Gramáticas regulares y analizadores léxicos

Pedro O. Pérez M., PhD.

Implementación de métodos computacionales
Tecnológico de Monterrey

pperezm@tec.mx

03-2021

Contenido I

Introducción

El concepto de lenguaje incluye una variedad de categorías aparentemente distintas que incluyen lenguajes naturales, lenguajes informáticos y lenguajes matemáticos. Una definición general de lenguaje debe abarcar todos estos diversos tipos de lenguajes.

Durante las siguientes sesiones analizaremos una definición puramente teórica de lenguaje: un lenguaje es un conjunto de cadenas sobre un alfabeto. El alfabeto es el conjunto de símbolos del lenguaje y una cadena sobre el alfabeto es una secuencia finita de símbolos del alfabeto.

También, conoceremos la familia de lenguajes definidos por expresiones regulares. Una expresión regular describe un patrón y el lenguaje asociado con la expresión regular consta de todas las cadenas que coinciden con el patrón.

Por último, examinaremos el uso de expresiones regulares en la búsqueda y la coincidencia de patrones.

Cadenas y lenguajes

- Un *alfabeto* (Σ) es un conjunto finito de símbolos.
- Una *cadena* (o *string*) de un alfabeto es una secuencia finita de símbolos obtenidos de ese alfabeto. La longitud de una cadena s, usualmente escrito como $|s|$, el número de símbolos en s.
- Un *lenguaje* es cualquier conjunto contable de *cadenas* sobre un alfabeto determinado.
-
- El string vacío, ϵ, es identidad bajo la concatenación.
- Concatenación se define como sigue: s^0 es ϵ, y para $i > 0$, s^i es $s^{i-1}s$.

Definición: Sea Σ un alfabeto. Σ^*, el conjunto de cadenas sobre Σ, se define recursivamente como:

1. **Base:** $\epsilon \in \Sigma^*$.
2. **Recursión:** Si $w \in \Sigma^*$ y $a \in \Sigma$, entonces $wa \in \Sigma^*$.
3. **Cerradura:** $w \in \Sigma^*$ sólo si se puede obtener a partir de ϵ mediante un número finito de aplicaciones del paso recursivo.

Definición: Sea $\Sigma = a, b, c$. Σ^* incluye cadenas de...

1. longitud 0: ϵ
2. longitud 1: a, b, c
3. longitud 2: aa, ab, ac, ba, bb, bc, ca, cb, cc
4. ...

Definición: Un *lenguaje* sobre un alfabeto Σ es un subconjunto de Σ^*.

Definición: Sean u, $v \in \Sigma^*$. La *concatenacin* de u y v, escrita como uv, es una operación binaria sobre Σ^* que se define como:

1. **Base:** Si $|v| = 0$, entonces $v = \epsilon$ y $uv = u$.
2. **Recursión:** Sea v una cadena de $|v| = n$, siendo $n > 0$.Entonces $v = wa$, si para alguna cadena w de longitud $n - 1$ y $a \in \Sigma$, y $uv = (uw)a$.

Definición: La concatenación es asociativa. Es decir, sea u, v, $w \in \Sigma^*$. Entonces, $(uv)w = u(vw)$.

Definición: Sea u una cadena en Σ^*. La inversión de u, denotada u^R, se define como:

1. **Base:** Si $|u| = 0$, entonces $u = \epsilon$ y $\epsilon^R = \epsilon$.
2. **Recursión:** Si $|u| = n$ siendo $n > 0$, entonces $u = wa$ para alguna cadena w con longitud $n - 1$ y para alguna $a \in \Sigma$, y $u^R = aw^R$.

Definición: Sea u, $v \in \Sigma^*$. Entonces, $(uv)^R = v^R u^R$.

Especificación finita de lenguajes

Recuerda que, un lenguaje se define como un conjunto de cadenas sobre un alfabeto. Los lenguajes de interés no consisten en conjuntos arbitrarios de cadenas, sino más bien en cadenas que satisfacen algunos requisitos sintácticos prescritos. Un lenguaje finito define explícitamente enumerando sus elementos. Varios lenguajes infinitos con requisitos sintácticos simples se definen de forma recursiva en los ejemplos que siguen.

