IC-5701
Compiladores e Intérpretes

Profesor:
Ing. Allan Rodríguez Dávila, MGP

Agenda

TEC

- Proceso de Análisis Sintáctico
- Gramáticas Libres de Contexto
- Árboles Sintácticos

Autómatas Finitos

Autómatas Finitos

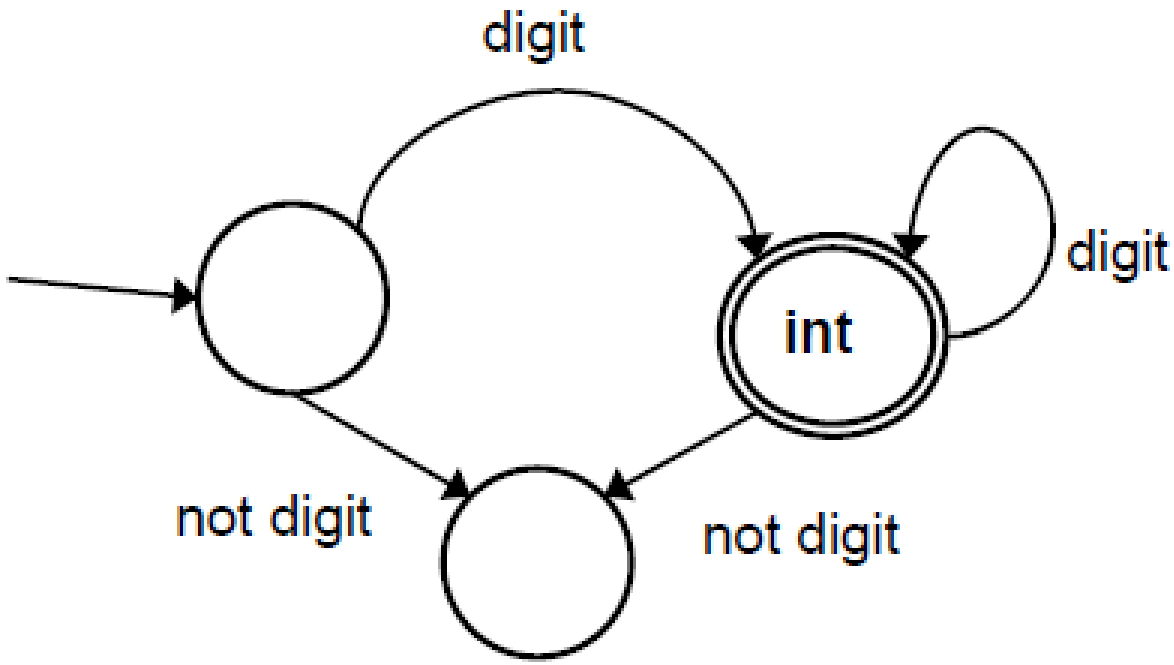
- Grafos de transición de estados.
- Son reconocedores.
- Son capaces de reconocer lenguajes regulares.
  - Reconocen tokens

Autómatas Finitos

- Todo lenguaje definido por un autómata finito es también definido por una expresión regular
- Todo lenguaje definido por una expresión regular es también definido por un autómata finito
- Son utilizados para construir analizadores léxicos

TEC

Autómatas Finitos



Clasificación

- Autómatas Finitos No Deterministas
  - No tienen restricciones en etiquetas
- Autómatas Finitos Deterministas
  - Para cada estado tienen una única arista con el símbolo que sale del estado.

De Regex a AFN

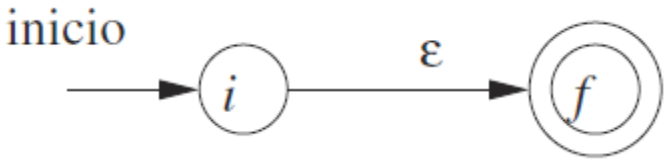
- Algoritmo conocido McNaughton-Yamada-Thompson.
- Entrada: Una expresión regular r sobre el alfabeto Σ
- Salida: Un AFN N que acepta a $L(r)$.
- Se basa en un análisis sintáctico de la expresión regular.

De Regex a AFN

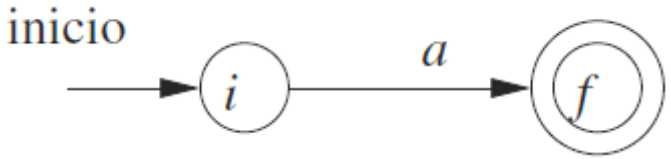
- Para cada subexpresión, el algoritmo construye un AFN con un solo estado aceptante
- Aplica reglas simples para subexpresiones sin operadores.
- Aplica reglas inductivas para subexpresiones largas

