

Tema 1 | Procesadores de Lenguajes

1.1 Conceptos previos.

- 1.1.1 Concepto de traducción.
- 1.1.2 Requisitos para construir un traductor.
- 1.1.3 Esquema de traducción.

1.2 Aplicaciones de los traductores.

1.3 Visión general de un traductor.

- 1.3.1 Diseño del traductor desde sus requerimientos.
- 1.3.2 Fase de análisis, problemas y restricciones de aplicación.
- 1.3.3 Fase de síntesis.
- 1.3.4 Modelo general de un traductor.

1.4 Tipos y evolución de los traductores.

- 1.4.1 Conceptos y evolución de los lenguajes de programación.
- 1.4.2 Evolución de los traductores.
- 1.4.3 Compiladores.
- 1.4.4 Intérpretes.

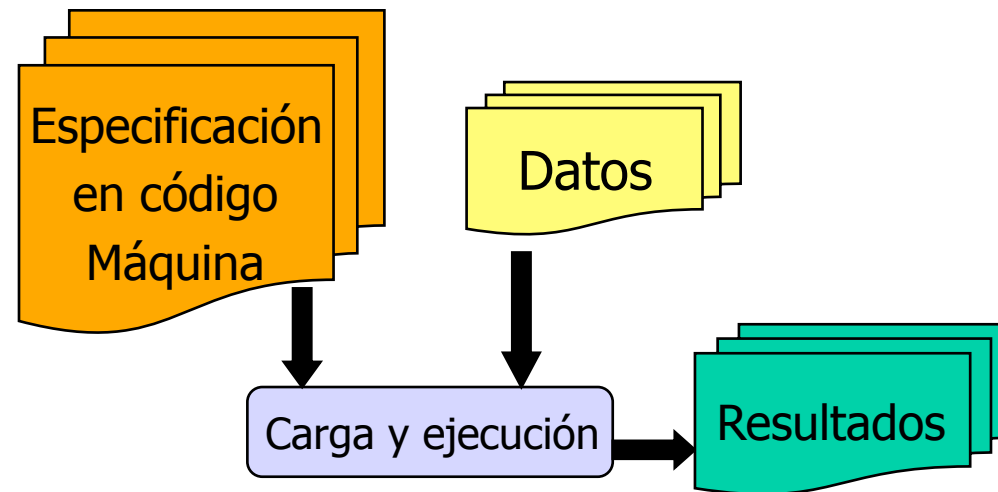
1.5 Arquitectura de procesadores de lenguajes.

20/Feb/2011

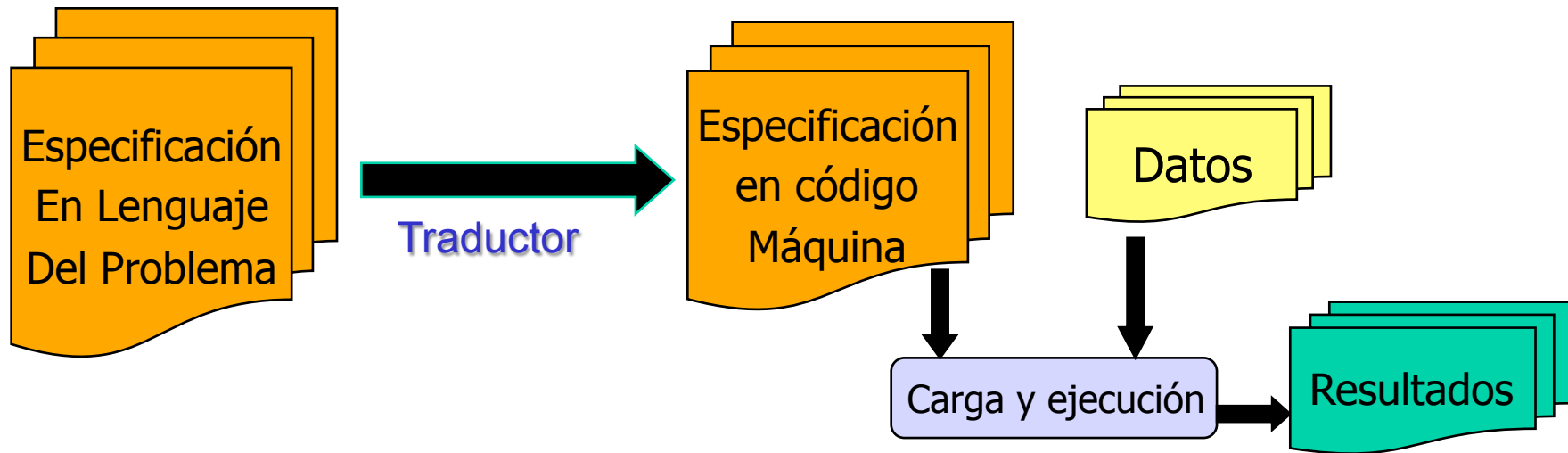
PROCESADOR DE LENGUAJE:

Estudia el proceso necesario para que una especificación de entrada sea ejecutada en una **máquina real**.

Proceso para ejecutar una especificación en lenguaje máquina en una Máquina Real



Procesamiento de lenguajes: Cuando el lenguaje no es máquina



Cuando el lenguaje de especificación es diferente al lenguaje máquina, Será necesario traducirlo a lenguaje máquina para completar el procesamiento del lenguaje.

Todo lenguaje de programación (no máquina) define una **MÁQUINA VIRTUAL**.

Estrategias para implementar el traductor en el procesamiento del lenguaje

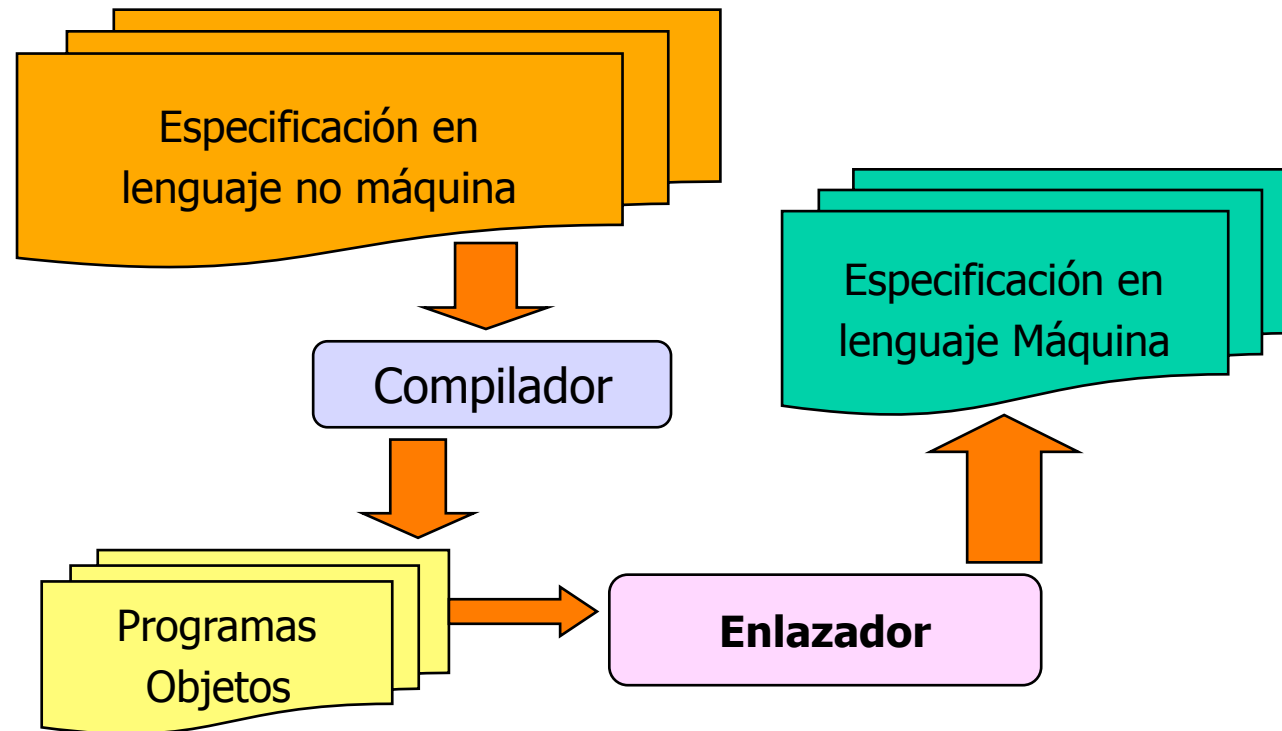
- Traducción a lenguaje máquina

Obtener una especificación en lenguaje máquina equivalente.

- Interpretación.

Cargar un intérprete que cargue la especificación y la ejecute.

Estrategia de traducción



Estrategia de traducción (2)

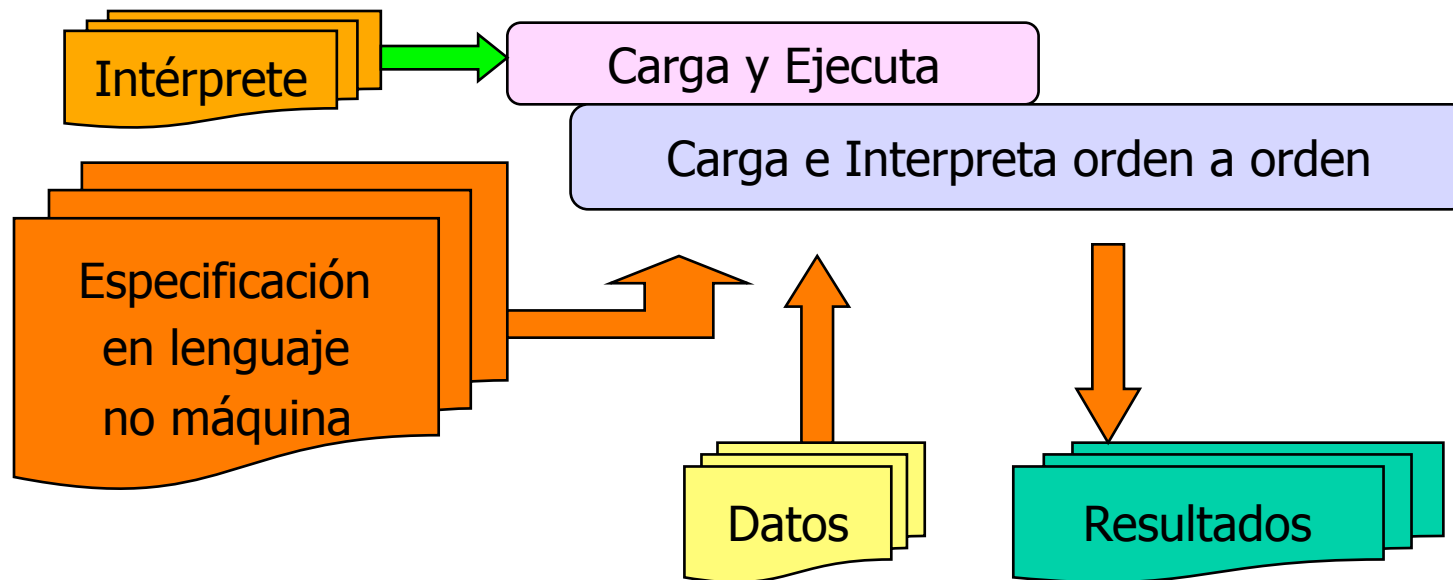
- **COMPILACIÓN:**

Traduce la especificación de entrada a lenguaje máquina incompleto y con instrucciones máquina incompletas.

