Procesadores de Lenguaje

Tema 1.1. Un lenguaje mínimo y su procesador: *Introducción*

Profesor
Federico Peinado

Elaboración del material
José Luis Sierra
Federico Peinado

Ingeniería en Informática
Facultad de Informática – Universidad Complutense de Madrid
Curso 2009-2010

¿Qué son los Procesadores de Lenguaje?

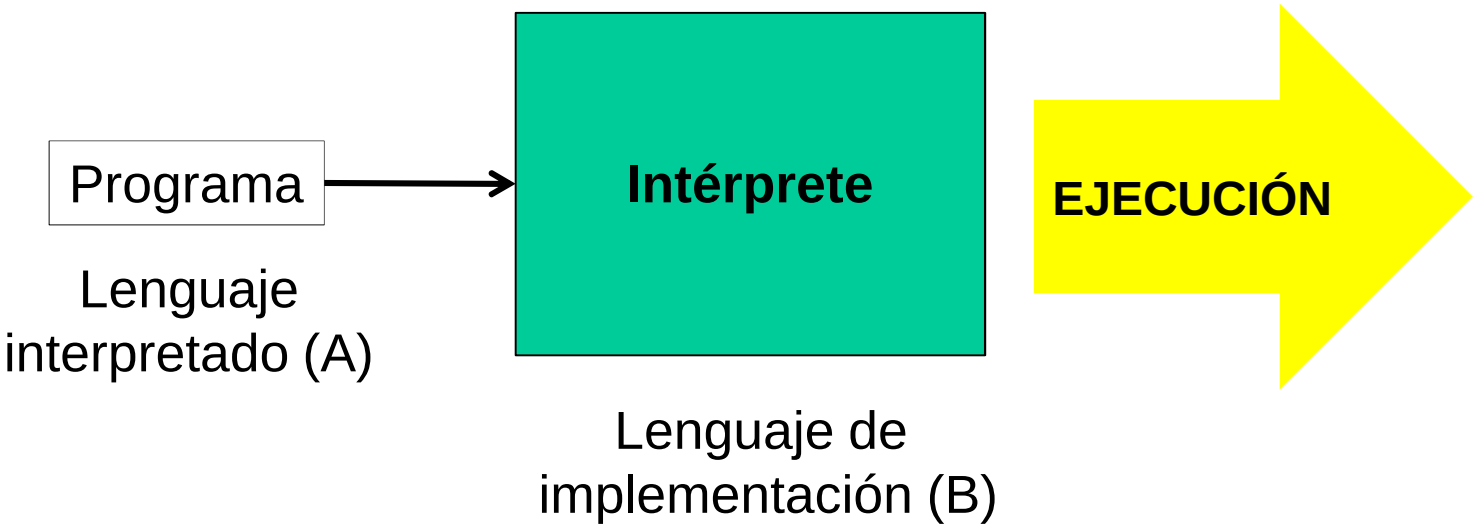
¡Metaprogramación!

- **Programas informáticos** que tienen como datos de entrada y de salida ficheros (documentos) escritos en uno o varios lenguajes
  - Lenguajes *formales* normalmente (no lenguajes *naturales* como el chino, el español, etc.)
  - Lenguajes *de programación* habitualmente (son **programas que trabajan sobre otros programas** → ¡tema complicado pero atractivo e imprescindible para los informáticos!)
- Existen muchos **tipos de procesadores de lenguaje**
  - Intérpretes
    - De máquina a pila (* Como el de la práctica anual)
    - …
  - Traductores
    - Ensambladores
    - Compiladores (* Como el de la práctica anual)
    - …
  - …

Intérpretes

- **Intérprete de un lenguaje X:** Programa que, tomando como entrada un programa en lenguaje X, emula su ejecución
  - La ejecución se produce al mismo tiempo que se procesa la entrada, de forma intercalada
  - El intérprete se comporta como una **máquina** (computadora), sólo que **virtual**

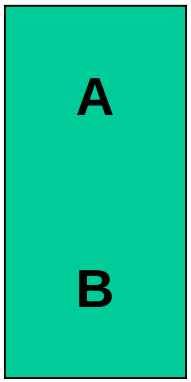
Un intérprete involucra a 2 lenguajes



- A y B pueden ser distintos lenguajes… o ser el mismo
- Ejemplo: A= *Python* y B= *C++*

Diagrama en bloque de un intérprete

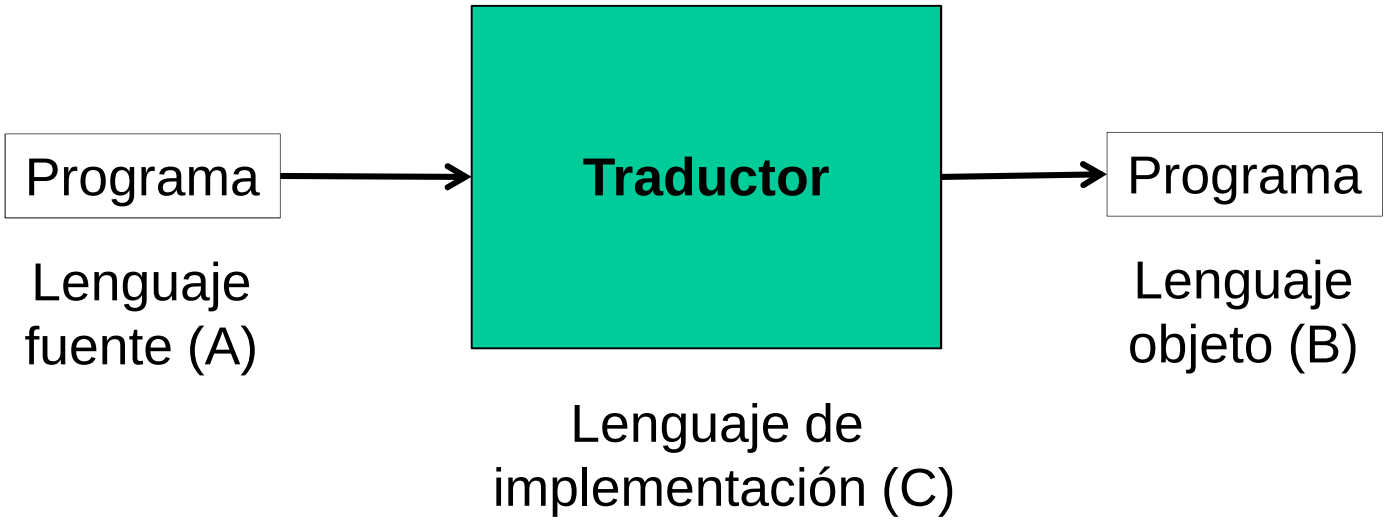
- Notación para representar a un intérprete (con sus 2 lenguajes involucrados):



Traductores

- **Traductor de A a B:** Programa que, tomando como entrada un programa en el lenguaje A produce como salida un programa en el lenguaje B
  - La traducción normalmente se realiza *frase a frase*
  - La ejecución del programa B será posible *una vez haya acabado por completo el proceso de traducción*
- Como decíamos, existen varios **tipos de traductores**
  - **Ensamblador:** Traductor de un lenguaje ensamblador a un lenguaje máquina (binario)
  - **Compilador:** Traductor de un lenguaje de alto nivel (como C) a otro de bajo nivel (como un ensamblador)
  - …

