







Expresiones regulares

Ing. Karina Cabrera Chagoyan

- 
- 
- El origen de las expresiones regulares surge de la teoría de autómatas y la teoría de lenguajes formales, ambas parte de la ciencias computacionales teórica. Este campo estudia los modelos computacionales (autómata) y la manera de describir y clasificar los lenguajes formales. Un lenguaje formal puede ser especificado de varias maneras, tales como :
 - Cadenas producidas por alguna gramática formal
 - Cadenas producidas por expresiones regulares
 - Cadenas aceptadas por algunos autómatas como las maquinas de Turing o autómatas de estado finito

- 
- 
- A las expresiones regulares frecuentemente se les llaman patrones, ya que son expresiones que describen a un conjunto de cadenas. Frecuentemente son usadas para dar una descripción concisa de un conjunto, sin tener que listar todos sus elementos
 - Las expresiones regulares pueden ser expresadas en términos de la teoría de lenguajes formales. Consisten de constantes y operadores que denotan el conjunto de cadenas y operaciones sobre estos conjuntos, respectivamente. Dado un alfabeto las siguientes constantes son definidas:
 - Conjunto vacío: $L(\emptyset)$ denota el conjunto $\{\emptyset\}$
 - Cadena vacía: $L(\epsilon)$ denota el conjunto $\{\epsilon\}$
 - Carácter del alfabeto: $L(a)$, a elemento de denota el conjunto $\{ "a" \}$



Expresiones regulares

- Definen las cadenas validas de un lenguaje mediante una descripción algebraica (formula)
- Los lenguajes que pueden describirse mediante expresiones regulares se denominan **lenguajes regulares**
- Se utilizan como lenguaje de entrada en muchos sistemas de proceso de cadenas

Operaciones básicas en expresiones regulares

► Selección de alternativas (unión)

Si r y s son expresiones regulares, entonces $r \mid s$ es una expresión regular que define cualquier cadena que concuerda con r o con s . En términos de lenguajes decimos que $r \mid s$ es la unión de los lenguajes de r y s , o $L(r \mid s) = L(r) \cup L(s) = \{ \alpha\beta \mid \alpha \text{ esta en } r \text{ o } \beta \text{ esta en } s \}$.

Ejemplo:

$$\text{a) } L(a \mid b) = L(a) \cup L(b) = \{ a, b \}$$

$$\text{b) } L(a \mid \varepsilon) = \{ a, \varepsilon \}$$

$$\text{c) } L(a \mid b \mid c \mid d) = \{ a, b, c, d \}$$

Concatenación

- La concatenación de dos expresiones regulares ***r*** y ***s*** se escribe como ***rs*** (yuxtaposición) y corresponde a cualquier cadena que sea la concatenación de dos cadenas, con la primera de ellas correspondiendo a ***r*** y la segunda a ***s***. Por ejemplo la expresión regular ***ab*** corresponde a la cadena { *ab* }, mientras que la expresión regular ***(a | b)c*** corresponde a las cadenas { *ac*, *bc* }. De esta forma la operación de concatenación para expresiones regulares se puede definir como $L(rs) = L(r) L(s) = \{ \alpha \beta \mid \alpha \text{ esta en } r \text{ y } \beta \text{ esta en } s \}$.

ejemplo:

$$a) L(a \mid b) c = L(a \mid b) L(c) = \{ a, b \} \{ c \} = \{ ac, bc \}$$

$$\begin{aligned} b) L(ab \mid c)(d \mid ef) &= L(ab \mid c)L(d \mid ef) \\ &= \{ ab, c \} \{ d, ef \} = \{ abd, abef, cd, cef \} \end{aligned}$$

Repetición (cerradura de Kleene)


- Se escribe r^* , donde r corresponde a la expresión regular. La expresión regular r^* corresponde a cualquier concatenación finita de cadenas, cada una de las cuales corresponde a r . Por ejemplo, la expresión regular a^* corresponde a las cadenas $\{\epsilon, a, aa, aaa, \dots\}$, (concuerda con ϵ por que ϵ es la concatenación de ninguna cadena concordante con a). En términos de lenguaje podemos decir que:

$$S^* = \bigcup_{n=0}^{\infty} S^n$$

Donde $S^n = S \dots S$ es la concatenación de S n veces ($S^0 = \{\epsilon\}$).

ejemplo:

- a) $L(a \mid b)^* = \{a, bb\}^* = \{\epsilon, a, bb, aa, abb, bba, bbbb, aaa, abba, abbbb, \dots\}$
- b) $L(a \mid b^*) = \{a, b^*\} = \{\epsilon, a, b, bb, bbb, \dots\}$
- c) $L(ab \mid c)^* = \{ab, c\}^* = \{\epsilon, ab, c, abab, abc, cab, cc, ababab, \dots\}$