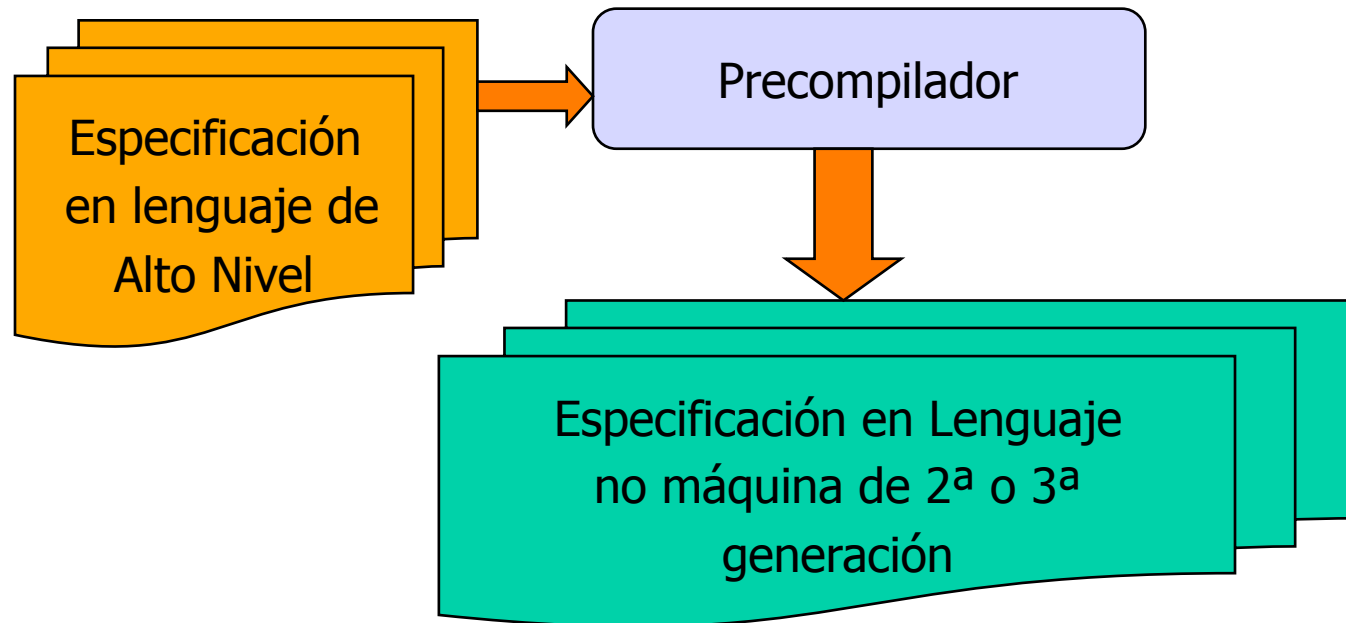
- **ENLAZADOR:**

Enlaza los programas objetos y completa las instrucciones máquina incompletas, generando un ejecutable.

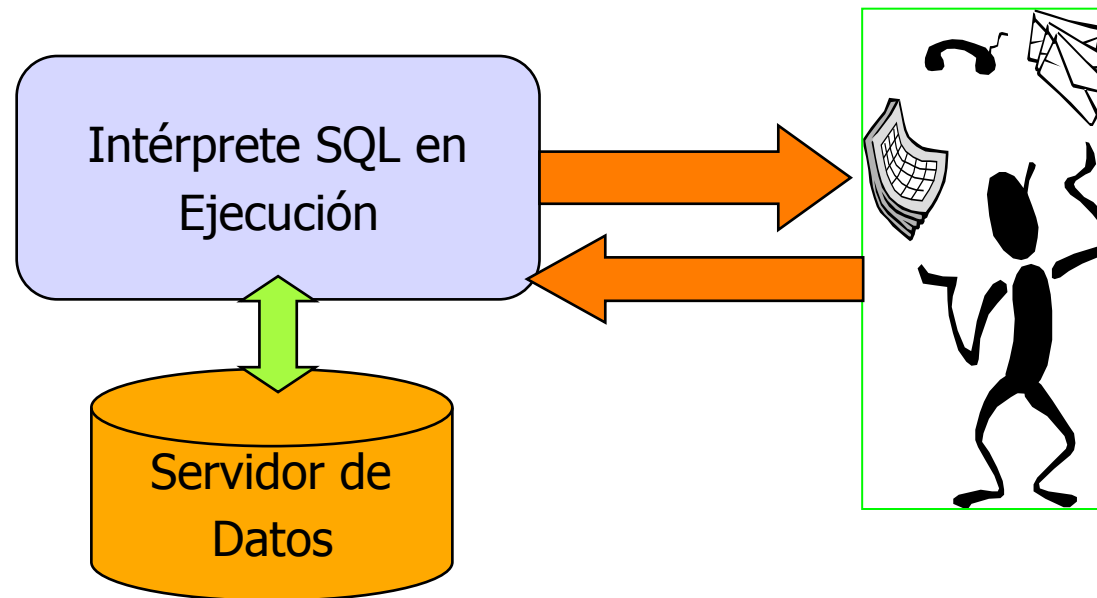
Estrategia de Interpretación



Lenguajes Generadores de Programas. Macroensambladores



Manejadores de Bases de Datos

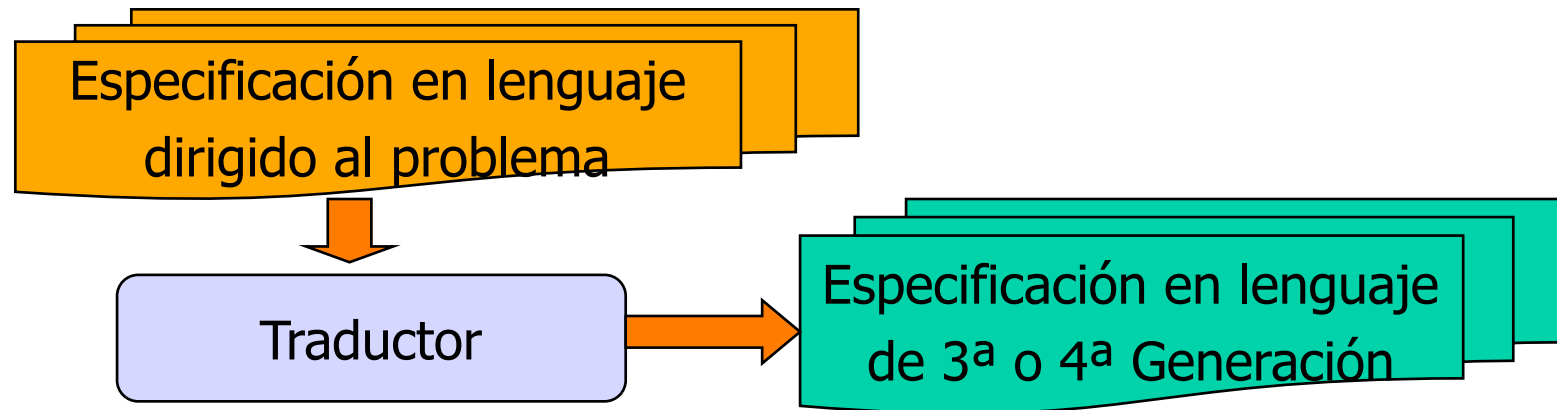


Lenguajes de 3ª o 4ª Generación con interfaz DBMS

- Se procesan como las especificaciones a generadores de programas o macro-ensambladores y
- Generando llamadas a funciones de acceso al DBMS como interpretación de las instrucciones de alto nivel de manejo de datos.