Ejemplo 1. Sea L, un lenguaje sobre $\Sigma = a, b$, en el que cada cadena comienza con una a y tiene una longitud par, se define:

1. **Base:** aa, $ab \in L$.
2. **Recusión:** Si $u \in L$, entonces uaa, uab, uba, $ubb \in L$.
3. **Cerradura:** Una cadena $u \in L$ sólo si puede obtenerse de los elementos base mediante un número finito de aplicaciones del paso recursivo.

Ejemplo 2. Sea L, un lenguaje sobre $\Sigma = a, b$ definido por:

1. **Base:** $\epsilon \in L$.
2. **Recusión:** Si $u \in L$, entonces ua, $uab \in L$.
3. **Cerradura:** Una cadena $u \in L$ sólo si puede obtenerse de los elementos base mediante un número finito de aplicaciones del paso recursivo.

consta de cadenas en las que cada aparición de b está precedida por una a. Por ejemplo, ϵ, a, $abaab$ están en L, mientras que bb, abb no están.

Ejemplo 3. Sea L, un lenguaje sobre $\Sigma = a, b$ definido por:

1. **Base:** $\epsilon \in L$.
2. **Recusión:** Si $u \in L$ y u puede ser escrito como $u = xyz$, entonces $xaybz \in L$ y $xaybz \in L$.
3. **Cerradura:** Una cadena $u \in L$ sólo si puede obtenerse de los elementos base mediante un número finito de aplicaciones del paso recursivo.

consta de cadenas con el mismo número de a's y b's.

Definición: La concatenación de lenguajes X y Y, indicada como XY, es el lenguaje $XY = \{uv \mid u \in X \text{ y } v \in Y\}$. La concatenación de X con el mismo n se indica como X^n, donde X^0 es ϵ.

Definición: Sea X un conjunto. Entonces, $X^* = \cup_{i=0}^{\infty} X^i$ y $X^+ = \cup_{i=1}^{\infty} X^i$

Sea L el conjunto de caracteres $a..z, A..Z$ y D el conjunto de dígitos 0.,9:

- $L \cup D$ es el conjunto de caracteres y dígitos.
- LD es conjunto de string de longitud 2 integrados por una letra y un dígito.
- L^4 el conjunto de string de 4 letras.
- L^* es el conjunto de todos los string de letras, incluyendo ϵ.
- $L(L \cup D)^*$ es el conjunto de todos los strings de letras y dígitos que empiezan con una letra.
- D^+ es el conjunto de todos los string de uno o más dígitos.

Ejercicios

Por ejemplo, sea $\Sigma = \{a, b\}$,

- Define un lenguaje que consista de todas las cadenas que contengan la subcadena bb.
- Define un lenguaje que consista de todas las cadenas que empiecen con aa o terminen con bb.
- Define un lenguaje que consista de todas las cadenas que empiecen y terminen con a y que contengan al menos una b.

Conjuntos y expresiones regulares

Un conjunto de cadenas es regular si se puede generar a partir del conjunto vacío, el conjunto que contiene la cadena nula y conjuntos que contienen un solo elemento del alfabeto mediante la unión, la concatenación y la operación de cerradura de Kleene ($*$).

Definición: Sea Σ, un alfabeto. El conjunto regular sobre Σ se define recursivamente como:

1. **Base:** $\emptyset$, ϵ y a, para cada $a \in \Sigma$, son conjuntos regulares sobre Σ.
2. **Recusión:** Sean X y Y, conjuntos regulares sobre Σ. Los conjuntos $X \cup Y$, XY, X^* son conjuntos regulares sobre Σ.
3. **Cerradura:** X es un conjunto regular sobre Σ sólo si puede obtenerse de los elementos base mediante un número finito de aplicaciones del paso recursivo.

