Tema 2. Análisis léxico

1. Introducción
2. Especificación de un analizador léxico
    - 2.1. Expresiones regulares
    - 2.2. Autómatas finitos
3. Implementación de un analizador léxico
    - 3.1. De una E.R. a un AF mínimo
    - 3.2. Problemas prácticos
4. Generadores de analizadores léxicos

1. Introducción

Funciones del analizador léxico

- Lee secuencia de caracteres y devuelve secuencia de símbolos:
  - *Manejo de fichero de entrada*
  - *Salta del código fuente comentarios, espacios en blanco, tabuladores,…*
  - *Hace corresponder los mensajes de error con la línea de código donde se produce.*
  - *A veces realiza labores de preprocesador*

- No tiene porqué ser una fase individual.

Definiciones

- ***Token***:
  grupo o clase de caracteres con un significado colectivo.

- ***Lexema:***
  Instancia particular de un token: Secuencia de caracteres en el programa fuente que se corresponden con un patrón.

- ***Patrón***:
  Regla que describe cómo se forma un token.

  ¿Cómo se especifican los tokens (patrones)?
  Mediante expresiones regulares

2. Especificación de un A.L.

2.1. Expresiones regulares

Repaso: algunas definiciones

Alfabeto : Conjunto finito de símbolos. Ej. {a, b, c}

Cadena sobre un alfabeto: Secuencia finita y ordenada de símbolos de un alfabeto (también llamada sentencia o palabra).

Lenguaje: Conjunto de cadenas sobre un alfabeto.

Operaciones sobre lenguajes (conjuntos):

unión de L y M, $L \cup M = \{s \mid s \text{ está en L o s está en M}\}$

concatenación de L y M

$LM = \{st \mid s \text{ está en L y t está en M}\}$

Clausura de Kleene de L, $L^* = L^0 \cup L^1 \cup L^2 + \ldots$

$L^0 = \{\varepsilon\}$, $L^1 = L$, $L^2 = L\,L$, $L^3 = L^2\,L$

Clausura positiva de L, $L^+ = L^1 \cup L^2 \cup \ldots$

Ejemplo:

L={aa, bb, cc}, M = {abc}

Expresión regular (ER)

- *Dado un alfabeto* **S**, *una expresión regular (ER) sobre* **S** *se define como:*
  - ∅ es una ER que denota al conjunto ∅.
  - **ε** es una ER que denota al conjunto { **ε** }.
  - **a** ∈ **S** es una ER que denota al conjunto { **a** }.
  - Si **r** y **s** son ER denotando a los lenguajes L_r y L_s entonces:
    - **r** | **s** es una ER que denota $L_r \cup L_s$
    - **r** · **s** es una ER que denota $L_r \cdot L_s$
    - **r*** es una ER que denota L_r^*

Definición regular

- Podemos dar nombres a las ER para construir ER más complejas
  - d_1 -> r_1 $r_1 \in$ ER sobre Σ^*
  - d_2 -> r_2 $r_2 \in$ ER sobre $\Sigma^* \cup \{d_1\}$
  - ...
  - d_n -> r $r_n \in$ ER sobre $\Sigma^* \cup \{d_1, .., d_{n-1}\}$

- Ejemplo:

  letra -> A | B | C | ... | Z | a | b | .... | z

  digito -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

  identificador -> letra (letra | digito) *

letra → a | ... | z | A | ... | Z

dígito → 0 | ... | 9

identificador → letra (letra | dígito)*

constante_numérica → constante_entera | constante_real

constante_entera → dígito (dígito)*

constante_real → constante_entera . (dígito)*

símbolo_especial → asignación | operador_relacional
| subrango | operador_aritmético
| separador | palabra_reservada

símbolos → identificador | constante_numérica

| símbolo_especial

2.2. Autómatas finitos

Reconocedores: AFN

- Un ***reconocedor*** es un programa que toma una cadena x como entrada y responde "si" si es una sentencia del lenguaje, o "no" en otro caso.
- Una ER puede ser convertida en un reconocedor construyendo un autómata finito, que podrá ser determinista o no-determinista.
- Un autómata finito no-determinista (AFN) es un modelo matemático que consiste en: (una 5-tupla)

$$NFA = (Q, \Sigma, \delta, q_0, F)$$

$$F \subseteq Q; \quad q_0 \in Q; \quad \delta: Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow P(Q)$$

