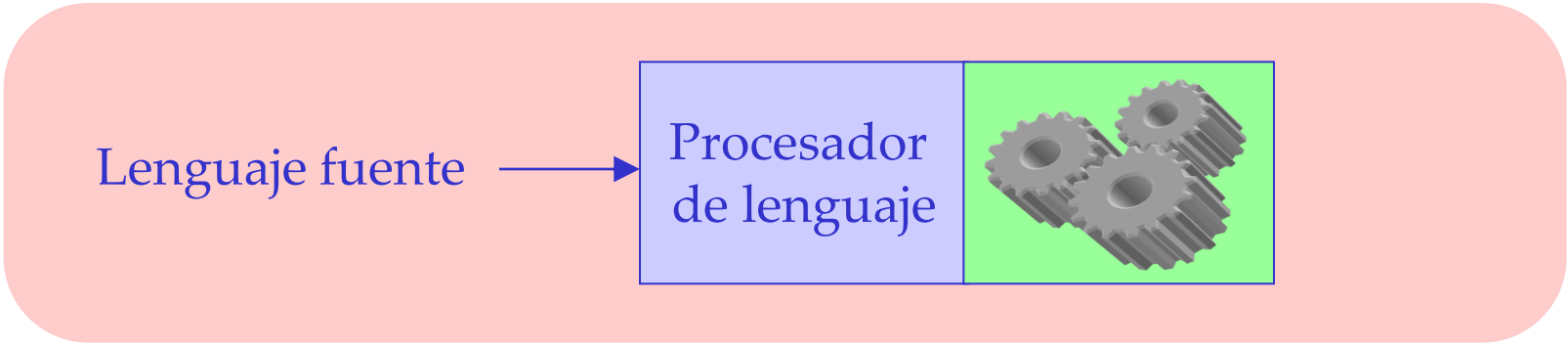
Universidad de Huelva

Tema 1

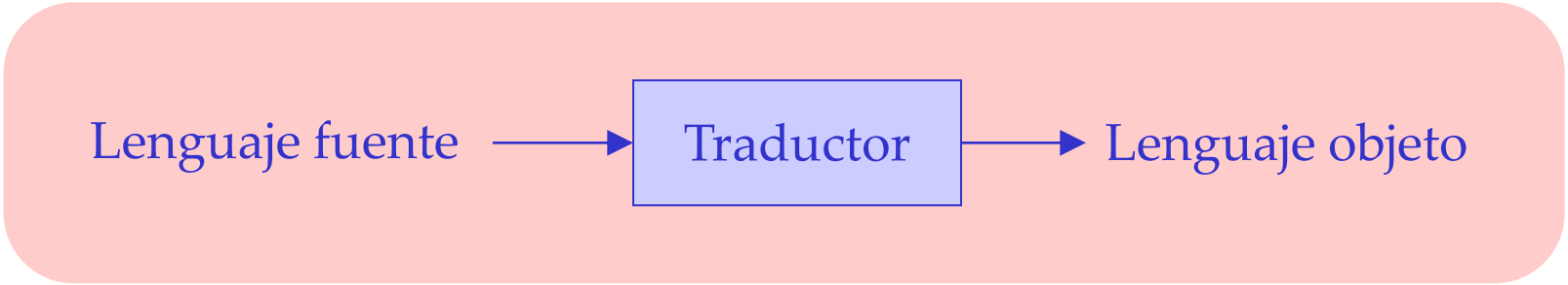
Introducción

Índice

Índice



- Procesador de lenguaje: aplicación que toma como entrada un conjunto de archivos descritos en un cierto lenguaje formal (lenguaje fuente)
- Traductor: procesador de lenguaje cuya función consiste en re-escribir los archivos de entrada en otro lenguaje formal (lenguaje objeto)



- Ensamblador: traductor de lenguaje ensamblador a código máquina
- Compilador: traductor de lenguaje de alto nivel a ensamblador o a código máquina

- Intérprete: Analiza el código fuente y lo ejecuta sin generar un código objeto.
- Emulador: Intérprete en el que el código fuente es el código máquina de una plataforma diferente.
- Compiladores vs intérpretes:
  - Los programas compilados son mucho más rápidos
  - Los compiladores consumen muchos recursos
  - Los intérpretes utilizan código independiente de la plataforma
  - Los intérpretes pueden ejecutar código generado en tiempo de ejecución.