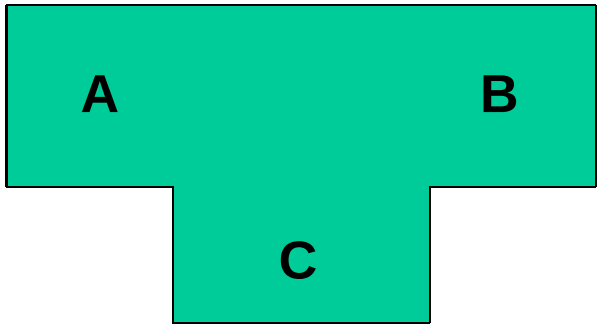
Un traductor involucra a 3 lenguajes



- A, B y C pueden ser distintos lenguajes… o los mismos
- Ejemplo: A= *Java*, B= *Bytecode* y C = *C++*

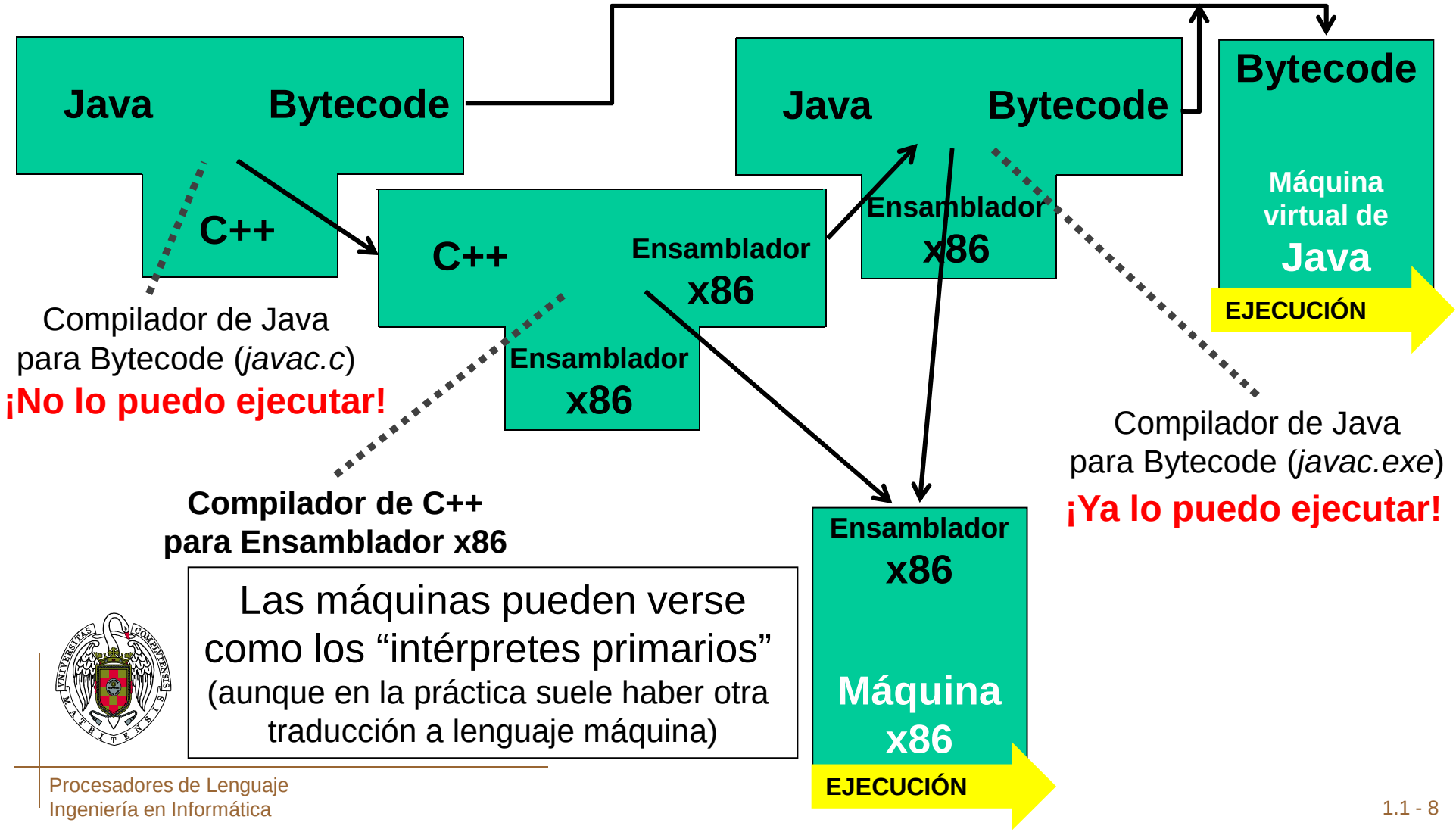
Diagrama en T de un traductor

- Notación para representar a un traductor (con sus 3 lenguajes involucrados):



Conexión entre traductores

- Gracias a la conexión entre el trabajo que realizan varios traductores es posible la Informática actual:



Las máquinas pueden verse como los “intérpretes primarios”
(aunque en la práctica suele haber otra traducción a lenguaje máquina)

