Lenguajes regulares

César Ignacio García Osorio
Área de Lenguajes y Sistemas Informáticos
Universidad de Burgos
Febrero, 1999

Conjuntos y expresiones regulares

- **Conjunto regular**: Cualquier conjunto de cadenas que se pueda formar mediante las operaciones de unión, concatenación y cierre.
- Las **expresiones regulares** se utilizan para denotar conjuntos regulares y se definen del siguiente modo recursivo (dado un alfabeto Σ:
  - ε la expresión regular que denota el conjunto {ε}
  - $\forall a \in \Sigma$, a es una expresión regular que designa al conjunto regular $\{a\}$
  - Si α y β son expresiones regulares que designan los conjunto regulares C_α y C_β respectivamente, entonces:
    - (α)|(β) es la expresión regular que designa al conjunto regular $C_\alpha \cup C_\beta$
    - (α)(β) es la expresión regular que designa al conjunto regular $C_\alpha \cdot C_\beta$
    - (α)* es la expresión regular que designa al conjunto regular $C_\alpha{}^*$
    - (α) es la expresión regular que designa al conjunto regular C_α
  - Φ es la expresión regular que representa la conjunto vacío (que es regular)
  - Ninguna otra cosa es un expresión regular.

Propiedades de las expresiones regulares (1)

- **Expresiones regulares equivalentes**: aquellas que aún siendo distintas representan el mismo lenguaje.
- Algunas propiedades de las e.r. en cuanto a su equivalencia son:
  - $r|s=s|r$ (la unión o selección es conmutativa)
  - $r|(s|t) = (r|s)|t$ (la selección es asociativa)
  - $(rs)t=r(st)$ (la concatenación es asociativa)
  - $r(s|t)=rs|rt$ (la concatenación es distributiva por la izquierda respecto de la unión)
  - $(r|s)t=rt|st$ (la concatenación es distributiva por la derecha respecto de la unión)

Propiedades de las expresiones regulares (2)

- $\varepsilon r=r\varepsilon =r$ (ε es el elemento neutro de la concatenación)
- $\Phi|r = r|\Phi = r$ (Φ es el elemento neutro de la unión)
- $(r^*)^*=r^*$ (el cierre es idempotente)
- $r^* = r^*r^* = (\varepsilon|r)^* = r^*|\varepsilon = r^*(r^*|\varepsilon) = (r|\varepsilon)r^* = \varepsilon|rr^* = \varepsilon|r^*r$
- $(r^*|s^*)^* = (r^*s^*)^* = (r|s)^* = (r^*s)^*r^* = r^*(sr^*)^*$
- $r(sr)^* = (rs)^*r$
- $\varepsilon^*=\Phi^*=\varepsilon$
- $r^+ = rr^* = r^*r$
- $r^*=r^+ \mid \varepsilon$
- $\Phi^+ = \Phi$
- $r\Phi = \Phi r = \Phi$
- $(r^*s)^* = \varepsilon \mid (r|s)^*s$
- $(rs^*)^* = \varepsilon \mid r(r|s)^*$
- $r^* \subset (r|t)^*$
- $r^n \subset r^*$
- si $r \subset t$ entonces $r^* \subset t^*$
- si $r_1 \subset t_1$ y $r_2 \subset t_2$ entonces $r_1r_2 \subset t_1t_2$

Definición regular sobre Σ

- Conjunto de definiciones de la forma:

$d_1 \rightarrow r_1$	donde r_1 es una expresión regular sobre Σ
$d_2 \rightarrow r_2$	donde r_2 es una expresión regular sobre $\Sigma \cup d_1$
$d_3 \rightarrow r_3$	donde r_3 es una expresión regular sobre $\Sigma \cup d_1 \cup d_2$
..... .	. ...
$d_i \rightarrow r_i$	donde r_i es una expresión regular sobre $\Sigma \cup \bigcup^{i}_{j=1} d_j$
..... .	. ...
$d_n \rightarrow r_n$	donde r_n es una expresión regular sobre $\Sigma \cup \bigcup^{n}_{j=1} d_j$

- Es decir, es una especie de gramática en la que no se permiten definiciones recursivas.