- Preprocesador: pequeño compilador dedicado a expandir las macros e incluir ficheros.
- Enlazador (linker): programa que enlaza los diferentes módulos compilados para generar el ejecutable.
- Compilador cruzado: compilador que genera código para una máquina diferente a la utilizada para compilar.
- Autocompilador: compilador escrito en el mismo lenguaje que va a compilar.
- Metacompilador: herramienta de ayuda al diseño de compiladores.
- Descompilador: programa que produce una representación de alto nivel a partir del código máquina.

- 1940's:
  - Primeros ordenadores.
  - Programación directa en código máquina.
  - Nace el ensamblador (traducido manualmente)
  - Se automatiza la etapa de ensamblado
- 1950's
  - (1950) John Backus dirige en IBM un proyecto de lenguaje algebraico
  - (1954-1958) Nace el FORTRAN (FORmulae TRANslator)
  - Primer compilador de FORTRAN para IBM modelo 704

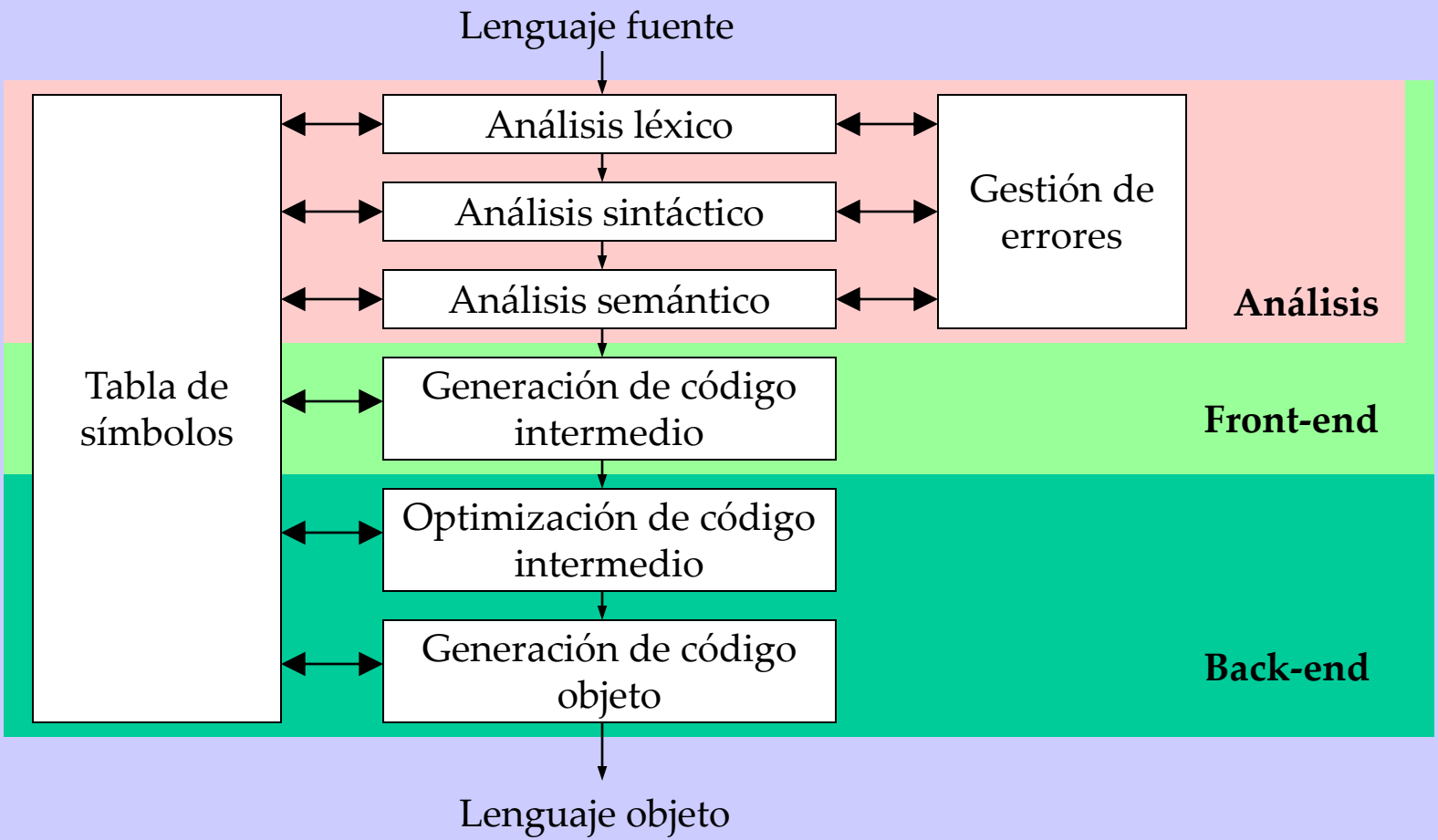
- 1950's (mediados):
  - Noam Chomsky publica sus estudios sobre la estructura de los lenguajes y las gramáticas formales
- 1950's (finales):
  - F.L. Bauer (Univ. Munich) dirige un proyecto de lenguaje formal
  - Se crea un comité conjunto con la Association for Computing Machinery en el que participa Backus
  - Se define IAL (International Algebraic Language)
  - Posteriormente se denomina ALGOL 58 (ALGOrithmic Language)