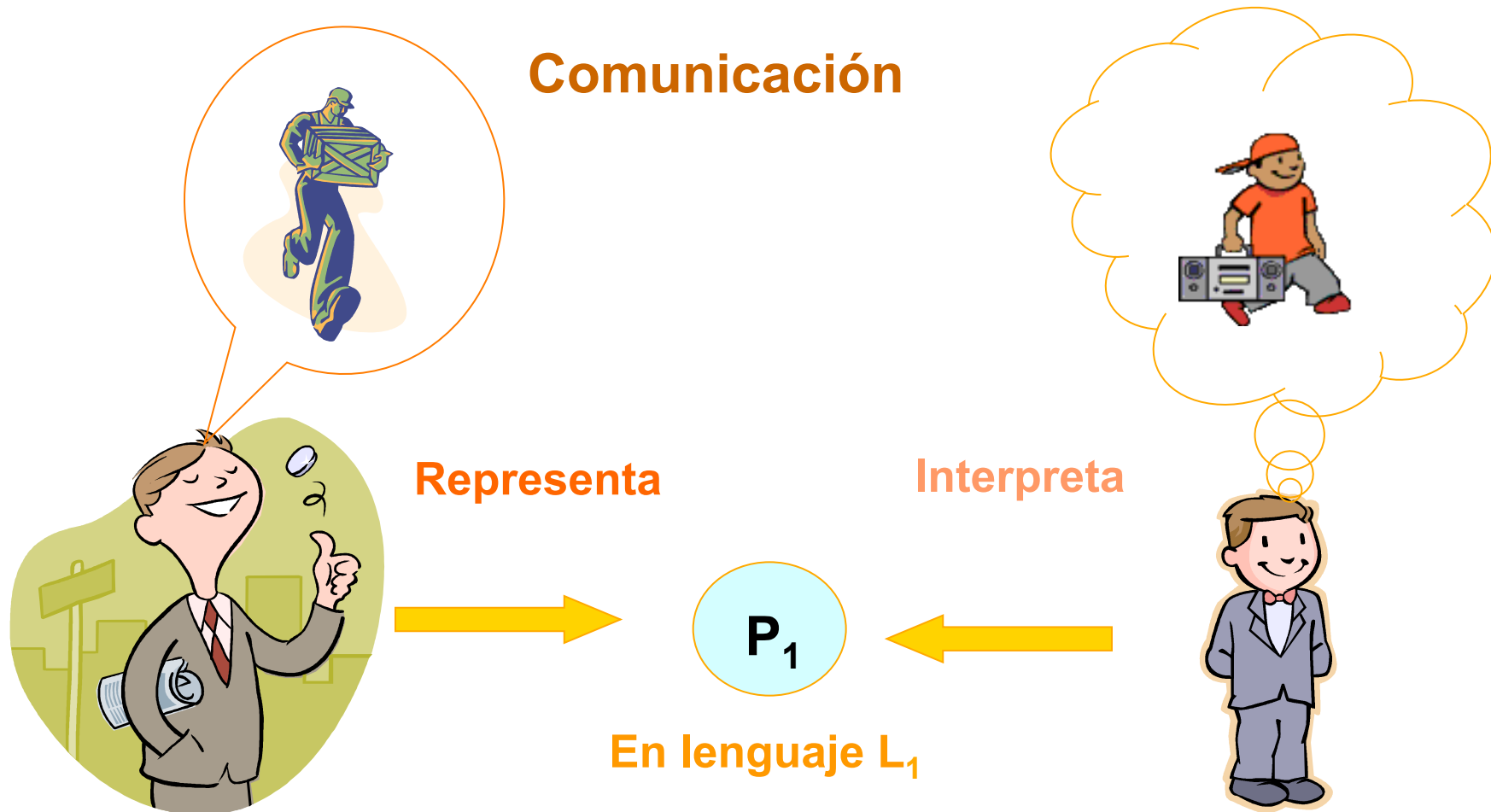
Lenguajes dirigidos al Problema

- Alternativa para el desarrollo de sistemas complejos:
 - Diseñar un **Lenguaje Orientado al Problema**.
 - Construir su **Traductor** (*Compilador o Intérprete*).

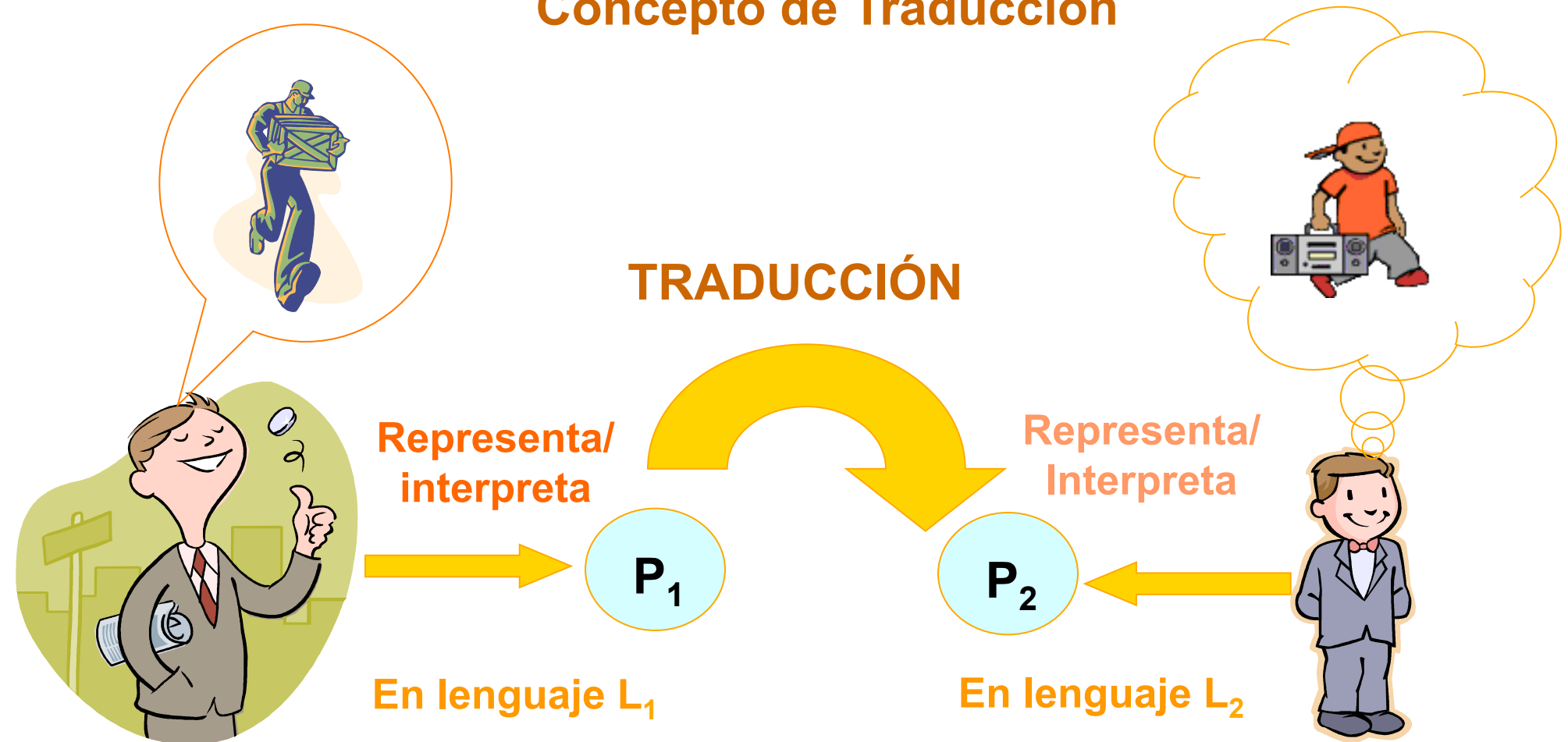


Ejemplo de Lenguajes Dirigidos al Problema : Lenguaje de especificación de un sistema genérico sectorial

- Imaginemos la gestión de un almacén, donde se dispone de *artículos*, *clientes*, *proveedores*, así como de las acciones de *compra*, *venta*, *baja* y *pedido*.
- Se podría diseñar un lenguaje en donde se expresaran las acciones de la forma:
 - *venta* cantidad_1 articulo_1 *a* cliente_1
 - *si* cantidad_1 < 20 *entonces* *pedido* articulo_1 *a* proveedor_1
 - *baja* cantidad_1 *existencia actual* cantidad_2




Concepto de Traducción



Concepto de Traducción

- La traducción relaciona a dos o más
 - **Modelos Semánticos (Culturas)**



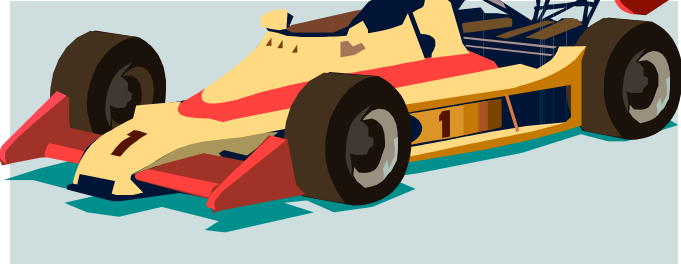
SEMÁNTICA
LENGUAJE
INTERPRETACIÓN

Dominio Semántico

Valores Semánticos

Elementales

Estructurados



coche



Juan



Juan conduce el coche

Lenguaje

- Conjunto de construcciones simbólicas formadas por secuencias de símbolos de un alfabeto
 - Se utiliza para representar los valores semánticos de un dominio semántico
 - Un lenguaje puede definirse:
 - Por Extensión
 - Por Comprensión
- GRAMÁTICA

Interpretación

Sea D el dominio semántico y $L(G)$ el lenguaje definido por la gramática G , Se define la **interpretación** como una aplicación I :

$$I : L(G) \longrightarrow D$$

Se nota el modelo semántico o modelo de la interpretación por

$$M = (L(G), D, I)$$

Principio de Traducción

Sea $M_1 = (L_1(G_1), D, I_1)$ y $M_2 = (L_2(G_2), D, I_2)$

dos modelos semánticos, tal que

$$I_1 = L_1(G_1) \longrightarrow D \text{ y } I_2 = L_2(G_2) \longrightarrow D$$

Para que pueda haber traducción deberá cumplirse:

$$\forall \alpha \in L_1(G_1), \exists \beta \in L_2(G_2) / I_1(\alpha) = I_2(\beta)$$

Esquema de Traducción

- Dados los modelos semánticos:

$$M_1 = (L_1(G_1), D, I_1) \text{ y } M_2 = (L_2(G_2), D, I_2)$$

Se define el Esquema de Traducción:

$$T : L_1(G_1) \longrightarrow L_2(G_2)$$

Tal que:

$$\forall \alpha \in L_1(G_1), \exists \beta \in L_2(G_2) \text{ con } T(\alpha) = \beta \text{ y } I_1(\alpha) = I_2(\beta)$$

Requisitos para la construcción de traductores

- Dados dos lenguajes $L_1(G_1)$ y $L_2(G_2)$

Se necesitan dos modelos semánticos:

$$M_1 = (L_1(G_1), D_1, I_1) \text{ y } M_2 = (L_2(G_2), D_2, I_2)$$

Tal que se debe cumplir:

$$D_1 \subseteq D_2 \text{ y}$$

$$\forall \alpha \in L_1(G_1), \exists \beta \in L_2(G_2) / I_1(\alpha) = I_2(\beta)$$

Aplicaciones de los Traductores (1)

- Lenguajes de Programación
- Lenguaje dirigidos al problema
- Lenguajes de programación híbridos.
- Desarrollo de sistemas software

Aplicaciones de los Traductores (2)

- Traductores de lenguajes de programación:
 - El dominio semántico (evaluación y control) de los lenguajes procedurales coincide con el dominio semántico del lenguaje máquina.
- Traductores dirigidos al problema:
 - El problema se puede expresar en términos de valores semánticos elementales y estructurados del modelo semántico del problema.
 - Podremos definir un lenguaje dirigido al problema que permita formular el problema más cercana a la cultura del usuario.