Más utilidades de la conexión entre traductores

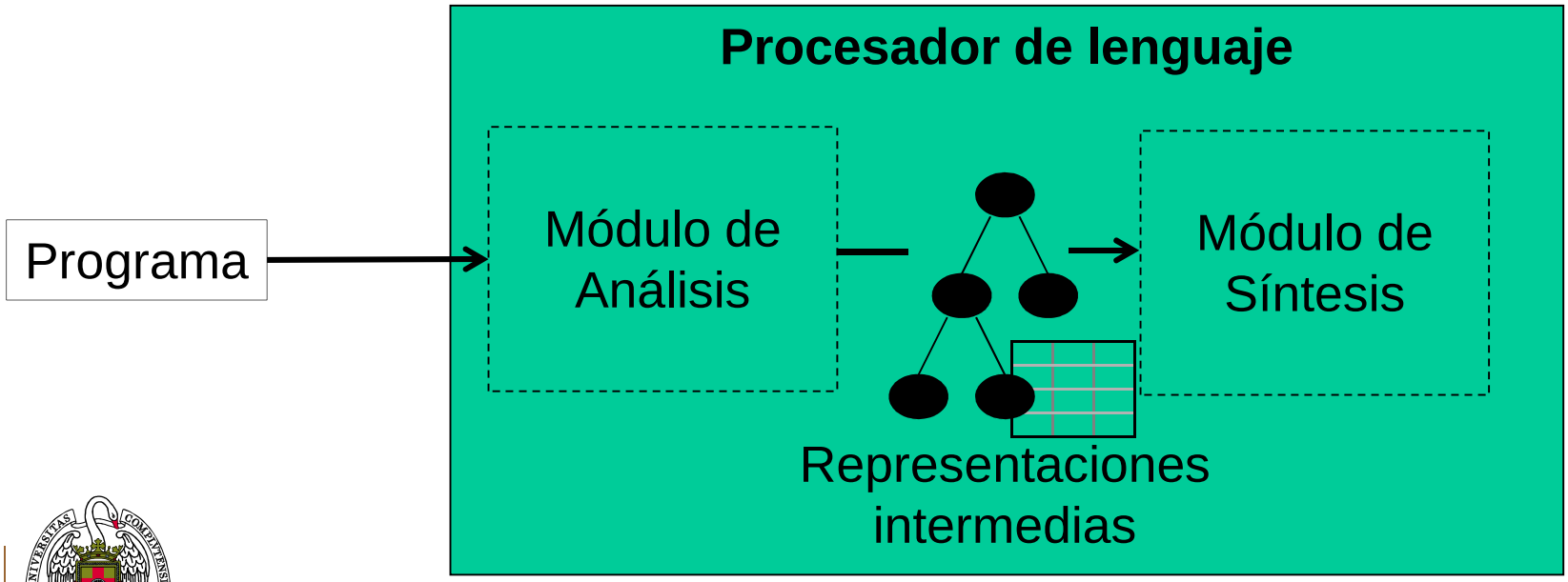
- **Traductor de A a A:** Tiene sentido para mejorar la eficiencia de un programa, *ofuscar* el código, etc.
- ***Bootstrapping:*** Conexión entre traductores que permite ir construyendo cada vez lenguajes de más alto nivel usando el lenguajes de bajo nivel disponible
  - **Traductor de A a B (implementado en B):** Tiene sentido cuando B es el lenguaje de bajo nivel disponible y necesitamos un compilador para un lenguaje A de alto nivel que ha sido creado
  - **Traductor de A' a B (implementado en A):** Tiene sentido cuando B es el lenguaje de bajo nivel disponible y necesitamos un compilador para un lenguaje A', siendo este una *ampliación* del más alto nivel que el lenguaje A para el que ya tenemos un compilador
    - Al repetir esta operación (para A'', A'''…), conseguimos tener disponible cada vez un lenguaje de más alto nivel

El caso particular de los compiladores

- Cuando se está usando un compilador, también se suelen usar otros procesadores de lenguaje auxiliares
  - **Antes** de ejecutar un compilador:
    - **Entornos Integrados de Desarrollo** con editores “inteligentes” que procesan fragmentos del programa según los vamos escribiendo y nos avisan de posibles errores, nos ayudan a rellenar huecos en el código, etc.
    - **Preprocesadores** que procesan el programa justo antes de compilarlo para quitar los comentarios, expandir las macros, atender las directivas de compilación, etc.
  - **Después** de ejecutar un compilador:
    - **Ensambladores** que, como dijimos, traducen el resultado del compilador (normalmente un programa en un lenguaje ensamblador) al lenguaje máquina que sea necesario
    - **Enlazadores** que, cuando el programa ha sido compilado en varios ficheros, los junta en uno solo; a veces incluso cargando ficheros *dinámicamente* = en ejecución (**Cargador-enlazador**)

El modelo de análisis-síntesis

- Es el *planteamiento típico* que se suele tomar para diseñar **la estructura de un procesador de lenguaje**
  - Un módulo de **análisis**
  - Una o varias **representaciones intermedias** (estructuras de datos como árboles o tablas…)
  - Un módulo de **síntesis**



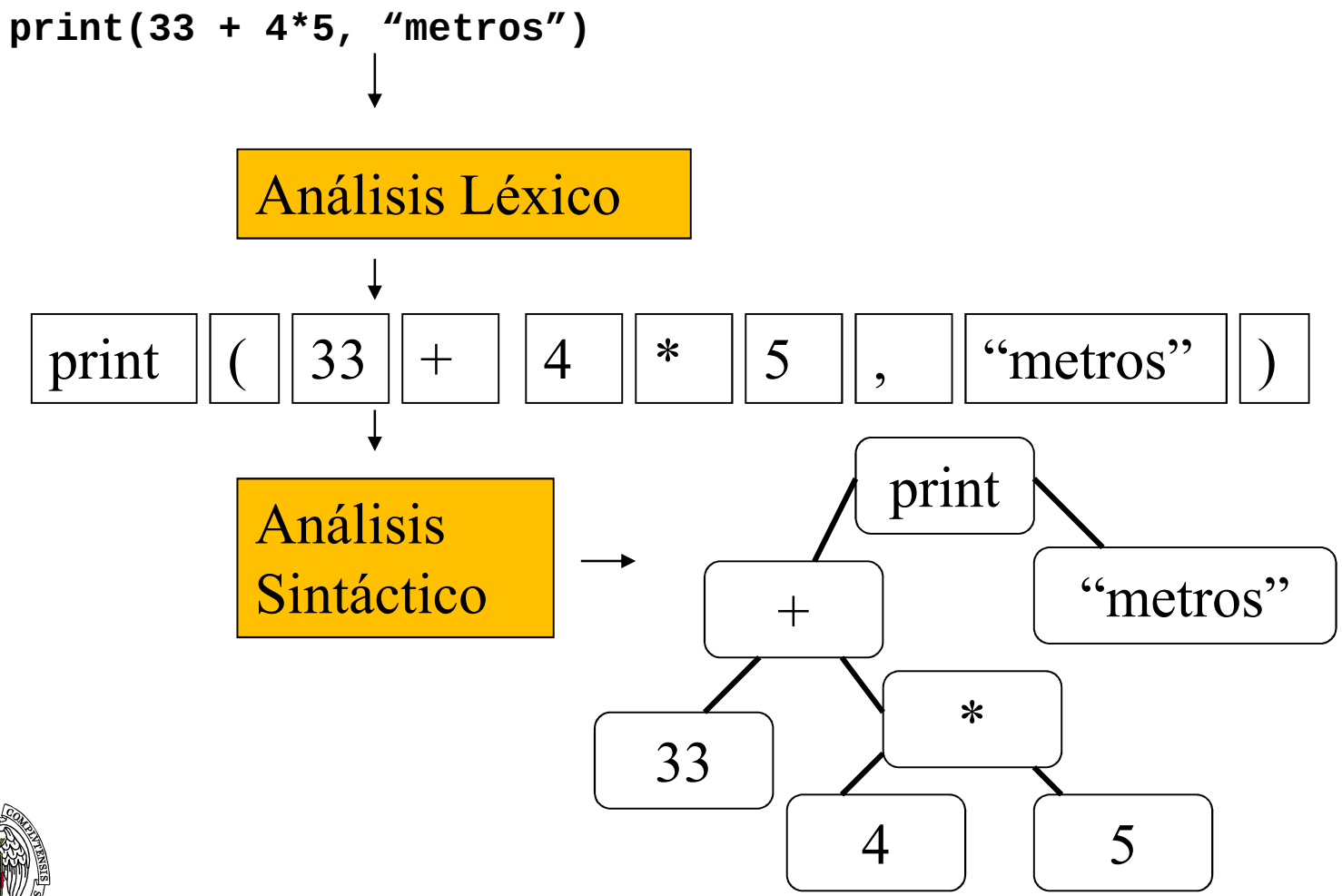
El módulo de análisis

1. Recibe el programa a procesar
2. Analiza sus elementos, su forma y su significado
3. Genera representaciones intermedias que describen:
   - Los símbolos que se utilizan en el programa
   - La estructura que tiene el programa
   - Los errores detectados en el programa (si los hay)
   - …

