Análisis Léxico

Albert Rubio

Procesadores de Lenguajes, FdI, UCM

Doble Grado Matématicas e Informática

Análisis Léxico

Contenidos

Analizador léxico

Carácterísticas principales:

- Procesa uno a uno los caracteres del programa fuente.
- agrupar dichos caracteres en unidades más grandes llamadas unidades léxicas ("tokens") $\Longrightarrow$ analizador sintáctico

$$Analizador\_lexico : Char^* \rightarrow Token^*$$

- Los tokens se agrupan en un número finito de clases.

Unidades léxicas

Ejemplos de clases de unidades léxicas:

- identificadores
- palabras reservadas (una clase por cada palabra). Por ejemplo **while**. No pueden ser identificadores.
- constantes literales (una clase por tipo de datos)
- símbolos (una clase por símbolo)
- operador infijo (como $+$, $*$, etc).

Reconocimiento de unidades léxicas

- Las UL se describen con un *lenguaje regular*, que se reconoce con un *autómata finito*.
- Para cada UL el analizadir devuelve su clase y cero o más atributos:
  - nombre del identificador o valor de la constante
  - número de línea y de columna donde aparece la UL
  - ...
- Puede procesar toda la entrada o ir token a token, pero no tiene información sobre la estructura del texto.
- Otras tareas del analizador léxico:
  - Pasar las constantes a su representación interna: "123" $\rightarrow$ 123
  - Asociar una clave numérica única a cada clase.
  - Eliminar comentarios y separadores.
  - Detectar errores léxicos.

Características léxicas de los lenguajes

Juego de caracteres y su codificación: ASCII (7 bits), Unicode (UTF-8 o UTF-16), etc. Propiedades relevantes

- si los números y las letras tienen códigos consecutivos,
- relación entre la codificación de minúsculas y mayúsculas,
- codificación del carácter blanco
- ...

Decisiones a tomar:

- Distinción entre minúsculas y mayúsculas (case sensitive).
- Incio y caracteres permitidos en los identificadores. P.e. inicio letra o _
- Relevancia de los blancos y otros separadores
- Formato comentarios. Linea // o bloque /*...*/. Anidamiento?

Contenidos

Notación

La sintaxis para describir ERs puede variar "ligeramente" de una herramienta a otra pero siguen siempre un mismo patrón:

- Alfabeto finito de simbolos $\Sigma = \{a, b, c, \ldots\}$.
- La palabra vacía: ϵ
- Σ^* palabras usando Σ (incluida ϵ). $\overline{A} = \Sigma - A$, con $A \subseteq \Sigma$.
- xy es la concatenación de las palabras x e y.
- x^n para concatenar n veces la misma palabra x

Un lenguaje sobre Σ es un subconjunto $L \subseteq \Sigma^*$. Operaciones:

- Unión: $L_1 \cup L_2$.
- Concatenación: $L_1 L_2 = \{xy \mid x \in L_1, y \in L_2\}$
- Complementario: $\overline{L} = \Sigma^* - L$.
- Repetición: $L^n = L \overset{n}{\cdots} L$. Estrella: $L^* = \bigcup_{n \geq 0} L^n$

Lenguajes regulares

Definición inductiva del lenguaje descrito con expresiones regulares sobre Σ.

1. La ER $\emptyset$ denota el lenguaje vacio.
2. La ER ϵ denota el lenguaje $\{\epsilon\}$ y a el lenguaje $\{a\}$ con $a \in \Sigma$.
3. Si r_1 y r_2 son ER que describen L_1 y L_2 entonces
   - $r_1 \mid r_2$ es una ER que describe el lenguaje $L_1 \cup L_2$
   - $r_1 r_2$ es una ER que describe el lenguaje $L_1 L_2$
   - r_1^* es una ER que describe el lenguaje L_1^*

Tomamos la precedencia $* > \mathit{concat} > |$

Un *lenguaje regular* es un lenguaje que puede ser definido por una expresión regular: $L(r)$
Si L es regular entonces $\overline{L}$ también es regular.

