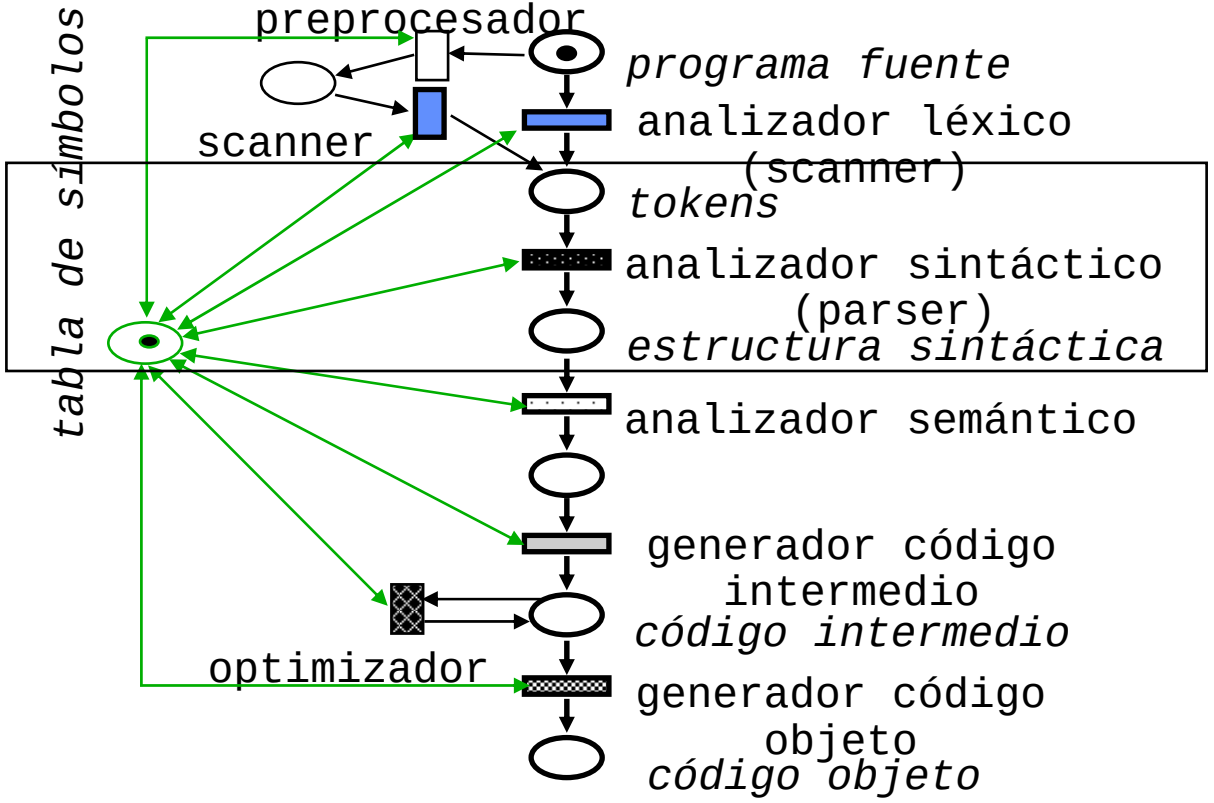
Lección 3: Fundamentos para el análisis sintáctico

1) Introducción
2) Gramáticas. Definiciones y clasificación
3) GLC. Notaciones
4) GLC. Árboles de análisis sintáctico
5) GLC. Derivación a dcha. y a izda.
6) GLC. Ambigüedad y transformación de gramáticas