- Características de ALGOL 58
  - Definición de la notación BNF (Backus-Naur Form)
  - Declaración de tipos de variables
  - Estructura modular
  - Funciones recursivas
  - Paso de parámetros por valor o por nombre
  - Precursor de Pascal, Ada o Modula

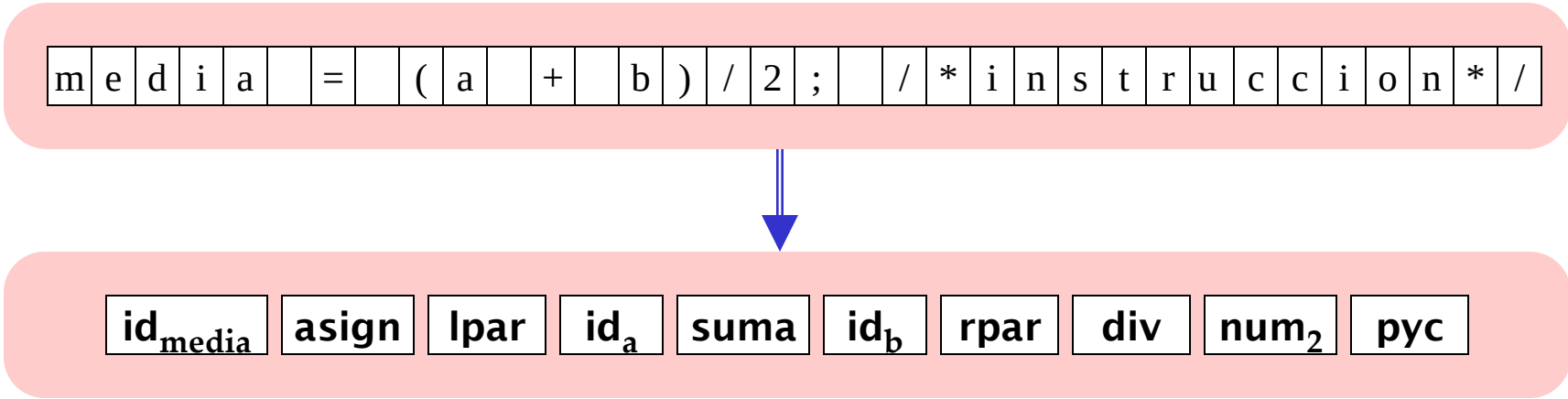
- 1960's: Primeras técnicas de desarrollo de compiladores
  - 1958: Strong y otros proponen dividir el compilador en dos fases: front-end y back-end y promueven la creación de un lenguaje intermedio universal (UNCOL – UNiversal Computer Oriented Language).
  - 1959: Rabin y Scott proponen utilizar autómatas deterministas para el análisis léxico
  - 1961: Primer analizador sintáctico descendente recursivo
  - Durante toda la década se estudian intensamente las gramáticas LL.

- 1970's:
  - Se estudian las gramáticas LR
  - Se definen los métodos de análisis ascendente SLR y LALR
  - Se crean numerosas herramientas de ayuda al diseño de compiladores
  - A mediados de la década aparecen las herramientas *lex* y *yacc* que se convierten en estándares de facto al distribuirse con UNIX.
  - La Free Software Fundation distribuye estas herramientas bajo los nombres de *flex* y *bison*

- Generación de código:
  - Inicialmente la gestión de la memoria era estática
  - Cuando aparecen las funciones recursivas se desarrolla la pila como forma de gestionar la memoria
  - La gestión de memoria dinámica conduce a la definición del montículo (heap) como zona de gestión de esta memoria
  - Surgen nuevos problemas como las referencias perdidas o la recogida de basura
- Últimos avances:
  - Optimización de código
  - Máquinas virtuales y auge de los lenguajes interpretados

