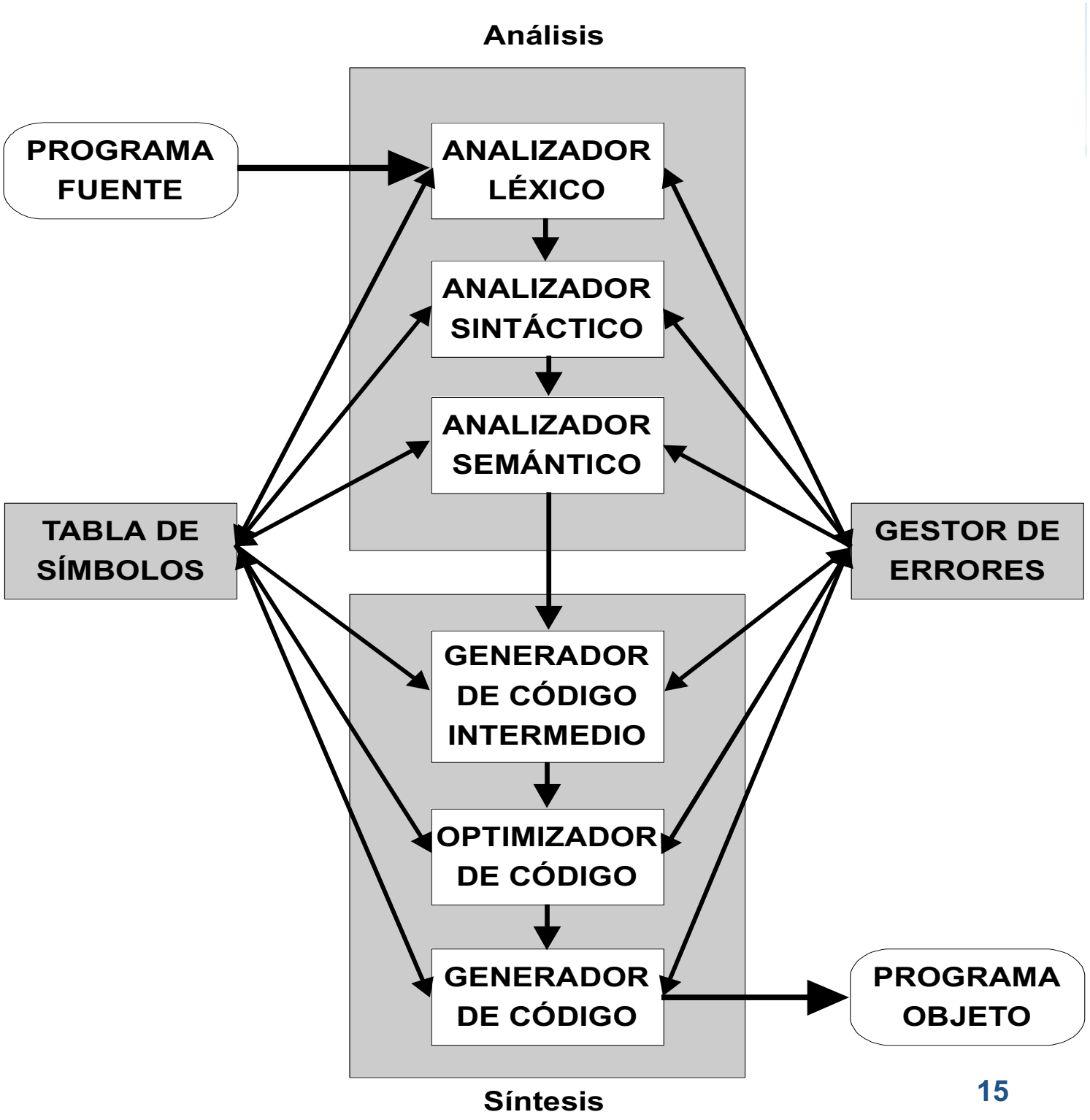
Semántica denotacional

El significado viene dado por un objeto matemático (una función), estando la noción de estado presente