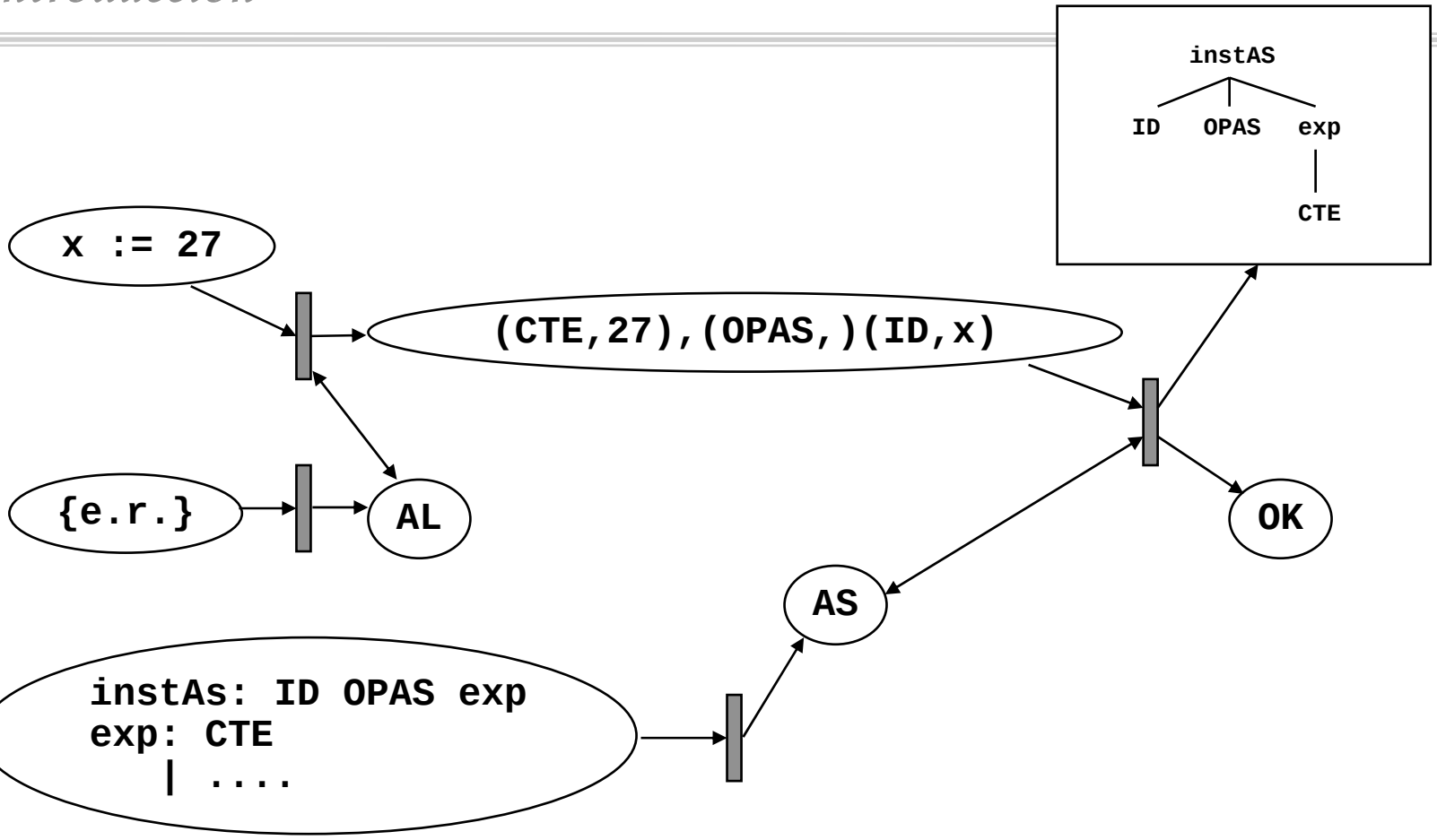
- Recordar, esquema general de un compilador:



- El analizador sintáctico:
  - Objetivo: agrupar los tokens suministrados por el scanner para reconocer "*frases*"
    - » determinar si es sintácticamente correcto
    - » establecer la estructura subyacente
  - La sintaxis se suele especificar formalmente mediante una GLC
  - El parser recibe tokens y los agrupa de acuerdo a

    `producciones`

    especificadas por la GLC
  - El parser detecta errores sintácticos


instAS
ID
OPAS
exp
CTE
x := 27
(CTE,27),(OPAS,)(ID,x)
{e.r.}
AL
OK
AS
instAs: ID OPAS exp
exp: CTE
| ....