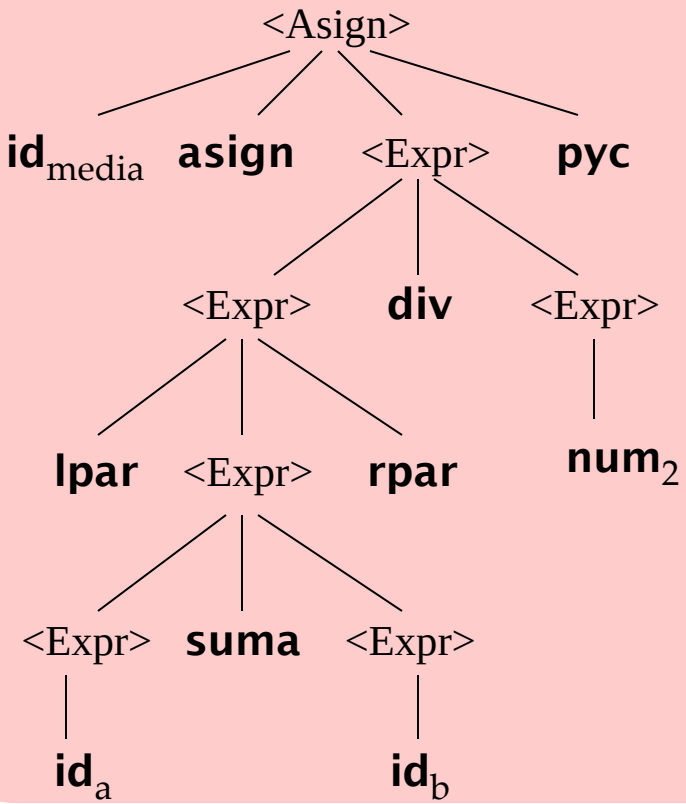

Lenguaje fuente
Análisis léxico
Análisis sintáctico
Análisis semántico
Gestión de errores
Análisis
Tabla de símbolos
Generación de código intermedio
Front-end
Optimización de código intermedio
Generación de código objeto
Back-end
Lenguaje objeto

- Objetivo:
  - Leer caracteres e identificar componentes léxicos (*tokens*)
  - Filtrar comentarios
  - Detectar errores léxicos
- Ejemplo:

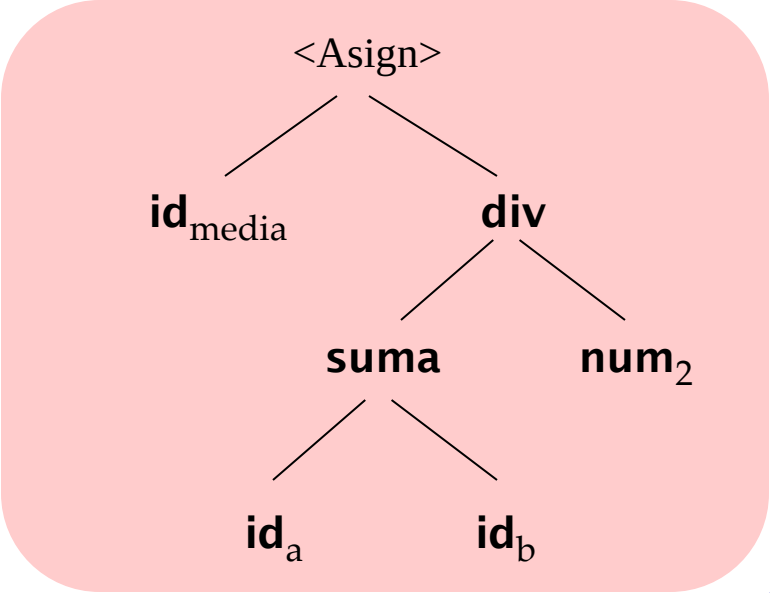


- Objetivo:
  - Verificar estructuras gramaticales
  - Detectar errores de sintaxis
- Especificación:
  - Gramáticas libres de contexto
- Ejemplo:

<Asign> → **id asign** <Expr> **pyc**
<Expr> → <Expr> **suma** <Expr>
<Expr> → <Expr> **div** <Expr>
<Expr> → **lpar** <Expr> **rpar**
<Expr> → **id**
<Expr> → **num**



- Objetivo:
  - Verificar restricciones semánticas (predefinición de variables, consistencia de tipos, llamadas a funciones)
  - Generar un Árbol de Sintaxis Abstracta
- Especificación:
  - Gramáticas atribuidas
- Ejemplo:



- Objetivo:
  - Generar una descripción en código de bajo nivel independiente de la plataforma
- Especificación:
  - Código de tres direcciones (destino, operando1, operando2, operación)
- Ejemplo:

```
T1 := a
T2 := b
T3 := T1 + T2
T4 := 2
T5 := T3 / T4
media := T5
```

1.3.5 Optimización de código intermedio

- Objetivo:
  - Reducir el número de operaciones para mejorar el rendimiento
- Técnicas:
  - Eliminar saltos consecutivos
  - Reutilizar valores intermedios
  - Extraer instrucciones de los bucles
  - Expandir los bucles
  - Etc.