Lenguajes regulares

Abreviaturas:

- $r^+ = rr^*$
- $r? = r \mid \epsilon$
- $[a..z] = a \mid \ldots \mid z$
- Si $L(r) = A \subseteq \Sigma$ entonces $\overline{r}$ denota $\overline{A}$.

Ejemplos:

- $(0 \mid 1)^*$
- $(01)^*$
- 01^*
- $(0 \mid 1)^*00(0 \mid 1)^*$, pero no se puede expresar que el prefijo y el sufijo son iguales.

ERs para definir ULs

$$
\begin{array}{lcl}
\mathit{digito} & \rightarrow & [0..9] \\
\mathit{letra} & \rightarrow & [a..zA..Z] \\
\mathit{literalEntero} & \rightarrow & [+ \mid -]\mathit{digito}^{+} \\
\mathit{id} & \rightarrow & \mathit{letra}(\mathit{letra} \mid \mathit{digito})^{*} \\
\mathit{opRel} & \rightarrow & < \mid <= \mid > \mid >= \mid == \mid != \\
\mathit{add} & \rightarrow & + \\
 & \ldots & \\
\mathit{sep} & \rightarrow & \mathit{SP} \mid \mathit{TAB} \mid \mathit{CR} \mid \mathit{NL} \\
\mathit{seps} & \rightarrow & \mathit{sep}^{+} \\
\mathit{comentario} & \rightarrow & //(\overline{CR})^{*}\mathit{CR}
\end{array}
$$

Contenidos

Lenguajes regulares y autómatas finitos

Un lenguaje regular queda determinado por

- una expresión regular (ER)
- un *autómata finito no determinista* con transiciones vacías (AFN)
- un *autómata finito determinista* (AFD)

Pasaremos de una ER que describen las UL a un AFD que las reconoce:

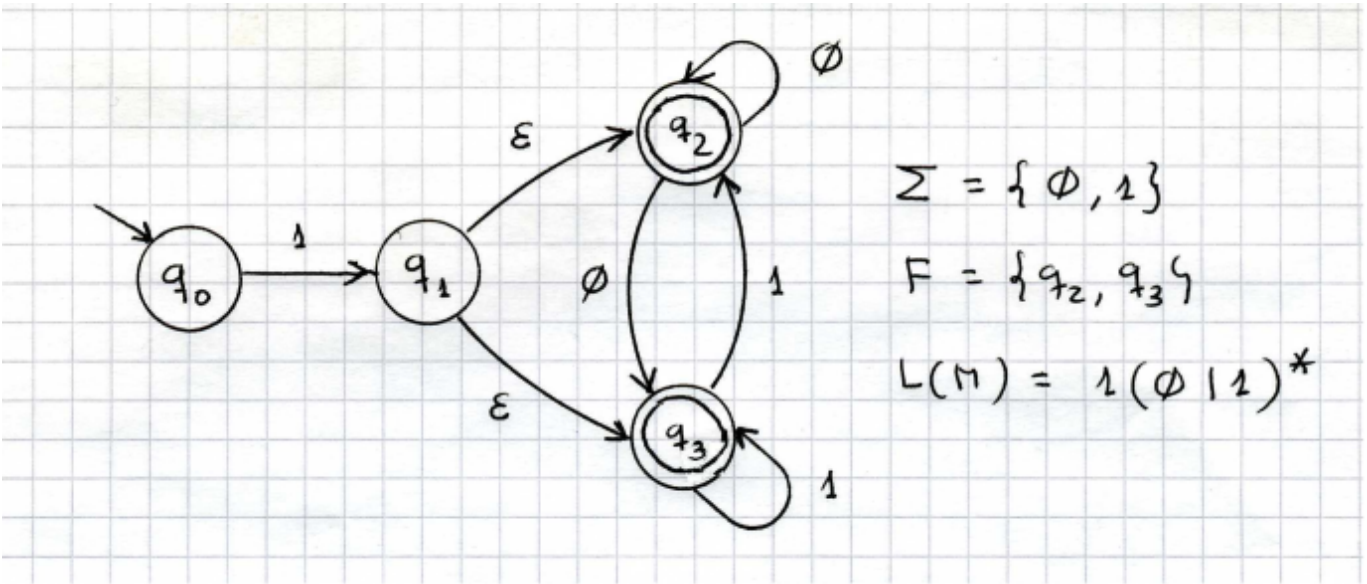
$$\boxed{\text{ER}} \longrightarrow \boxed{\text{AFN}} \longrightarrow \boxed{\text{AFD}}$$

Todos son equivalentes.