- El estudio de gramáticas es anterior al de lenguajes de programación
  - Empezó con el estudio del lenguaje natural
- Punto de referencia: Noam Chomsky

gramática

Una gramática G=(N,T,S,P) es una 4-tupla donde:
1) N es el conjunto de No-Terminales
2) T es el de Terminales tal que N ∩ T = ∅
3) S∈N, y se denomina Símbolo Inicial
4) P es el conjunto de Producciones

$S \rightarrow A\ C\ b$

$C\ b \rightarrow A\ b$

$C\ b \rightarrow a\ b\ c$

$A \rightarrow a$

- El <u>símbolo inicial</u> es el único no terminal que se utiliza para generar todos los strings del lenguaje

- Un <u>terminal</u> genera un único string (él mismo)
- Una <u>producción</u> es una regla que establece una transformación de strings
- Forma de una producción (usando BNF) $\alpha \rightarrow \beta$
- <u>Significado</u>: si δ es un string, al aplicarle la producción, cualquier* aparición de α en δ se puede sustituir por β

*: En realidad, <u>algunas apariciones</u>

- Establezcamos dos operadores
- Sea G=(N,T,S,P) una gramática

$\alpha A \delta$ deriva directamente $\alpha\beta\delta$ en la gramática G

derivación directa

Sean $\alpha, \delta \in (N \cup T)^*$ **y sea** $A \rightarrow \beta \in P$. **Entonces**

$$\alpha A \delta \Rightarrow_G \alpha\beta\delta$$

derivación

α_0 deriva α_n en cero o más pasos en la gramática G

Sean $\alpha_0 \ldots \alpha_n \in (N \cup T)^*$ **t.q.**

$$\alpha_0 \Rightarrow_G \alpha_1 \Rightarrow_G \ldots \Rightarrow_G \alpha_n \text{ , } n>=0$$

Entonces

$$\alpha_1 \overset{*}{\Rightarrow}_G \alpha_n$$

$S \rightarrow A\ C\ b$

$C\ b \rightarrow A\ b$

$C\ b \rightarrow a\ b\ c$

$A \rightarrow a$

¿$S \overset{*}{\Rightarrow} a\ a\ b$?

- Algunas definiciones:

forma de frase

Cualquier string que se pueda derivar del símbolo inicial (string de símbolos gramaticales)

frase

Cualquier forma de frase con sólo elementos terminales

lenguaje definido por una gramática

Conjunto de todas las frases de la gramática

- Es fácil ver que:
  - los siguiente elementos pertenecen al lenguaje generado por la gramática:
    - » *el perro está cerca*
    - » *la perro está lejos*
    - » *el gata está cerca*
    - » .....
  - los siguiente elementos NO pertenecen al lenguaje generado por la gramática:
    - » *el perro*
    - » *la lejos perro está*
    - » .....
  - Por desgracia, no siempre será tan sencillo

- Ejercicio 1: Sea G=(N,T,S,P) con
  - N={*S*}
  - T={a,b}
  - P={*S*→aSb, *S*→ab}
  - S=*S*

$L(G)=\{a^n b^n \mid n>=1\}$

- Ejercicio 2: Sea G=(N,T,S,P) con
  - N={S,A,B}
  - T={a,b}
  - P={ *S*→aB, *S*→bA, *A*→a, *A*→aS, *A*→bAA, *B*→b, *B*→bS, *B*→aBB}
  - S=*S*

$L(G)=\{w \in T^* \mid |w|_a=|w|_b\}$

- Ejercicio 3: ¿Cuál es el lenguaje generado por la siguiente gramática?

S → [S
S → S1
S1 → [a]

- Hemos visto cómo la aplicación de producciones genera las frases del lenguaje
- Para compiladores, el proceso es en sentido contrario:
  - dado un string:
    - » ¿Pertenece al lenguaje?
    - » ¿Cuáles son las producciones aplicadas para derivarlo del símbolo inicial?
- Este proceso lo lleva a cabo el **analizador sintáctico** ("parser")

- Ejercicio 3: Sea G=(N,T,S,P) con
  - N = {*programa, bloque, insts, inst, opas, ident, const, punto*}
  - T = {PROGRAM, BEGIN, END, =, A, B, 1, 0, .}
  - S = *programa*