De Regex a AFN

- Expresión ε

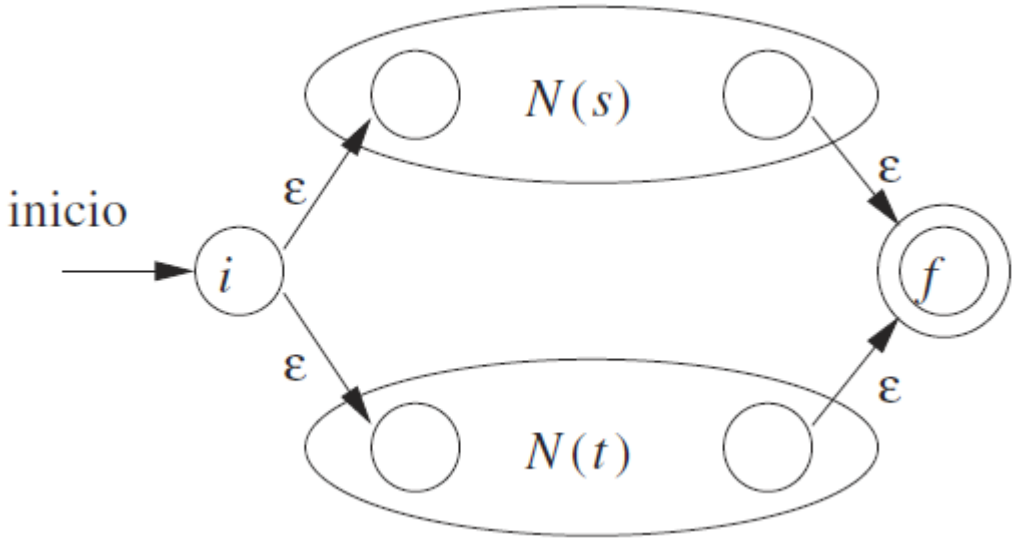


- Expresión a



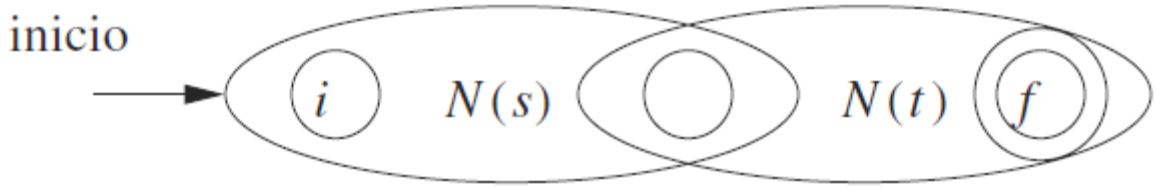
De Regex a AFN

- Reglas de Inducción
  - Expresión `r = s|t`



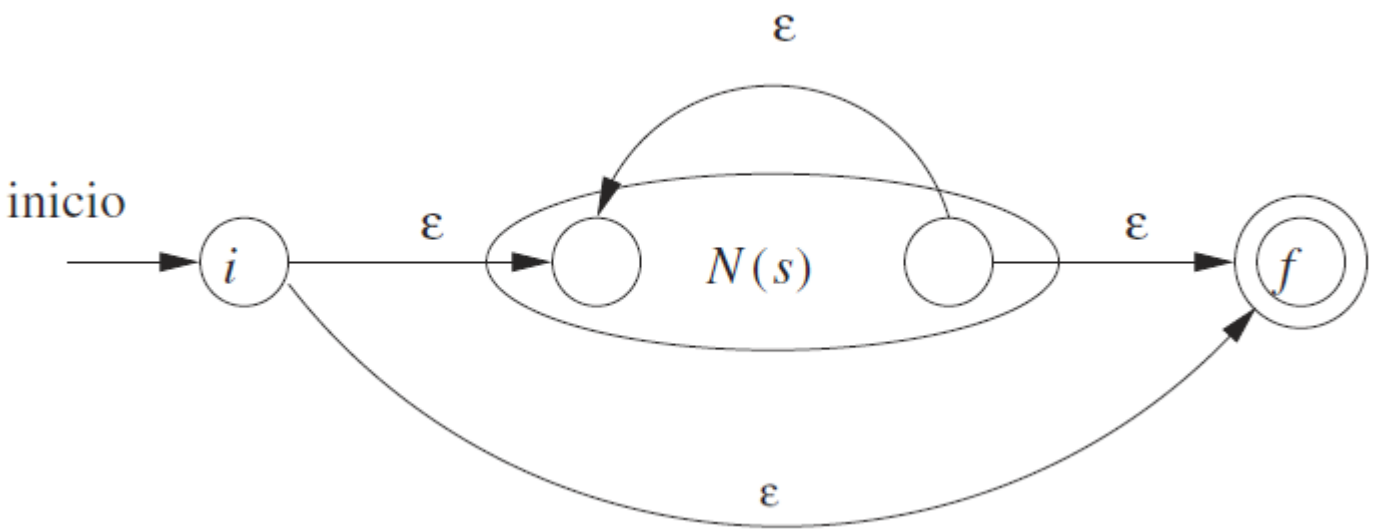
De Regex a AFN

- Reglas de Inducción
  - Expresión `r = st`



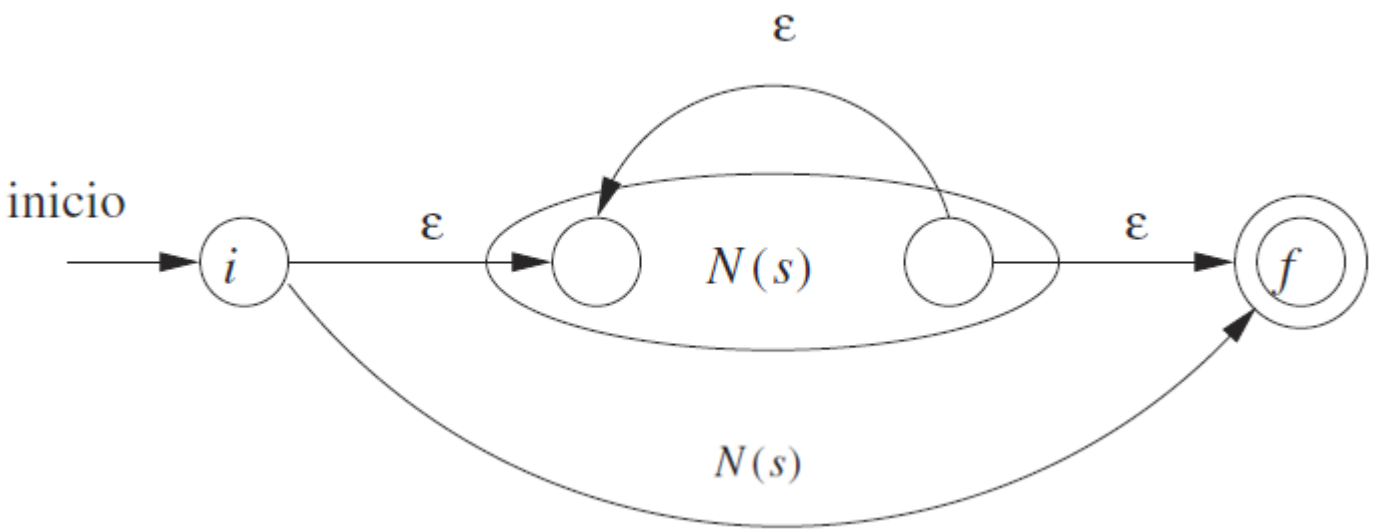
De Regex a AFN

- Reglas de Inducción
  - Expresión `r = s*`



De Regex a AFN

- Reglas de Inducción
  - Expresión $r = s^+$



Ejemplo

- Desarrolle los AFN de los lenguajes denotados por las siguientes expresiones regulares:

a?b((ac)+|ad)*d

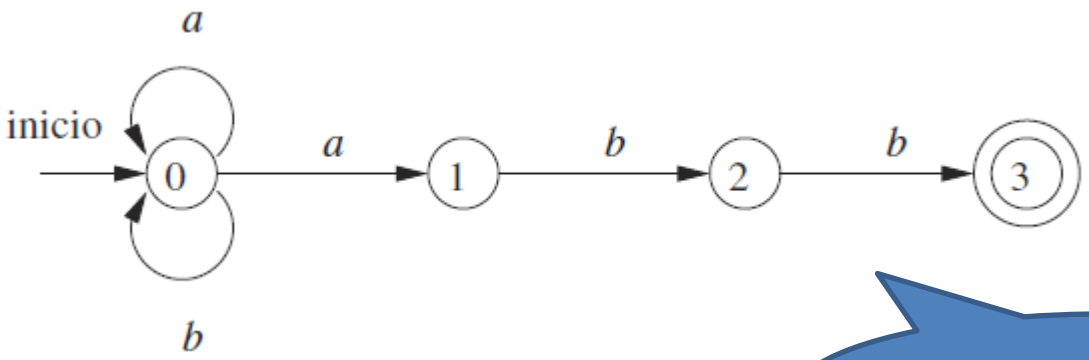
Autómatas Finitos Deterministas

Autómatas Finitos **Deterministas**

- Es un caso especial de AFN.
- Para cada estado ***s*** y cada símbolo de entrada ***a***, hay exactamente una línea (arista) que surge de ***s*** con etiqueta ***a***
- No hay movimientos en la entrada **ε**

Autómatas Finitos Deterministas

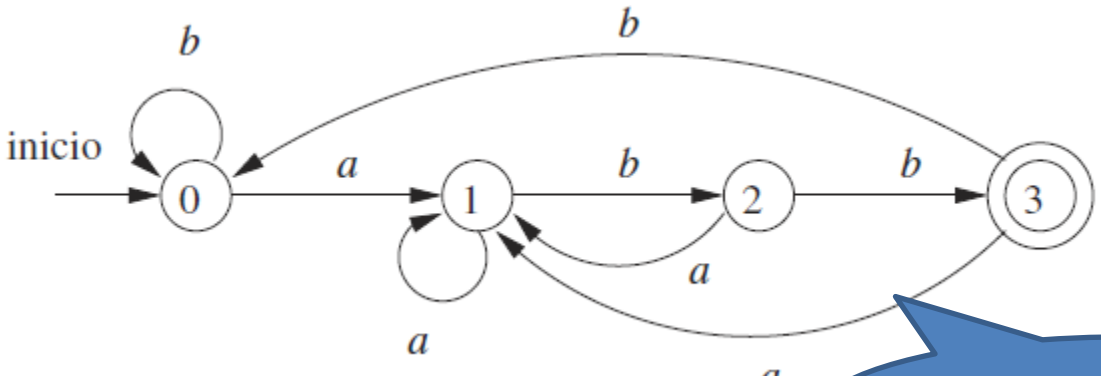
- Grafo de transición para un AFN que reconoce el lenguaje de la expresión regular `(a|b)*abb`.