Autómata finito no determinista

$$M = \langle\ Q\ ,\ \Sigma\ ,\ q_0\ ,\ \delta\ ,\ F \rangle$$

- Q conjunto de *estados* (finito)
- Σ *alfabeto* (finito)
- $q_0 \in Q$ *estado inicial*
- $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ *relación de transición*
- $F \subseteq Q$ *estados finales*



Autómata finito no determinista

Configuraciones de un autómata:

- (q, w) con $q \in Q$ y $w \in \Sigma^*$ es una *configuración*
- (q_0, w) es una *configuración inicial*
- (q_f, ϵ) con $q_f \in F$ es una *configuración final*
- $(q, aw) \vdash_M (p, w)$ con $a \in \Sigma$ si $\delta(q, a, p)$
- $(q, ww') \vdash_M^* (p, w')$ cierre reflexivo y transitivo de $\vdash_M$

δ no es determinista: dados q y a puede haber varios p tal que $\delta(q, a, p)$. Además están las transiciones con ϵ

Lenguaje reconocido por M:

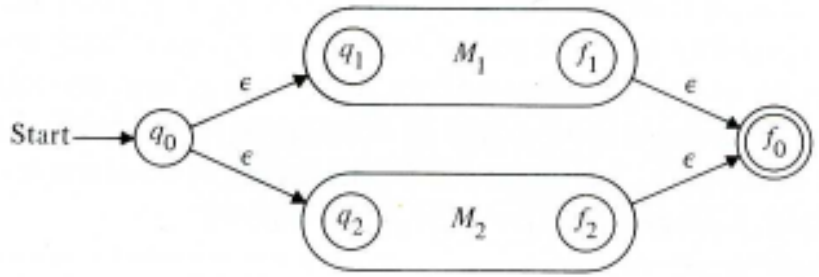
$$L(M) = \{w \in \Sigma^* \mid (q_0, w) \vdash_M^* (q_f, \epsilon), q_f \in F\}$$

De ER a AFN

Dada una ER r simpre existe un AFN M tal que $L(r) = L(M)$

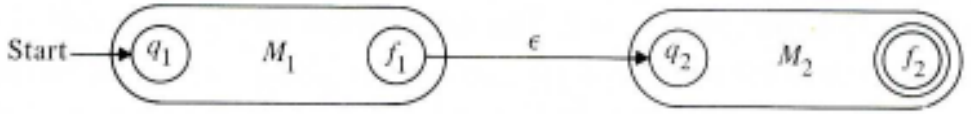
M se construye inductivamente a partir de r

- $r = \emptyset$: $M = \langle \{q_0, q_f\}, \Sigma, q_0, \emptyset, \{q_f\} \rangle$
- $r = \epsilon$: $M = \langle \{q_0, q_f\}, \Sigma, q_0, \{(q_0, \epsilon, q_f)\}, \{q_f\} \rangle$
- $r = a$: $M = \langle \{q_0, q_f\}, \Sigma, q_0, \{(q_0, a, q_f)\}, \{q_f\} \rangle$
- $r_1 \mid r_2$:

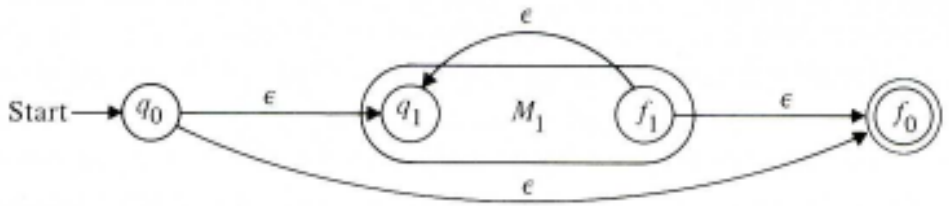


y simplificar

- $r_1 r_2$:



