# Especificación de un Lenguaje de Programación Mínimo y Construcción de su Procesador (2)

## Procesadores de Lenguaje

# Contenidos

**(Capítulos 1 y 2 de Aho et al.)**

1. **Especificación de un lenguaje de programación mínimo y construcción de su procesador**.
   - Introducción a los lenguajes de programación y a los procesadores de lenguajes: el modelo análisis/síntesis.
   - Elementos de la teoría de lenguajes formales. **Introducción a las gramáticas de atributos: definiciones dirigidas por la sintaxis y semántica de los lenguajes de programación.**
   - **Definición del lenguaje fuente:**
     - **aspectos léxicos**
     - **aspectos sintácticos**
     - tabla de símbolos
     - restricciones contextuales.

# Elementos de la Teoría de Lenguajes Formales: gramáticas de atributos

**Los lenguajes G-2 son incontextuales, ello quiere decir que con las gramáticas utilizadas para definirlos no pueden expresarse restricciones, o dependencias, contextuales entre los elementos que componen una frase.**

**En las frases -programas- de la mayoría de los lenguajes de programación existen dependencias y restricciones contextuales. La mayoría de ellas relacionadas con la declaración y el tipo de los elementos más básicos del lenguaje**

***ejercicio: hacer una lista razonada de las restricciones contextuales más habituales en un lenguaje de programación***

***discusión : ¿la conclusión podría ser que hay que utilizar gramáticas del tipo G-1 para definir los lenguajes de programación?***

***reconsiderar el ejemplo del lenguaje*** $L = \{ x \mid x = a^n b^n c^n \}$



# Elementos de la Teoría de Lenguajes Formales: gramáticas de atributos

**La solución adoptada desde el diseño de algol 60, y posteriormente formalizado por D. Knuth en el 69, es utilizar gramáticas de atributos, o algún mecanismo equivalente.**

***ejercicio: buscar las referencias a las gramáticas de atributos en Aho et al.***

**Gramáticas de atributos**

**- Son una extensión de las gramáticas tipo G2 (G3)** ***¿cuál es su potencia expresiva?***

**- En una gramática de atributos cada símbolo del lenguaje (terminal o no-terminal) tiene asociado cero, uno, o más, atributos cuyos valores contribuyen a dar significado al símbolo**

- **Un símbolo tiene asociado el mismo conjunto de atributos en toda la gramática**
- **Distintos símbolos de la gramática pueden tener asociados distintos conjuntos de atributos**

# Elementos de la Teoría de Lenguajes Formales: gramáticas de atributos

**Los valores de los atributos de un símbolo se expresan mediante ecuaciones semánticas asociadas a cada producción de la gramática**

**En estas ecuaciones se utilizan, además de constantes y operadores, atributos de los demás símbolos de la producción, incluidos otros atributos del símbolo al que pertenece el atributo. No se pueden utilizar atributos de símbolos que no están en la producción**

***Este modelo es muy apropiado para una descripción e implementación modular de un lenguaje y su procesador.***

**Los atributos pueden tomar valores de cualquier tipo (cada atributo concreto siempre del mismo tipo): enteros, reales, caracteres, cadenas de caracteres, listas, tablas, etc.**



# Elementos de la Teoría de Lenguajes Formales: gramáticas de atributos

**Ejemplo: dada la siguiente gramática de atributos (D.Knuth, 69)**

| | |
|---|---|
| **<lista-bits> --> <lista-bits> <bit>** | |
| | ***$<\text{lista-bits}>_0.val = 2*<\text{lista-bits}>_1.val + <\text{bit}>.val$*** |
| **<lista-bits> --> <bit>** | ***<lista-bits>.val = <bit>.val*** |
| **<bit> --> 0** | ***<bit>.val = 0*** |
| **<bit> --> 1** | ***<bit>.val = 1*** |

**y las frases, 11001, 01001, construir su árbol de derivación y evaluar sus atributos (decorar el árbol).** ***Es bueno dar una interpretación al significado de cada símbolo y comprobar como éste está representado por sus atributos.***

***Ejercicios (notar que son dos usos muy diferentes de las GA):***

***- escribe una gramática de atributos para generar los números binarios decimales y calcular su valor, p.e. 10.001, 0.011***

***- escribe una gramática de atributos para el lenguaje*** $L = \{ x \mid x = a^n b^n c^n \}$

# Elementos de la Teoría de Lenguajes Formales: gramáticas de atributos

**Ejemplo, dada la siguiente gramática de atributos**

```
exp --> exp + term        exp0.val = exp1.val + term.val
exp --> term              exp.val = term.val
term --> term * fact      term0.val = term1.val * fact.val
term --> fact             term.val = fact.val
fact --> NUM              fact.val = NUM.val
```

**y las frases 33+4*5 , 10* 5 *2 +7 construir el árbol de derivación y evaluar sus atributos (decorar el árbol)**

***discusión:***

***¿coincide esta gramática con tu respuesta a los ejercicios anteriores?***

***¿qué representa y como se obtiene el terminal NUM y el valor de su atributo***

***ejercicio:***

***- modificar esta gramática de atributos para obtener la lista de instrucciones de una máquina a pila que sirva para calcular el valor de cada expresión (el ejemplo utilizado al explicar el modelo análisis-síntesis).***



# Elementos de la Teoría de Lenguajes Formales: atributos sintetizados

**Los ejemplos anteriores utilizan atributos sintetizados, significando ésto que las ecuaciones semánticas expresan siempre el valor de un atributo de una categoría sintáctica en función de los valores de los atributos de los símbolos que componen esa categoría sintáctica, formalmente:**

$X_0 \text{ --> } X_1\ X_2 \ldots X_n \qquad X_0.x_{0j} = f(X_1.x_{1k}, X_2.x_{2l}, \ldots X_n.x_{nm})$

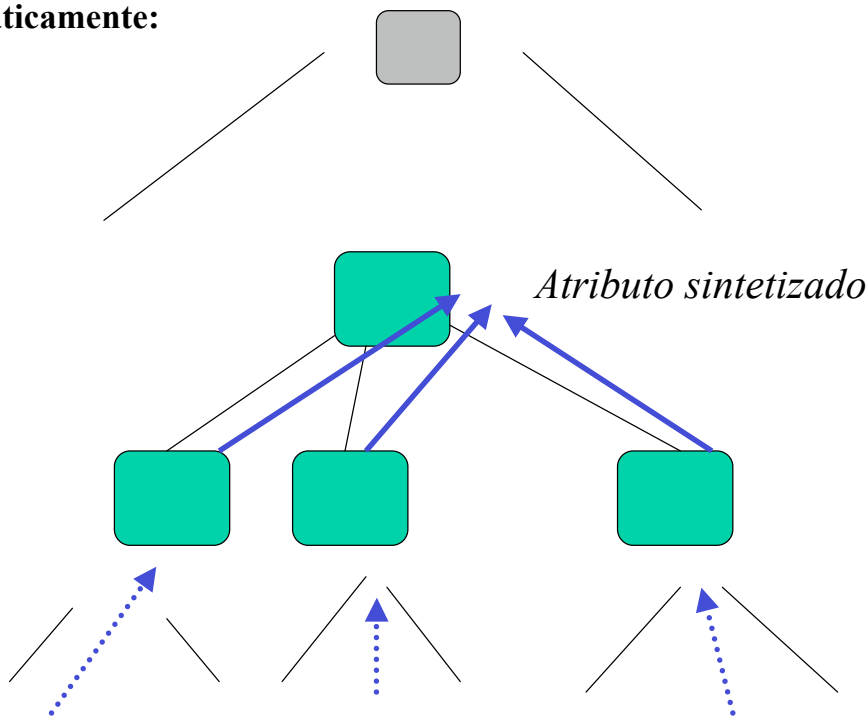
**ejemplos**

```
exp --> exp + term        exp0.val = exp1.val + term.val
```

```
<lista-bits> --> <lista-bits> <bit>
        <lista-bits>0.val = 2*<lista-bits>1.val + <bit>
```