Descomposición del análisis en pasos

1. **Análisis léxico**
   - Recibe el programa a procesar
   - Comprueba su *validez léxica*
   - Genera la secuencia de **componentes léxicos** o *tokens* (= ocurrencias de **categorías léxicas)** que hay en el programa
2. **Análisis sintáctico**
   - Recibe la secuencia de *componentes léxicos*
   - Comprueba su validez estructural
     - Puede tener en cuenta **restricciones contextuales**, *como el hecho de que para poder usar un identificador hay que haberlo declararlo previamente*
   - Genera las representaciones intermedias correspondientes

Ejemplo de análisis (Simplificado)



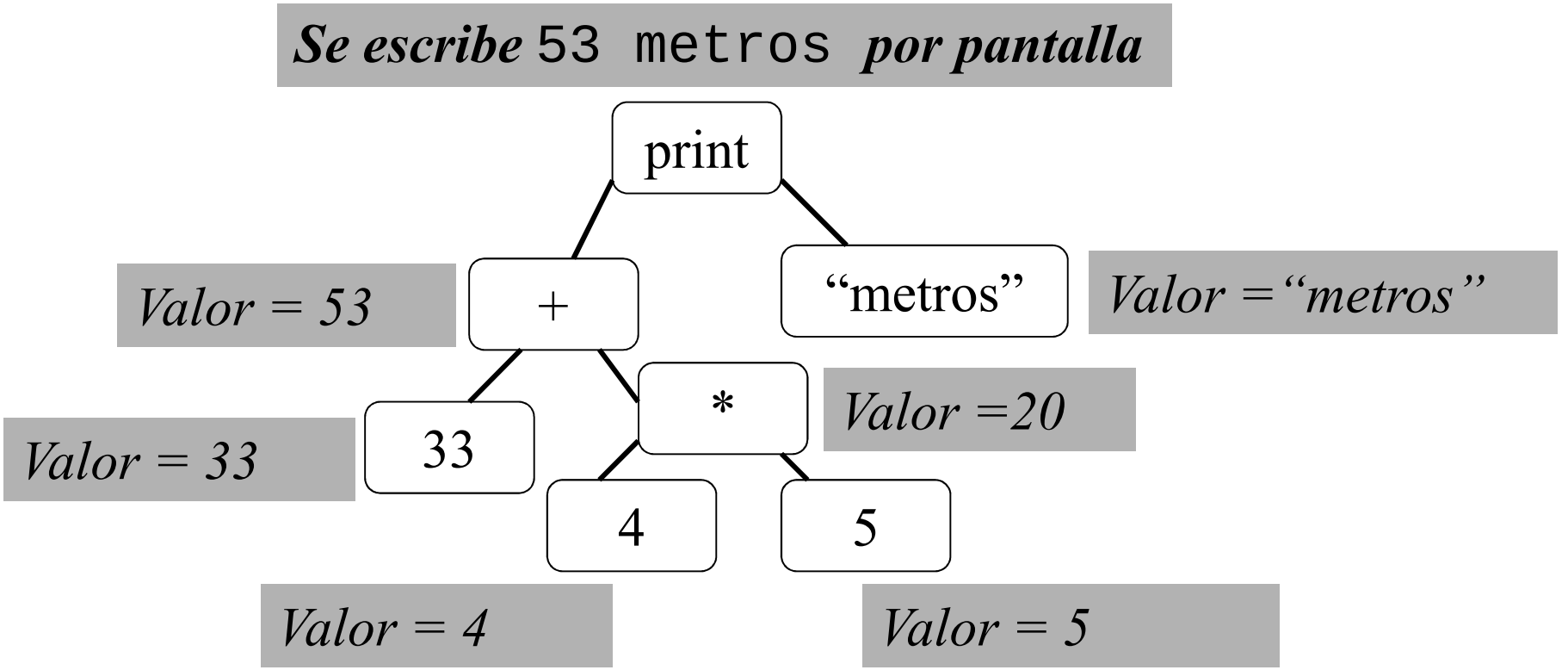
El módulo de síntesis

1. Recibe las representaciones intermedias
2. Cada *porción de información* que se encuentra en las representaciones intermedias se sintetiza en forma de:
   - Ejecución de una instrucción
     (en el caso de los intérpretes)
   - Generación de una frase del programa objeto
     (en el caso de los traductores)
3. Se repite el paso 2 hasta que se acaba la ejecución o se termina de generar el programa objeto