Semántica operacional

El significado se define mediante una máquina abstracta (con estados) y secuencias de cómputos sobre dicha máquina

3. Modelo conceptual de compilador


Análisis
PROGRAMA FUENTE
ANALIZADOR LÉXICO
ANALIZADOR SINTÁCTICO
ANALIZADOR SEMÁNTICO
TABLA DE SÍMBOLOS
GESTOR DE ERRORES
GENERADOR DE CÓDIGO INTERMEDIO
OPTIMIZADOR DE CÓDIGO
GENERADOR DE CÓDIGO
PROGRAMA OBJETO
Síntesis

Ejemplo

TDS	
...	...
ind_x	x
ind_y	y
ind_z	z
...	...

Cod	
...	...
cod_+	+
cod_*	*
$cod_/$	/
...	...

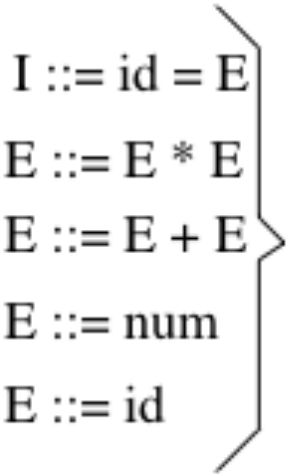
$x = y + z * 4$

AL

$(id,ind_x)(opasig)(id,ind_y)(op,cod_+)\ (id,ind_z)(op,cod_*)(num,val_4)$

Ejemplo

AS



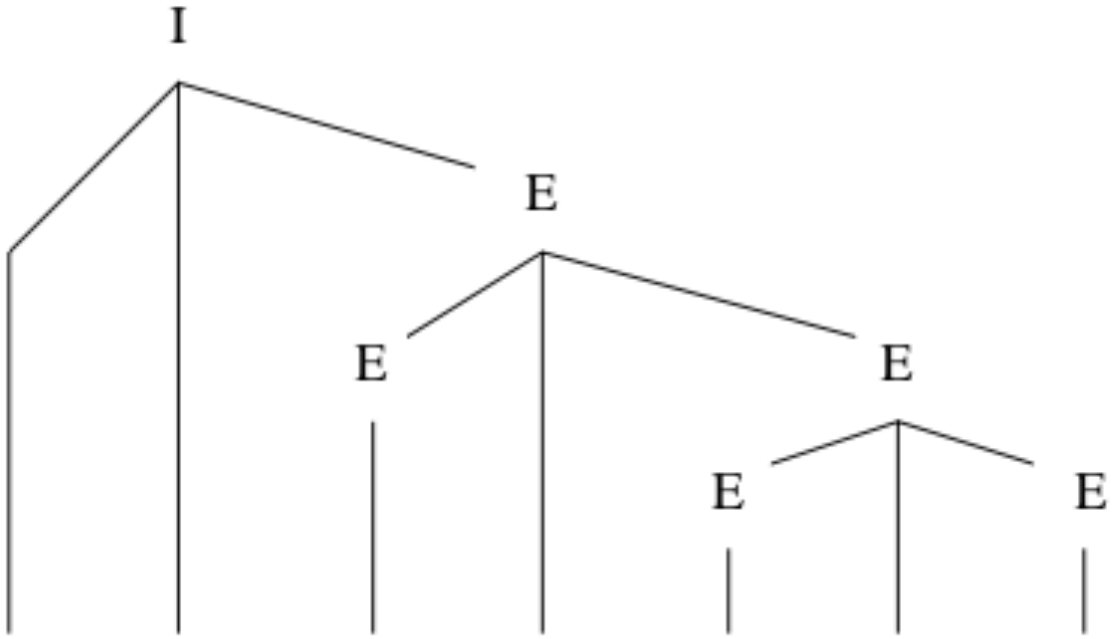
I ::= id = E

E ::= E * E

E ::= E + E

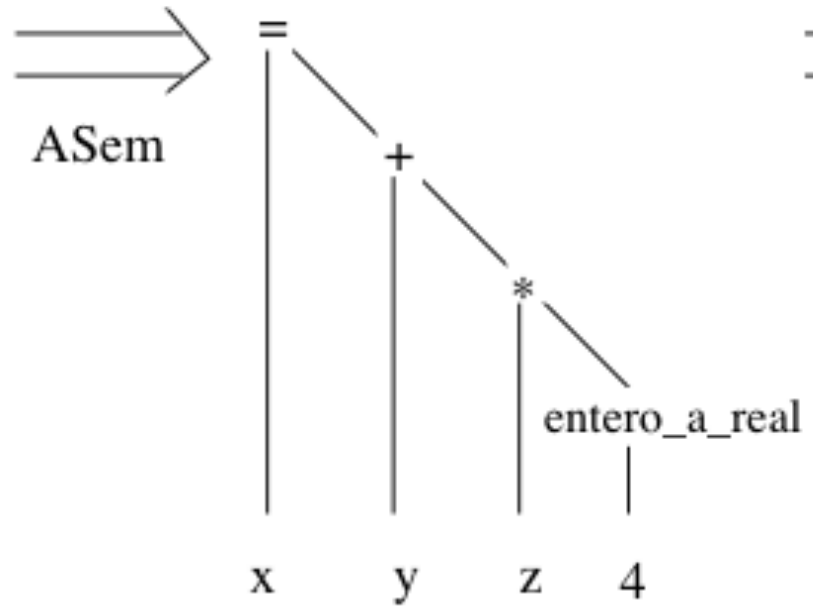
E ::= num

E ::= id



(id, ind_x) (opasig) (id, ind_y) (op, cod_+) (id, ind_z) (op, cod_*) (num, val_4)

Ejemplo



T D S

	id	tipo	dir
...	...	...	...
ind_x	x	t_x	d_x
ind_y	y	t_y	d_y
ind_z	z	t_z	d_z
...	...	...	...