# Elementos de la Teoría de Lenguajes Formales: atributos sintetizados

**Esquemáticamente:**






# Elementos de la Teoría de Lenguajes Formales: atributos sintetizados y heredados

**Los atributos sintetizados sirven para expresar traducciones donde el significado del "todo", la categoría sintáctica (e.g. la expresión, o la lista de bits), depende exclusivamente del significado de sus "componentes", los elementos del lado derecho de la producción.**

**En los lenguajes de programación hay construcciones donde el significado de una categoría sintáctica depende del contexto en que esta aparece.**

***discusión: re-interpretar así la lista de restricciones contextuales elaborada anteriormente***

**La forma más conveniente ¡pero no la única! de expresar estas construcciones es utilizando atributos heredados**

# Elementos de la Teoría de Lenguajes Formales: atributos sintetizados y heredados

***ejemplo: definir un lenguaje para una calculadora con memoria, la memoria es el contexto en el que se evalúa una expresión:***

**1ª aproximación, el lenguaje objetivo consta de frases como:**

*M=5 31+M*5+M*2. 30+2. M=20 3*5+M.*

```
<calculadora> --> <memoria> <expresion> .
<memoria> --> M = NUM
<memoria> --> λ
<expresion> --> <expresion> + <termino>
<expresion> --> <termino>
<termino> --> <termino> * <factor>
<termino> --> <factor>
<factor> --> NUM
<factor> --> M
```



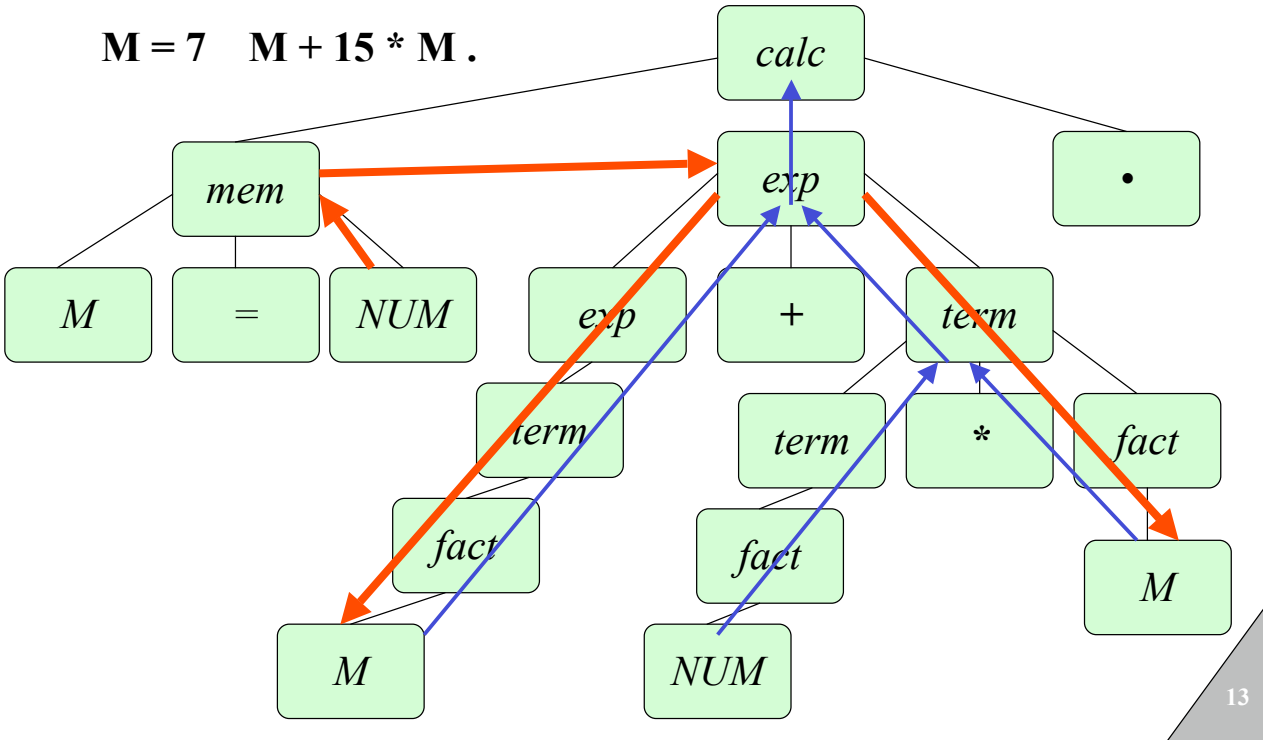
# Elementos de la Teoría de Lenguajes Formales: atributos sintetizados y heredados

- ***ejercicio: comprobar que la gramática anterior genera el tipo de frases que buscamos y les da una estructura sintáctica conveniente (construir los árboles de derivación de las frase ejemplo)***

- **el problema es especificar mediante una gramática de atributos un intérprete que evalúe cada `<expresion>` `<termino>` `<factor>` en el contexto del valor guardado en la memoria.**

- **esencialmente lo que necesitamos es guardar en un atributo del símbolo `<memoria>` el valor de la memoria, y pasarlo como un atributo heredado a todas las otras categorías sintácticas (`<expresion>` `<termino>` `<factor>`) que pudiesen necesitarlo.**

# Elementos de la Teoría de Lenguajes Formales: atributos sintetizados y heredados






# Elementos de la Teoría de Lenguajes Formales: atributos sintetizados y heredados

**Este cómputo podemos formalizarlo mediante una gramática de atributos (lo que sigue es un fragmento de la misma)**
***ejercicio: representar gráficamente cada producción y el flujo de la información contenida en los atributos en la misma:***

**<memoria> --> M = NUM** `<memoria>.mval = NUM.val`
***introducimos el atributo mval para guardar el valor de la memoria.***

**<calculadora> --> <memoria> <expresion> .**
`<expresion>.mvalh = <memoria>.mval`
***por claridad nombramos al nuevo atributo de <expresion> mvalh para indicar que se trata de un atributo heredado de un hermano***

**<expresion> --> <expresion> + <termino>**
$\texttt{<expresion>}_1\texttt{.mvalh} = \texttt{<expresion>}_0\texttt{.mvalh}$
`<termino>.mvalh = <expresion>`$_0$`.mvalh`
***nuevo atributo de <termino>, mvalh , heredado del padre***

# Elementos de la Teoría de Lenguajes Formales: atributos sintetizados y heredados

```
<termino> --> <termino> * <factor>
                          <termino>1.mvalh = <termino>0.mvalh
                            <factor>.mvalh = <termino>0.mvalh
<expresion> --> <termino>
                         <termino>.mvalh = <expresion>.mvalh
<termino> --> <factor>
                            <factor>.mvalh = <termino>.mvalh
<factor> --> M
                                <factor>.val = <factor>.mvalh
```

*discusión:*
*- ¿qué particularidad tiene la última producción y ecuación semántica?*
*- ¿sería posible especificar esta traducción sin utilizar atributos heredados?*
*ejercicios:*
*- escribe la gramática de atributos completa de este intérprete*
*- representa el flujo de información en el árbol de derivación decorado de una frase*
*- diseña una calculadora más compleja (varias memorias, paréntesis en las expresiones, etc.) -interesante su proximidad a la tabla de símbolos-*