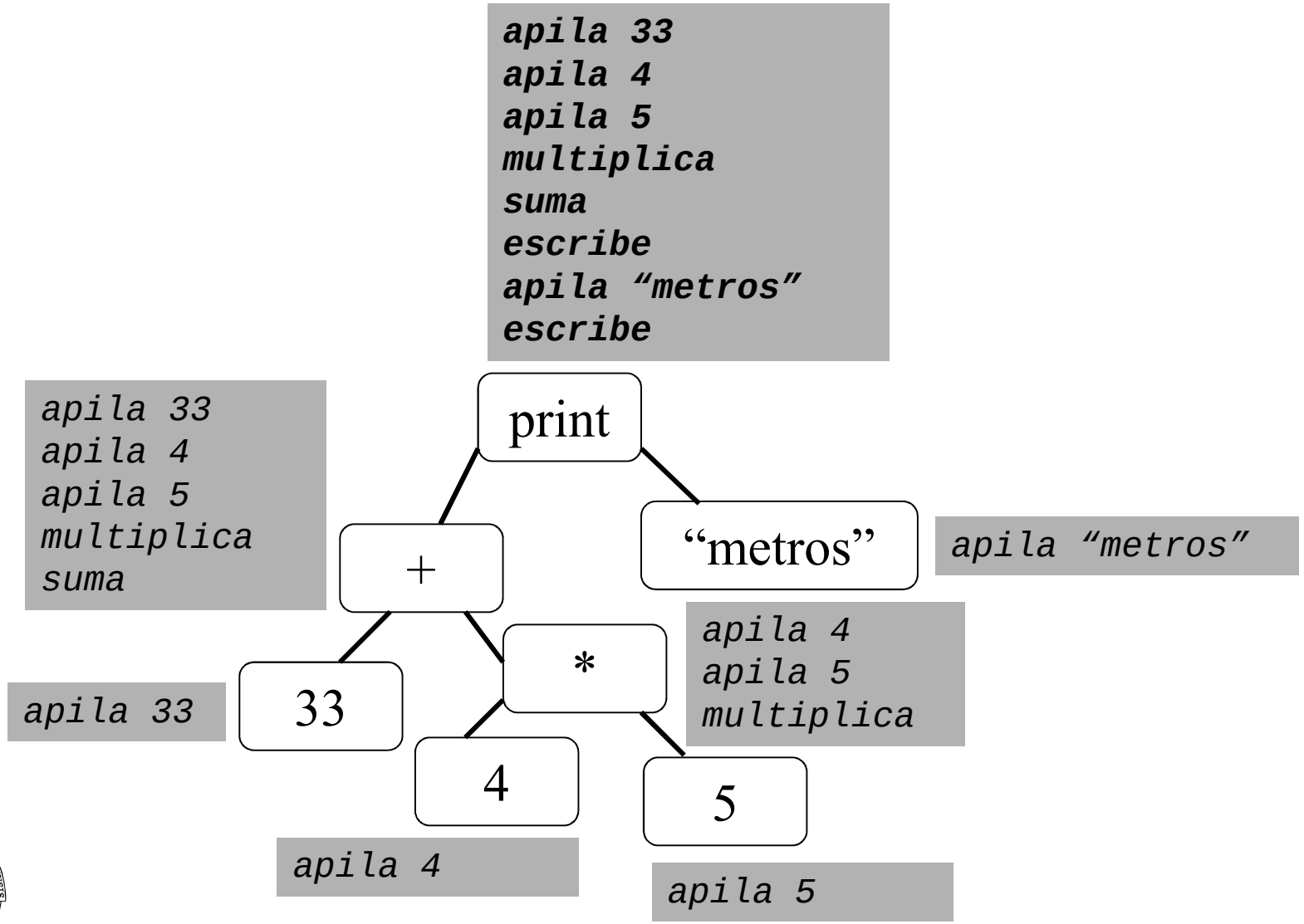
Descomposición de la síntesis en pasos

- **Traducción Dirigida por la Sintaxis** es el modelo habitual para hacer la síntesis *(se llama así, ya hablemos de traductores o de intérpretes)*
  1. Recibe las representaciones intermedias correspondientes
  2. Se realiza uno o varios recorridos de la estructura de datos (*árbol*, normalmente) que representa la estructura del programa
  3. Se calculan valores y/o realizan acciones en cada punto del recorrido (*nodos del árbol*, normalmente)

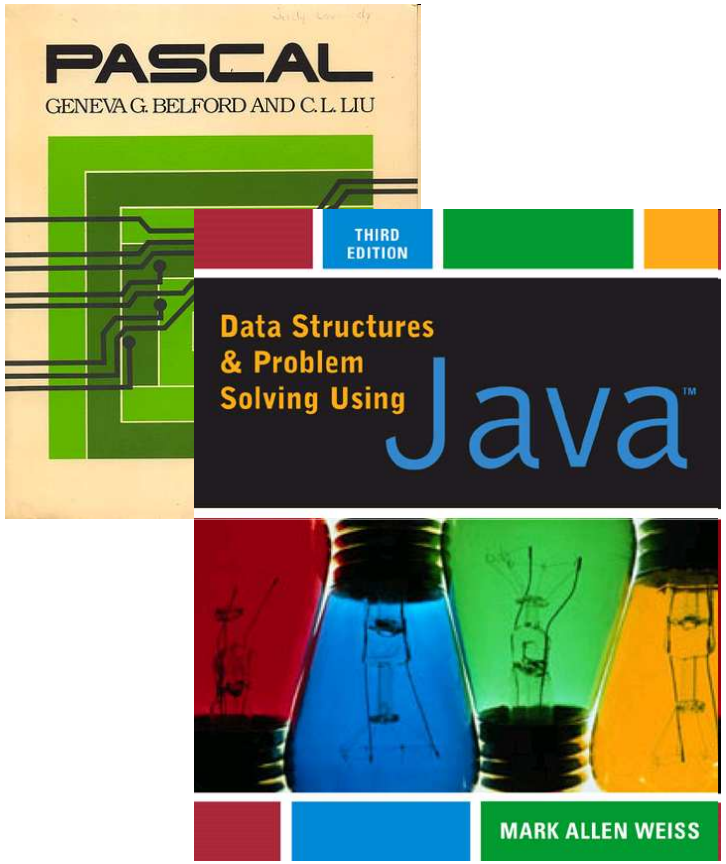
Ejemplo de síntesis en un intérprete (Simplificado)



Ejemplo de síntesis en un traductor (Simplificado)



Definición de un lenguaje de programación



- **Definir formalmente un lenguaje fuente o interpretado** es un requisito fundamental para poder construir procesadores robustos sobre dicho lenguaje
  - Especialmente para construir el módulo de análisis

Pasos para definir un lenguaje de programación

1. Descripción informal del lenguaje
   - Usando ejemplos para ilustrar sus características
2. Definición **léxica**
3. Definición **sintáctica incontextual**
   - Tratando sólo los aspectos *incontextuales* del lenguaje
4. Definición **sintáctica contextual**
   - Tratando también los aspectos *contextuales* del lenguaje
     - Se añaden **restricciones contextuales** a lo del paso 3
5. Definición **semántica**

Ejemplo de descripción informal: Expresiones aritméticas

1. Descripción informal del lenguaje
   - Los números reales son expresiones aritméticas
   - La suma **+**, la resta **-**, la multiplicación ***** y la división ***/*** de expresiones aritméticas son expresiones aritméticas
   - Los paréntesis **()** indican qué operandos se ven afectados por un determinado operador, y cuales no
   - *Sin paréntesis*, un operando que tenga operadores a ambos lados se verá afectado por el que tenga mayor **prioridad**. Multiplicación y división tienen la misma prioridad, que es mayor que la que tienen suma y resta.
   - *Sin paréntesis,* un operando que tenga operadores a ambos lados *con la misma prioridad,* se verá afectado por el que tenga a su izquierda (los operadores **asocian** a izquierdas**)**

   - *Ejemplos de frases en este lenguaje son:*
     - `(456.87 + 78) *  20`
     - `456 - 5 / 67.6`

Definición léxica

- Documento técnico que especifica los **aspectos léxicos** (a veces llamados *morfología* o *microsintaxis*) del lenguaje que se está definiendo
  1. Identifica elementos básicos: **las categorías léxicas**
  2. Describe formalmente cada uno de dichos elementos

- **Categorías léxicas**
  - *Ej: Identificador, Número, CadenaDeCaracteres, ParéntesisAbierto, ParéntesisCerrado, etc.*
  - Se pueden ver como pequeños **lenguajes formales**, de hecho asumimos que deben ser **lenguajes regulares**
    - Ej: La categoría léxica *Número* es simplemente el conjunto infinito de cadenas finitas formadas con los dígitos: *0, 1, 2, 3, 4, 5, 6, 7, 8 y 9*
    - **REPASO: LENGUAJES FORMALES (Expresiones Regulares)**

Definición léxica mediante expresiones regulares

- Está formada por una serie de **categorías léxicas** definidas cada una mediante una expresión regular
  - *nombre-categoría1* ≡ *expresión regular*
  - *nombre-categoría2* ≡ *expresión regular*
  - …
- En las expresiones regulares pueden salir mencionadas las categorías léxicas previamente definidas
  - Su nombre debe escribirse entre llaves: *{nombre-categoría}*
- También puede ser necesario hacer algunas **definiciones léxicas auxiliares**, que no son categorías léxicas propiamente dichas (no interesan para el análisis sintáctico), pero sirven de apoyo para definirlas

Definición léxica mediante gramáticas regulares

- **REPASO: LENGUAJES FORMALES (Gramáticas regulares)**
- Las expresiones regulares son en realidad una *alternativa* a la forma más habitual de definir un lenguaje, que es usando gramáticas (en este caso regulares)