- 
- Para evitar los paréntesis se asume que la *repetición* tiene la precedencia mas alta, luego la *concatenación* y al final la *unión*. Si no existiera ambigüedad los paréntesis pueden ser omitidos . Por ejemplo , $(ab)c$ se escribe como abc y $a \mid (b(c^*))$ puede ser escrito como $a \mid bc^*$.
 - De manera que los lenguajes regulares deben su nombre al hecho de que presentan “regularidades” o repeticiones de los mismo componentes, como por ejemplo el lenguaje L_1 :

$$L_1 = \{ab, abab, ababab, abababab, \dots\}$$



- Una expresión regular se construye a partir de expresiones regulares mas simples utilizando un conjunto de reglas definitorias (*operaciones básicas*). Cada expresión regular r representa un lenguaje $L(r)$. Las reglas de definición especifican como se forma $L(r)$ combinando de varias maneras los lenguajes representados por las subexpresiones de r .
- Se dice que un lenguaje designado por una expresión regular es un conjunto regular. Es importante recordar que la *especificación* de una *expresión regular* es un ejemplo de *definición recursiva*.

Definición

- Es una forma de simplificación, dando un nombre a las expresiones regulares y definiendo nuevas expresiones regulares utilizando dichos nombres como si fueran símbolos. Por ejemplo, podríamos desarrollar una *expresión regular* para una secuencia de uno o mas dígitos, generando inicialmente una *definición regular* para un dígito.

dígito = 0 | 1 | 2 | ... | 9 (*definición regular*)


dígito dígito* (*expresión regular para números enteros sin signo*)

- 
- 
- A partir de aquí ya se puede elaborar la una definición de expresiones regulares para la identificación de los componentes de un lenguaje determinado. Los tokens de lenguajes de programación tiende a caer dentro de varias categorías limitadas que son bastante estandarizadas, como palabras reservadas, símbolos especiales, identificadores y literales (numérica/cadena).
 - Por ejemplo los identificadores en la mayoría de los lenguajes de programación es el conjunto de cadenas de *letras* y *dígitos* que empiezan con una *letra*. Para ello generamos las *definiciones regulares* para *letra* y *digito*:

letra = a | b | c | ... | z | A | B ... | Z

digito = 0 | 1 | 2 | ... | 9

identificador = letra (letra | digito)*

- 
- Los números con signo son literales numéricas constituidas por el punto decimal después de un dígito y por una literal que represente la parte del exponente. Con esta información desarrollamos las definiciones regulares y las expresiones regulares necesarias:

$$((+|-|\epsilon)\text{digito digito}^*) ((.\text{ digito})|\epsilon) ((E(+|-)|)\text{digito})|\epsilon$$

entero = $(+|-|\epsilon)\text{ digito digito}^*$

fracción = $(.\text{ digito})|\epsilon$

exponente = $(E(+|-|\epsilon)\text{ digito})|\epsilon$

numero = $\text{digito fracción exponente}$

- Esta definición establece que fracción es un punto decimal seguido de uno o mas dígitos, o esta ausente. Un exponente, que es E seguido de un signo + ó - ó ausente, seguidos de uno o mas dígitos, o la ausencia de exponente. Considere que, como mínimo debe existir un dígito después del punto. De manera que numero concuerda con 1 y con 1.0

Abreviaturas

Uno o mas casos

- el operador unitario postfijo $+$ significa “uno o mas casos de”, de manera que la expresión regular r que designa al lenguaje $L(r)$, entonces r^+ es una expresión que designa al lenguaje $L(r)^+$. Así la expresión regular a^+ representa al conjunto de todas las cadenas de una o mas a . El operador $+$ tiene la misma precedencia que la repetición, las identidades algébricas son $r^* = r \mid \varepsilon$ y $r^+ = r r^*$.