P=

programa* → *PROGRAM ident punto bloque
bloque* → *BEGIN insts END punto
insts* → *insts punto inst
insts* → *inst
inst* → *ident opas const
***opas* → =**
***ident* → A**
***ident* → B**
***const* → 1**
***const* → 0**
***punto* → .**

```
PROGRAM A.
BEGIN
   B=1.
   A=0
END.
```

```
PROGRAM A.
BEGIN
   B=1.
   A=0.
END.
```

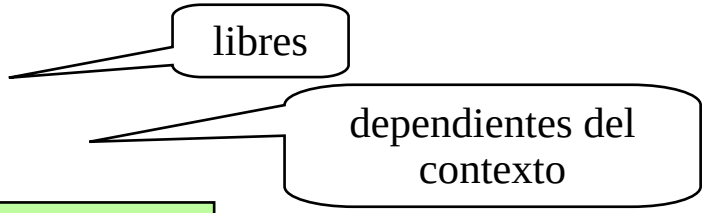
¿Sintácticamente correctos?

- Una producción "general" tiene la forma

$$\alpha \rightarrow \beta$$

- Imponiendo restricciones a las posibles formas de las producciones, se obtienen distintos tipos de gramáticas
  - restricciones respecto a qué pueden ser α y β
  - restricciones de dónde se puede aplicar la transformación establecida por la producción

- Clasificación de Chomsky
  - **gramáticas de Tipo 0**
  - **gramáticas de Tipo 1**



$$\alpha A \beta \rightarrow \alpha \delta \beta$$

  - » $\alpha,\beta,\delta \in (N \cup T)^*$
  - » δ no vacío y A un no terminal
  - » A se transforma en δ sólo cuando va precedido por α y seguido por β
  - » La producción ya no se aplica siempre, sino sólo en determinados contextos
  - » Muchos lenguajes de prog. tienen aspectos que requieren una gramática de este tipo
    - demasiado complicadas de manejar, por lo que se usan del tipo siguiente, con "algunos apaños"

- **gramáticas de Tipo 2**

libres de contexto

$$A \rightarrow \beta$$

  - » A es un (único) no terminal
  - » Cada vez que aparece A, se puede sustituir por β
  - » Se corresponde con la notación BNF
  - » Pascal (en la mayoría de sus aspectos) es una gramática de tipo 2

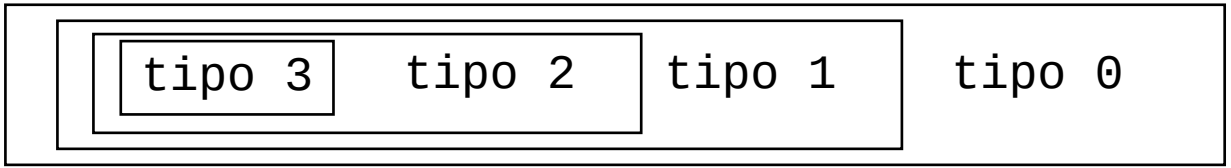
- **gramáticas de Tipo 3**

$$A \rightarrow w \qquad A \rightarrow wB$$

regulares

  - » A ,B son no terminales
  - » w es una secuencia de terminales
  - » poco potentes para lenguajes
  - » fundamentales para comp. léxicos
  - » expresiones regulares

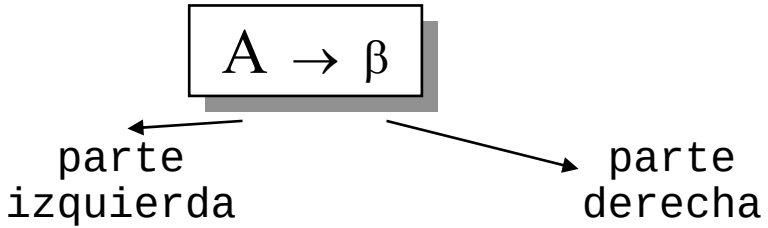
- En forma de diagrama:



- Cada gramática de un tipo es también del tipo anterior
- Cuanto menos restrictiva es una gramática, más complejo es su análisis
- Las más sencillas son las de Tipo 3, que pueden ser reconocidas por autómatas de estados finitos (recordar análisis léxico)