Definición léxica mediante autómatas finitos

- **REPASO: LENGUAJES FORMALES (Autómatas finitos)**
- En la implementación de analizadores léxicos, se usan AFDs para **reconocer las categorías léxicas**
  - Tendremos **un estado final *como mínimo* por cada categoría léxica** que queramos reconocer
- Existen incluso programas que son **generadores automáticos de analizadores léxicos** (por ejemplo Lex)
  - **Traductores que pasan de ERs a AFDs *optimizados***

Para especificar lenguajes regulares, siempre es más cómodo usar ERs (o gramáticas regulares) que AFs

Ejemplo de definición léxica: Expresiones aritméticas

2. Definición léxica *(usando expresiones regulares)*
   - parte-entera ≡ 0|[1-9][0-9]*
   - parte-decimal ≡ .[0-9]+
   - número ≡ {parte-entera}{parte-decimal}?
   - ~~suma ≡ \+~~
   - ~~resta ≡ \-~~
   - ~~multiplicación ≡ *~~
   - ~~división ≡ /~~
   - ~~paréntesis-apertura ≡ \(~~
   - ~~paréntesis-cierre ≡ \)~~
   - + ≡ \+
   - - ≡ \-
   - * ≡ *
   - / ≡ /
   - (≡ \(
   -) ≡ \)

Definición sintáctica

- Documento técnico que especifica los **aspectos sintácticos** del lenguaje que se está definiendo
  (cómo es la **estructura sintáctica** de todas las posibles **secuencias de componentes léxicos** a procesar)

  1. Identifica elementos básicos: **las categorías sintácticas**
  2. Describe formalmente cada uno de dichos elementos

- Es un requisito imprescindible para poder implementar **analizadores sintácticos** robustos de un lenguaje

Definición sintáctica incontextual

- Aquella primera parte de la definición sintáctica en la que se trabaja, para simplificar, con esta hipótesis:
  - Hipótesis de Incontextualidad: Considerando que **no hay restricciones contextuales** en el lenguaje de todas las secuencias posibles de componentes léxicos que pueden formarse
    → *La sintaxis de cualquier lenguaje de programación la vamos a poder expresar con un* ***lenguaje incontextual***

Definición sintáctica incontext. mediante gramáticas incontext.

- **REPASO: LENGUAJES FORMALES (Gramáticas incontextuales)**
- Los *terminales* serán las **categorías léxicas** (ya no usamos el alfabeto de símbolos alfanuméricos)
  (Para distinguirlos se pondrán entre **< >** como en BNF)
- Los *no terminales* serán las **categorías sintáticas** (a su vez también se pueden ver como sublenguajes formales)
  - *Ej: Programa, SecciónDeclaraciónVariables, DeclaraciónVariable, etc.*
- Las producciones de un mismo no terminal se pueden agrupar en una sola producción
  (usando **|** como en BNF para separar alternativas en la parte derecha)

Definición sintáctica incontext. mediante gramáticas incontext.

- Los *árboles de derivación,* en el contexto de la definición sintáctica, se llamarán **árboles de análisis sintáctico** y serán de mucha ayuda para hacer explícita la estructura de las sentencias de un lenguaje

- También existen **programas que generan automáticamente analizadores sintácticos** (y hasta traductores enteros) como JavaCC o YACC

El problema de la ambigüedad

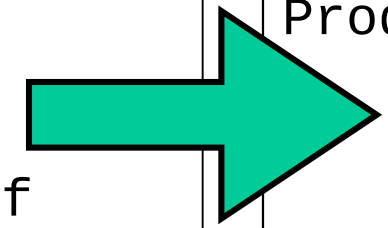
- En la documentación destinada al usuario de un lenguaje de programación *a veces hay ambigüedades*, que se resuelven con *explicaciones en lenguaje natural*
- Sin embargo, **nosotros siempre evitaremos usar gramáticas ambiguas** (aquellas donde una sentencia tiene varios análisis posibles) para definir la sintaxis de un lenguaje de programación
  - Por una **cuestión pragmática**, en realidad: queremos que nuestro procesador de lenguaje tenga **sólo una forma válida** de "entender" cada programa
  - Además ¡si tenemos una gramática ambigua es porque nos falta algo por especificar en el lenguaje! → *Hay que averiguar qué es y proponer otra* ***gramática equivalente*** *pero que no sea ambigua y que especifique completamente el lenguaje*

Ejemplos sobre cómo evitar las gramáticas ambiguas

- La clave está en que al definir lenguajes de programación importa **la estructura sintática** de sus sentencias y no sólamente sus sentencias

G_{fbf}

Terminales: p, $\wedge$, $\vee$, $\neg$
No terminales: fbf
Axioma: fbf
Producciones:
  fbf $\rightarrow$ p
  fbf $\rightarrow$ $\neg$ fbf
  fbf $\rightarrow$ fbf $\wedge$ fbf
  fbf $\rightarrow$ fbf $\vee$ fbf



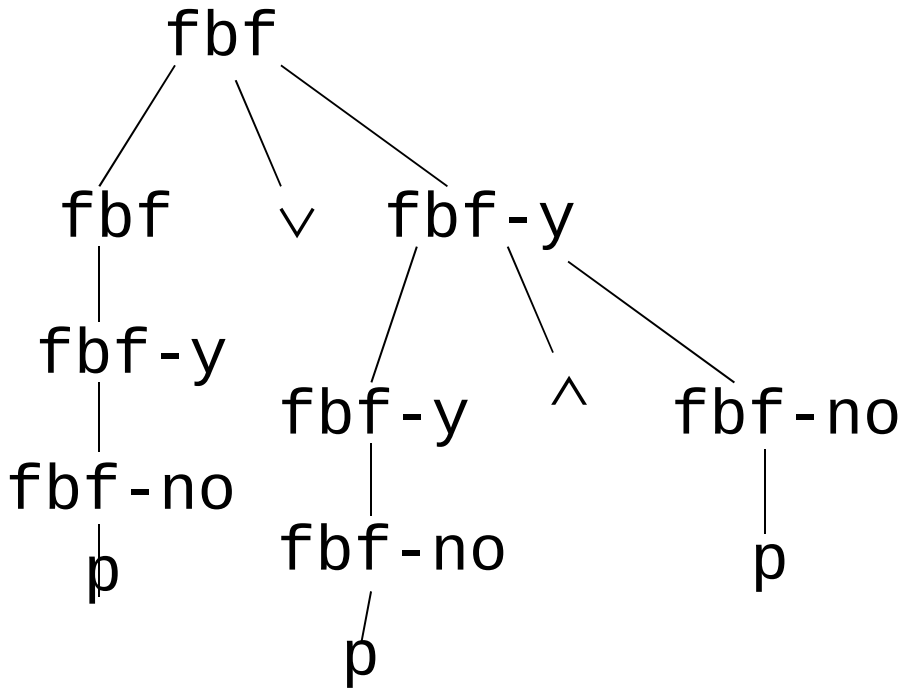
$G_{fbf\text{-}no\text{-}ambigua}$

Terminales: p, $\wedge$, $\vee$, $\neg$
No terminales: fbf, fbf-y,
      fbf-no
Axioma: fbf
Producciones:
      fbf $\rightarrow$ fbf $\vee$ fbf-y
      fbf $\rightarrow$ fbf-y
      fbf-y $\rightarrow$ fbf-y $\wedge$ fbf-no
      fbf-y $\rightarrow$ fbf-no
      fbf-no $\rightarrow$ $\neg$ fbf-no
      fbf-no $\rightarrow$ p