AFN

Autómatas Finitos Deterministas

- Grafo de transición para un AFD que reconoce el lenguaje de la expresión regular `(a|b)*abb`.



De AFN a AFD

- Se utiliza el algoritmo de construcción de subconjuntos *T*.
- La entrada es el AFN.
- La salida es un AFD que acepta el mismo lenguaje que el AFN.

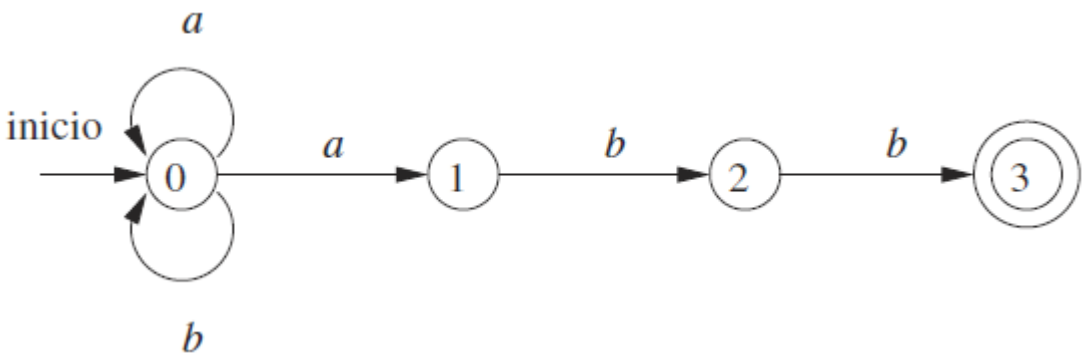
De AFN a AFD

- Se construye una tabla Dtran para el AFD.
- Cada estado en Dtran es un subconjunto de estados del AFN que se alcanzan desde otro estado Dtran,por medio de la función de transición, iniciando en $\{S_0\}$.

  `mover(T,a)`

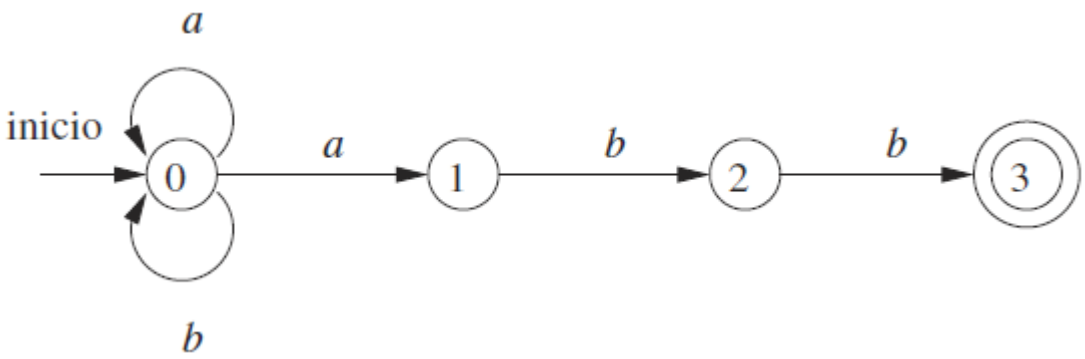
- Los estados de aceptación del AFD son aquellos conjuntos que contienen un estado de aceptación del AFN

De AFN a AFD



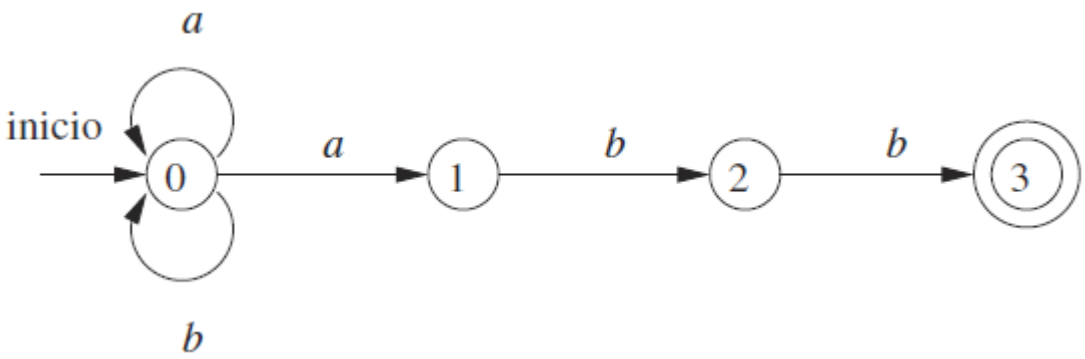
Estado AFN	Estado AFD	a	b
{0}	A	{0,1}	{0}

De AFN a AFD



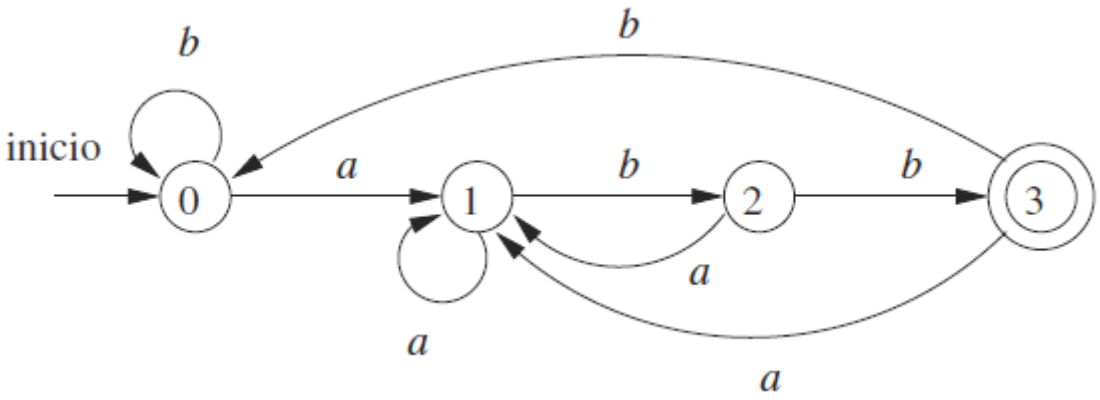
Estado AFN	Estado AFD	a	b
{0}	A	{0,1}	{0}
{0,1}	B	{0,1}	{0,2}

De AFN a AFD



Estado AFN	Estado AFD	a	b
{0}	A	{0,1}	{0}
{0,1}	B	{0,1}	{0,2}
{0,2}	C	{0,1}	{0,3}
{0,3}	D	{0,1}	{0}