- Por claridad, para GLC usaremos los siguiente convenios:
  - consideraremos terminales:
    - » primeras minúsculas del abecedario
    - » símbolos de operación y puntuación
    - » dígitos
    - » palabras en negrita: **perro**,**begin**
  - consideraremos NO terminales:
    - » primeras mayúsculas del abecedario
    - » palabras en cursiva: *sujeto*,*expresión*
    - » la S, suele representar el símbolo inicial
  - ...,X,Y,Z: representan símbolos gramaticales (terminal, No terminal)
  - letras minúsculas de "en medio" (u,v,..) representan cadenas de terminales

- letras griegas minúsculas representan **formas de frase** (strings de símbolos gramaticales)
  - » Así, en una GLC, una producción se escribe siempre como



- varias producciones con igual parte izda. se pueden agrupar en una producción con alternativas

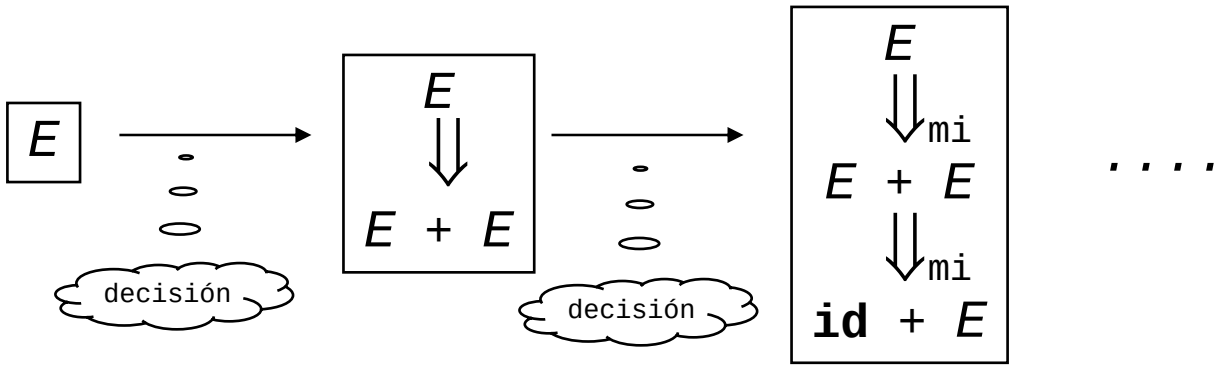
$$\begin{array}{l} A \rightarrow \alpha_1 \\ A \rightarrow \alpha_2 \\ \quad \ldots \\ A \rightarrow \alpha_n \end{array} \quad = \quad A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$$

- Nuestro problema
  - Dada una gramática G y un **w∈T***, determinar si **w∈L(G)**
- Para ello, trataremos de construir un ***árbol de sintaxis***
  - teniendo presente que queremos un proceso automático
- Características:
  - el **nodo raíz** se etiqueta con el símbolo inicial
  - cada **nodo no hoja** se etiqueta con un no terminal
  - los hijos de un nodo son los **símbolos** (de izda. a dcha.) que aparecen en una de las **producciones** que tienen dicho nodo como parte izda.
  - cuando se ha derivado una frase, las hojas del árbol son terminales

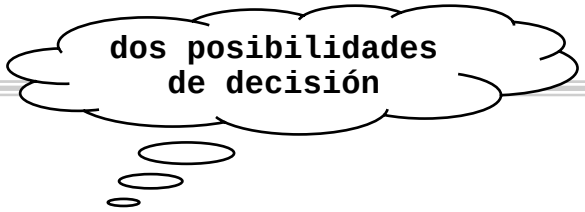
- Ejemplo:

G: $E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$

W: $id+id*id$



¡podemos mirar el siguiente terminal que viene: token!