Ejemplos sobre cómo evitar las gramáticas ambiguas

- Con las nuevas producciones de $\mathbf{G_{fbf\text{-}no\text{-}ambigua}}$ **hacemos explícita la prioridad y la asociatividad** de los operadores, que es lo que faltaba en $\mathbf{G_{fbf}}$
  - Ahora sólo hay una estructura posible por sentencia

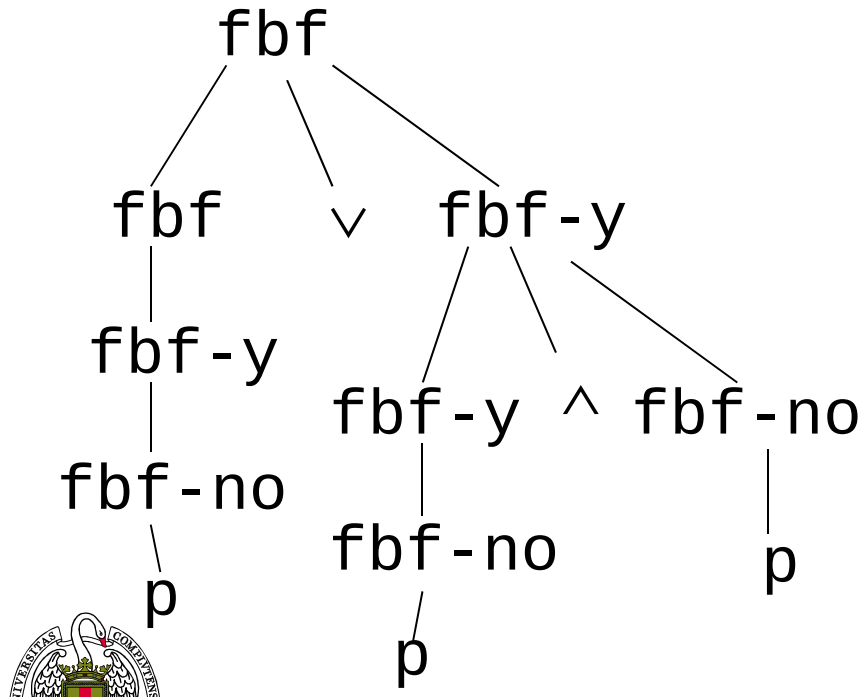
p ∨ p ∧ p



Prioridades y asociatividad

- El orden de prioridad es *¬, ∧ y luego ∨*, justo **el inverso al de las producciones** y los operadores asocian *a izquierdas* **igual que la recursión de las producciones**

p ∨ p ∧ p



$G_{\text{fbf-no-ambigua}}$

Terminales: p, ∧, ∨, ¬
No terminales: fbf, fbf-y, fbf-no
Axioma: fbf
Producciones:

- **fbf → fbf ∨** fbf-y
- fbf → fbf-y
- **fbf-y → fbf-y ∧** fbf-no
- fbf-y → fbf-no
- fbf-no → ¬ fbf-no
- fbf-no → p

Prioridades y asociatividad

- La prioridad de los operadores se “codifica” en la sintaxis organizando las categorías sintácticas **por niveles**
  - Una categoría sintáctica se define en términos de aquellas otras que tienen mayor nivel de prioridad
- La asociatividad de un operador binario ⊗ se “codifica” en la sintaxis **según se use la recursión**
  - Asociatividad a izquierdas: recursión a izquierdas
    exp-a::= exp-a ⊗ *exp-b*
  - Asociatividad a derechas: recursión a derechas
    exp-a ::= exp-b ⊗ *exp-a*
  - No asociatividad: no recursión
    exp-a ::= exp-b ⊗ exp-b

Utilidad de los delimitadores

- Los **delimitadores** son un mecanismo sintáctico que permite hacer sentencias con **terminales similares** pero **muy diversas estructuras**, a pesar de las reglas de prioridad y asociatividad
  - Ej: Los *paréntesis* **()** de una expresión aritmética
  - Ej: El *punto y coma* **;** del final de una instrucción
- Las gramáticas con **pocos delimitadores y mucho azúcar sintáctico** (varias maneras de expresar una misma cosa) generan lenguajes **más cómodos de usar**, pero tienen más riesgo de ser ambigüas
  - "Syntactic sugar causes cancer of the semicolom" (Alan Perlis)
  - Ejemplo de paréntesis incómodos: LISP
  - ¡Encontrar el equilibrio es uno de los puntos clave en el diseño de la sintaxis de un lenguaje de programación!

Ejemplo: Los paréntesis

- Usamos ciertos terminales (ej. paréntesis) para tener varias estructuras anidadas posibles y sin ambigüedad