Autómatas (1)

- **Autómata finito determinista**: es una quintupla (Σ, Q, f, q_0, F), donde:
  - Σ es un **alfabeto de entrada**.
  - Q es un alfabeto de **estados**.
  - f es la **función de transición**:
    $$f : Q \times \Sigma \rightarrow Q$$
  - q_0 es el **estado inicial**.
  - $F \subset Q$ es el conjunto de **estados finales** o **estados de aceptación**

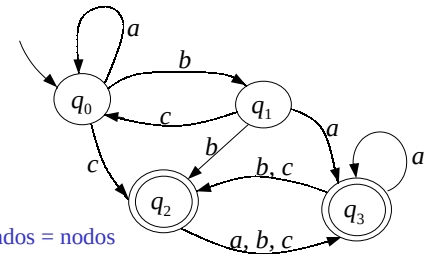
Autómatas (2)

- **Autómata finito no determinista**: es una quintupla (Σ, Q, f, q_0, F), donde:
  - Σ es un **alfabeto de entrada**.
  - Q es un alfabeto de **estados**.
  - f es la **función de transición**:
    $$f : Q \times (\Sigma \cup \varepsilon) \rightarrow P(Q)$$
  - q_0 es el **estado inicial**.
  - $F \subset Q$ es el conjunto de **estados finales** o **estados de aceptación**

Autómatas (3)

- La parte fundamental de un autómata finito es su función de transición por eso es habitual representarlo mediante una tabla que da dicha función o mediante un grafo

	a	*b*	*c*
$\rightarrow \mathbf{q_0}$	q_0	q_1	q_2
$\mathbf{q_1}$	q_3	q_2	q_2
$\mathbf{(q_2)}$	q_3	q_3	q_3
$\mathbf{(q_3)}$	q_3	q_2	q_2



estados = nodos

transiciones = arcos

estados finales = nodo con doble circulo

estado inicial = nodo con flecha de entrada

Autómatas (4)

- **Extensión de *f* a palabras**: la función f toma como argumentos un estado y un carácter, pero se puede definir una función que tome un estado y una palabra:

$\tilde{f}: Q \times \Sigma^* \rightarrow Q \qquad e \in \Sigma \quad x \in \Sigma^*$

$\tilde{f}(q, e) = f(q, e) \qquad \tilde{f}(q, ex) = \tilde{f}(f(q,e), x)$

La función $\tilde{f}(q, w)$, da el estado al que se puede llegar partiendo de q y siguiendo las transiciones según w.

- El **lenguaje reconocido** por un autómata $M = (\Sigma, Q, f, q_0, F)$ se define como:

$L(M) = \{w \in \Sigma^*: \tilde{f}(q_0, w) \in F\}$ AFD

$L(M) = \{w \in \Sigma^*: \tilde{f}(q_0, w) \cap F \neq \varnothing\}$ AFND

Simulación de un AFD

$s \leftarrow q_1$;
$c \leftarrow$ *sgtecar*();
while $c \neq$ e.o.f do
begin
  $s \leftarrow$ *mueve*(s,c);
  $c \leftarrow$ *sgtecar*();
end
if $s \in F$ then return "SI" else return "NO"

mueve(s,c) es una función que implementa la función de transición, dado un estado y un carácter nos devuelve el estado siguiente

sgtecar() es una función que accede al siguiente carácter de la cadena de entrada

Simulación de un AFND

$S \leftarrow$ *cerr*-ε($\{s_0\}$);
$c \leftarrow$ *sgtecar*();
while $c \neq$ e.o.f do
begin
  $S \leftarrow$ *cerr*-ε(*mueve*(S,*c*));
  $c \leftarrow$ *sgtecar*();
end
if $S \cap F \neq \varnothing$ then return "SI" else return "NO"

mueve(s,c) función de transición

sgtecar() accede al siguiente carácter de la entrada

cerr-ε (s) conjunto de estados a los que se puede llegar a partir de ***s*** siguiendo arcos etiquetados con **ε**

Paso de un AFND a un AFD

- cerr-ε(q) = $\{\tilde{f}(q, \varepsilon^n)\} \cup \{q\}$. Es decir el conjunto de estados a los que puedo llegar a partir del estado q mediante transiciones ε (sin consumir entrada)
- cerr-ε(T)= $\bigcup_{q \in T}$ cerr-ε(q)
- **Algoritmo: Construcción de subconjuntos**