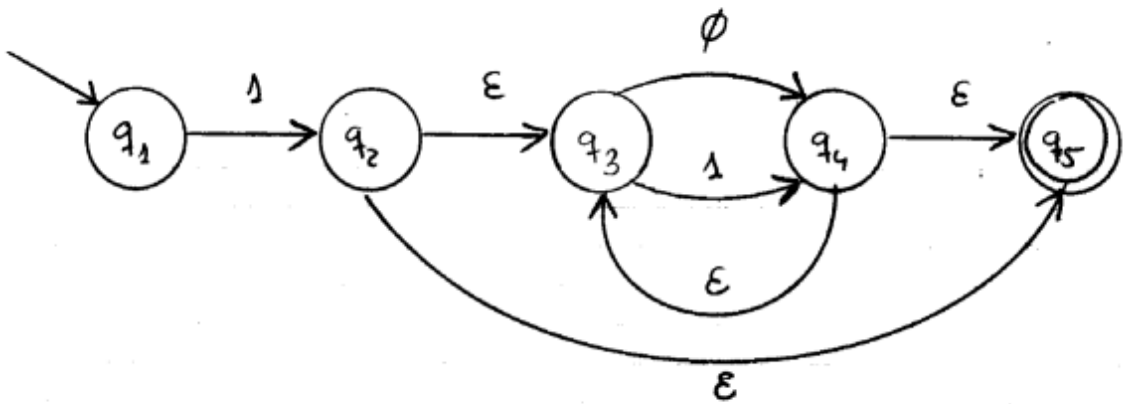
- r_1^*:



Ejemplo de construcción

$r = 1\ (0\ \mid\ 1)^*$

Un posible AFN:



¿Cómo se obtiene?

Autómata finito determinista

$$M = \langle\ Q\ ,\ \Sigma\ ,\ q_0\ ,\ \delta\ ,\ F \rangle$$

donde $\delta : Q \times \Sigma \to Q$ es ahora una función (parcial)
Notad que las transiciones con ϵ ya no se permiten!

- Dada una configuración (q, aw) existe a lo sumo una configuración (p, w) tal que $(q, aw) \vdash_M (p, w)$, es decir si $\delta(q, a) = p$

Lenguaje reconocido por M:

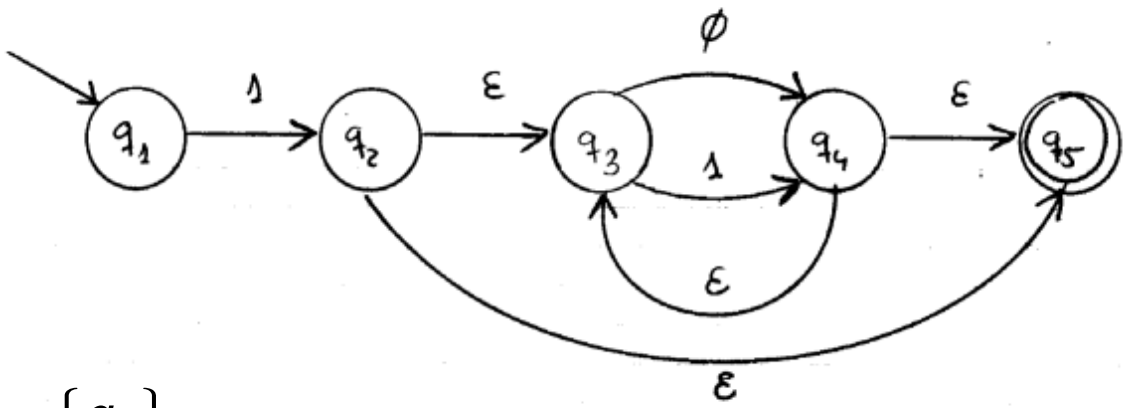
$$L(M) = \{w \in \Sigma^* \mid (q_0, w) \vdash_M^* (q_f, \epsilon), q_f \in F\}$$