Aplicaciones de los Traductores (3)

Lenguajes Dirigidos al Problema (ejemplo)

- Imaginemos la gestión de un almacén, donde se dispone de *artículos*, *clientes*, *proveedores*, así como de las acciones de *compra*, *venta*, *baja* y *pedido*.
- Se podría diseñar un lenguaje en donde se expresaran las acciones de la forma:

- *venta* cantidad_1 articulo_1 *a* cliente_1
- *si* cantidad_1 < 20 *entonces* *pedido* articulo_1 *a* proveedor_1
- *baja* cantidad_1 *existencia actual* cantidad_2

Aplicaciones de los Traductores (4)

Traductores Dirigidos al Problema (ejemplo)

- Este lenguaje nos permitiría poder expresar nuestro problema a nivel de diseño, permitiendo realizar la fase de codificación de forma automática, mediante un traductor desde el lenguaje diseñado a un lenguaje de alto o bajo nivel.

Aplicaciones de los Traductores (5)

Lenguajes de Programación Híbridos

- Lenguajes que integren los paradigmas:
 - Procedural-Lógico.
 - Lógico – Funcional.
 - Funcional –Procedural
 - Lógico – Funcional – Orientado a Objetos
 - ...

Aplicaciones de los Traductores (6)

Desarrollo de Sistemas Software

- Uso de metodologías orientadas a obtener una gramática de lenguaje de órdenes:
- **Fase de análisis**, Se distinguen los niveles:
 - **Tareas:**
 - Entidades conceptuales, y
 - tareas con entidades
 - **Semántico:**
 - Entidades conceptuales,
 - conjunto de órdenes,
 - procedimientos,
 - métodos.

Aplicaciones de los Traductores (7)

Fase de Análisis

– Sintáctico:

- Ordenes,
- Argumentos,
- Descriptores,
- Contexto de órdenes,
- Variables de estado,
- Areas de Display
- Procedimientos sintácticos,
- Métodos sintácticos.

Aplicaciones de los Traductores (8)

Fase de Análisis

– Interacción con el entorno:

- Parte de interacción
- Dispositivos de entrada,
- Acciones Primitivas,
- Reglas de control del diálogo,
- Reglas de ordenes,
- Reglas de acciones de usuario y sistema,
- Procedimientos y Métodos de interacción.

Construcción de Traductores (1)

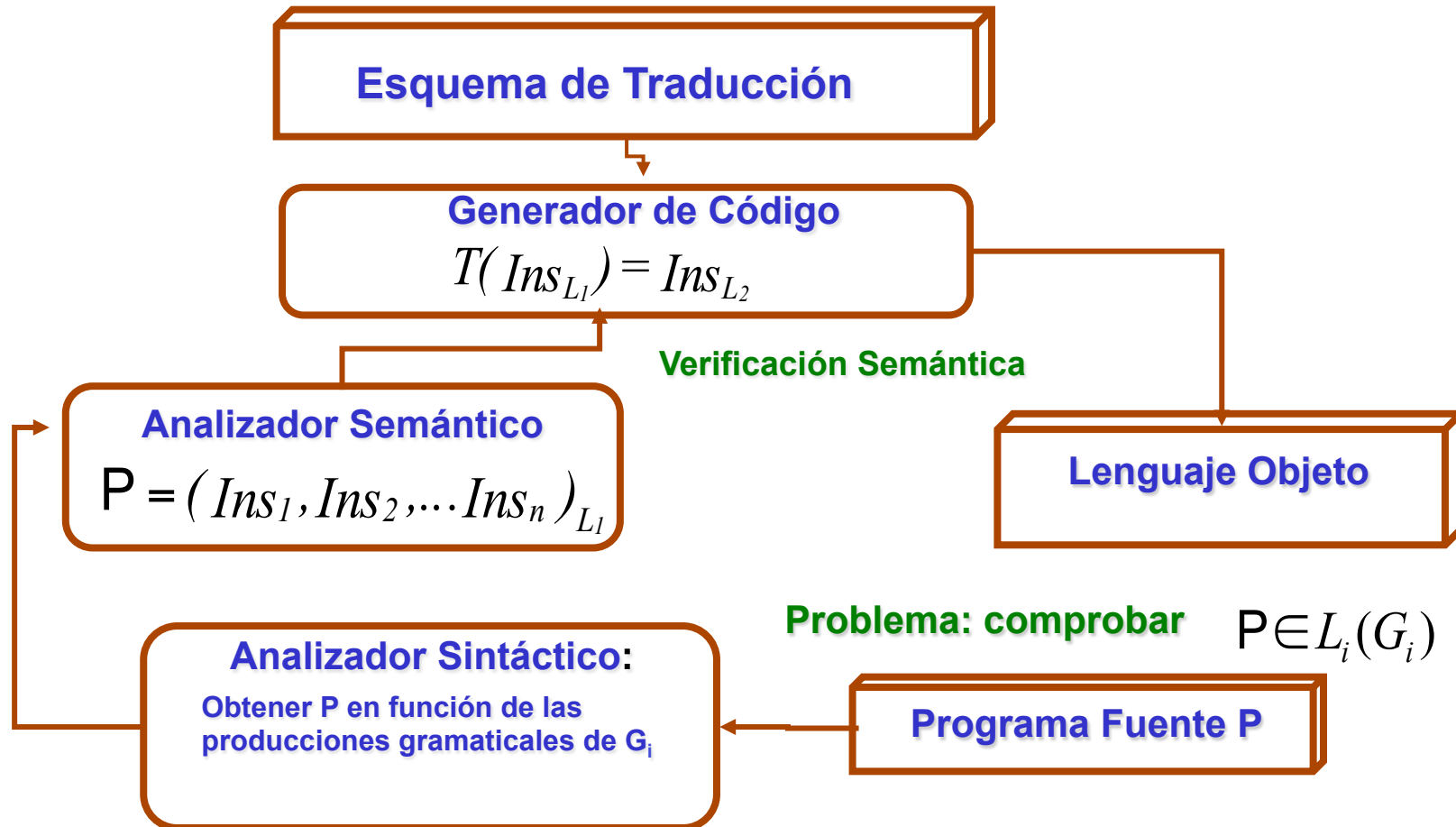
Dados los modelos semánticos $\left\{ \begin{array}{l} M_1 = (L_1(G_1), D_1, I_1) \\ M_2 = (L_2(G_2), D_2, I_2) \end{array} \right.$ y

y su esquema de traducción $T : L_1(G_1) \longrightarrow L_2(G_2)$

Para traducir un programa $P \in L_1(G_1)$

¿Qué tareas y en qué orden se deben realizar?

Construcción de Traductores (2)



Construcción de Traductores (3)

Problema: Verificación Sintáctica

- La complejidad de la verificación sintáctica depende del tipo de gramática que define el lenguaje.
- Una gramática se define mediante una cuádrupla (**N,T,P,S**), donde:
 - **N** representa el conjunto de símbolos no terminales,
 - **T** es el conjunto de símbolos terminales,
 - **P** conjunto de producciones (o reglas) gramaticales y
 - **S** símbolo (no terminal) inicial de la gramática.

Construcción de Traductores (4)

Problema: Verificación Sintáctica

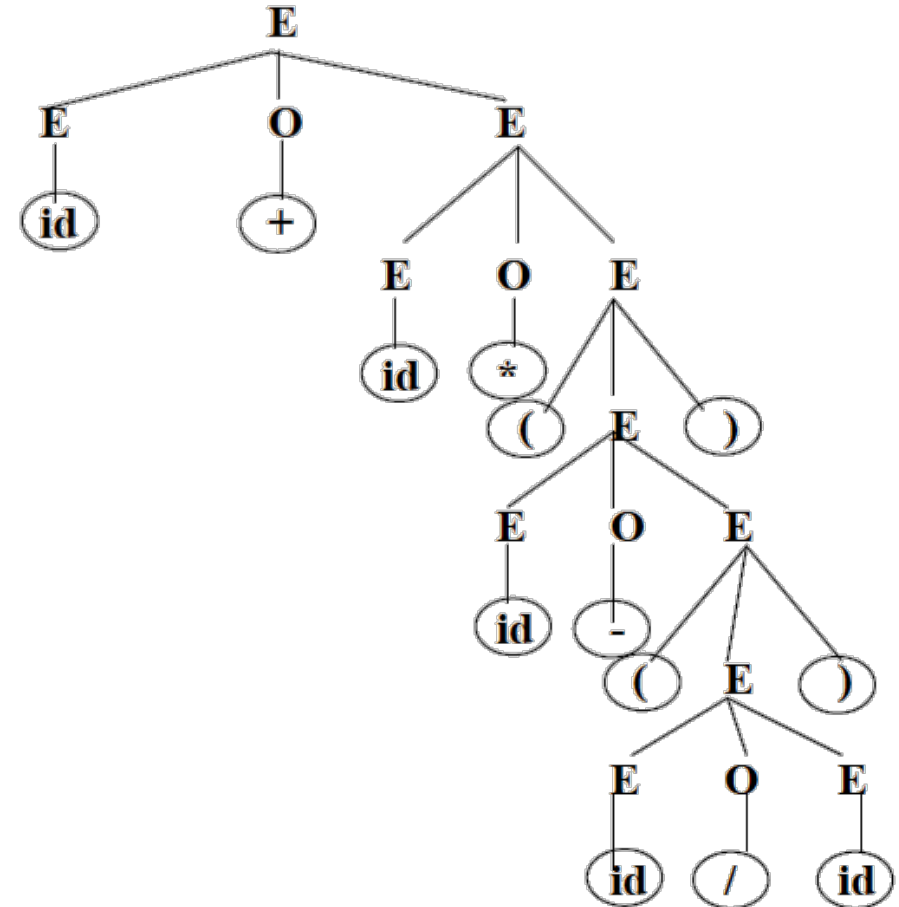
$$P = \{ \begin{array}{l} E \rightarrow EOE \\ E \rightarrow (E) \\ E \rightarrow id \\ O \rightarrow + \mid - \mid * \mid / \end{array} \}$$