De AFN a AFD

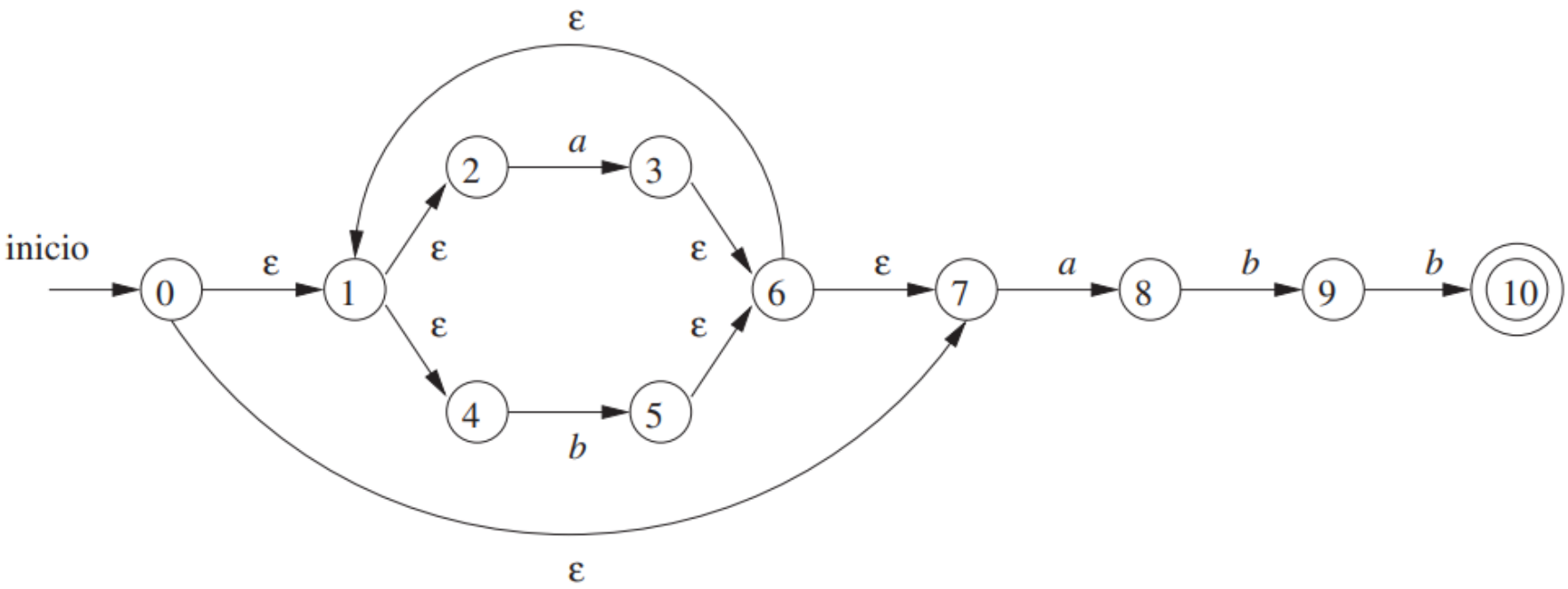


Estado AFN	Estado AFD	a	b
{0}	A	{0,1}	{0}
{0,1}	B	{0,1}	{0,2}
{0,2}	C	{0,1}	{0,3}
{0,3}	D	{0,1}	{0}

De AFN (ε) a AFD

- Se utiliza el algoritmo de construcción de subconjuntos *T*.
  - Con un manejo especial de ε.
- El estado inicial T_0 es ε-cerradura(S_0).
  - Para cada entrada a, la función de transición será:
    - ε-cerradura(mover(T, a))
- Se agregan como nuevos estos los subconjuntos de estados *T* resultantes de la función de transición

De AFN (ε) a AFD



Manejo de errores Léxicos

Manejo de errores

- Se detectan al intentar reconocer componentes léxicos y la entrada no se ajusta a lo esperado.
- Utilización de caracteres inválidos.
- Escribir mal.

Manejo de errores

- Nombres ilegales
  - Un identificador contiene caracteres inválidos
  - Empieza con un dígito
- Números incorrectos
  - Un número contiene caracteres o formado incorrectamente

Manejo de errores

- Errores en palabras reservadas
  - Caracteres omitidos, adicionales o cambiados
- Fin de archivo
  - Se llega a fin de archivo a mitad de un componente léxico

Recuperación de errores

- Eliminar un caracter del resto de la entrada.
- Insertar un caracter faltante en el resto de la entrada
- Sustituir un caracter por otro
- Intercambiar dos caracteres consecutivos

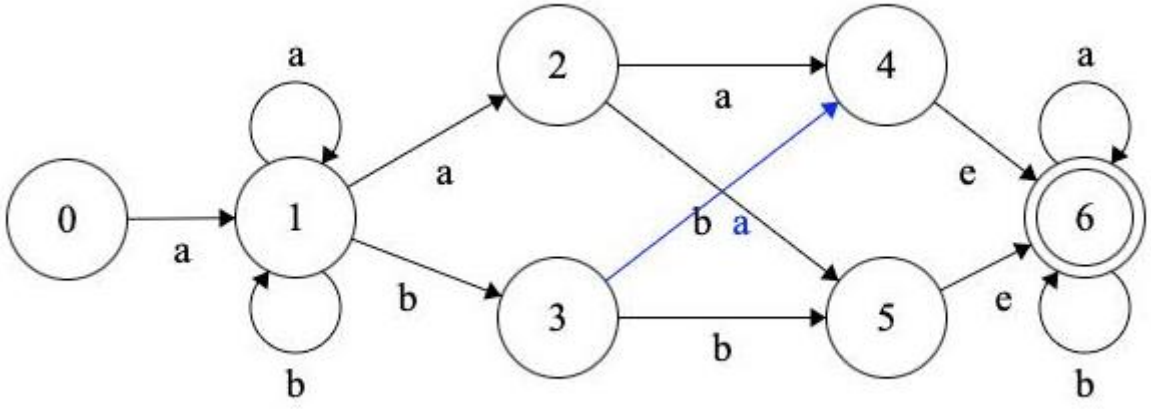
TEC

Recuperación de errores

- Al encontrar un error se debe:
  1. Manejar el error
  2. Reportarlo
  3. Continuar

Ejemplo AFN a AFD

- AFN del siguiente lenguaje denotado por la siguiente expresión regular:
  - a(a|b)*(a|b)(a|b)+



Portafolio 2, #3.1

- Desarrolle los **AFN** de los lenguajes denotados por las siguientes expresiones regulares:
  - a(a|bc+)*c
  - c*dc*dc*(c|d)c*