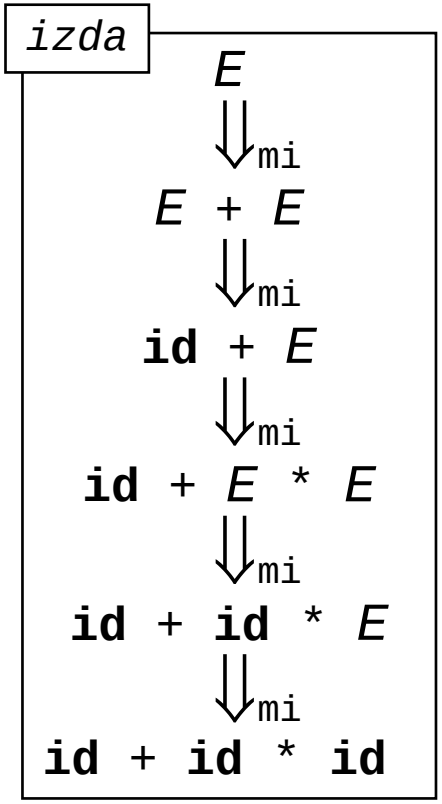
- En cada paso en una derivación:
  - hay que elegir qué no terminal sustituir
  - elegido uno, optar por una de las posibles producciones que lo tengan como parte izda.
- Si siempre se sustituye el de más a la izda.

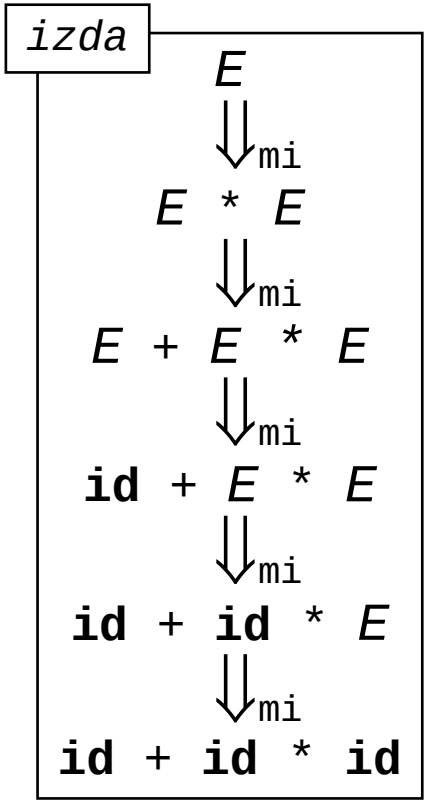
derivación por la izda.

- Si siempre se sustituye el de más a la dcha.

derivación por la dcha.

- Aun optando por una derivación m_i ó m_d, tomar decisiones puede generar problemas
- ¿Cuál de las dos opciones nos gusta más?





- Una gramática es ambigua si existe al menos una frase ambígua
- Una frase es ambígua si existe más de un árbol para ella (`mi` ó `md`)
  - produce indeterminismo
  - es importante eliminarla, cuando se pueda
- A veces, es posible eliminar la ambigüedad

transformar la gramática en una equivalente que no sea ambigua

G'

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid -E \mid id$$

- Hay lenguajes inherentemente ambiguos

- Ejemplo: Considerar la instrucción "**if**" en Pascal
- Si la gramática es de la forma

```
inst → if exp then inst
     | if exp then inst else inst
     | otras instrucciones
```

```
if exp1 then if exp2 then inst2 else inst3
```