Un lenguaje es llamado regular si está definido por un conjunto regular.

Definición: Sea Σ, un alfabeto. Las expresiones regulares sobre Σ se define recursivamente como:

1. **Base:** $\emptyset$, ϵ y $a \in \Sigma$, son expresiones regulares sobre Σ.
2. **Recusión:** Sean u y v, expresiones regulares sobre Σ. Los conjuntos $(u \cup v)$, (uv), u^* son expresiones regulares sobre Σ.
3. **Cerradura:** u es una expresión regular sobre Σ sólo si puede obtenerse de los elementos base mediante un número finito de aplicaciones del paso recursivo.

Podemos quitar ciertos paréntesis si adoptamos las siguiente convenciones:

- El operador unitario $*$ tiene la más alta prioridad y es asociativo por la izquierda.
- La concatenación es segundo en precedencia y asociativo por la izquierda.
- $|$ tiene la precedencia más baja y es asociativo por la izquierda.

Expresiones regulares y búsqueda de texto

- Una aplicación común de las expresiones regulares es la especificación de patrones para buscar documentos y archivos.
- La principal diferencia entre el uso de expresiones regulares para la definición de lenguajes y para la búsqueda de texto es el alcance de la coincidencia deseada. Una cadena está en el lenguaje definido por una expresión regular si toda la cadena coincide con el patrón especificado por la expresión regular.

LAW	DESCRIPTION
$r\|s = s\|r$	$\|$ is commutative
$r\|(s\|t) = (r\|s)\|t$	$\|$ is associative
$r(st) = (rs)t$	Concatenation is associative
$r(s\|t) = rs\|rt;\ (s\|t)r = sr\|tr$	Concatenation distributes over $\|$
$\epsilon r = r\epsilon = r$	ϵ is the identity for concatenation
$r^* = (r\|\epsilon)^*$	ϵ is guaranteed in a closure
$r^{**} = r^*$	* is idempotent

Figure 3.7: Algebraic laws for regular expressions

Example 3.6: Unsigned numbers (integer or floating point) are strings such as `5280`, `0.01234`, `6.336E4`, or `1.89E-4`. The regular definition

$$
\begin{array}{rcl}
\textit{digit} & \rightarrow & \texttt{0} \mid \texttt{1} \mid \cdots \mid \texttt{9} \\
\textit{digits} & \rightarrow & \textit{digit}\ \textit{digit}^{*} \\
\textit{optionalFraction} & \rightarrow & \texttt{.}\ \textit{digits} \mid \epsilon \\
\textit{optionalExponent} & \rightarrow & (\ \texttt{E}\ (\ \texttt{+} \mid \texttt{-} \mid \epsilon\ )\ \textit{digits}\ ) \mid \epsilon \\
\textit{number} & \rightarrow & \textit{digits}\ \textit{optionalFraction}\ \textit{optionalExponent}
\end{array}
$$

- $+$: Una o más instancias.
- ?: Cero o una instancia.
- Clases de caracteres: Una expresión regular del tipo $a_1|a_2|...|a_n$ donde $a_i's$ son símbolos del alfabeto puede ser reemplazado por $[a_1a_2...a_n]$.

Example 3.7: Using these shorthands, we can rewrite the regular definition of Example 3.5 as:

$$\begin{array}{rcl} \mathit{letter\_} & \rightarrow & \texttt{[A-Za-z\_]} \\ \mathit{digit} & \rightarrow & \texttt{[0-9]} \\ \mathit{id} & \rightarrow & \mathit{letter\_}\ (\ \mathit{letter} \mid \mathit{digit}\ )^* \end{array}$$

The regular definition of Example 3.6 can also be simplified:

$$\begin{array}{rcl} \mathit{digit} & \rightarrow & \texttt{[0-9]} \\ \mathit{digits} & \rightarrow & \mathit{digit}^+ \\ \mathit{number} & \rightarrow & \mathit{digits}\ (\texttt{.}\ \mathit{digits})?\ (\ \texttt{E}\ \texttt{[+-]}?\ \mathit{digits}\ )? \end{array}$$

Ejercicios

Exercise 3.3.5: Write regular definitions for the following languages:

a) All strings of lowercase letters that contain the five vowels in order.

b) All strings of lowercase letters in which the letters are in ascending lexicographic order.

c) The "digits" in a hexadecimal number (choose either upper or lower case for the "digits" above 9).