TEC

Portafolio 2, #3.2

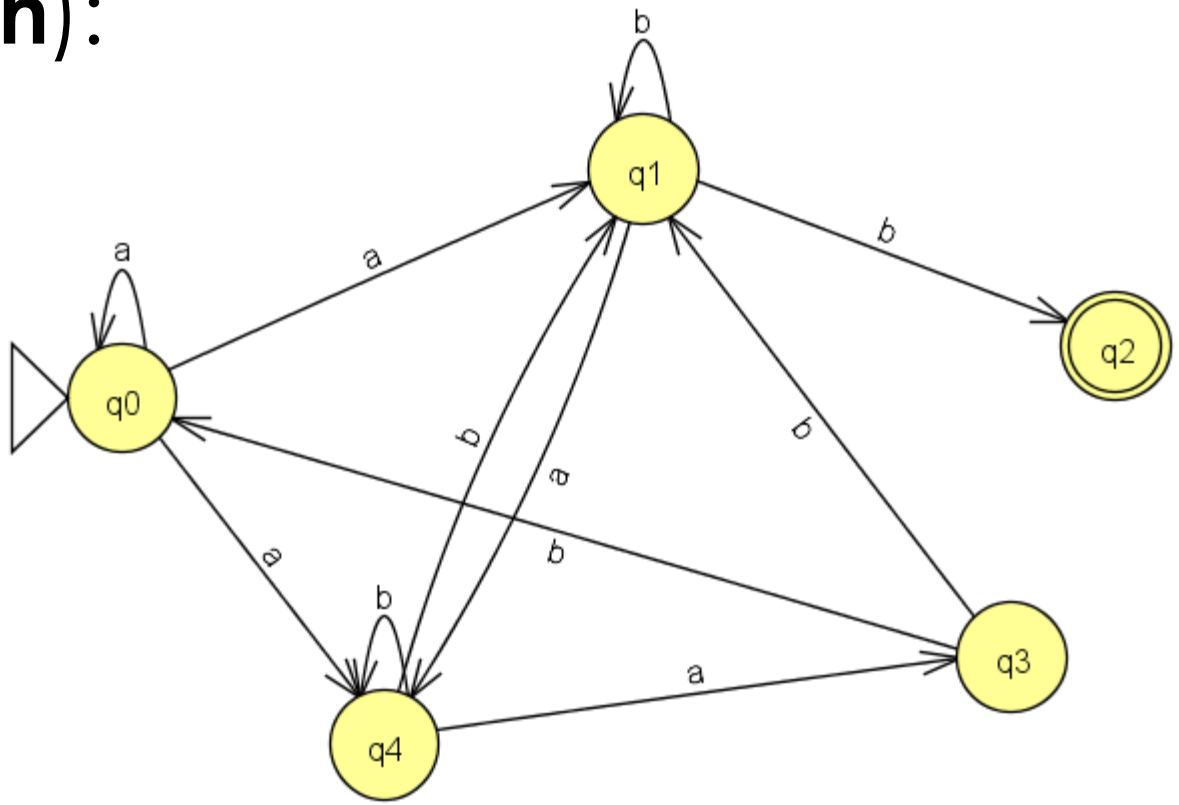
- Desarrolle las **tablas de transición** de los AFN de los lenguajes denotados por las siguientes expresiones regulares:
  - a(a|bc+)*c
  - c*dc*dc*(c|d)c*

Portafolio 2, #3.3

- Desarrolle los **AFD** de los lenguajes denotados por las siguientes expresiones regulares(con **tablas Dtran**):
  - a(a|bc+)*c
  - c*dc*dc*(c|d)c*

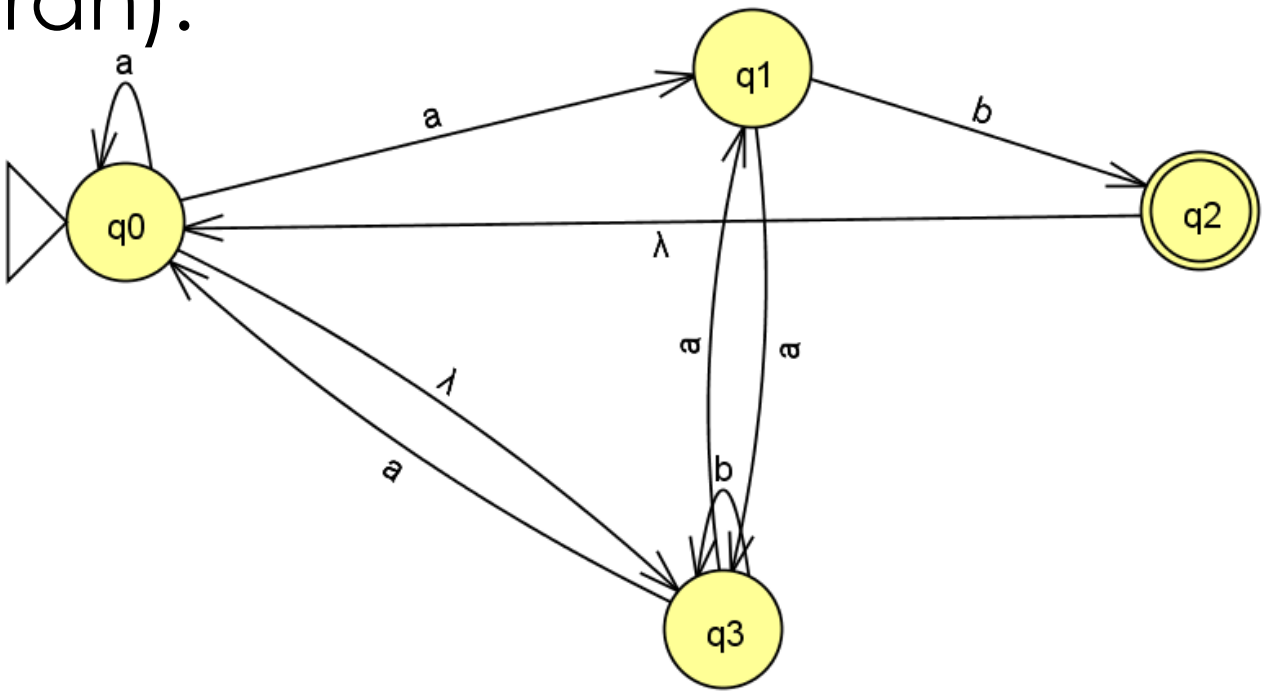
Bonus

- Desarrolle los **AFD** el siguiente AFN(con **tablas Dtran**):



Bonus

- Desarrolle los AFD el siguiente AFN(+ tablas Dtran):

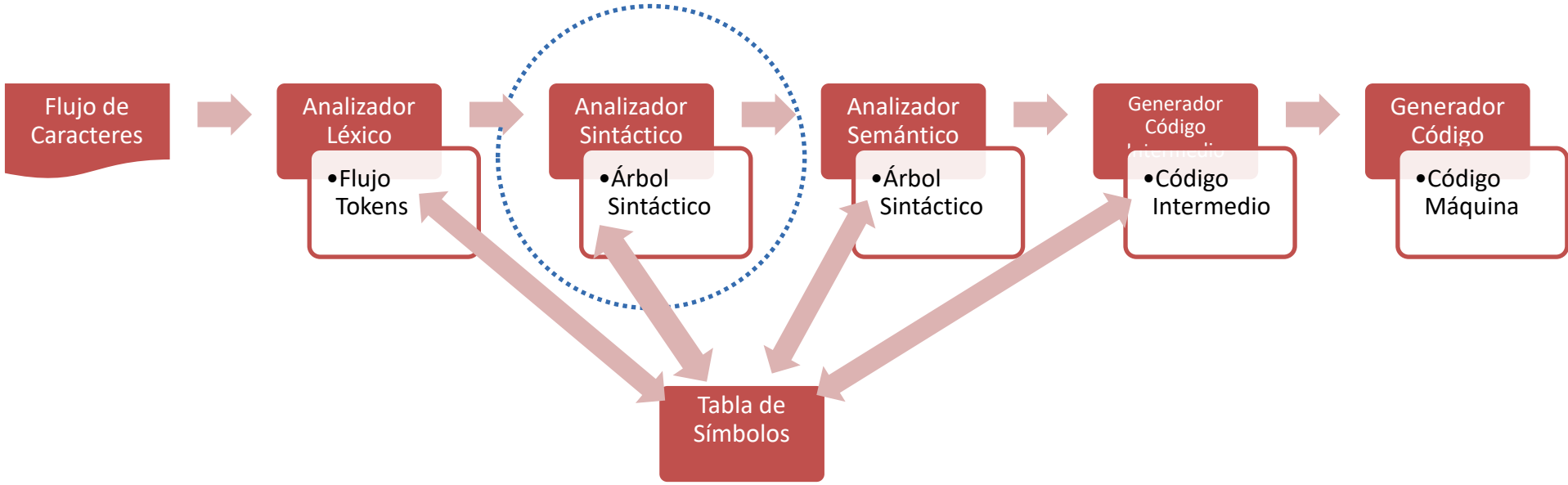


Análisis Sintáctico

Análisis Sintáctico

- Segunda fase del compilador
- Conocido como parser.
- Utiliza los tokens para crear una representación que describa la estructura gramatical
- Interactúa con la tabla de símbolos

Compilación



Análisis Sintáctico

- Determina cómo generar una cadena mediante una gramática.
- Complejidad máxima es $O(n^3)$.
- Toma una cadena de tokens y verifica que puede generarse por la gramática del lenguaje.