- Objetivo:
  - Obtener la representación en ensamblador para la máquina de destino
  - Optimizar el código
- Técnicas.
  - Almacenar las variables en registros
  - Utilizar instrucciones específicas
  - Etc.

- Objetivo:
  - Almacenar toda la información de los diferentes símbolos (variables, constantes, tipos, funciones)
- Características:
  - Cada etapa requiere una información diferente
  - Comportamiento dinámico (variables definidas en módulos)
  - Optimizar el tiempo de acceso (tablas hash)

- Objetivo:
  - Localizar errores
  - Informar de los detalles del error
  - Recuperación del error (tokens de sincronismo)

- Alfabeto (Σ): conjunto finito de símbolos
- Cadena: secuencia finita de símbolos
- Cadena vacía (λ): cadena de longitud cero
- Σ^k: Conjunto de cadenas de longitud k
- Clausura del alfabeto (Σ^*): Conjunto de todas las cadenas

$$\Sigma^* = \bigcup_{k=0}^{\infty} \Sigma^k = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$$

- Lenguaje formal: Subconjunto de Σ^*

- Gramáticas formales:
  - Forma de describir lenguajes formales
  - Cuatro elementos (N, Σ, P, <S>):
  - N: Alfabeto de símbolos no terminales
  - Σ : Alfabeto de símbolos terminales
  - P: Conjunto de producciones o reglas

$$P \subset (N \cup \Sigma)^* N (N \cup \Sigma)^* \rightarrow (N \cup \Sigma)^*$$

  - <S>: Símbolo inicial (no terminal)

- Derivación directa: η deriva directamente en γ ($\eta \Rightarrow \gamma$) si
  - $\eta = \omega_1 \alpha \omega_2$
  - $\gamma = \omega_1 \beta \omega_2$
  - $\alpha \rightarrow \beta \in P$
- Derivación: η deriva en γ ($\eta \overset{*}{\Rightarrow} \gamma$)si
  - existen $\xi_1,\ldots, \xi_n$ tales que η deriva directamente en ξ_1, ξ_i deriva directamente ξ_{i+1} y ξ_n deriva directamente en γ
- Forma sentencial: cadena $\alpha \in (N \cup \Sigma)^*$ tal que $<S> \overset{*}{\Rightarrow} \alpha$
- Sentencia: forma sentencial perteneciente a Σ^*
- Lenguaje generado por la gramática G:
  - $L(G) = \{ x \in \Sigma^* \mid <S> \overset{*}{\Rightarrow} x \}$

- Ejemplo:
  - G = ({<S>, <E>}, { id,+,* }, P, <S> }, con las siguientes reglas:
    - <S> → <E>
    - <E> → <E> + <E>
    - <E> → <E> * <E>
    - <E> → id
- Ejemplo de derivación:
    - <S> ⇨ <E>
    - <S> ⇨ <E> + <E>
    - <S> ⇨ <E> + <E> * <E>
    - <S> ⇨ id + <E> * <E>
    - <S> ⇨ id + id * <E>
    - <S> ⇨ id + id * id

- La jerarquía de Chomsky:
  - Gramáticas no restringidas: $\alpha \rightarrow \beta$
    - $\alpha \in (N \cup \Sigma)^* N (N \cup \Sigma)^*$
    - $\beta \in (N \cup \Sigma)^*$
  - Gramáticas sensibles al contexto: $\alpha<A>\beta \rightarrow \alpha \gamma \beta$
    - $\alpha, \beta, \gamma \in (N \cup \Sigma)^*$
    - $<A> \in N$
  - Gramáticas independientes de contexto: $<A> \rightarrow \alpha$
    - $<A> \in N$
    - $\alpha \in (N \cup \Sigma)^*$
  - Gramáticas regulares: $<A> \rightarrow a <B>$ ó $<A> \rightarrow a$
    - $<A>, <B> \in N$
    - $a \in \Sigma$

- Dispositivos reconocedores de lenguajes:

Tipo de gramática	Dispositivo
Regular	Autómata de estados finitos
Libre de contexto	Autómata de pila
Sensible al contexto	Autómata de memoria limitada
No restringida	Máquina de Turing