# Elementos de la Teoría de Lenguajes Formales: atributos sintetizados y heredados

**Atributos heredados: el significado, o una parte del significado, de una categoría sintáctica depende del contexto, esto es, puede depender de los atributos del padre, de los hermanos, o de la propia categoría. Formalmente:**

$X_0 \rightarrow X_1\ X_2 \ldots X_n \qquad X_i.x_{ij} = f(X_0.x_{0k}, X_1.x_{1l}, X_2.x_{2m}, \ldots, X_n.x_{np})$
$0 \le i \le n$

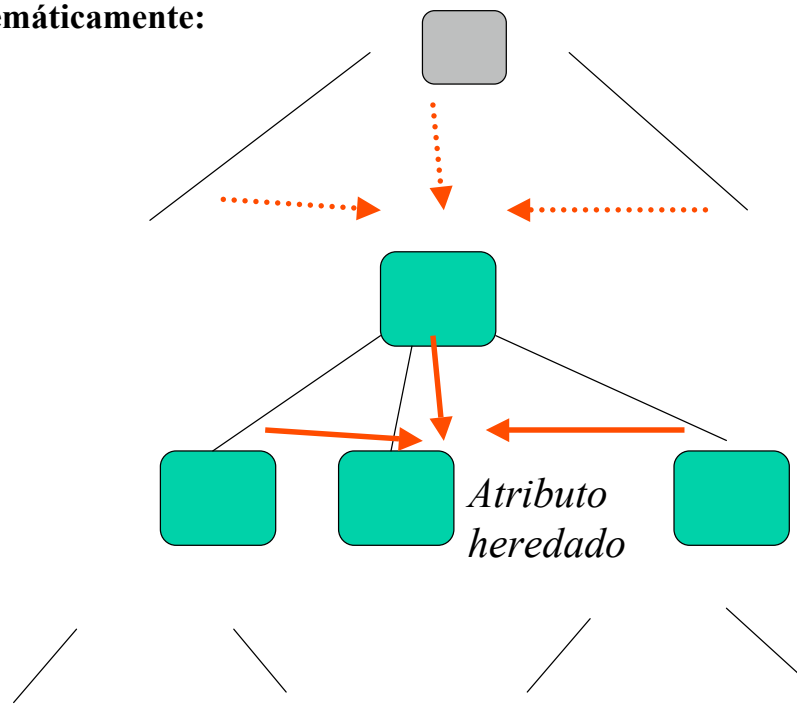
*ejemplos:*

```
<calculadora> --> <memoria> <expresion> .
                          <expresion>.mvalh = <memoria>.mval
<termino> --> <termino> * <factor>
                          <termino>1.mvalh = <termino>0.mvalh
                            <factor>.mvalh = <termino>0.mvalh

<factor> --> M             <factor>.val = <factor>.mvalh
```

# Elementos de la Teoría de Lenguajes Formales: atributos heredados

**Esquemáticamente:**






# Elementos de la Teoría de Lenguajes Formales: traducción dirigida por sintaxis

**Cuando se utilizan gramáticas de atributos para expresar como se lleva a cabo el proceso de traducción de un lenguaje fuente a un lenguaje objeto se habla de "definiciones dirigidas por sintaxis" o "traducciones dirigidas por sintaxis"** ***ver el capítulo 2.3 del libro de Aho***

**Definiciones S-atribuidas: solo se utilizan atributos sintetizados**

**Definiciones L-atribuidas: se utilizan además atributos heredados del padre y/o de hermanos a la izquierda**

***discusión: ¿se te ocurre porqué son importantes la definiciones L-atribuidas?***

***un caso de estudio interesante: dar una definición S-atribuida de un procesador que acepte como lenguaje fuente el lenguaje de las expresiones regulares y produzca como lenguaje objeto la tabla de la función de transición de un AFN capaz de reconocer el lenguaje definido por la expresión regular de entrada.***

# Elementos de la Teoría de Lenguajes Formales: traducción dirigida por sintaxis

**Una gramática para las expresiones regulares:**

```
<exp-reg> --> <exp-reg> '|' <concat>
<exp-reg> --> <concat>
<concat> --> <concat> <rep>
<concat> --> <rep>
<rep> --> <elem>*
<rep> --> <elem>
<elem> --> SIMBOLO
<elem> --> ( <exp-reg>)
```

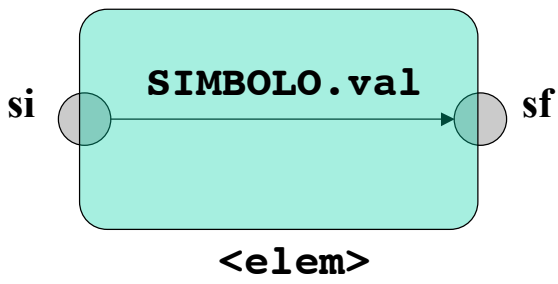
**¡Importante!** **antes de formalizar mediante una gramática de atributos una traducción, debemos tener una idea muy clara de cómo vamos a realizar la traducción, en este caso, aplicaremos el algoritmo de Thompson (capítulo 3.7 del libro de Aho et al.)**



# Elementos de la Teoría de Lenguajes Formales: traducción dirigida por sintaxis

**algoritmo de Thompson**

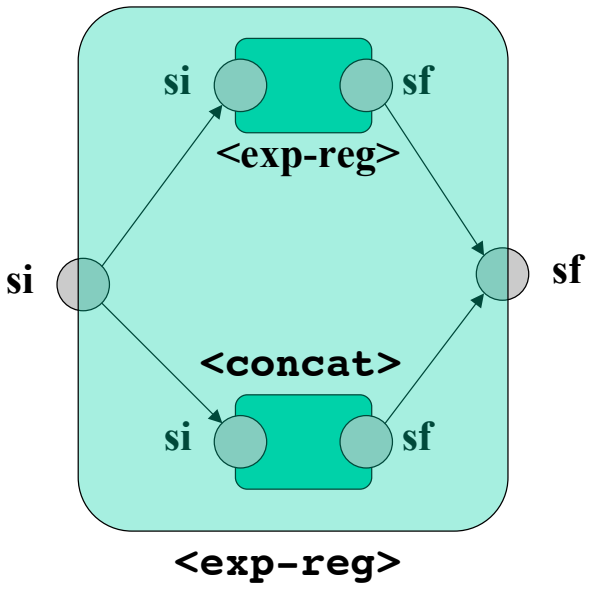
**<elem> --> SIMBOLO**



# Elementos de la Teoría de Lenguajes Formales: traducción dirigida por sintaxis

**algoritmo de Thompson**

```
<exp-reg> --> <exp-reg> | <concat>
```

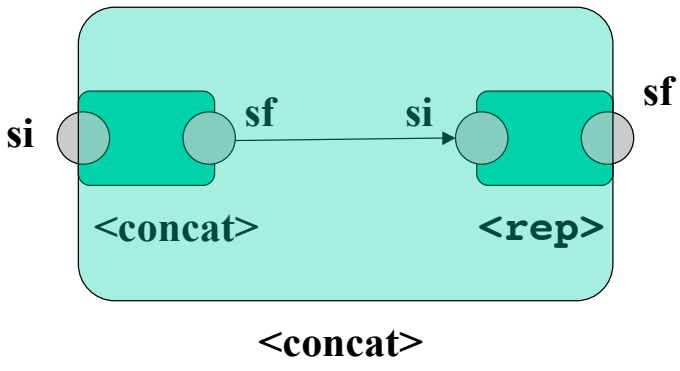





# Elementos de la Teoría de Lenguajes Formales: traducción dirigida por sintaxis

**algoritmo de Thompson** ***(aquí difiere del propuesto en Aho et al. notar que de esta forma solo necesitamos atributos sintetizados)***