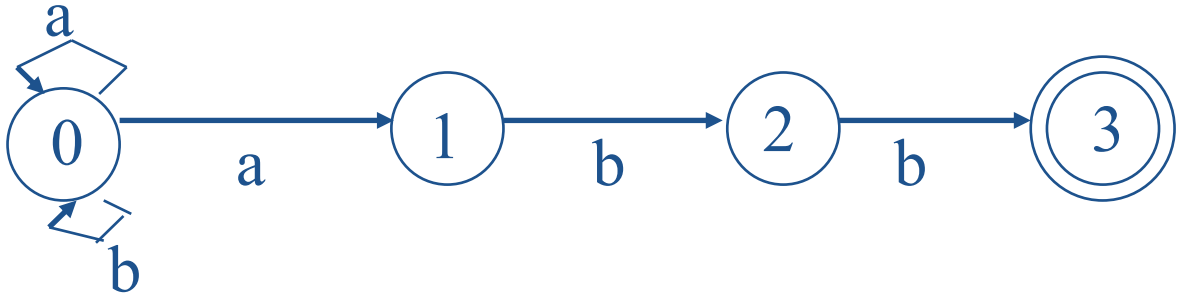
  - Un conjunto de estados Q
  - Un conjunto de símbolos de entrada
  - Una función de transición.
  - Un estado q_0 que se distingue como estado inicial
  - Un conjunto de estados F que se distinguen como estados finales.

AFN

- Un AFN es **no-determinista** si:
  - Un mismo símbolo puede etiquetar dos o más transiciones que salen del mismo estado
  - La cadena vacía etiqueta alguna transición.

- Un AFN **acepta** una cadena de entrada x si y solo si hay algún camino en el grafo de transición etiquetado con los símbolos de la cadena desde el estado inicial a algún estado de aceptación.

- Ejemplo:
  - ¿Qué lenguaje reconoce el AFN?