D← *cerr*-ε(q_0);
while *haya un estado* T *sin marcar en* D do
  marcar T;
  for *cada símbolo de entrada* $a \in \Sigma$ do
    U ← *cerr*-ε(*mueve*(T, *a*));
    if U∉ D then *añadir* U *sin marcar a* D
    tranD[T, a] ← U

AFND(Σ, Q, f, q_0, F) →AFD(Σ,D, tranD, cerr-ε(q_0), **F**) **F**=$\{U \in D: U \cap F \neq \varnothing\}$

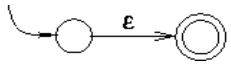
Teoremas de análisis y síntesis

- **Teorema de síntesis**: Todo conjunto regular es un lenguaje aceptado por un reconocedor finito.
  - **e.r** → AFD
  - Procedimientos:
    - Método de Thompson
    - Método de las derivadas
    - Método de Aho-Sethi-Ullman
- **Teorema de análisis**: Todo lenguaje aceptado por un reconocedor finito es un un conjunto regular.
  - AFD → **e.r**
  - Procedimientos:
    - Método 1
    - Método de las ecuaciones características

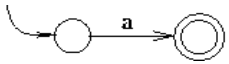
Método de Thompson (1)

1 Φ

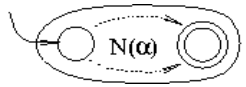
2 ε



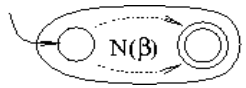
3 a



4 α

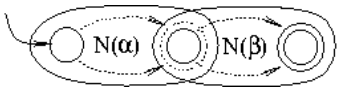


5 β

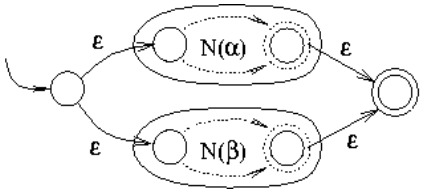


Método de Thompson (2)

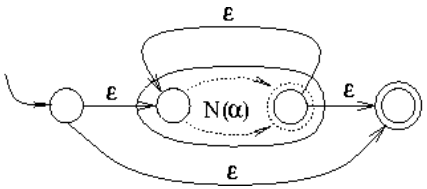
6 $(\alpha)(\beta)$



7 $(\alpha) \mid (\beta)$



8 $(\alpha)^*$



Método de las derivadas (1)

- Consideremos una expresión regular α, que designa un conjunto regular L_α, y consideremos el conjunto L_α^e formado por todas las cadenas de L_α que empiezan por un determinado símbolo *e*. Definimos el cociente izquierdo de L_α entre *e*, denotado $L_\alpha \backslash e$, como el conjunto resultante de suprimir *e* en todas las cadenas de L_α^e.

$$L_\alpha \backslash e = \{w : ew \in L_\alpha\}$$

y definimos la **derivada de α respecto al símbolo *e***, denotado $D_e(\alpha)$, como la expresión regular de $L_\alpha \backslash e$.

Método de las derivadas (2)

- Algunas propiedades de las derivadas así definidas son:
  - $D_{e_1}(e_2) = \varepsilon$ si $e_1=e_2$ $\quad\Phi$ si $e_1 \neq e_2$
  - $D_e(\varepsilon) = D_e(\Phi) = \Phi$
  - $D_\varepsilon(\alpha)=\alpha$
  - $D_{ew}(\alpha)=D_w[D_e(\alpha)]$ $\quad D_{we}(\alpha)=D_e[D_w(\alpha)]$
  - $D_e(\alpha|\beta)=D_e(\alpha)|D_e(\beta)$
  - $D_e(\alpha\beta)=D_e(\alpha)\ \beta|\delta(\alpha)D_e(\beta)$ donde $\delta(\alpha)= \emptyset$ si $\varepsilon \notin L_\alpha$ ε si $\varepsilon \in L_\alpha$
  - $D_e(\alpha^*)=D_e(\alpha)\alpha^*$