$$N = \{E, O\}$$

$$T = \{id, (,), +, -, *, /\}$$

$$S = E$$

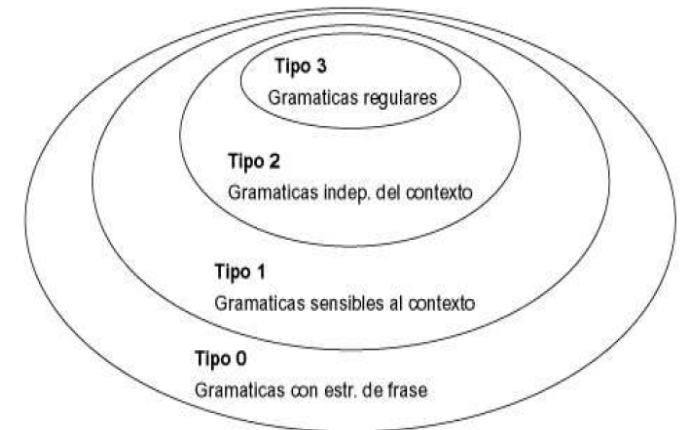
$$id + id * (id - (id / id))$$



Construcción de Traductores (5)

Clasificación de las Gramáticas (según Chomsky)

Tipo 3	Gramáticas regulares (Secuencias de símbolos: Palabras) $P = \{A \rightarrow aB ; A \rightarrow a / a \in T, B \in (V_N \cup V_T)^*\}$ Son reconocidas por un <i>autómata finito</i>
Tipo 2	Gramáticas independientes del contexto (Secuencias de palabras) $P = \{A \rightarrow \alpha / A \in V_N ; \alpha \in (N \cup T)^*\}$ Son reconocidas por un <i>autómata con pila (push-down)</i>
Tipo 1	Gramáticas sensibles al contexto $P = \{\alpha A \beta \rightarrow \alpha \gamma \beta / A \in V_N ; \gamma \in (V_N \cup V_T)^+ ; \alpha, \beta \in (V_N \cup V_T)^*\}$ Son reconocidas por un <i>autómata lineal limitado</i>
Tipo 0	Gramáticas con estructura de frase $P = \{\alpha \rightarrow \beta / \alpha \in (V_N \cup V_T)^+ ; \beta \in (V_N \cup V_T)^*\}$ Son reconocidas por una <i>máquina de Turing</i>



Construcción de Traductores (6)

Complejidad de la Verificación Sintáctica

- Para asegurar que una cadena de símbolos pertenece a un lenguaje , es necesario aplicar todas las combinaciones posibles sobre las producciones de la gramática y comprobar que puede generarse . Esto no siempre es posible.
- Supongamos la gramática:

$$E \rightarrow EOE, E \rightarrow id, O \rightarrow + \mid -$$

- Si aplicamos una estrategia descendente, nunca se termina.
- ¿CÓMO SOLUCIONAMOS ESTE PROBLEMA?

Construcción de Traductores (7)

Complejidad de la Verificación Sintáctica

- El **análisis sintáctico** se simplifica en base a los criterios:
 - Elegir **lenguajes** que estén definidos por **gramáticas** de tipo 3 ó 2.
 - Diseñar el lenguaje formado por **frases** y las frases como concatenación de **palabras** y las palabras formadas como concatenación de símbolos del alfabeto.
 - Si las frases obedecen a una estructura sintáctica, entonces el análisis sintáctico se puede descomponer en:
 - **Léxico**: Se identifican las palabras con la misma misión sintáctica.
 - **Sintáctico**: Donde se verifica la sintaxis de la secuencia de palabras. Aplicando **técnicas predictivas** (sin vuelta atrás).

Construcción de Traductores (8)

Complejidad de la Verificación Sintáctica: ANÁLISIS LÉXICO

- **Componente Léxico (token):** Conjunto de palabras que hacen la misma misión sintáctica. Ejemplo verbo, sujeto, Los tokens son definidos por expresiones regulares (sublenguaje).
- **Lexema:** palabra del conjunto de palabras que definen un token.

Construcción de Traductores (9)

Complejidad de la Verificación Sintáctica: ANÁLISIS LÉXICO

$$\begin{aligned}
 S &\rightarrow A \mid C \\
 A &\rightarrow id := E \\
 C &\rightarrow if E then S \\
 E &\rightarrow EOE \mid (E) \mid id \\
 O &\rightarrow + \mid - \mid * \mid / \\
 id &\rightarrow letra \mid id\ digito \mid id\ letra \\
 letra &\rightarrow a \mid b \mid \dots \mid z \\
 digito &\rightarrow 0 \mid 1 \mid \dots \mid 9
 \end{aligned}$$

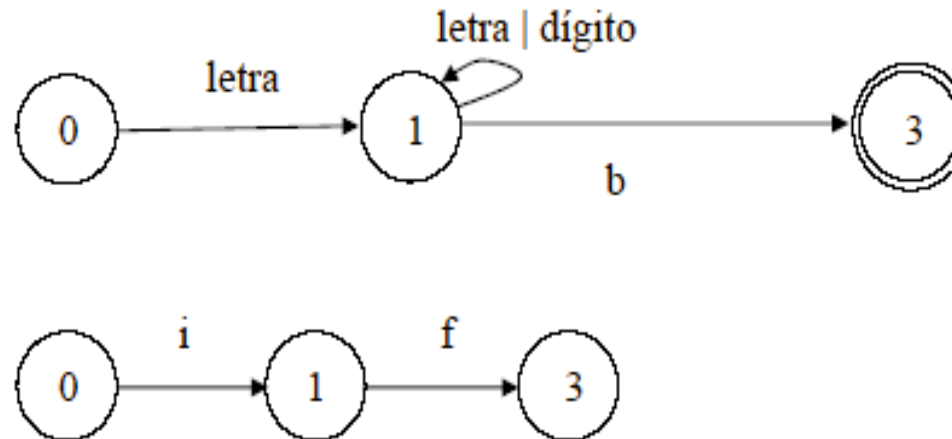
Token	Patrón / Expresión regular	Código
ID	letra(letra digito)*	256
ASIGN	" : = "	257
IF	" if "	258
THEN	" then "	259
PARIZQ	" ("	260
PARDER	") "	261
OPEBIN	" + " " - " " * " " / "	262

Ejemplo de id: su39, sus, a30r

Construcción de Traductores (10)

Complejidad de la Verificación Sintáctica: ANÁLISIS LÉXICO

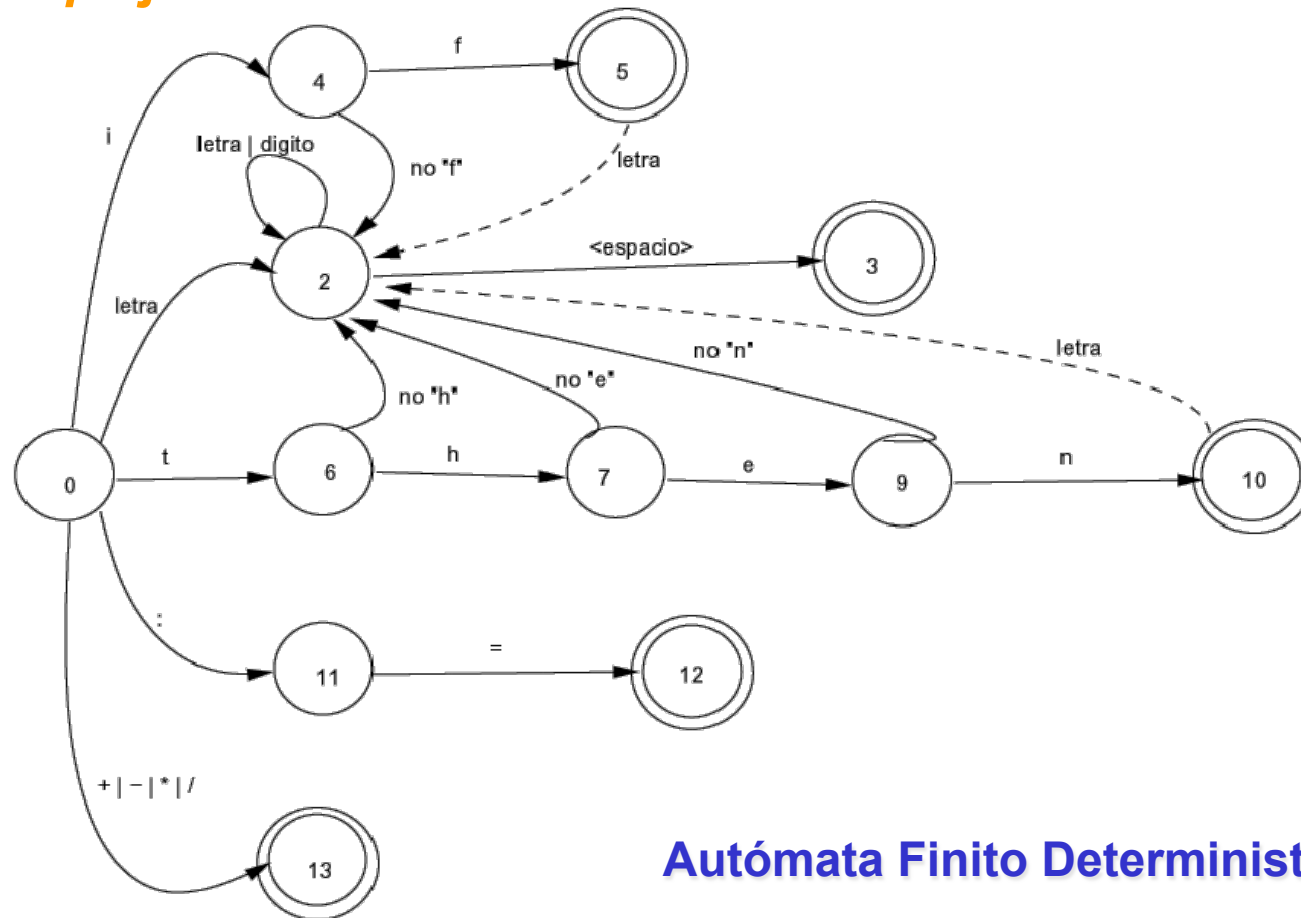
PATRONES: Cada patrón es reconocido por un Autómata finito determinista



Problema de reconocer más de un patrón a la vez...