Diagramas de transición

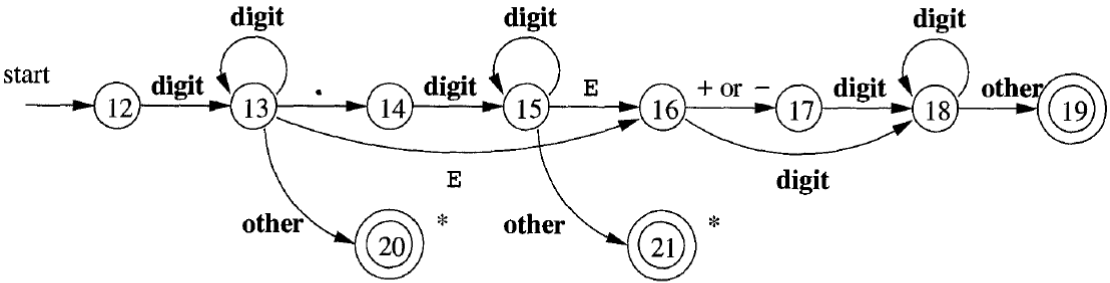


Figure 3.16: A transition diagram for unsigned numbers

Autómata finito

Un autómata finito es esencialmente un grafo, como los diagramas de transición, pero con algunas diferencias:

- Los autómatas finitos son "`reconocedores`".
- Pueden ser de dos tipos:
  - Autómata Finito No Determinista, AFN (Non-deterministic finite automata, NFA). No tiene restricciones en las etiquetas de los arcos. Un símbolo puede aparecer en diferentes arcos que salgan del mismo estado y ϵ es una posible etiqueta.
  - Autómata Finito Determinista, AFD (Deterministic finite automata, DFA). Tiene para cada estado y para cada símbolo del alfabeto exactamente un arco saliendo de ese estado.

Autómata Finito No Determinista (AFN)

Un autómata finito no determinista (AFN) consiste de:

- Un conjunto finito de estados, S.
- Un conjunto de símbolos de entrada Σ (alfabeto de entrada). Asumimos que ϵ nunca es miembro de Σ.
- Una *función de transición* que, para cada estado, y para cada símbolo en $\Sigma \cup \epsilon$ un conjunto de siguientes estados.
- Un estado s_0 del S considerado como estado inicial.
- Un conjunto de estados, F, subconjunto de S, que son identificados como estado de aceptación (o estados finales).

Tablas de transición

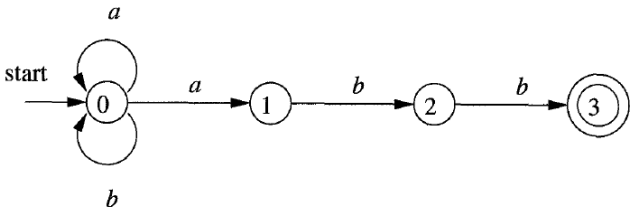


Figure 3.24: A nondeterministic finite automaton

STATE	a	b	ϵ
0	$\{0,1\}$	$\{0\}$	$\emptyset$
1	$\emptyset$	$\{2\}$	$\emptyset$
2	$\emptyset$	$\{3\}$	$\emptyset$
3	$\emptyset$	$\emptyset$	$\emptyset$

Figure 3.25: Transition table for the NFA of Fig. 3.24

Autómata Finito Determinista (AFD)