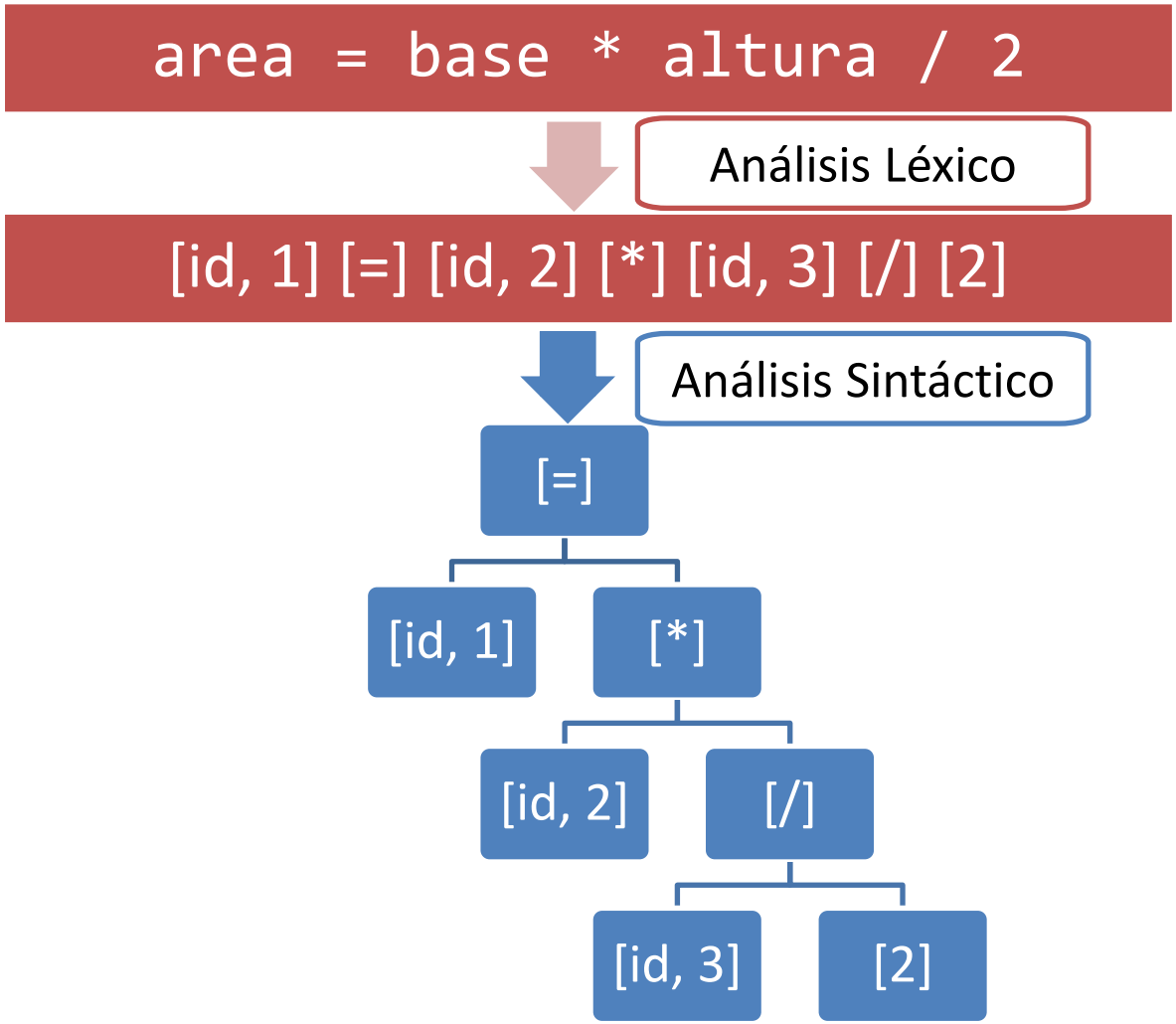
Gramáticas

```
programa -> bloque

bloque -> { decls instr }

….

instr -> if (expr) instr else instr
```

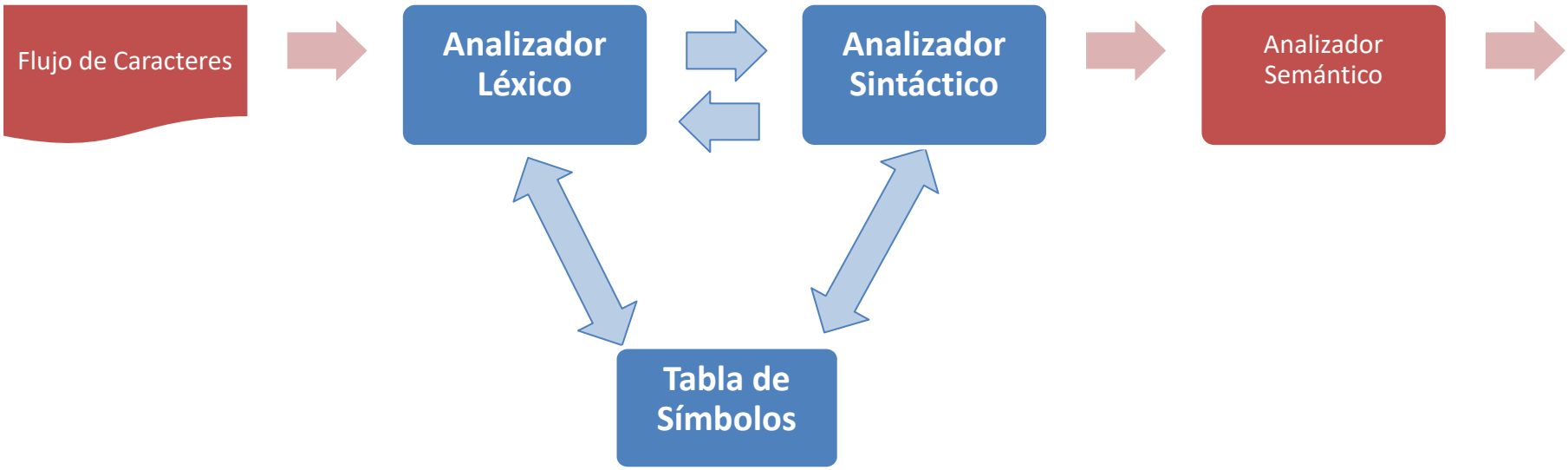
Análisis Sintáctico

- Se espera que el compilador ayude a localizar y rastrear los errores.
  - Formación incorrecta de instrucciones
- Utiliza un árbol sintáctico y lo pasa a las fases siguientes.
  - Puede o no construirse explícitamente