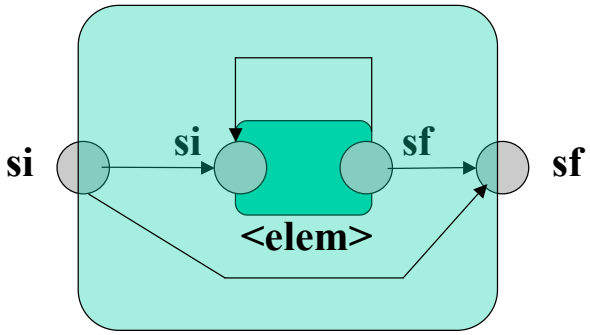
```
<concat> --> <concat> <rep>
```



# Elementos de la Teoría de Lenguajes Formales: traducción dirigida por sintaxis

**algoritmo de Thompson**

```
<rep> --> <elem> *
```



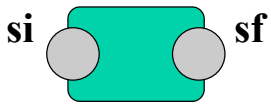

**<exp-reg>**



# Elementos de la Teoría de Lenguajes Formales: traducción dirigida por sintaxis

**producciones que solo copian o transmiten los atributos**

```
<exp-reg> --> <concat>
<concat> --> <rep>
<rep> --> <elem>
<elem> --> ( <exp-reg>)
```




**las ecuaciones semánticas para estas producciones seguirían este modelo:**

```
<elem> --> ( <exp-reg>)
                          <elem>.tabla = <exp-reg>.tabla
                          <elem>.si = <exp-reg>.si
                          <elem>.sf = <exp-reg>.sf
```

# Elementos de la Teoría de Lenguajes Formales: traducción dirigida por sintaxis

**el modelo para otras producciones sería:**

```
<elem> --> simbolo
       <elem>.si = creaestado()
       <elem>.sf = creaestado()
       <elem>.tabla = [<elem>.si; simbolo.codigo; <elem>.sf]

<exp-reg> --> <exp-reg> '|' <concat>
       <exp-reg>0.si = creaestado()
       <exp-reg>0.sf = creaestado()
       <exp-reg>0.tabla = <exp-reg>1.tabla
          ||<concat>.tabla
          ||[<exp-reg>0.si ; lambda; <exp-reg>1.si, <concat>.si]
          ||[<exp-reg>1.sf ; lambda; <exp-reg>0.sf]
          ||[<concat>.sf ; lambda; <exp-reg>0.sf]
```



# Elementos de la Teoría de Lenguajes Formales: traducción dirigida por sintaxis

***Ejercicio:***
***- Escribe la gramática de atributos completa para el ejemplo anterior***
***- Dada la expresión ab(c\|d)a*, construye el árbol de derivación, decora los nodos con sus atributos y marca con flechas el flujo de la información sobre dicho árbol***
***- ¿Podrías optimizar la traducción dirigida por sintaxis anterior? (*)***

***Nota***
***- Formalmente una gramática es una cuádrupla, pero en una especificación más informal, todos los elementos de esa cuádrupla se pueden deducir del listado de las producciones.***

***- De la misma forma una gramática de atributos requiere que especifiquemos el conjunto de atributos de cada uno de los elementos de su vocabulario. Aunque más informalmente, estos conjuntos de atributos se pueden deducir de las ecuaciones semánticas de la gramática***

# Elementos de la Teoría de Lenguajes Formales: algoritmo de Thompson (Aho et al.)

**(*) El algoritmo de Thompson del Aho concatena los componentes del autómata utilizando como estado final del primer sub-autómata el inicial del segundo**

**¿cómo organizar esta traducción?**

**1.- podría ser S-atribuida pero ¿es la solución más apropiada?**

**2.- la solución más apropiada puede ser L-atribuida pero puede resultar compleja de expresar.**

***Ejercicio***

***- dar una gramática de atributos que permita construir el AFN de una expresión regular utilizando el algoritmo de Thompson tal y como se define en el Aho (no se utiliza una transición vacía para concatenar símbolos).***

***- discutir los puntos más conflictivos de esta gramática de atributos***

***- comparar con una solución basada en la gramática de atributos que da D. Knuth para la máquina de Turing.***



# Elementos de la Teoría de Lenguajes Formales: algoritmo de Thompson (Aho et al.)

**- La especificación siguiente está basada el la que hace el propio Knuth para una máquina de Turing**

**- Se amplia la gramática con una producción adicional para inicializar los atributos que caracterizan al autómata**

**- Se introduce un atributo remoto en el sentido de Knuth para manejar de forma más simple y eficiente la tabla de la función de transición del autómata.**

**- Notar que aun así la especificación puede resultar compleja de escribir y de leer**

# Elementos de la Teoría de Lenguajes Formales: algoritmo de Thompson (Aho et al.)

***- ¡Atención! los atributos heredados introducen en las definiciones dirigidas por sintaxis el problema del orden de evaluación. Incluso puede ocurrir que ésta no sea posible (la definición no es válida si existen referencias circulares)***

***- ¡Importante! Las gramáticas de atributos no especifican el orden de evaluación. El orden de evaluación se especificará después cuando se introduzcan detalles de implementación: en los esquemas de traducción***

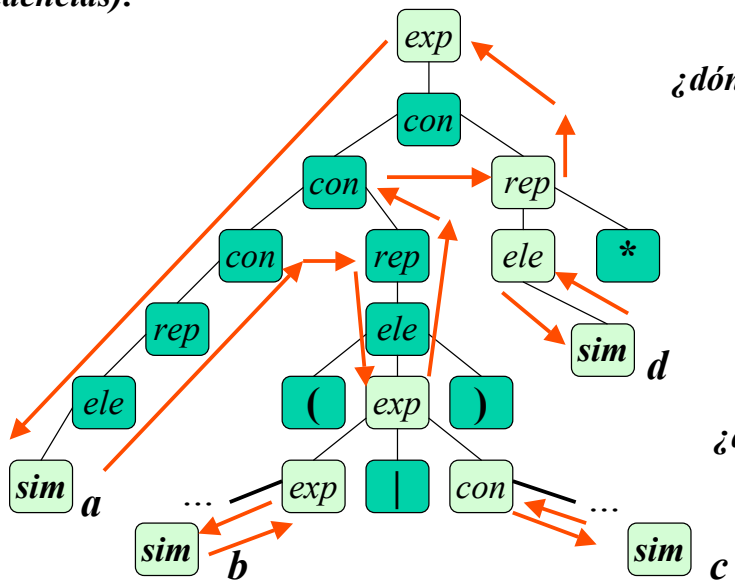
***- La definición que sigue es L-atribuida por tanto es posible mediante un solo recorrido en profundidad decorar el árbol de derivación de cualquier frase del lenguaje***

***- El resultado de la traducción es la quíntupla que caracteriza el autómata finito no determinista***



# Elementos de la Teoría de Lenguajes Formales: algoritmo de Thompson (Aho et al.)

***Antes de construir la gramática de atributos estudiar de forma global como se va a realizar la trducción, p.e., representar el árbol de derivación de una expresión regular, a(b|c)d* , y marcar sobre éste el flujo de la información (grafo de dependencias):***




***¿dónde se crean estados?***

***¿dónde se reutilizan?***

***¿dónde se propagan?***

# Elementos de la Teoría de Lenguajes Formales: algoritmo de Thompson (Aho et al.)

*Los (*) marcan ecuaciones que se introducen para mayor claridad pero que no aportan información nueva*

```
S --> <exp-reg>
          <exp-reg>.sih = creaestado()
          S.tabla = []
          S.Q = [<exp-reg>.sih, <exp-reg>.sf]
          S.q_o = <exp-reg>.sih
          S.q_f = <exp-reg>.sf
          S.V = []
```