Un autómata finito determinista (AFD) es un caso especial de AFN donde

- No existen movimientos bajo ϵ.
- Para cada estado s y para símbolo de entrada a, existe exactamente un arco que sale de s etiquetado como a.

```
s = s0;
c = nextChar();
while ( c != eof ) {
        s = move(s, c);
        c = nextChar();
}
if ( s is in F ) return "yes";
else return "no";
```

Figure 3.27: Simulating a DFA

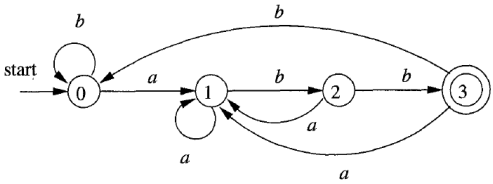


Figure 3.28: DFA accepting (**a**|**b**)*__abb__

Ejercicios

Exercise 3.6.3: For the NFA of Fig. 3.29, indicate all the paths labeled *aabb*. Does the NFA accept *aabb*?

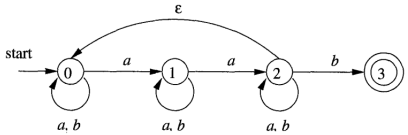


Figure 3.29: NFA for Exercise 3.6.3

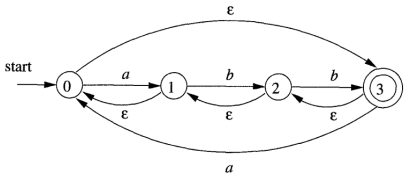


Figure 3.30: NFA for Exercise 3.6.4

Exercise 3.6.4: Repeat Exercise 3.6.3 for the NFA of Fig. 3.30.

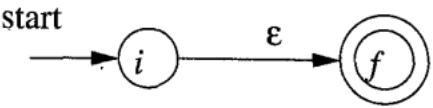
3.7 De expresiones regulares a autómatas

Las expresiones regulares son la notación empleada para describir un analizador léxico, así como otros software de procesamiento de patrones. Sin embargo, la implementación de este software requiere de una simulación de un AFD o, quizás, de un AFN.