Análisis Sintáctico



Interacción



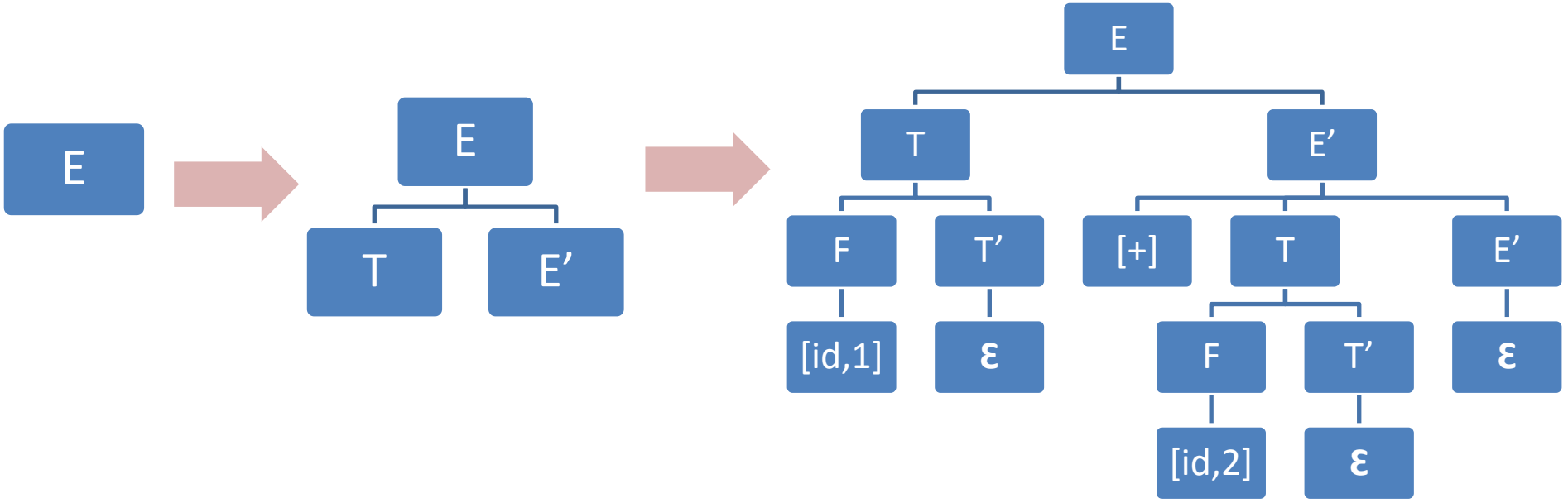
Análisis Sintáctico

- Tipos de analizadores:
  - Universales
  - Descendentes
  - Ascendentes

Análisis Sintáctico Descendente

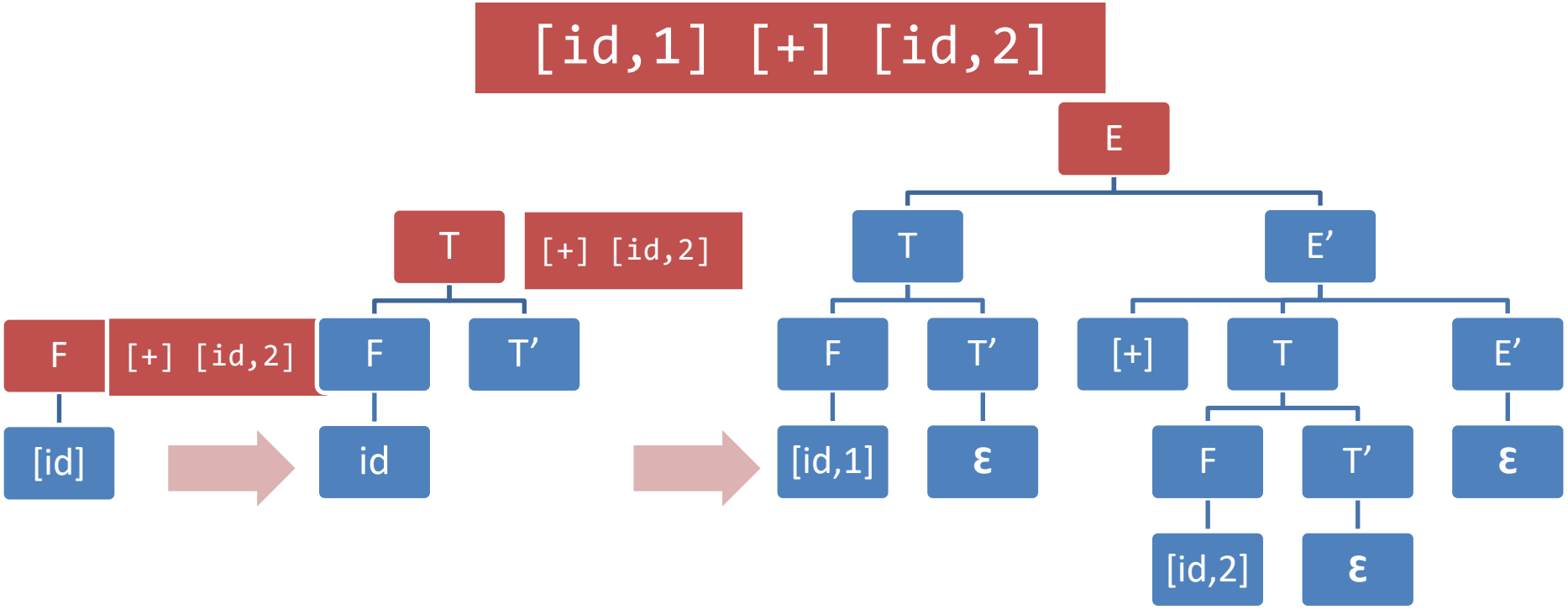
- Construye el árbol sintáctico desde la raíz y creando los nodos en preorden.

```
[id,1] [+] [id,2]
```



Análisis Sintáctico Ascendente

- Construye el árbol sintáctico iniciando desde las hojas hacia la raíz.



Árboles Sintácticos

Árboles Sintácticos

- Forma gráfica para representar la derivación de una cadena a partir del símbolo inicial de un lenguaje.

- Filtra el orden de aplicación de las producciones.

Árboles Sintácticos

- La raíz se etiqueta con el símbolo inicial.
- Cada hoja se etiqueta con un terminal o ε
- Cada nodo interior se etiqueta con un no terminal.

Árboles Sintácticos

digito ::= [0-9]