1

```
if exp1 then
   if exp2 then
        inst2
     else
        inst3
```

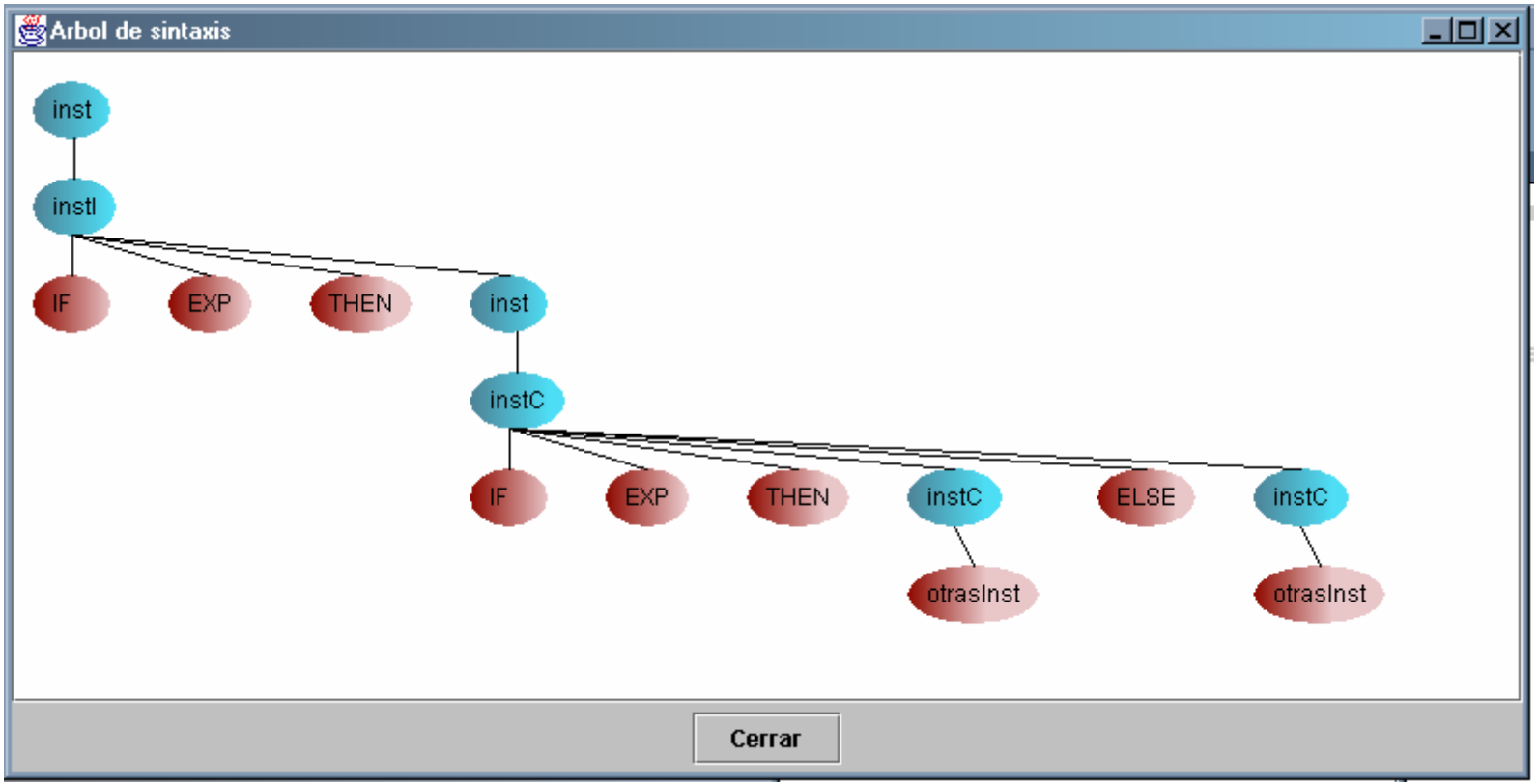
2

```
if exp1 then
   if exp2 then
      inst2
else
   inst3
```

- ¿Con cuál nos quedamos?
- Generalmente, se aplica la misma regla que aplica Pascal:
  - cada else se empareja con el then más próximo
  - es decir, la buena versión es la 1
- ¿Se puede expresar esta regla en una gramática transformada?
  - en este caso, sí
  - no siempre será posible

```
inst → instC | instI
instC → if exp then instC else instC
      | otras instrucciones
instI → if exp then inst
      | if exp then instC else instI
```

GLC. Ambigüedad



- Ejercicio: "Convencerse" de que genera el mismo lenguaje que la anterior y se ha eliminado la ambigüedad
- ¿Es posible saber si una gramática dada es ambigua?