De AFN a AFD

Dado un AFN M siempre existe un AFD M' tal que $L(M) = L(M')$

- Es necesario el *cierre* $-\ \epsilon$ de los estados de M:

$$cierre - \epsilon(q) = \{p \mid (q, \epsilon) \vdash_M^* (p, \epsilon)\}$$



$cierre - \epsilon(q_1) = \{q_1\}$
$cierre - \epsilon(q_2) = \{q_2, q_3, q_5\}$
$cierre - \epsilon(q_3) = \{q_3\}$
$cierre - \epsilon(q_4) = \{q_4, q_3, q_5\}$
$cierre - \epsilon(q_5) = \{q_5\}$

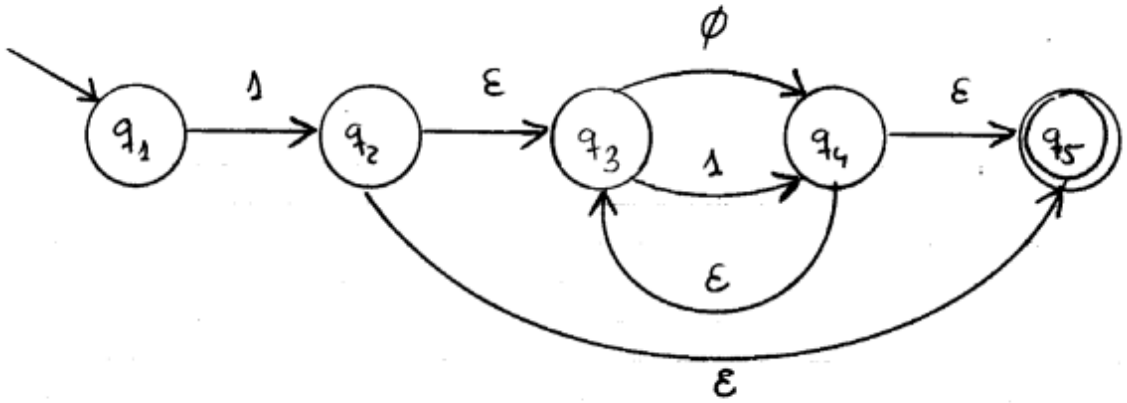
De AFN a AFD

Como funciona la transformación:

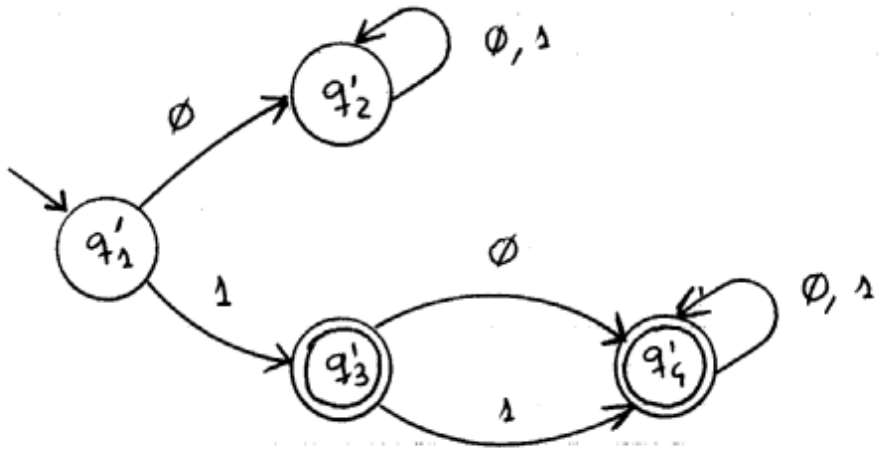
- Extendemos el *cierre* $-\epsilon$ a conjuntos de estados: $cierre - \epsilon(S) = \bigcup_{q \in S} cierre - \epsilon(q)$
- Los estados de M' son subconjuntos de los estados de M ($Q_{M'} \subseteq \mathcal{P}(Q_M)$.
- Dos estados p y q de M estarán en el mismo estado de M' si existe una palabra w tal que $(q_0, w) \vdash_M (p, \epsilon)$ y $(q_0, w) \vdash_M (q, \epsilon)$. Computable.
- El estado inicial de M' es $cierre - \epsilon(q_0)$
- Estado finales de M' son los estados S de M' que contienen algun estado final de M.