numero ::= {digito}+

expresion ::= *expresion* + *expresion*

expresion ::= *expresion* - *expresion*

expresion ::= *expresion* * *expresion*

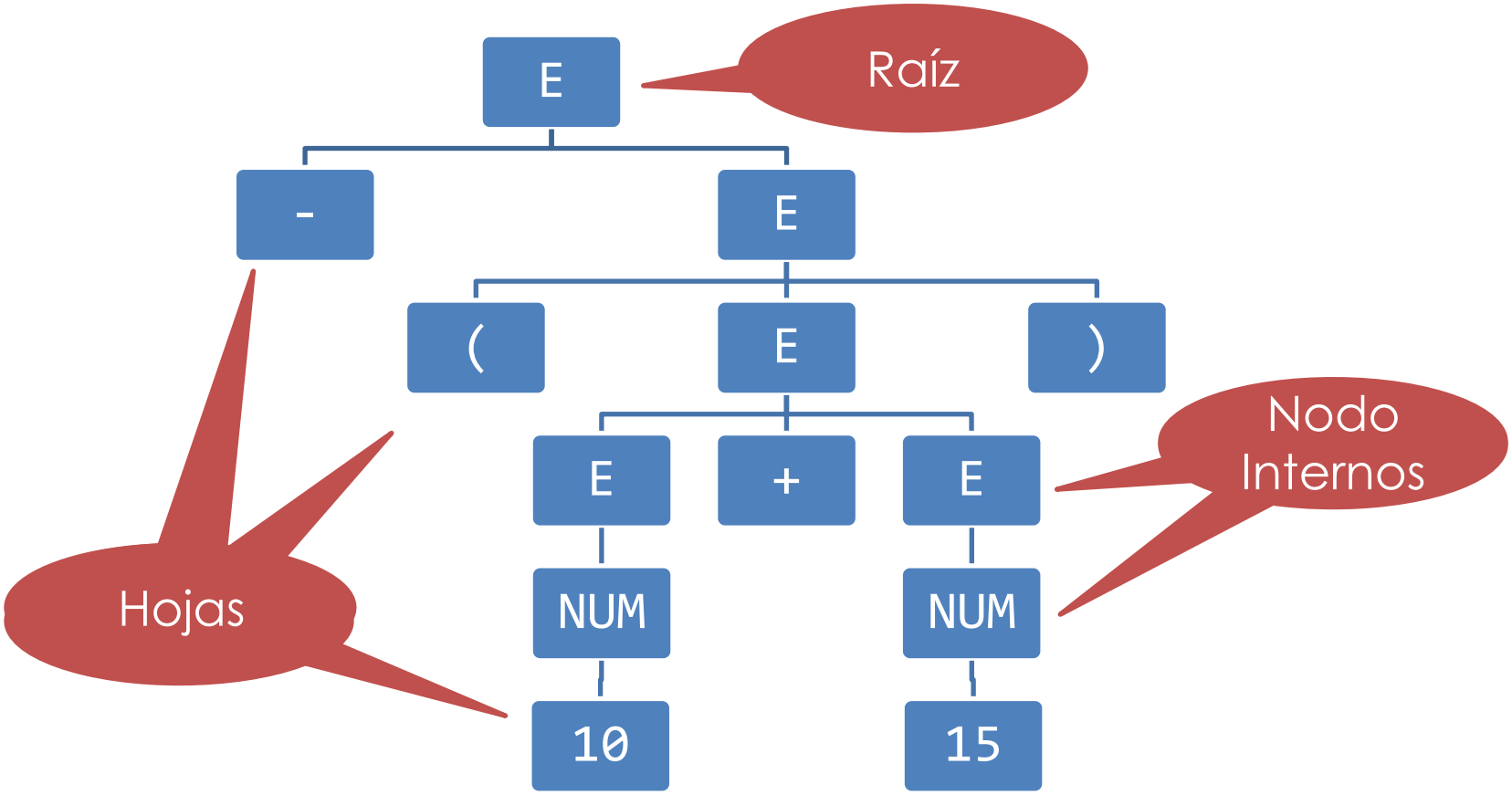
expresion ::= - *expresion*

expresion ::= (*expresion*)

expresion ::= *numero*

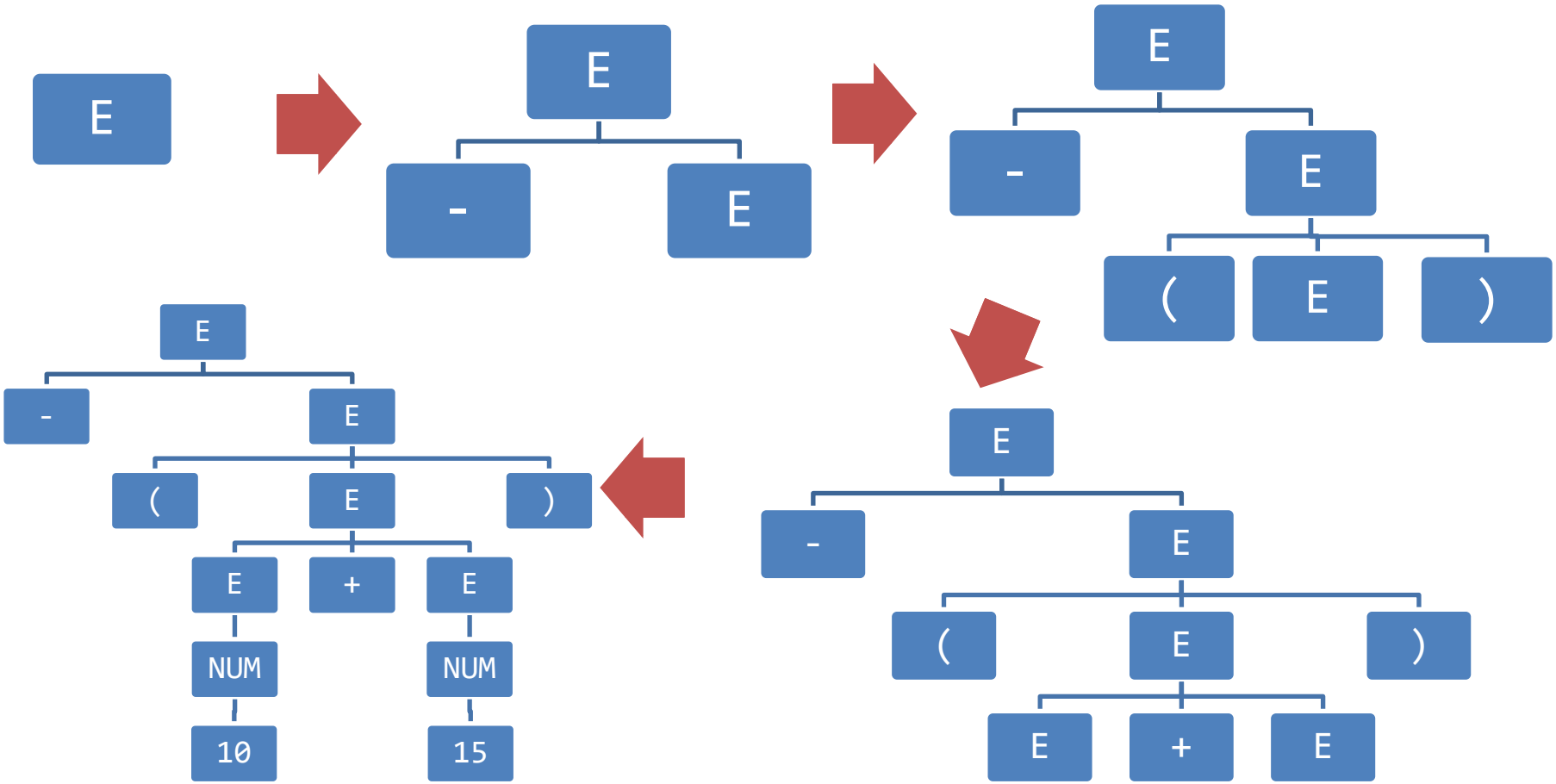
Árboles Sintácticos

-(10 + 15)



Construcción Árboles Sintácticos

-(10 + 15)



TEC

Construcción Árboles Sintácticos

- **13 + 7 * 23**
- **(100*(-(1000)))**
- **(-(625)+(-144+(-(169))))**

Construcción Árboles Sintácticos

sentenciaAsignacion → *var* := *expresion*

expresion → *expresion* + *termino*

expresion → *expresion* – *termino*

expresion → *termino*

termino → *termino* * *factor*

termino → *termino* / *factor*

termino → *factor*

factor → (*expresion*)

factor → *var*

factor → *num*

var → [a-zA-Z][a-zA-Z0-9]*

num → [1-9][0-9]*(. [0-9]⁺)?

Portafolio #2: Árbol Sintáctico

- `varPrincipal:=((3.14+radio)*diametro)/2`
- `TEMPORAL:=(varPrincipal*2)+diametro`
- `Auxiliar:=(TEMPORAL-3.14)*(3.14*2)`


TEC
Tecnológico
de Costa Rica