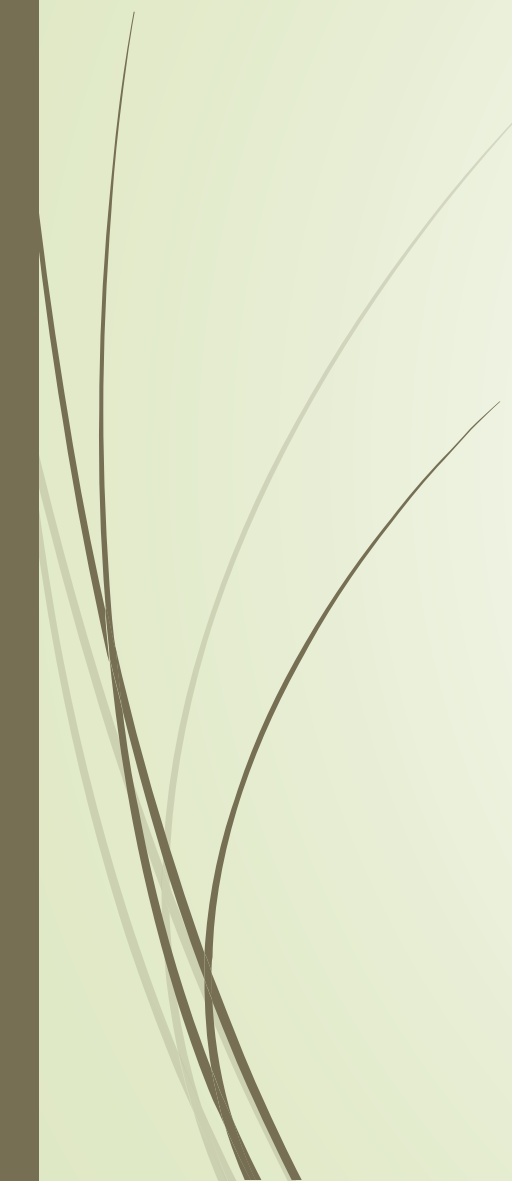




Cero o un caso

- el operador unitario postfijo **?** significa “cero o un caso de”. La notación $r?$ es la abreviatura de $r \mid \epsilon$. Si r es una expresión regular, entonces $(r)?$ es una expresión regular que designa el lenguaje $L(r) \cup \{\epsilon\}$.

Clases de caracteres

- una clase abreviada de caracteres como **[a – z]** designa la expresión regular $a \mid b \mid c \mid \dots \mid z$.

- 
- 
- A menudo en la descripción de los tokens de lenguaje de programación utilizamos expresiones regulares, algunas cadenas se pueden definir mediante varias expresiones regulares diferentes, Por ejemplo, cadenas tales como **if** y **while** podrían ser identificadores o palabras reservadas.
 - Una definición de lenguaje de programación debe establecer cual interpretación se observará, y las expresiones regulares por si mismas no pueden hacer esto. En realidad, una definición de lenguaje debe proporcionar reglas de no ambigüedad que expliquen cual significado es el conveniente para cada uno de tales casos.

- 
- 
- De esta manera el reconocimiento de componentes léxicos podría realizarse partiendo del conjunto de cadenas dadas por *expresiones regulares*. Por ejemplo:
 - **if** = *if*
 - **else** = *else*
 - **oprelacion** = < | <= | = | <> | > | >=
 - **identificador** = letra (letra | digito)*
 - **entero** = (+ | -)? digito+
 - **real** = (+ | -)? digito+ (. digito+)? (E (+ | -)? digito+)?

Expresión regular	Descripción	Cadenas que lo forman
a^*	Cero o mas a	$\epsilon, a, aa, aaa, aaaa, \dots$
a^+	Una o mas a	$a, aa, aaa, aaaa, \dots$
a^*b	Cero o mas a seguida de una b	$b, ab, aab, aaab, \dots$
ab^*	Una a seguida de cero o mas b	$a, ab, abb, abbb, abbb, \dots$
a^+b	Una o mas a seguida de una b	$ab, aab, aaab, aaaab, \dots$
ab^+	Una a seguida de una o mas b	$ab, abb, abbb, abbbb, \dots$
$(ab)^*$	Cero o mas ab	$\epsilon, ab, abab, ababab, \dots$
$(ab)^+$	Una o mas ab	$ab, abab, ababab, \dots$
$a \cup b$	Una a o una b	a, b
$a^* \cup b^*$	Cero o mas a o cero o mas b	$\epsilon, a, aa, aaa, \dots, b, bb, bbb, \dots$
$a^+ \cup b^+$	Una o mas a o una o mas b	$a, aa, aaa, \dots, b, bb, bbb, \dots$
$a^*b \cup b^*a$	Cero o mas a seguida de una b o cero mas b seguida de una a	$b, a, aab, aaab, \dots, a, ba, bba, bbba, \dots$
$(ab)^* \cup (ba)^*$	Cero o mas ab o cero o mas ba	$\epsilon, ab, abab, ababab, \dots, ba, baba, bababa, \dots$
Ing. Karina Cabrera Chagoyan		