*se introducen 5 atributos remotos: atributos del axioma que se modifican desde cualquier otra producción de la gramática, está implícita la idea de que estos atributos son atributos heredados que se transmiten a través del árbol en un recorrido en profundidad del mismo.*



# Elementos de la Teoría de Lenguajes Formales: algoritmo de Thompson (Aho et al.)

```
<exp-reg> --> <exp-reg> '|' <concat>

                          <exp-reg>0.si = <exp-reg>0.sih (*)
                          <exp-reg>1.sih = creaestado()
                          S.Q = S.Q || [<exp-reg>1.sih]
                          <concat>.sih = creaestado()
                          S.Q = S.Q || [<concat>.sih]
                          <exp-reg>0.sf = creaestado()
                          S.Q = S.Q || [<exp-reg>0.sf]
   S.tabla = S.tabla
        ||[<exp-reg>0.sih; λ; <exp-reg>1.sih, <concat>.sih]
        ||[<exp-reg>1.sf; λ; <exp-reg>0.sf]
        ||[<concat>.sf; λ; <exp-reg>0.sf]
```

*el orden de evaluación de las ecuaciones no está determinado en la definición*

# Elementos de la Teoría de Lenguajes Formales: algoritmo de Thompson (Aho et al.)

```
<exp-reg> --> <concat>
                          <exp-reg>.si = <exp-reg>.sih (*)
                          <concat>.sih = <exp-reg>.sih
                          <exp-reg>.sf = <concat>.sf
```

***no es necesario escribir las ecuaciones que transmiten a través del árbol los atributos remotos, sin embargo, sí hacemos explícitas las ecuaciones que manipulan los estados inicial y final de los sub-autómatas , es decir, no consideramos atributos remotos a si, sf.***



# Elementos de la Teoría de Lenguajes Formales: algoritmo de Thompson (Aho et al.)

```
<concat> --> <concat> <rep>
                          <concat>0.si = <concat>0.sih (*)
                          <concat>1.sih = <concat>0.sih
                          <rep>.sih = <concat>1.sf
                          <concat>0.sf = <rep>.sf
```

***no se introduce ninguna entrada en la tabla de la función de transición del autómata, solamente se pasa la información: el estado inicial de la categoría sintáctica*** **<rep>** ***debe ser el final de*** **<concat>$_1$**

```
<concat> --> <rep>
                          <concat>.si = <concat>.sih (*)
                          <rep>.sih = <concat>.sih
                          <concat>.sf = <rep>.sf>
```

# Elementos de la Teoría de Lenguajes Formales: algoritmo de Thompson (Aho et al.)

```
<rep> --> <elem>*
                    <rep>.si = <rep>.sih (*)
                    <elem>.sih = creaestado()
                    S.Q = S.Q || [<elem>.sih]
                    <rep>.sf = creaestado()
                    S.Q = S.Q || [<rep>.sf]
          S.tabla = S.tabla
                || [<rep>.sih ; lambda; <elem>.sih, <rep>.sf]
                || [<elem>.sf ; lambda; <rep>.sf, <elem>.sih]

<rep> --> <elem>
                    <rep>.si = <rep>.sih (*)
                    <elem>.sih = <rep>.sih
                    <rep>.sf = <elem>.sf
```



# Elementos de la Teoría de Lenguajes Formales: algoritmo de Thompson (Aho et al.)

```
  <elem> --> simbolo
                    <elem>.si = <elem>.sih (*)
                    <elem>.sf = creaestado()
                    S.Q = S.Q || [<elem>.sf]
                    S.V = S.V || [simbolo.codigo]
          S.tabla = S.tabla
                || [<elem>.sih; simbolo.codigo; <elem>.sf]


  <elem> --> ( <exp-reg>)
                    <elem>.si = <exp-reg>.sih (*)
                    <exp-reg>.sih = <elem>.sih
                    <elem>.sf = <exp-reg>.sf
```

# Elementos de la Teoría de Lenguajes Formales. Gramáticas de Atributos: **resumen final**

***- extienden la capacidad expresiva de las gramáticas incontextuales***

***- se pueden utilizar para dar una interpretación a las frases del lenguaje definido por una gramática incontextual***

***- se pueden utilizar para imponer restricciones -limitar- el conjunto de frases que forman el lenguaje incontextual (*)***

***- se pueden utilizar para transformar -traducir- las frases del lenguaje de entrada en frases de un lenguaje objeto ¡aunque la gramática de atributos no define el lenguaje destino o lenguaje objeto!***

***- el resultado de la traducción puede tener o no tener una interpretación sencilla como un elemento (una frase) del lenguaje objeto***

***ejercicio***

***escribe una gramática de atributos para generar los números binarios decimales y calcular su valor utilizando un atributo heredado que nos de el peso de la posición ocupada por cada bit en la secuencia, p.e. en la secuencia 1110.0111 el primer bit tiene peso 3($2^3$) y el último -4 ($2^{-4}$)***



# Contenidos

**(Capítulos 1 y 2 de Aho et al.)**

1. **Especificación de un lenguaje de programación mínimo y construcción de su procesador**.
   - Introducción a los lenguajes de programación y a los procesadores de lenguajes: el modelo análisis/síntesis.
   - Elementos de la teoría de lenguajes formales. Introducción a las gramáticas de atributos: definiciones dirigidas por la sintaxis y semántica de los lenguajes de programación**.**
   - **Definición del lenguaje fuente:**
     - **aspectos léxicos**
     - **aspectos sintácticos**
     - tabla de símbolos
     - restricciones contextuales.

# Definición del Lenguaje Fuente

- **Antes de construir el procesador para un lenguaje es necesario definir rigurosamente el lenguaje que se va a procesar.**
- **Dicha definición será utilizada para construir los módulos del procesador.**
- **En un lenguaje hay que definir:**
  - **Sus aspectos léxicos.**
  - **Sus aspectos sintácticos (independientes de contexto).**
  - **La *tabla de símbolos*, parecida a un diccionario. Es una estructura de datos auxiliar que se utiliza para imponer las restricciones contextuales y para realizar la síntesis.**
  - **Sus restricciones contextuales o dependencias contextuales (e.g. los identificadores usados deben haber sido declarados previamente).**
- ***Es importante, insistir en la conveniencia de separar la definición de tokens (lexemas) de la definición de estructuras sintácticas.***
- ***se utilizarán los conceptos formales vistos anteriormente***



# Definición del Lenguaje Fuente

- **Ejemplo inicial: el lenguaje de las *expresiones aritméticas*.**
- **La descripción informal (inductiva) de este lenguaje es:**
  - **Los números son expresiones aritméticas.**
  - **La suma, la resta, la multiplicación y la división de expresiones aritméticas son expresiones aritméticas.**
  - **Los operadores asocian a izquierdas. Además, la multiplicación y la división tienen igual prioridad, y mayor prioridad que la suma y la resta. La prioridad puede cambiarse utilizando paréntesis.**
- **Ejemplos de sentencias en este lenguaje:**

```
(456.87 + 78) *  20
 456 - 5 / 67.6
```

- ***¡Atención! el lenguaje de las expresiones aritméticas es el primer lenguaje que podemos utilizar en el trabajo de prácticas.***