  ¡¡ NO !! (Hopcroft y Ullman)

- Una cuestión interesante: la simplificación de gramáticas
  - puesto que puede haber varias gramáticas equivalentes, busquemos alguna más simple
  - posibles simplificaciones:
    - » eliminación de no terminales inútiles
    - » eliminación de producciones-ε
    - » eliminación de producciones unitarias
    - » otras

- Ejemplo: Considerar la gramática

S	→	***A***	|	***B***
A	→	**a**		
B	→	***B***	**b**	
C	→	**c**		

- Verifica que:
  - el no terminal C no es alcanzable desde S
  - el no terminal B no deriva ningún string terminal
- Los no terminales B y C se denominan inútiles
- No terminales inútiles
  - pueden ser eliminados
  - se obtiene una gramática equivalente

- Un no terminal X es **útil** si existe

$$S \Rightarrow_* \alpha \, \mathbf{X} \, \beta \Rightarrow_* \mathbf{w}$$

para algún α,β y tal que $w \in T^*$

- Dos condiciones necesarias de "utilidad"
  1) debe haber alguna cadena de terminales que sea derivable de X
  2) X debe aparecer en alguna forma de frase derivable desde S
- Proceso:
  1) eliminar los no terminales que no deriven ninguna frase
  2) eliminar los no terminales que no sean alcanzables desde S

- Un no terminal es "terminable" cuando es capaz de derivar alguna frase

¿$G' \cong G$?
¿Termina?
¿Correcto?
¿Por qué $L(G)<>\emptyset$ en la Pre?
¿Coste?

```
Algoritmo  eliminaNoTerminables
                    (E G:GLC; S: G':GLC)
--Pre: G=(N,T,P,S) t.q. L(G)<>∅
--Post: G'=(N',T,P',S) ∧ G' ≅ G ∧
--      ∀X'∈N'.∃w∈T*.X'⇒* w
Vars viejo,nuevo: conj. de no terminales
Principio
  viejo:={}
  nuevo:={A∈N|A→w∈P,w∈T*}
  Mq viejo<>nuevo
    viejo:=nuevo
    nuevo:=viejo ∪
              {B∈N|B→α∈P,α∈(T ∪ viejo)*}
  FMq
  N':=nuevo
  P':={A→w∈P|A∈N', w∈(N' ∪ T)*}
Fin
```

- Segundo paso: eliminar los símbolos que no sean accesibles desde el símbolo inicial

```
Algoritmo  eliminaNoAccesibles
                    (E G:GLC; S: G':GLC)
--Pre: G=(N,T,P,S) t.q. L(G)<>∅ ∧
--    ∀X∈N.X es terminable
--Post: G'=(N',T',P',S) ∧ G' ≅ G ∧
--      ∀X∈(N'∪T').∃α,β∈(N'∪T')*.
--      S ⇒* αXβ
Vars viejo,nuevo: conj. símbolos gram.
Principio
  viejo:={S}
  nuevo:={X∈(N ∪ T)|S→ αXβ ∈ P}∪{S}
  Mq viejo<>nuevo
    viejo:=nuevo
    nuevo:=viejo ∪
           {Y∈(N ∪ T)|A→ αYβ ∈P,A∈viejo}
  FMq
  <N',T'>:=<nuevo ∩ N, nuevo ∩ T>
  P':={A→ w ∈ P|A∈N',w∈(N'∪T')*}
Fin
```

- Un no terminal X se dice "**anulable**" cuando $X \Rightarrow_* \varepsilon$
- Transformación interesante:

  obtener una gramática equivalente a otra "sin ε"

- Una gramática es "sin ε" ssi:

  1) no hay ninguna regla $X \rightarrow \varepsilon$

  2) como mucho, hay una producción $S \rightarrow \varepsilon$, pero entonces S no aparece en la parte derecha de ninguna otra producción

- Otra transformación interesante: eliminación de producciones unitarias
  - una producción de la forma $A \rightarrow B$ se dice **unitaria**
- Conclusión: desarrollo de **formas normales**
  - formas normales de Chomsky y de Greibach

GLC. Comparación de gramáticas

- Asumamos una gramática para un lenguaje
- ¿Es correcta?
  - ¿"Expresa" el lenguaje que queríamos expresar?
- Notar que la gramática es la propia definición del lenguaje
- Formas de verificación:
  - hacer "algunos" tests para ver resultados
  - comparar la equivalencia con una gramática "correcta"
  - NO existe un algoritmo general para GLC