Método de las derivadas (3)

Algoritmo de síntesis (Brozozowaki, 1962)

- Para construir un reconocedor del lenguaje denotado por la expresión regular, se puede seguir el siguiente procedimiento
  - Calcular $\{D_w(\alpha): w \in \Sigma^*\}$
  - El estado inicial es $q_1=\alpha$; los otros son las diferentes $D_w(\alpha)$
  - La función de transición es $f(\alpha, e)= D_e(\alpha)$; $f(D_w(\alpha), e)= D_{we}(\alpha)$;
  - El conjunto de estado finales es
    $$F=\{D_w(\alpha): \varepsilon \in L(D_w(\alpha))\}$$

Método de Aho-Sethi-Ullman (1)

- Construir la e.r. ampliada $\alpha\$$ $\quad \$ \notin \Sigma$
- Obtener el árbol sintáctico de $\alpha\$$
- Responder a las preguntas:
  - ¿Que letras pueden aparecer como primera letra de la e.r. representada por α cuáles como últimas?
  - ¿Supuesto que yo se que letra acaba de aparecer, que letras pueden venir a continuación en las palabras representadas por α?
  - La idea es tratar de numerar la posición de las letras que aparecen en la e.r. Una vez que tengo el árbol puedo hacer una numeración, lo cual me da una ordenación parcial. Se establece una equivalencia entre los **estados significativos** del autómata y las posiciones del árbol (nodos hoja).

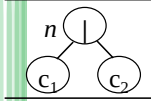
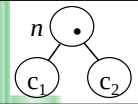
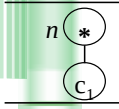
Método de Aho-Sethi-Ullman (2)

- Permite construir directamente un AFD a partir de una *expresión regular aumentada* ($\alpha\$ \quad \$ \notin \Sigma$).
- Primero se construye un árbol sintáctico *T* para $\alpha\$$ con las posiciones de las hojas numeradas y después se calculan cuatro funciones: *anulable*, *primera-pos*, *última-pos* y *siguiente-pos* haciendo recorridos sobre *T*.
- Por último se construye el AFD a partir de la función *siguiente-pos*. Las funciones *anulable*, *primera-pos* y *última-pos* se definen sobre los nodos del árbol sintáctico y se usan para calcular *siguiente-pos*, que está definida en el conjunto de posiciones de las hojas en el árbol.

Método de Aho-Sethi-Ullman (3)

- La noción de una posición concordando con un símbolo de entrada será definida en cuanto a la función *siguiente-pos* en posiciones del árbol sintáctico. Si *i* es una posición, *siguiente-pos*(*i*) es el conjunto de posiciones *j* tales que hay alguna cadena de entrada ...*cd*... Tal que *i* corresponde a esta aparición de *c* y *j* a esta aparición de *d*.
- Para calcular la función *siguiente-pos*, es necesario conocer qué posiciones pueden concordar con el primer o último símbolo de una cadena generada por una determinada subexpresión de una expresión regular. Para esto es también necesario conocer qué nodos son las raíces de las subexpresiones que generan lenguajes que incluyen la cadena vacía (nodos *anulables*)

Método de Aho-Sethi-Ullman (4)

Nodo	*anulable*(*n*)	*primera-pos*(*n*)	*última-pos*(*n*)
n es una hoja con símbolo ε	**true**	∅	∅
n es una hoja con la posición *i*	**false**	{*i*}	{*i*}
	*anul*(c_1)∨*anul*(c_2)	*pripos*(c_1)∪*pripos*(c_2)	*últpos*(c_1)∪*últpos*(c_2)
	*anul*(c_1)∧*anul*(c_2)	**if** *anul*(c_1) **then** *pripos*(c_1)∪*pripos*(c_2) **else** *pripos*(c_1)	**if** *anul*(c_2) **then** *últpos*(c_1)∪*últpos*(c_2) **else** *últpos*(c_2)
	**true**	*pripos*(c_1)	*últpos*(c_1)

Método de Aho-Sethi-Ullman (5)

- La función *siguiente-pos*(*i*) indica qué posiciones pueden seguir a la posición *i* en el árbol sintáctico. Dos reglas definen todas las formas en que una posición puede seguir a otra:
  - Si *n* es un *nodo-cat* con hijo izquierdo c_1 e hijo derecho c_2, e *i* es una posición dentro de *última-pos*(c_1), entonces todas las posiciones de *primera-pos*(c_2) están en *siguiente-pos*(*i*).