# Definición del Lenguaje Fuente
## Definición Léxica

***consultar capítulos del Aho: introducción 2.6, ampliar 3.3, 3.4 y 3.6***

- **Objetivo: definir los componentes básicos del lenguaje fuente: *categorías léxicas* o *tokens*.**
- **Además: informar al programador (usuario del lenguaje) y servir como referencia para la construcción del módulo de análisis léxico del procesador.**
- **Ejemplos de categorías léxicas: *identificador*, *número*, *paréntesis de apertura*, *paréntesis de cierre, etc.***
- **Cada categoría léxica es un *lenguaje* en el sentido formal del término: conjunto (puede ser de un solo elemento) de cadenas finitas sobre un conjunto no vacío de símbolos, o alfabeto.**
- ***discusión: es muy importante identificar los lenguajes y los alfabetos/vocabularios que están en juego***



# Definición del Lenguaje Fuente
## Definición Léxica

- **Las categorías léxicas son lenguajes regulares:**
  - **Pueden describirse mediante expresiones o gramáticas regulares.**
  - **Pueden reconocerse mediante autómatas finitos.**
- **La definición léxica (morfología o micro-sintaxis) de un lenguaje supone:**
  - **Identificar sus categorías léxicas.**
  - **Describir formalmente cada categoría léxica.**
- **En la descripción de las categorías léxicas se utilizarán expresiones regulares.**
- ***Discusión:***

  ***- ¿qué ventaja tiene utilizar expresiones regulares en lugar de autómatas finitos o gramáticas regulares?***

  ***- ¿es posible prescindir de la definición léxica (y del módulo de análisis léxico) en la construcción de un lenguaje (y su procesador)?***

# Definición del Lenguaje Fuente
## Definición Léxica

- **Expresiones regulares ER(V) sobre un alfabeto (o vocabulario) V :**
  - **$\varnothing \in$ ER(V). Denota el lenguaje {}.**
  - **$\lambda \in$ ER(V). Denota el lenguaje {$\lambda$}, donde $\lambda$ es la cadena vacía.**
  - **Si x $\in$ V, x $\in$ ER(V). Denota el lenguaje {x}.**
  - **Si A $\in$ ER(V) y B $\in$ ER(V), con lenguajes denotados $L_A$ y $L_B$ respectivamente:**
    - **(AB) $\in$ ER(V). Denota el lenguaje {ab | a $\in L_A$ y b $\in L_B$}.**
    - **(X|Y) $\in$ ER(V). Denota el lenguaje $L_A \cup L_B$.**
    - **A* $\in$ ER(V). Denota el menor lenguaje $L_{A^*}$ que satisface $L_{A^*} = \{\lambda\} \cup \{a\alpha \mid a \in L_A \text{ y } \alpha \in L_{A^*}\}$. Dicho de otro modo, la concatenación de cero o más cadenas de $L_A$**
  - **Si E $\in$ ER(V) debe haber sido obtenida por alguno de los casos anteriores.**



# Definición del Lenguaje Fuente
## Definición Léxica

- **Importante! para evitar paréntesis se supone que * tiene mayor prioridad que la concatenación, y ésta que |**
- **Ejemplos:**
  - **Lenguaje de las cadenas que empiezan por *a* y van seguidas por una secuencia de *a*s o *b*s.**

    `a(a|b)*`
  - **Lenguaje de las cadenas que empiezan por *a*, van seguidas de un número indeterminado de *b*s y *c*s, y terminan con *d*.**

    `a(b|c)*d`
  - **Las frases siguientes no son ambiguas**

    `a|bc  ab|c*  ab|bcd*a`

# Definición del Lenguaje Fuente
## Definición Léxica

- **A la hora de escribir, de forma práctica, definiciones léxicas, conviene ampliar la notación del lenguaje básico de expresiones regulares:**
  - **Opcionalidad A? ≡ A\| λ  ? tiene la misma prioridad que ***
  - **Cierre positivo A+ ≡ A(A)*  + tiene la misma prioridad que ***
  - **Rango [$c_0$- $c_1$]. Tiene sentido sobre alfabetos totalmente ordenados (e.j. sobre juegos de caracteres, donde los caracteres están ordenados por su código). Ejemplo: `[0-9]`: expresión regular que denota los dígitos (0,1, 2, …), suponiendo un juego de caracteres en el que estos aparecen de forma consecutiva (e.j. ASCII, Latin-1, Unicode).**
  - **Opcionalidad. La opcionalidad también se representa utilizando corchetes, e.g. A \| B \| C es equivalente a [A B C]**
  - ***Escape* de meta-caracteres. En muchas ocasiones, será necesario incluir en el lenguaje definido caracteres que son propios de las expresiones regulares. Para ello, se utilizará el carácter de escape **
    - *** : Expresión regular que denota el carácter ***
    - **\\ : Expresión regular que denota el carácter **



# Definición del Lenguaje Fuente
## Definición Léxica

- **En la definición léxica de un lenguaje de programación, las expresiones regulares no aparecen aisladas. Para poder distinguirlas y poder referenciarlas posteriormente se da un nombre a cada una de las categorías léxicas denotadas.**
- **Esto da lugar a una secuencia de *definiciones regulares*:**

  ***nombre-categoría* ≡ *expresión-regular***

  ***nombre-categoría* ≡ *expresión-regular***

  **….**

  ***nombre-categoría* ≡ *expresión-regular***
- **En las partes derechas de las definiciones regulares puede referirse a nombres de categorías léxicas previamente definidas. Para ello se encierra su nombre entre llaves ({…}).**
- ***Discusión:***
  - ***La notación con la que se ha extendido la notación básica ¿aumenta el poder expresivo del formalismo?***

# Definición del Lenguaje Fuente
## Definición Léxica

- **Definición léxica del lenguaje de ejemplo.**

```
parte-entera ≡ 0|[1-9][0-9]*
parte-decimal ≡ .[0-9]+
número ≡ {parte-entera}{parte-decimal}?
SUMA ≡ \+
RESTA ≡ \-
MUL ≡ \*
DIV ≡ /
PAP ≡  \(
PCIERRE ≡ \)
```



# Definición del Lenguaje Fuente
## Definición Léxica

**ejercicio:**

- **interpretar los siguientes fragmentos del manual de referencia de PHP (*)**
- **construir los AF equivalentes**

***Formally the possible structure for integer literals is:***

```
decimal     : [1-9][0-9]*
            | 0
hexadecimal : 0[xX][0-9a-fA-F]+
octal       : 0[0-7]+
integer     : [+-]?decimal
            | [+-]?hexadecimal
            | [+-]?octal
```

# Definición del Lenguaje Fuente
## Definición Léxica

***Formally the possible structure for float, doubles or real numbers is:***

```
LNUM        :[0-9]+
DNUM        :([0-9]*[\.]{LNUM}) | ({LNUM}[\.][0-9]*)
EXP_DNUM    :( ({LNUM} | {DNUM}) [eE][+-]? {LNUM})
```

***Formally the possible structure for names is:***

```
[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*
```



# Definición del Lenguaje Fuente
## **Definición Sintáctica**

### Sintaxis del lenguaje fuente

***consultar capítulos del Aho:***

***introducción 2.2, se puede ampliar en 4.2***

- **Objetivo:**

  **Identificar la estructura sintáctica de las <u>secuencias de</u> <u>*tokens*</u>.**

# Definición del Lenguaje Fuente
## Definición Sintáctica

- **Objetivo: Identificar la estructura sintáctica de las secuencias de *tokens*.**
- **Además: Informar al programador y servir como base a la construcción del módulo de análisis.**
- **Hipótesis: a nivel sintáctico, las cadenas de *tokens* forman (fundamentalmente) un lenguaje incontextual:**
  - **Su sintaxis puede describirse mediante una gramática incontextual.**
  - **Esta sintaxis puede reconocerse mediante un autómata de pila.**
- **Fundamental: a nivel sintáctico se consideran cadenas de *tokens* (no de caracteres).**
  - **El conjunto de los *tokens* es el alfabeto/vocabulario del lenguaje que habitualmente identificamos como el lenguaje fuente.**
  - ***ejercicio: identificar las secuencias de tokens en las frases de un lenguaje de programación: Pascal, Java, C***