Construcción de Traductores (11)

Complejidad de la Verificación Sintáctica: ANÁLISIS LÉXICO



Autómata Finito Determinista con Anticipación

Construcción de Traductores (12)

Complejidad de la Verificación Sintáctica: ANÁLISIS LÉXICO

- Desde el análisis sintáctico, se obvia el lexema de cada componente léxico, sólo nos interesa la presencia de un determinado componente léxico. Pero desde la perspectiva del análisis semántico y generación de código, se necesita conocer el lexema concreto.
- **¿Cómo se soluciona este problema?**. Introduciendo los lexemas en una **tabla de símbolos**, y conservando para cada componente léxico un atributo que indica el lexema concreto que representa.
- **Herramienta para construir el analizador de léxico: LEX**

Construcción de Traductores (13)

Complejidad de la Verificación Sintáctica: ANÁLISIS SINTÁCTICO

Abstracción Léxica: Simplificar la gramática original en otra gramática que denominaremos gramática abstracta.

$$\begin{aligned}
 S &\rightarrow A \mid C \\
 A &\rightarrow id := E \\
 C &\rightarrow if E then S \\
 E &\rightarrow EOE \mid (E) \mid id \\
 O &\rightarrow + \mid - \mid * \mid / \\
 id &\rightarrow letra \mid id digito \mid id letra \\
 letra &\rightarrow a \mid b \mid \dots \mid z \\
 digito &\rightarrow 0 \mid 1 \mid \dots \mid 9
 \end{aligned}$$


$$\begin{aligned}
 S &\rightarrow A \mid C \\
 A &\rightarrow \mathbf{ID \ ASIGN \ E} \\
 C &\rightarrow \mathbf{IF \ E \ THEN \ S} \\
 E &\rightarrow \mathbf{E \ OPBIN \ E} \\
 &\quad \mid \mathbf{PARIZQ \ E \ PARDER} \\
 &\quad \mid \mathbf{ID}
 \end{aligned}$$

Estrategia de análisis

Descendente

Ascendente

Construcción de Traductores (14)

Complejidad de la Verificación Sintáctica: A. SINTÁCTICO PREDICTIVO

- Ante un símbolo de entrada poder decidir si procede, o no, para obtener una cadena del lenguaje, sin tener que generar todas las posibles cadenas del lenguaje.
- Es válido si la gramática no es ambigua.
- Existen técnicas de análisis predictivas dependiendo de la estrategia de análisis.
 - Descendente: LL(1),
 - Ascendente: Precedencia, LR, SLR, LALR
- **Herramientas para obtener el análisis sintáctico. YACC**

Construcción de Traductores: ANÁLISIS SEMÁNTICO (15)

- Desde el árbol sintáctico (secuencia de producciones aplicadas en el análisis) podemos identificar la secuencias de componentes léxicos con significado conjunto.
- Ya que partimos de una secuencia de **componentes léxicos** correcto sintácticamente, podemos analizar el **valor semántico estructurado** asociado con cada subsecuencia de componentes léxicos junto con su argumentos.

Construcción de Traductores: ANÁLISIS SEMÁNTICO (16)

- En el ejemplo anterior, tenemos dos símbolos no terminales de la gramática que identifica secuencias de entrada con significado conjunto:

$C \rightarrow \text{if } E \text{ then } S$ -----> $C \rightarrow \text{IF } E \text{ THEN } S$
 $A \rightarrow \text{id} := E$ -----> $A \rightarrow \text{ID ASIGN } E$

- Al aplicar la producción $A \rightarrow \text{ID ASIGN } E$, debemos comprobar que el lexema asociado con **ID** debe ser del mismo tipo que el de la expresión **E**. Esto nos facilita completar la información acerca de los lexemas. Nos permite completar la información de la tabla de símbolos.
- Verificaciones Semánticas.*

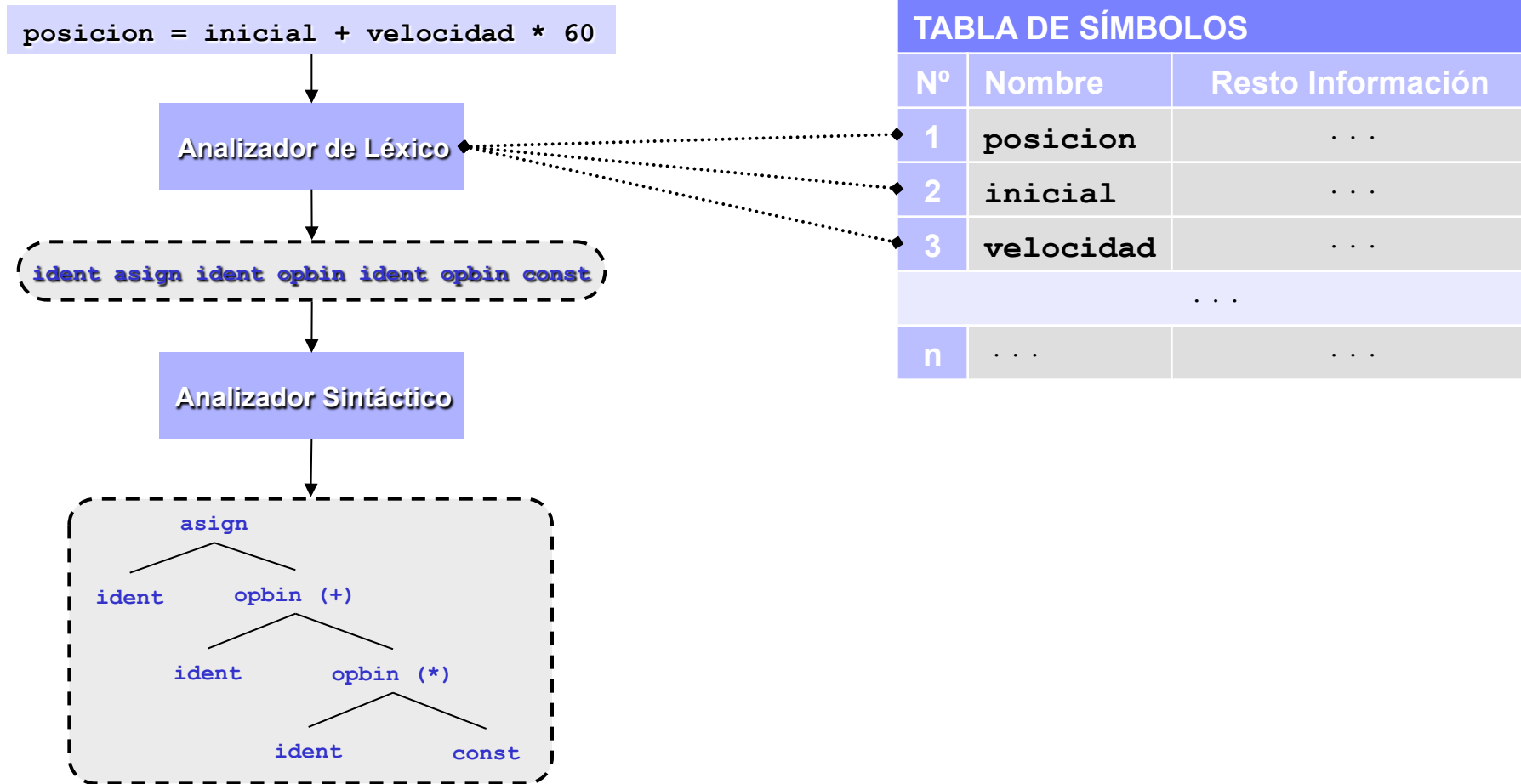
Construcción de Traductores: GENERACIÓN DE CÓDIGO (18)

- Consiste en aplicar de forma particularizada el **esquema de traducción** para cada **secuencia de lexemas** con sentido propio
- Un aspecto importante en los traductores de lenguajes de programación consiste en que deben generar códigos de una eficiencia comparable al generado directamente en lenguaje máquina
- Esta cualidad deseable de los traductores ha dado lugar al desarrollo de técnicas de **optimización de código**.

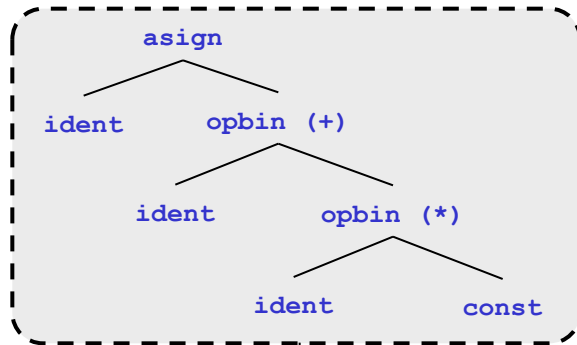
Construcción de Traductores: OPTIMIZACIÓN DE CÓDIGO (18)

- En toda secuencia de programa obtenido en lenguaje máquina u objeto, se caracteriza por la presencia de:
 - Secuencias de **órdenes de evaluación**.
 - **Órdenes de control**.