AFD

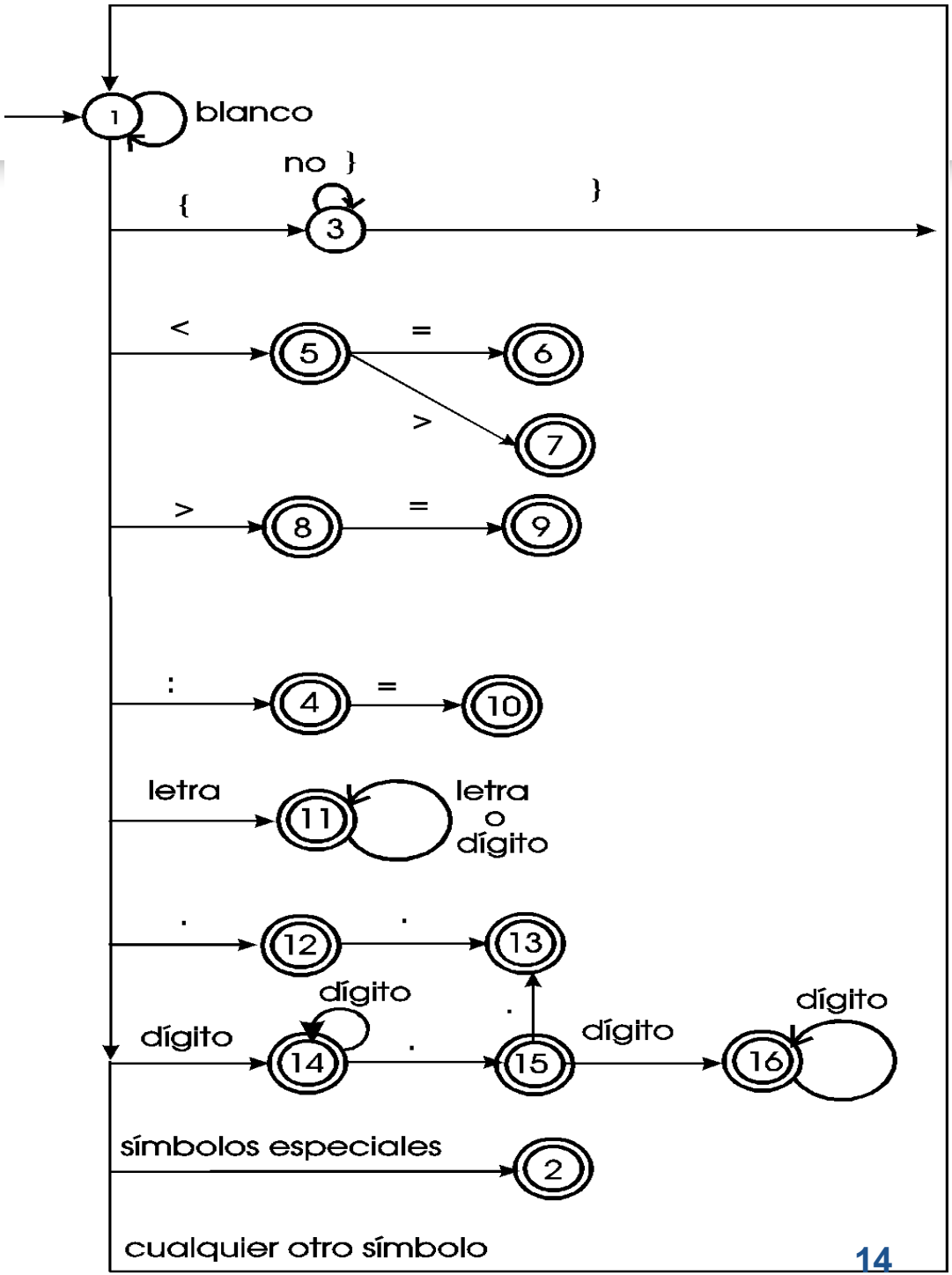
- Se puede mejorar la complejidad temporal usando autómatas finitos deterministas en lugar de AFN.

- Un AF es **determinista** (AFD) si
  - No hay transiciones vacías
  - No hay ningún estado con más de un arco de salida etiquetado con el mismo símbolo

$$AFD = (Q, \Sigma, \delta, q_0, F)$$

- $F \subseteq Q;$ $q_0 \in Q;$ $\delta : Q \times \Sigma \longrightarrow Q$

Ejemplo


1
blanco
no }
{
3
}
<
5
=
6
>
7
>
8
=
9
:
4
=
10
letra
11
letra
o
dígito
.
12
.
13
dígito
dígito
14
.
.
15
dígito
dígito
16
símbolos especiales
2
cualquier otro símbolo

3. Implementación de un A.L.

3.1. De una ER a un AF mínimo

Equivalencias

E.R. $\leftrightarrow$ AFND con ε-trans $\leftrightarrow$ AFND $\leftrightarrow$ AFD

- Construir un AFN para reconocer la unión de todos los patrones
- Convertir el AFN en un AFN sin ε-trans
- Convertir el AFN en un AFD.
- Minimizar el AFD.
- Implementar el AFD

3.2. Problemas prácticos

Problemas prácticos

- Las **palabras clave** pueden:
  - Escribirse como expresiones regulares y ser incluidas en el AFD. Esto incrementará el tamaño del AFD
  - Tratarlas como excepciones de los identificadores (buscándolas en una tabla de **palabras reservadas**)
- El analizador sintáctico **llama** al léxico y éste último lo devuelve un token.
- Cuando el analizador léxico encuentra "**==**" ¿Qué token o tokens reconoce?
  - Reconoce siempre el token más largo
  - Usa varios caracteres de anticipación

Problemas prácticos