# Definición del Lenguaje Fuente

**Un lenguaje sobre tokens (palabras)**

**La estructura jerárquica de las secuencias de tokens**

**Se define con una gramática incontextual sobre el conjunto de tokens (terminales) y el de categorías sintácticas (no terminales)**

*tokens*



**Se define con una expresión(es) regular(es) (alternativamente autómata(s) finito(s) o gramática(s) regular(es)) sobre el conjunto de caracteres**

**La estructura de las secuencias de caracteres (para formar un token)**

**Un lenguaje(s) sobre octetos/caracteres (códigos ASCII)**

*caracteres/ octetos*

*discusión: ¿todas las secuencias de octetos forman tokens válidos?*

# Definición del Lenguaje Fuente
## Definición Sintáctica

- **Gramáticas incontextuales como herramientas descriptivas de la sintaxis básica del lenguaje fuente.**
- **Formalmente, una gramática incontextual G es una cuádrupla**

  **$G \equiv <T, N, a, P>$ donde:**
  - **T es un alfabeto de terminales (los *tokens*).**
  - **N es un alfabeto de no terminales (las *categorías sintácticas*).**
  - **$a \in N$ es no terminal distinguido denominado *axioma*.**
  - **P es un conjunto de *reglas de producción*.**
- **Cada regla de producción es, a su vez, un par ($n$, $\alpha$) donde $n \in N$ y $\alpha \in (T \cup N)^*$. Se representan como $n \rightarrow \alpha$, o bien como $n ::= \alpha$ (notación BNF)**

***Ejercicio:***

***- La notación BNF es un metalenguaje ¿qué tipo de lenguaje es? ¿puedes definirlo formalmente?***

***- Lo mismo para la notación BNF extendida (EBNF)***



# Definición del Lenguaje Fuente
## Definición Sintáctica

- **Las cadenas de $(T \cup N)^*$ se denominan *formas sentenciales*. Las cadenas $T^*$ se denominan *sentencias* (o *frases)*.**
- **Sea $G \equiv <T,N,S,P>$ una gramática incontextual. G introduce en $(T \cup N)^*$ una *relación de derivación inmediata* $\Rightarrow_G$ como sigue: $\alpha \Rightarrow_G \beta$ sii:**
  - **$\alpha$ es de la forma $\alpha_o\, n\, \alpha_1$ .**
  - **$\beta$ es de la forma $\alpha_o\, \beta_1\, \alpha_1$.**
  - **Hay una producción $n \rightarrow \beta_1$ en P.**
- **El cierre reflexivo-transitivo $\Rightarrow^*_G$ de $\Rightarrow_G$ se denomina *relación de derivación*.**
- **Si $\alpha \Rightarrow_G^* \beta$ se dice que $\beta$ es *derivable* de $\alpha$.**

# Definición del Lenguaje Fuente
## Definición Sintáctica

- **El *lenguaje generado por G,* L(G) se define como sigue:**
  **$L(G) = \{\alpha \mid \alpha \in T^* \text{ y a } \Rightarrow^*_G \alpha\}$**
- **Es decir, es el conjunto formado por todas las posibles sentencias derivables del axioma.**
- **Si $\alpha \Rightarrow_G^* \beta$, entonces hay $\alpha_0, \ldots, \alpha_n$ tal que $\alpha \Rightarrow_G \alpha_0$, $\alpha_0 \Rightarrow_G \alpha_1$, …, $\alpha_n \Rightarrow_G \beta$. Esto puede escribirse de forma más compacta como $\alpha \Rightarrow_G \alpha_0 \Rightarrow_G \alpha_1 \Rightarrow_G \ldots \Rightarrow_G \beta$. Este tipo de cadenas se denominan *derivaciones* de $\beta$ desde $\alpha$.**
- **En general, si $\alpha \Rightarrow_G^* \beta$, hay muchas derivaciones de $\beta$ desde $\alpha$.**
- **De ellas interesan dos: las derivaciones *más a la izquierda* y las *más a la derecha* (el *n* en $\alpha_0\, n\, \alpha_1$ es el más a la izquierda –más a la derecha- de todos los posibles).**



# Definición del Lenguaje Fuente
## Definición Sintáctica

- **Aparte del concepto de derivación, que puede resultar útil para comprobar si una sentencia es, o no, sintácticamente válida, es interesante disponer también de un mecanismo que explicite la estructura sintáctica de las sentencias de L(G).**
- **Este mecanismo viene dado por el concepto de *árbol de derivación* (o *árbol de análisis sintáctico*).**
- **Como ejercicio se recomienda encontrar ejemplos de gramáticas regulares, incontextuales, normalizadas y no normalizadas , y comprobar la estructura asociada a la derivación de las frases del lenguaje.**

# Definición del Lenguaje Fuente
## Definición Sintáctica

- **Árboles de derivación de G ≡ <T, N, a, P>. Son árboles en los cuáles:**
  - **Los nodos están etiquetados por elementos de T∪N∪{λ}.**
  - **Los hijos de cada nodo están *ordenados* (es decir, tiene sentido hablar del hijo *primero*, del hijo *segundo*, etc. de un nodo dado).**
- **Más concretamente:**
  - **El árbol formado por un único nodo etiquetado por *a* es un árbol de derivación de G.**
  - **Sea *t* un árbol de derivación de G. Sea *h* una hoja de *t* etiquetada por *n* ∈ N. Sea *n* → α una producción de P. Entonces, es posible obtener un nuevo árbol de derivación *t'* como sigue:**
    - **Si α es λ, se añade a *h* un hijo etiquetado por λ.**
    - **En otro caso, se añade a *h* |α| hijos, y se etiquetan, por orden, con los símbolos de α.**

***discusión:***
***- Árboles de derivación, estrategias de análisis y derivaciones DMI, DMD***



# Definición del Lenguaje Fuente
## Definición Sintáctica

***Ejercicios:***
***1.- dada la gramática:***

```
S->ABC
A->a|Aa
B->b|Bb
C->c|Cc
```

***analizar, es decir, obtener el árbol de derivación de frases del leguaje p.e. "aaaabbccc"***

***a) utilizar una estrategia de análisis descendente por la izquierda, es decir, empezando por el axioma tratamos de reconocer el primer símbolo y siguientes de la frase***

***b) utilizar una estrategia de análisis ascendente por la izquierda, es decir, empezando por el primer terminal y siguientes tratamos de reducir la frase hasta el axioma***

***c) comprobar que en el caso a) el proceso de construcción del árbol equivale a construir la DMI y que en el caso b) el proceso de reducción equivale a construir la DMD en sentido inverso***

***d) ¿se te ocurre como transformar la gramática anterior en una gramática equivalente que permita un análisis descendente más simple?***

***2.- utilizando la gramática de las expresiones regulares escribe las DMI, DMD de las siguientes frases :***

***aa(b|c|d)*bc , a*(bb | cc)d****

# Definición del Lenguaje Fuente
## Definición Sintáctica

- **Los árboles de derivación cuyas hojas son todas, bien λ, bien terminales, proporcionan la estructura de una sentencia de L(G): la que se obtiene eliminando las λ y concatenando, en orden, el resto de las hojas.**
- **Para cada sentencia de L(G) habrá, al menos, un árbol de derivación…**
- **… pero puede haber más de uno!**
- **En caso de que una misma sentencia tenga más de un árbol de derivación, G se dice que es una *gramática ambigua***
- **En la descripción de lenguajes de programación, la ambiguedad sintáctica es, generalmente, un fenómeno a evitar.**