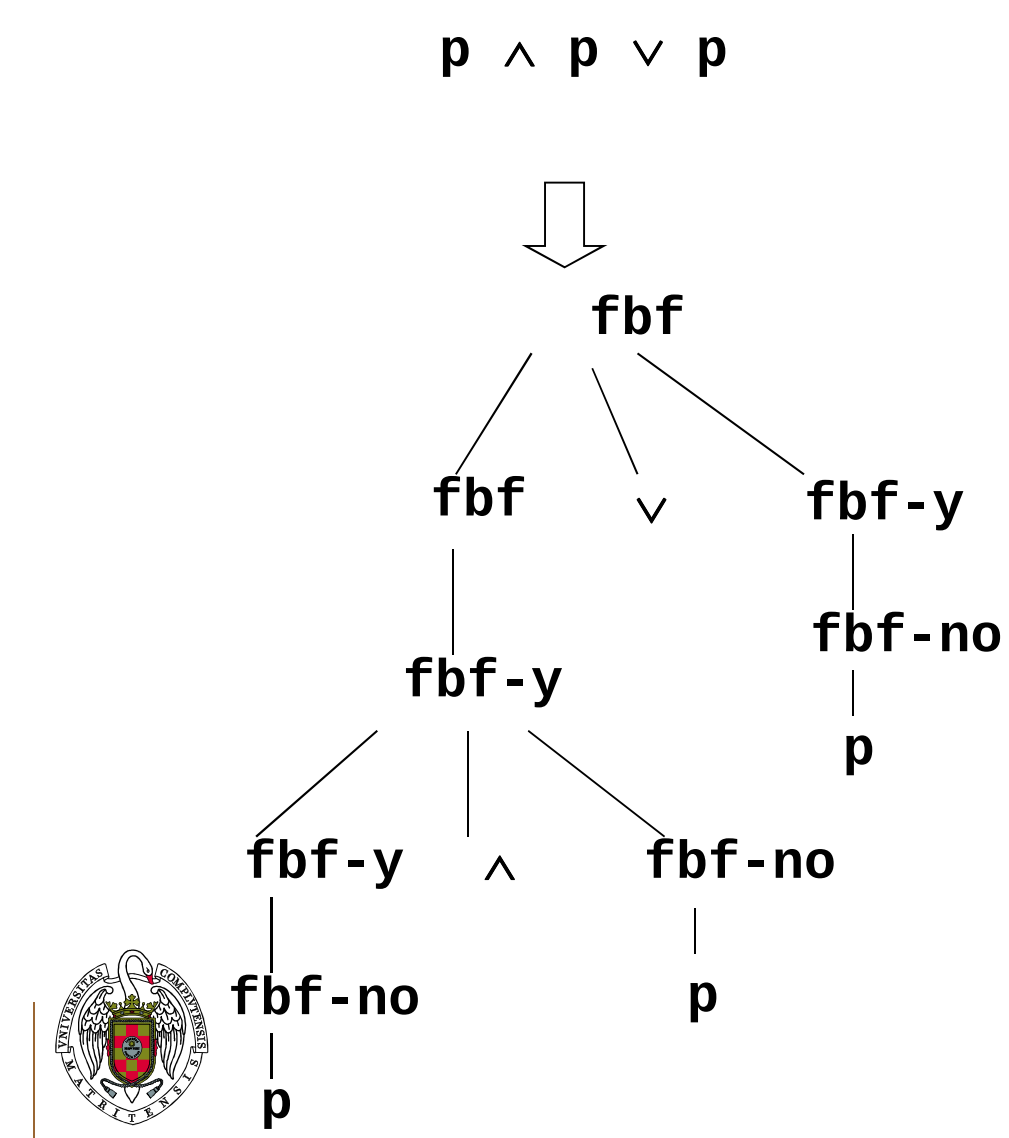
$G_{fbf\text{-}no\text{-}ambigua}$

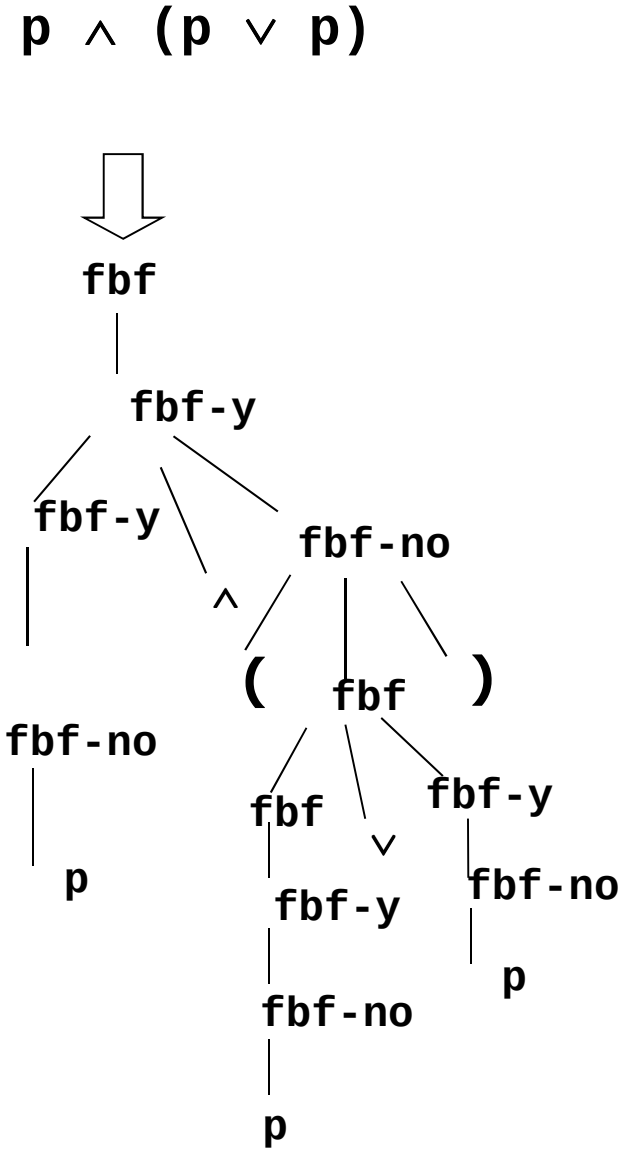
Terminales: p, $\wedge$, $\vee$, $\neg$
No terminales: fbf, fbf-y,
fbf-no
Axioma: fbf
Producciones:
fbf $\rightarrow$ fbf $\vee$ fbf-y
fbf $\rightarrow$ fbf-y
fbf-y $\rightarrow$ fbf-y $\wedge$ fbf-no
fbf-y $\rightarrow$ fbf-no
fbf-no $\rightarrow$ $\neg$ fbf-no
fbf-no $\rightarrow$ p

$G_{fbf\text{-}final}$

Terminales: p, $\wedge$, $\vee$, $\neg$, (,)
No terminales: fbf, fbf-y,
fbf-no
Axioma: fbf
Producciones:
fbf $\rightarrow$ fbf $\vee$ fbf-y
fbf $\rightarrow$ fbf-y
fbf-y $\rightarrow$ fbf-y $\wedge$ fbf-no
fbf-y $\rightarrow$ fbf-no
fbf-no $\rightarrow$ $\neg$ fbf-no
fbf-no $\rightarrow$ p
fbf-no $\rightarrow$ (fbf)

Ejemplo: Los paréntesis





Definición sintáctica incontextual mediante autómatas a pila

- **REPASO: LENGUAJES FORMALES (Autómatas a pila)**
- Los autómatas a pila son capaces de reconocer cualquier lenguaje incontextual, igual que las gramáticas incontextuales son capaces de generarlo
  - Toda gramática incontextual tiene asociado un **autómata a pila equivalente**
  - ¡Sólo las gramáticas incontextuales **deterministas** tienen asociado un **autómata a pila determinista equivalente!**
    - Esto hace que siempre busquemos definir la sintaxis de un lenguaje de programación con una **gramática incontextual determinista**
- *Para especificar un lenguaje incontextual siempre es más cómodo usar una gramática incontextual que un autómata a pila*

Ejemplo de definición sintáctica incontextual: Expresiones aritméticas

3. Definición sintáctica incontextual
(usando una gramática incontextual)

```
Exp ::= num     |
        Exp + Exp |
        Exp - Exp |
        Exp * Exp |
        Exp / Exp |
        ( Exp )
```

¡Ambigüedad!

```
Exp  ::= Exp OpAd Term
Exp  ::= Term
Term ::= Term OpMul Fact
Term ::= Fact
Fact ::= num | ( Exp )
OpAd ::= + | -
OpMul ::= * | /
```

Definición sintáctica contextual y definición semántica

Los siguientes pasos son más complejos y se estudian a lo largo de todo el tema…
(usando el ejemplo de las expresiones aritméticas)

4. Definición sintáctica contextual
   - Añadiendo **restricciones contextuales** sobre el paso 3
   - Detallando la estructura y el proceso de construcción de la **tabla de símbolos**: estructura de datos auxiliar (tabla) que se construye dinámicamente durante el análisis y se usa en todas las etapas de análisis y síntesis, dando soporte a las restricciones contextuales al procesar un programa
5. Definición semántica
   - Etiquetando las estructuras sintácticas con información adicional sobre el *significado* de cada categoría sintáctica

Críticas, dudas, sugerencias...

Federico Peinado
www.federicopeinado.es