Construcción de un AFN a partir de una expresión regular

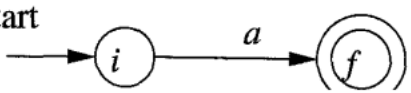
$r = \epsilon$

BASIS: For expression ϵ construct the NFA



$r = a$

For any subexpression a in Σ, construct the NFA



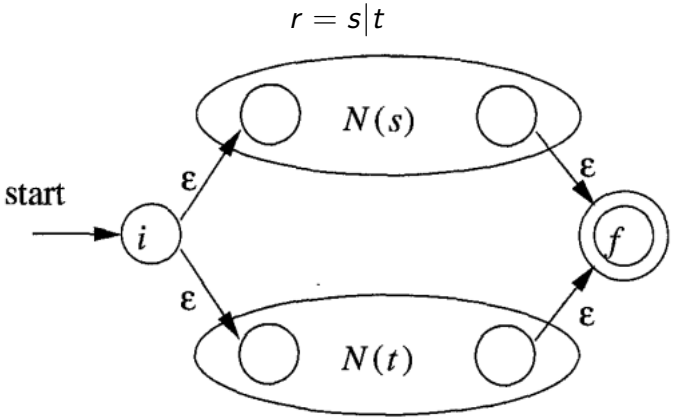


Figure 3.40: NFA for the union of two regular expressions

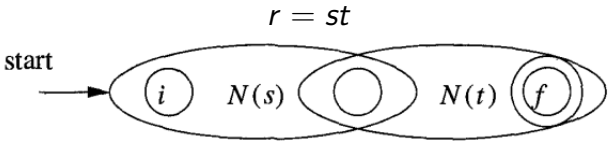


Figure 3.41: NFA for the concatenation of two regular expressions

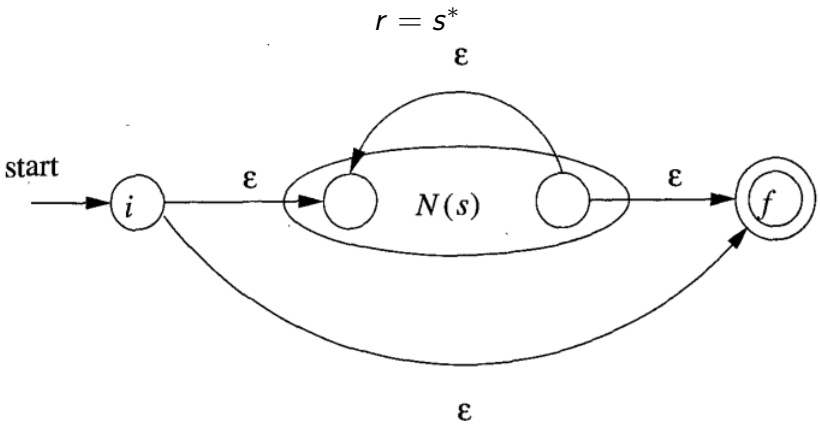


Figure 3.42: NFA for the closure of a regular expression

Ejercicios

Construye el AFN de la siguiente expresión:

$$r = (a|b)^* abb$$

Algoritmo de conversión de AFN a AFD

OPERATION	DESCRIPTION
ϵ-*closure*(s)	Set of NFA states reachable from NFA state s on ϵ-transitions alone.
ϵ-*closure*(T)	Set of NFA states reachable from some NFA state s in set T on ϵ-transitions alone; $= \cup_{s \text{ in } T}$ ϵ-*closure*(s).
move(T, a)	Set of NFA states to which there is a transition on input symbol a from some state s in T.

Figure 3.31: Operations on NFA states

```
while ( there is an unmarked state T in Dstates ) {
        mark T;
        for ( each input symbol a ) {
                U = ε-closure(move(T, a));
                if ( U is not in Dstates )
                        add U as an unmarked state to Dst
                Dtran[T, a] = U;
        }
}
```

Figure 3.32: The subset construction

```
push all states of T onto stack;
initialize ε-closure(T) to T;
while ( stack is not empty ) {
        pop t, the top element, off stack;
        for ( each state u with an edge from t to u labele
                if ( u is not in ε-closure(T) ) {
                        add u to ε-closure(T);
                        push u onto stack;
                }
}
```

Figure 3.33: Computing ϵ-*closure*(T)

$$r = (a|b)^*abb$$

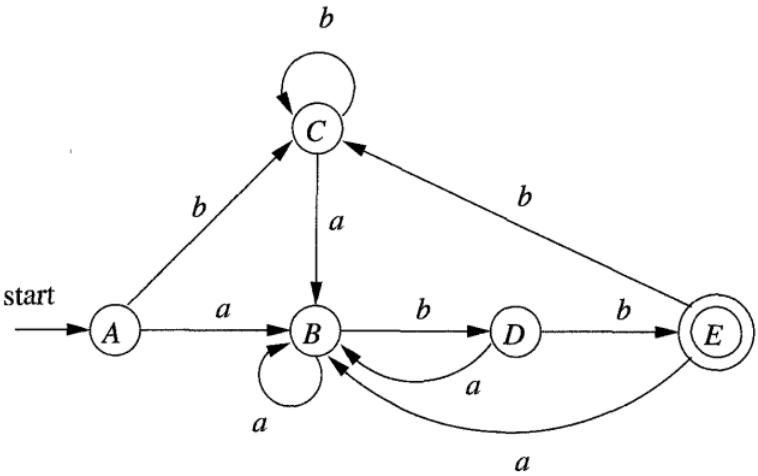


Figure 3.36: Result of applying the subset construction to Fig. 3.34

Simulación de un AFN

```
1)  S = ε-closure(s0);
2)  c = nextChar();
3)  while ( c != eof ) {
4)          S = ε-closure(move(S, c));
5)          c = nextChar();
6)  }
7)  if ( S ∩ F != ∅ ) return "yes";
8)  else return "no";
```

Figure 3.37: Simulating an NFA