    $\forall i \in$ *última-pos*(c_1) *siguiente-pos*(*i*) $\supset$ *primera-pos*(c_2)
  - Si *n* es un *nodo-ast*, e *i* es una posición dentro de *última-pos*(*n*), entonces todas las posiciones de *primera-pos*(*n*) están en *siguiente-pos*(*i*)

    $\forall i \in$ *última-pos*(c_1) *siguiente-pos*(*i*) $\supset$ *primera-pos*(c_1)

Método de Aho-Sethi-Ullman (6)

- **Algoritmo para obtener el AFD a partir de *siguiente-pos***

```
D← primera-pos (raíz);
while haya un estado T sin marcar en D do
    marcar T;
    for cada símbolo de entrada a∈ Σ do
            U ← ⋃ siguiente-pos(p);
                p∈T   p etiqueta una hoja con símbolo a
            if (U∉ D) ∧ (U≠∅) then añadir U sin marcar a D
            tranD[T, a] ← U
    end
end
```

AFD(Σ,D, tranD, *primera-pos*(*raíz*), **F**) **F**={U∈ D: *p* ∈ U y *p* la posición de $}

Demostración del teorema de análisis (1)

- Sea un AFD (Σ, Q, f, q_1, F), con n estados, $Q=\{q_1, q_2, ..., q_n\}$, con estados finales $F=\{q_{f_1}, q_{f_2}, ..., q_{f_r}\}$, el lenguaje aceptado es:

$$L(R)=\{w \mid \tilde{f}(w, q_1) \in F\}$$

o de forma equivalente

$$L(R)= \cup\{R_{1j} \mid q_j \in F\}$$

con $R_{ij}=\{x \mid \tilde{f}(x, q_i)= q_j\}$ (conjunto de cadenas que llevan q_i del estado al estado q_j). Basta entonces demostrar que los R_{ij} son conjuntos regulares.

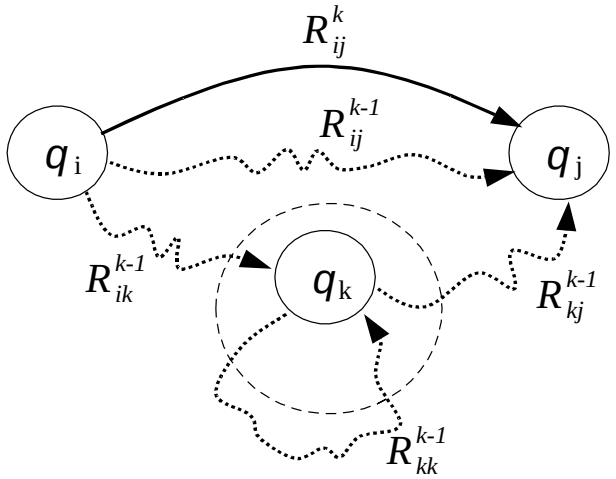
Demostración del teorema de análisis (2)

- Se define R^k_{ij} $(0 \le k \le n)$ como el conjunto de cadenas que llevan del estado q_i al estado q_j sin pasar por ningún estado q_l tal que $l > k$.
- Vamos a demostrar por inducción que los R^k_{ij} son conjuntos regulares:
  - Para k=0, esta claro que R^0_{ij} es regular
  - Supuesto que R^{k-1}_{ij} es regular se tiene que:

$$R^k_{ij} = R^{k-1}_{ij} \cup R^{k-1}_{ik} (R^{k-1}_{kk})^* R^{k-1}_{kj}$$

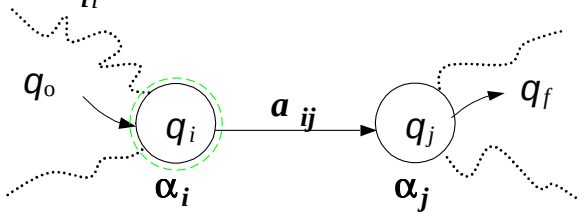
por tanto, R^k_{ij} es regular para todo k, y, en particular, $R_{ij} = R^n_{ij}$ es regular. Y $L(R)$ es regular