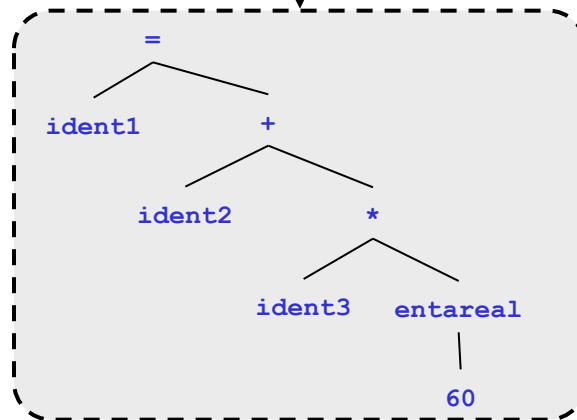
Ejemplo de Traducción (1)



Ejemplo de Traducción (2)



Analizador Semántico



Generador de
Código Intermedio

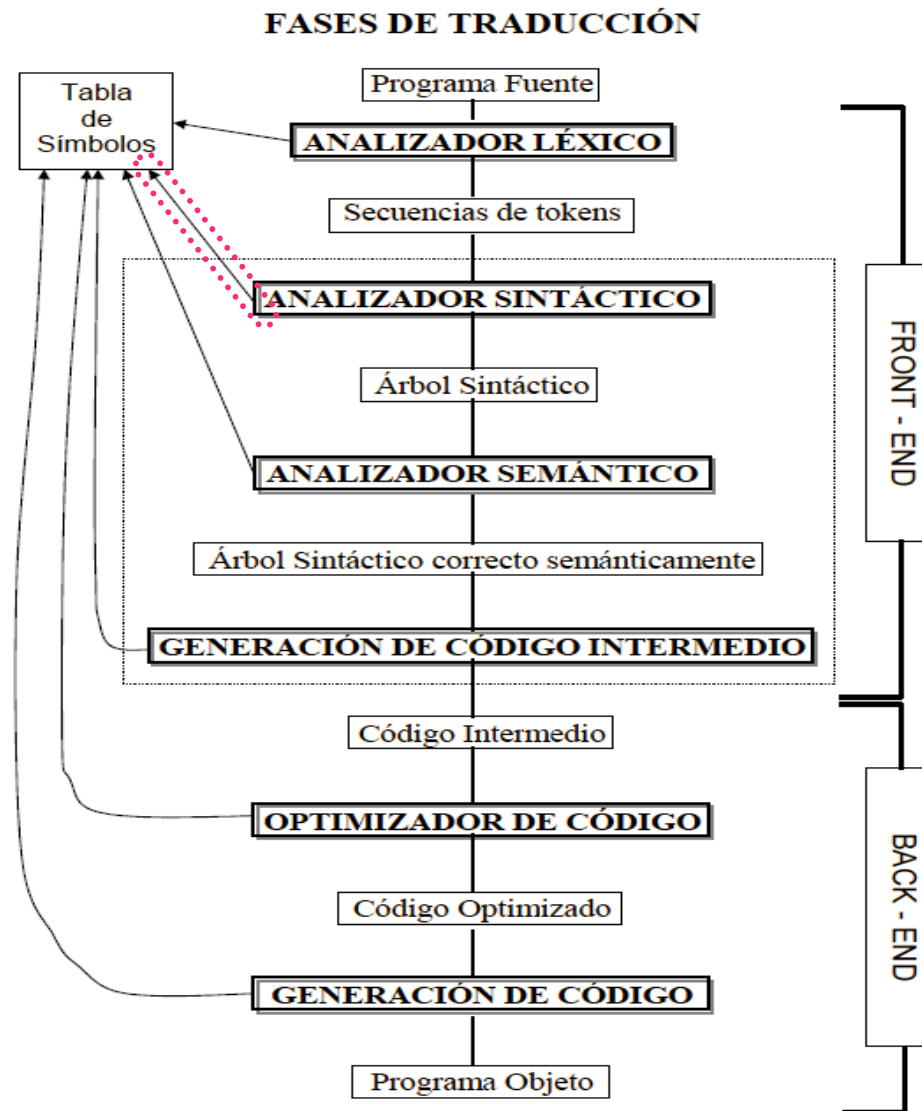
```
temp1 = entareal(60) ;
temp2 = ident3 * temp1 ;
temp3 = ident2 + temp2 ;
ident1 = temp3 ;
```

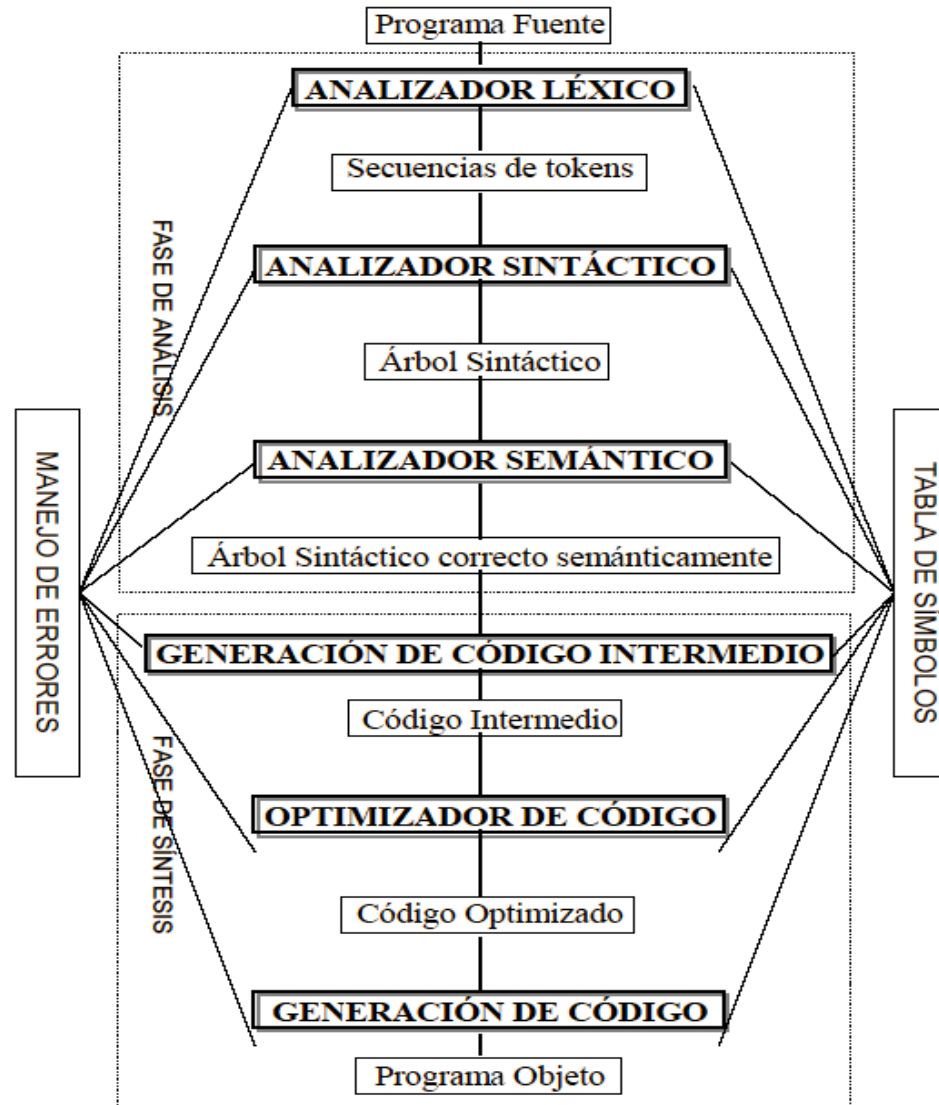
Optimizador de Código

```
temp1 = ident3 * 60.0 ;
Ident1 = ident2 + temp1 ;
```

Generador de Código

```
MOVF ident3, R2
MULF #60.0, R2
MOVF ident2, R1
ADDF R2, R1
MOVF R1, iden1
```



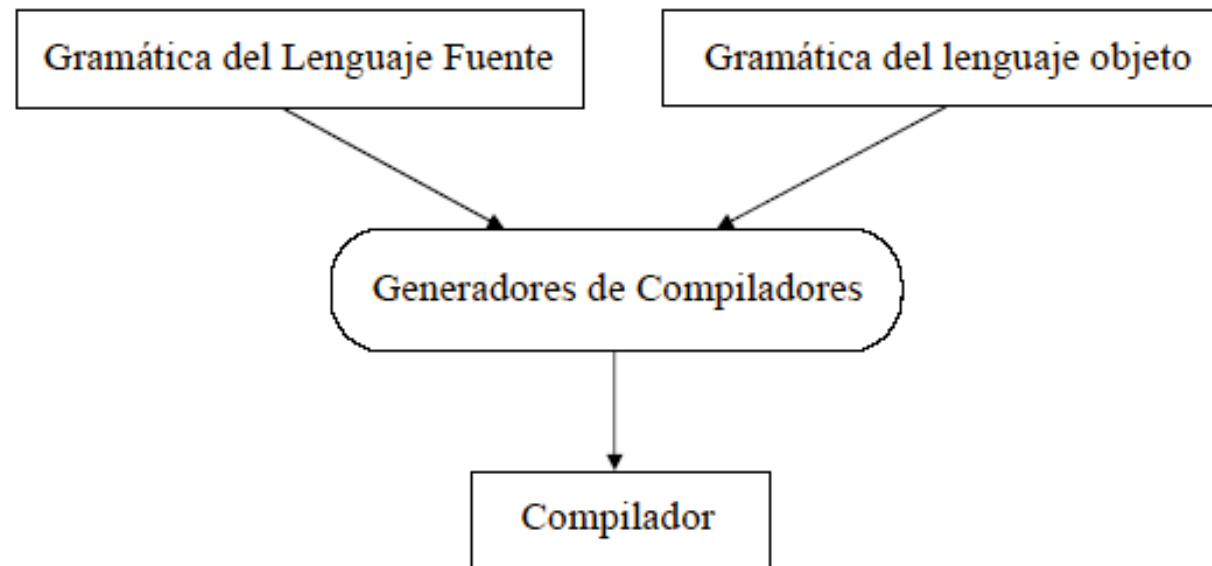


Evolución de los Traductores (1)