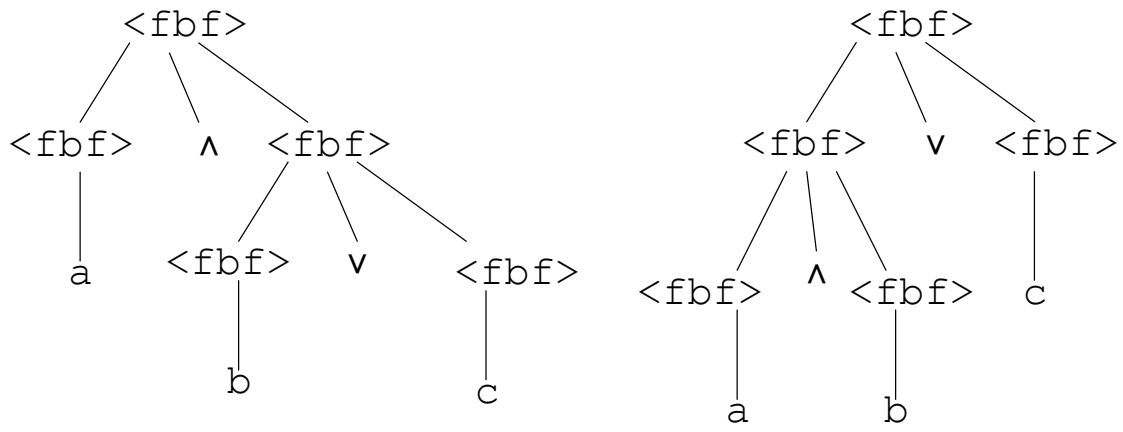


# Definición del Lenguaje Fuente
## Definición Sintáctica

Terminales: a,b,∧,∨,¬
No terminales: <fbf>
Axioma: <fbf>
Producciones:
<fbf> → a
<fbf> → b
<fbf> → ¬ <fbf>
<fbf> → <fbf> ∧ <fbf>
<fbf> → <fbf> ∨ <fbf>

- *¿qué tipo de gramática es?*
- *construir el árbol de derivación de a ∧ b ∨ c*

# Definición del Lenguaje Fuente
## Definición Sintáctica



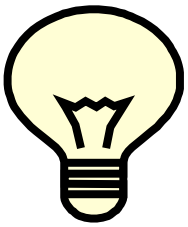

- **La gramática es ambigua!.**



# Definición del Lenguaje Fuente
## Definición Sintáctica

- **La ambiguedad en la gramática se traduce, a su vez, en una ambiguedad en la definición del lenguaje ⇒ hay alguna característica del lenguaje que no se está especificando.**
- **En este caso, ¿cuál es esa característica?.**
- **Una solución: Dada una gramática ambigua G, encontrar una gramática G' no ambigua equivalente (en el sentido de que L(G) = L(G')).**
- **… pero no vale cualquiera! ¿por qué?**

# Definición del Lenguaje Fuente
## Definición Sintáctica



En la definición de los lenguajes de programación importa la estructura (la forma de los árboles de derivación) y no únicamente el conjunto de cadenas derivables.

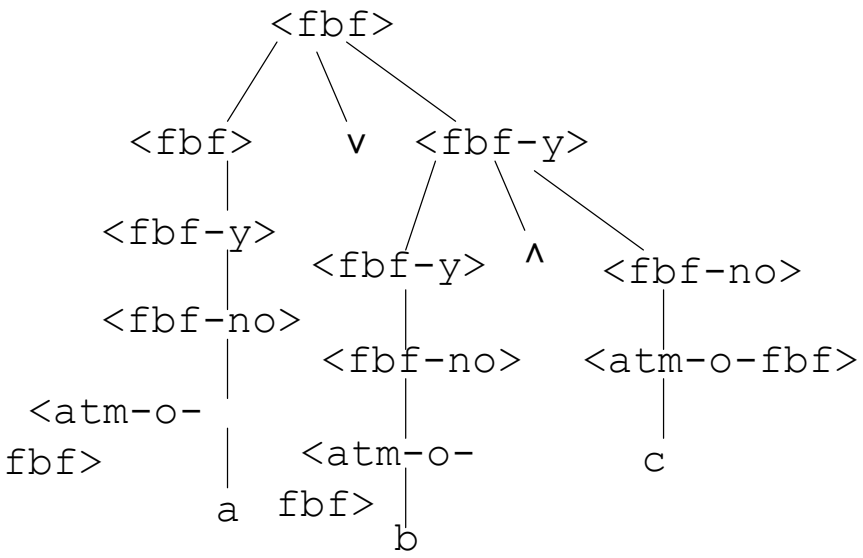


# Definición del Lenguaje Fuente
## Definición Sintáctica

```
<fbf> → <fbf> ∨ <fbf-y>
<fbf> →  <fbf-y>
<fbf-y> → <fbf-y> ∧ <fbf-no>
<fbf-y> → <fbf-no>
<fbf-no> → ¬ <fbf-no>
<fbf-no> → <atm-o-fbf>
<atm-o-fbf>→ a
<atm-o-fbf> → b
<atm-o-fbf> → ( <fbf> )
```

- **Gramática no ambigua (*¿equivalente?*) a la anterior.**
- **Se está explicitando la prioridad y asociatividad de los operadores (¿cuál es?).**

# Definición del Lenguaje Fuente
## Definición Sintáctica






# Definición del Lenguaje Fuente
## Definición Sintáctica

- **Algunos convenios:**
  - **Normalmente bastará con listar las producciones, empleando una letra distinta (ejemplo,cursiva) para los no terminales, o bien encerrando éstos entre <…>.**
  - **El axioma será la parte izquierda de la primera producción.**
  - **Para ahorrar espacio, las producciones de una misma categoría sintáctica (aquellas cuya parte izquierda coincide) se agrupan en una línea separando sus partes derechas por |**

    **(en realidad parte de la notación EBNF).**

    $n \rightarrow \alpha_0 \mid \alpha_1 \mid \ldots \mid \alpha_k$

# Definición del Lenguaje Fuente
## Definición Sintáctica

- **Cuando la descripción solo está orientada al programador pueden permitirse algunas ambiguedades, que se clarifican en lenguaje natural (¿algún ejemplo?).**
- **También es usual utilizar una notación extendida, más compacta (EBNF completa), o bien notaciones gráficas (grafos sintácticos o diagramas de Conway ).**
- **En las descripciones orientadas a la construcción de procesadores se evitarán las ambiguedades (¿por qué?).**
- **… y se utilizará la notación BNF (¿por qué?).**



# Definición del Lenguaje Fuente
## Definición Sintáctica

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

- **Discusión:**
  - **¿Es aceptable esta gramática para el lenguaje ejemplo?**

# Definición del Lenguaje Fuente
## Definición Sintáctica

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

- ***¿Cómo son aquí las prioridades y asociatividades de los operadores?***
- ***¿esperamos resultados correctos con esta gramática?***



# Definición del Lenguaje Fuente
## Definición Sintáctica

***ejercicio:***

- ***escribir una gramática para las expresiones aritméticas que no sea ambigua, que por defecto dé la misma prioridad a todos los operadores (se pueden poner paréntesis para establecer prioridades) y que asocie por la izquierda***
- ***lo mismo pero que asocie por la derecha***
- ***lo mismo pero para las expresiones regulares***