Demostración del teorema de análisis (3)



Método de las ecuaciones características (1)

- La ecuación característica de un estado q_i representa el conjunto de cadenas aceptadas por el estado q_i



$$\alpha_i = \sum_{j \in S(q_i)} a_{ij} \alpha_j \ (+ \varepsilon)$$

$S(q_i)$ representa el conjunto de índices j para los cuales q_j es un estado siguiente de q_i

Método de las ecuaciones características (2)

- Otra forma de definir la ecuación característica de un estado es:

$$\alpha_i = \sum_{j \in S(q_i)} a_{ij}\alpha_j + \sum_{k \in S_f(q_i)} b_k + \varphi(q_i)$$

- Donde
  - a_{ij}, $b_k \in \Sigma$,
  - $S(q_i)$ representa el conjunto de argumentos j para los cuales q_j es un estado siguiente de q_i ,
  - $S_f(q_i) \subset S(q_i)$ son sólo aquellos argumentos j para los cuales q_j es un estado siguiente de q_i y además $q_j \in F$
  - $\varphi(q_i) = \varepsilon$ si $q_i = q_0$ y $q_0 \in F$

Método de las ecuaciones características (3)

- Una vez que se obtiene los sistemas de ecuaciones para cada estado, se deben intentar resolver las ecuaciones de cada estado, de manera que tengan la forma $x_i = Ax_i + B$ donde $\varepsilon \notin A$, $x_i \notin B$, que es a lo que se denomina *ecuación característica*. Y se resuelve usando el resultado que da el lema de Arden.
- **Lema de Arden**: Una ecuación de la forma $X = AX + B$, donde $\varepsilon \notin A$ y $B \neq \emptyset$ tiene una solución única $X = A^*B$.

Lema de bombeo (1)

- Para todo lenguaje L regular existe una constante n, tal que para toda palabra $z \in L$ ($|z| \geq n$) existen u, v, w, $z = uvw$, $|uv| \leq n$, $|v| > 0$ y $(uv^*w) \in L$
- El lema de bombeo puede utilizarse para demostrar que un lenguaje determinado no es regular. Para ello basta demostrar que dicho lenguaje no cumple el lema de bombeo.
- El lenguaje L no es regular si para toda constante n existe una palabra $z \in L$ ($|z| \geq n$), tal que en toda descomposición de z en la forma $z = uvw$, con $|uv| \leq n$, $|v| > 0$ existe un entero i tal que $(uv^iw) \notin L$.

Lema de bombeo (2)

- **Demostrar que el lenguaje $L = \{a^n b^n : n \in \mathbb{N}^+\}$ no es regular:**
- El lenguaje $\{a^n b^n : n \in \mathbb{N}\}$ no es regular y podemos usar el lema de bombeo para demostrarlo. Siguiendo los siguientes pasos:
  - 1.- Se elige un valor arbitrario $N = k$
  - 2.- Seleccionamos una palabra z con $|z| \geq N$, por ejemplo $a^k b^k$
  - 3.- Se descompone z en la forma uvw, con $|uv| \leq N$ y $|v| > 0$, de todas las formas posibles. Las posibles descomposiciones son únicamente de la forma: $u = a^{k-l-h}$, $v = a^l$, $w = a^h b^k$ con $l > 0$, por tanto tomando $i = 2$, la palabra ya no pertenecería al lenguaje. Ya que tendríamos que el número de *aes* sería mayor que el número de *bes*.
- Si se hubiera escogido una constante $N = 2k$, y una palabra z de longitud igual a N, por ejemplo $z = a^k b^k$ habría que haber considerado tres posibles formas de descomponer la z, a saber:

  a) $u = a^{k-l-h}$, $v = a^l$, $w = a^h b^k$
  b) $u = a^{k-l}$, $v = a^l b^{k-h}$, $w = b^h$
  c) $u = a^k b^{k-l-h}$, $v = b^l$, $w = b^h$

Minimización de autómatas (1)

- En Postscript