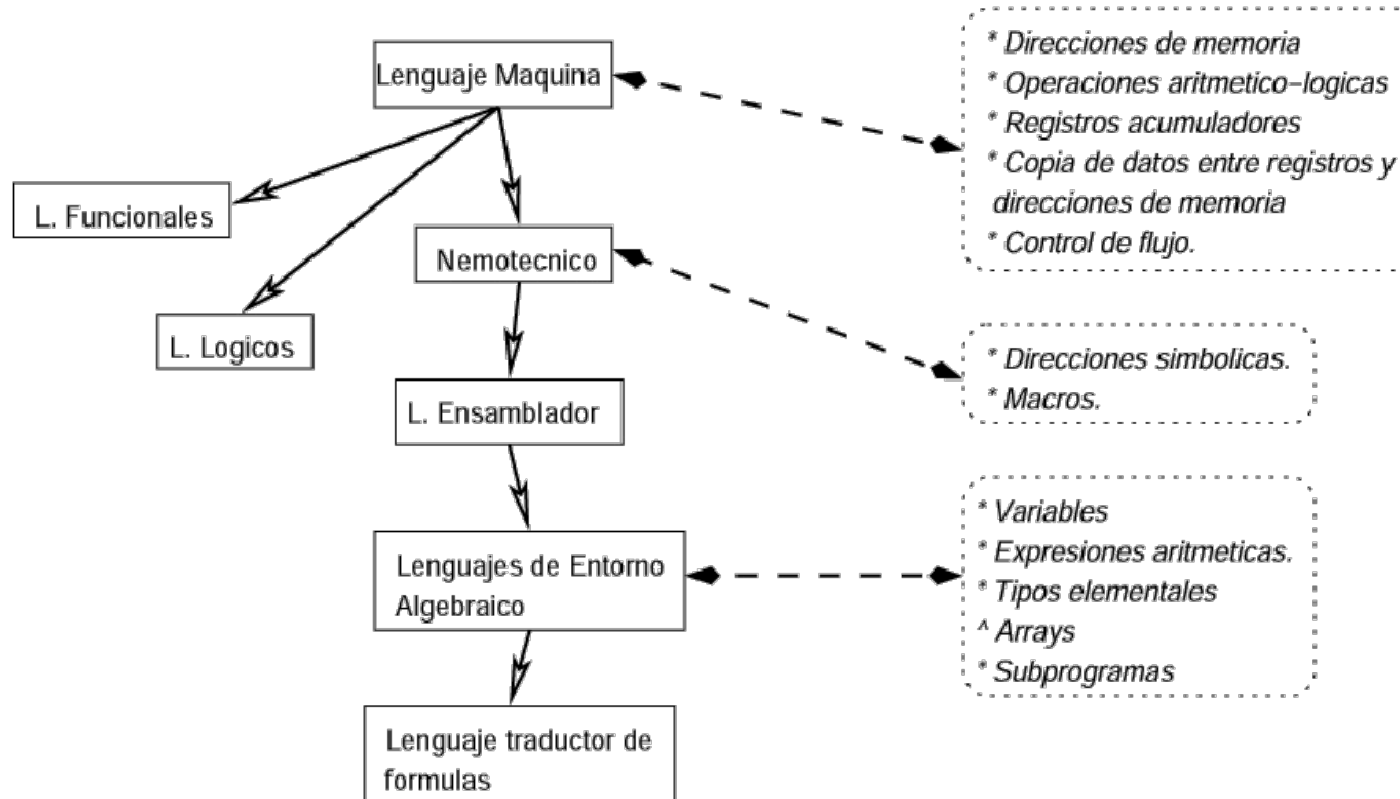
- **Década de los 50.**
 1. Examen del texto de entrada
 2. Construcción de la tabla de direcciones y símbolos
 3. Traducción de direcciones simbólicas a en direcciones de código.
 4. Generación de código.
- **Década de los 60.**
 1. Análisis de léxico.
 - Gramáticas regulares,
 - Gramáticas libre de contexto.
 2. Análisis Semántico
 3. Generación de código

Evolución de los Traductores (2)

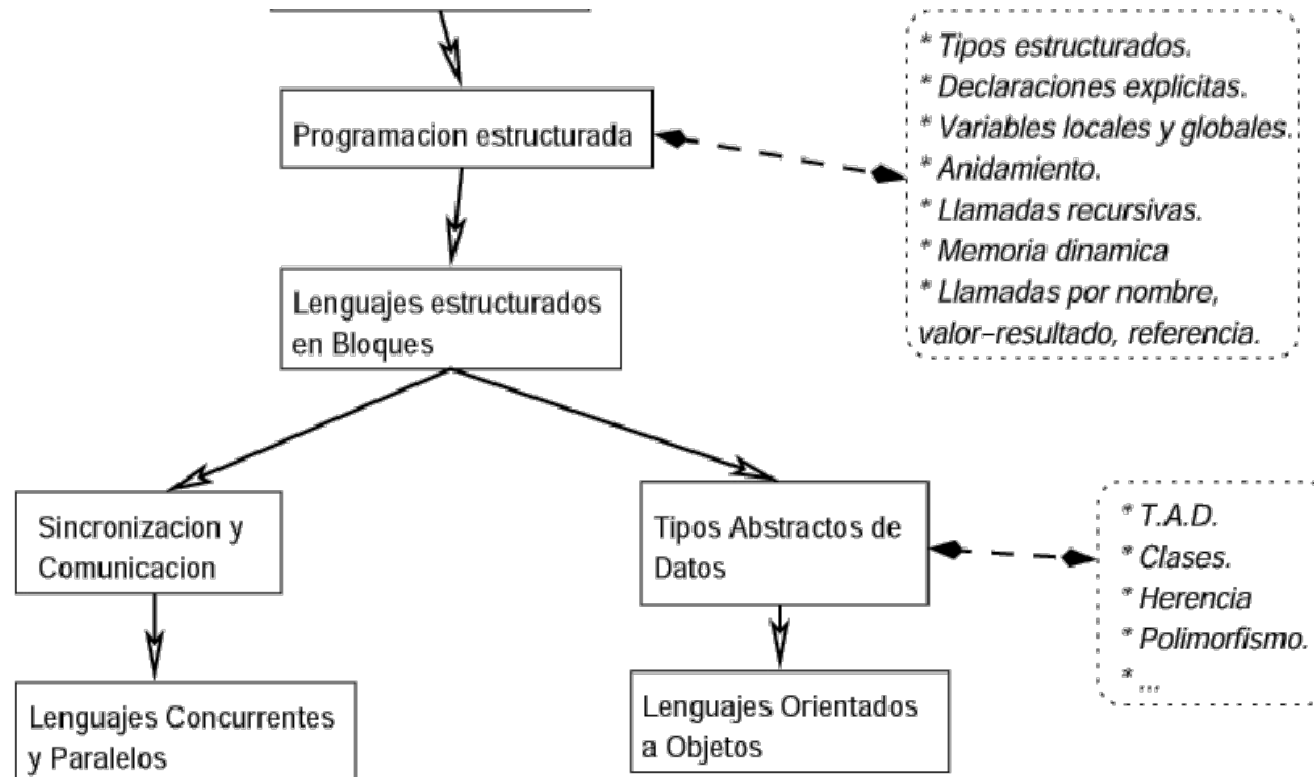
- **Generadores de Compiladores.** Traductores dirigidos por sintaxis.



Evolución de los Lenguajes de Programación (1)



Evolución de los Lenguajes de Programación (2)



Incidencia de los Conceptos de los Lenguajes de Programación en la Construcción de Traductores (1)

- **Lenguajes de entorno algebraico**, se caracteriza por el uso de:
 - **Variables**: obliga usar mapa de memoria estática.
 - **Transferencia de argumentos por referencia**: obliga a trabajar con la tabla de símbolos.

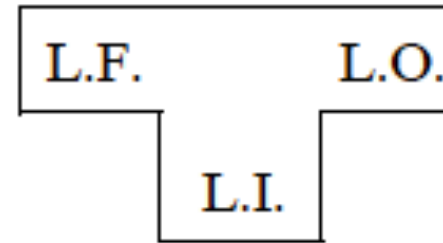
Incidencia de los Conceptos de los Lenguajes de Programación en la Construcción de Traductores (2)

- **Lenguajes con estructura de bloques:**
 - **Variables locales y globales:** usar tabla de símbolos por bloques.
 - **Dimensiones dinámicas:** Gestión dinámica de memoria.
 - **Subprogramas recursivos:** Gestión dinámica de memoria (dependiendo de tipo de enlace puede cambiar el resultado de la ejecución del programa).
 - **Transferencia de argumentos,**
 - **Anidamiento de instrucciones,**
 - **Lenguajes orientados a objetos.**

Arquitectura de Procesadores de Lenguajes (1)

- **Bootstrapping:** Truco o trampa de arranque. Facilidad que ofrece un lenguaje para poder compilarse a sí mismo.

Diagrama T:



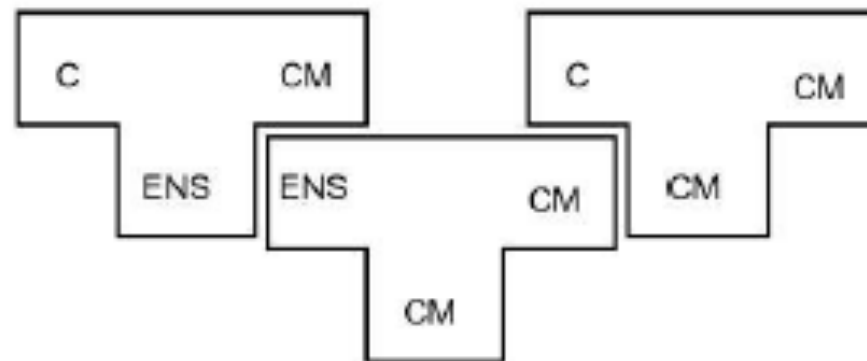
L.F.: Lenguaje fuente.

L.O.: Lenguaje objeto.

L.I. : Lenguaje de Implantación

Arquitectura de Procesadores de Lenguajes (2)

- Compilador escrito en ensamblador, debe ser ensamblado.



Arquitectura de Procesadores de Lenguajes (3)

- Bootstrapping:** traducir un lenguaje complejo usando un lenguaje simple