GCI

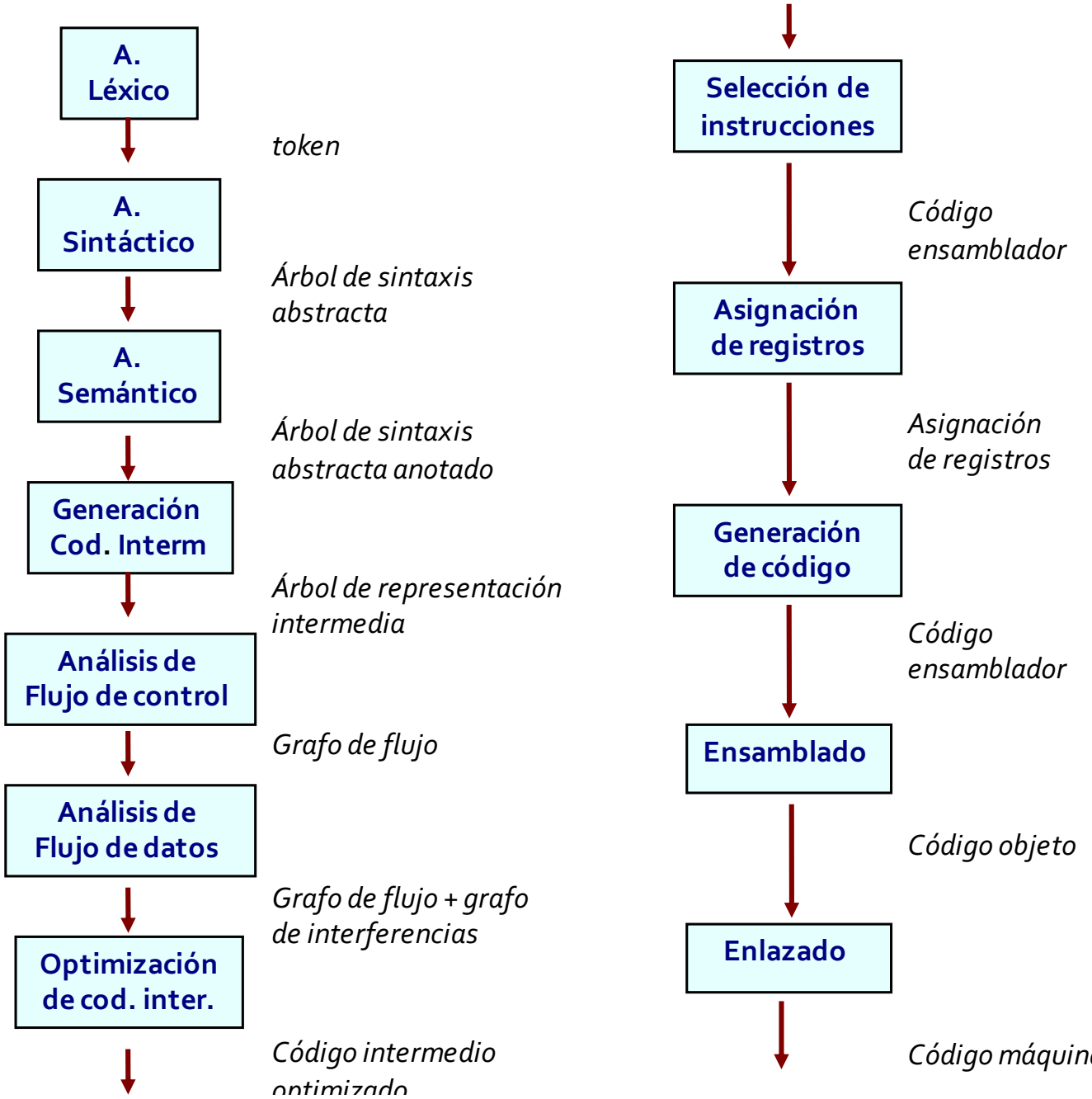
$d_{t1} \leftarrow$ entero_a_real 4
$d_{t2} \leftarrow d_z * d_{t1}$
$d_{t3} \leftarrow d_y + d_{t2}$
$d_x \leftarrow d_{t3}$

OCI

$d_{t2} \leftarrow d_z * 4.0$
$d_x \leftarrow d_y + d_{t2}$

GC

RMULT d_z 4.0 d_{t2}
RSUM d_y d_{t2} d_x


A. Léxico
token
A. Sintáctico
Árbol de sintaxis abstracta
A. Semántico
Árbol de sintaxis abstracta anotado
Generación Cod. Interm
Árbol de representación intermedia
Análisis de Flujo de control
Grafo de flujo
Análisis de Flujo de datos
Grafo de flujo + grafo de interferencias
Optimización de cod. inter.
Código intermedio optimizado
Selección de instrucciones
Código ensamblador
Asignación de registros
Asignación de registros
Generación de código
Código ensamblador
Ensamblado
Código objeto
Enlazado
Código máquina

4. Modelos de programación

- Lenguajes imperativos.
- Orientados a objetos
- Lenguajes funcionales. Lógica ecuacional. Lisp, Haskell, ML
- Lenguajes lógicos. Lógica clausal. Prolog
- Concurrencia. Más de un contexto de ejecución activo
- Lenguajes de script. Perl, Python, PHP.

5. Entornos de compilación

- Preprocesadores (macros, archivos incluidos, …)
- Enlazador (varios archivos-> un ejecutable)
- Editores sensibles al contexto
- Analizadores de código
- EDI (Entornos de Desarrollo Integrados): Manejo de proyectos grandes (varios programadores), crear interfaces de usuario