De AFN a AFD: ejemplo

De



Pasamos a



donde $q'_1 = \{q_1\}, q'_2 = \{\}, q'_3 = \{q_2, q_3, q_5\}, q'_4 = \{q_3, q_4, q_5\}$

Minimización de AFDs

En general, el AFD resultante puede ser innecesariamente grandes

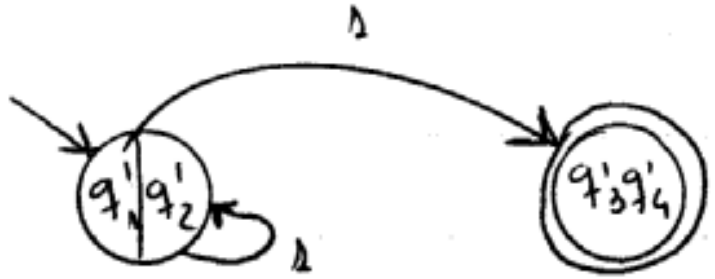
Se puede computar un AFD mínimo respecto a $\mid Q \mid$:

- Se separan inicialmente los estados en dos partes F y $Q - F$, ya que tienen comportamiento distinto.
- Se refina la partición iterativamente hasta que no hay ningún cambio: Cada partición se divide en subparticiones tales que todos sus estados se comporten igual para todas sus transiciones respecto a la partición anterior.

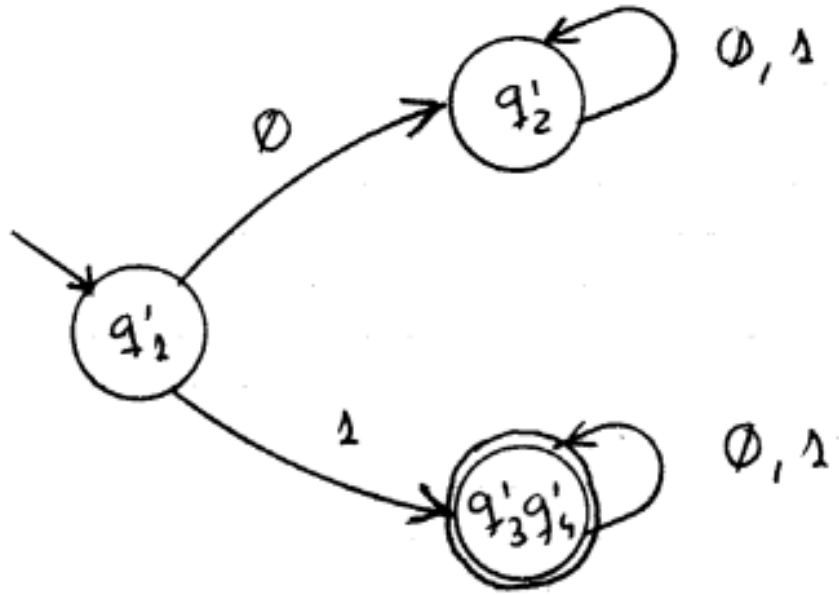
Minimización de AFDs

Ejemplo:

- Paso 1: $\{q'_1, q'_2\}$ y $\{q'_3, q'_4\}$



- Paso 2: refinamos $\{q'_1, q'_2\}$



- Paso 3: No hay más cambios

Construcción de analizadores léxicos

Una vez especificadas las unidades léxicas con ERs, se puede construir el analizador léxico:

- Manualmente:
  se obtiene el AFD (mínimo) y se programa explícitamente la función de transición y la generación de tokens aceptados. Se puede usar una tabla para δ e iterar sobre la entrada.
  Hay que implementar el tratamiento de errores: reportar y recuperarse.
- Automáticamante:
  toman como entrada la expresión regular y producen directamente un analizador dirigido por tabla y la tabla misma.
  Permiten asociar acciones de algún lenguaje de alto nivel como C++ o Java al reconocimiento de cada unidad léxica.
  Normalmente incluyen mecanismos para la gestión de errores.