**TEMA 2:**

# Lenguajes y Expresiones Regulares

Autómatas y Lenguajes Formales.
Grado en Informática (2° curso).

Dpto. de Ingeniería de la Información y las Comunicaciones.

UNIVERSIDAD DE MURCIA

# Índice general

# 1. Concepto de expresión regular y lenguaje regular

En Aritmética se usan fórmulas con constantes, variables y operadores numéricos, tal como $(2x+y)\times z$. Podría decirse que la expresión aritmética $(2x+y)\times z$ describe un conjunto infinito de números, que son los que se obtienen al evaluar la expresión con diferentes valores numéricos para las variables $x, y, z$.
En Teoría de Lenguajes Formales, donde se trata con cadenas en lugar de números, se usan otro tipo de fórmulas o expresiones algebraicas llamadas **expresiones regulares**. Cada expresión regular describe formalmente a un conjunto de cadenas que forman un **lenguaje regular**.

Por ejemplo, $0^*10^*$ es una expresión regular cuyo **significado o valor semántico** es el lenguaje que describe y consiste en el conjunto de cadenas formadas por cero o más repeticiones del símbolo 0 (indicado por la subexpresión $0^*$ al principio del patrón), seguidas de un 1 (lo indica la subexpresión 1 del centro) y seguidas de otra secuencia opcional de $0's$ (indicado por $0^*$ al final del patrón). Expresado de manera formal se dice que la expresión regular $0^*10^*$ describe al lenguaje infinito $\{0^i10^j \mid i, j \geq 0\}$ y por el hecho de poder describirse este lenguaje mediante una expresión regular se dice que es un **lenguaje regular**.

El **convenio de notación** matemática establecido hasta ahora es usar por defecto la letra $V$ para referirnos a un alfabeto, las primeras letras minúsculas $a, b, \ldots$ o dígitos para los símbolos de un alfabeto, las últimas letras minúsculas $w, x, y, z$ para cadenas arbitrarias, y la letra $L$ u otra mayúscula $A, B, \ldots$ para referirnos a un lenguaje, en todos los casos con posibles subíndices. Aunque no es obligatorio llamarlos de esa forma, ese convenio ayuda a distinguir cada objeto por el nombre. De la misma manera, vamos a usar la letra $R$ (con o sin subíndices) para hacer referencia a una expresión regular arbitraria y usamos $L(R)$ para denotar el lenguaje que describe la expresión regular $R$. Usaremos "ER" como abreviatura de "expresión regular" (en inglés se usa como abreviatura el término *regex*, de regular expression).

## 1.1. Sintaxis y semántica de las expresiones regulares

La **sintaxis de las expresiones regulares** establece la manera de escribir correctamente una ER (como en las expresiones aritméticas) y la **semántica de las expresiones regulares** establece el significado o valor semántico de la ER, que es el lenguaje $L(R)$ que describe la expresión regular $R$. La sintaxis y la semántica de las ER se puede definir conjuntamente de forma recursiva. Primero se introducen casos que definen expresiones regulares básicas, que son las que no tienen operadores. Luego se introducen los casos que definen ER con operadores a partir de ER más simples.

**Definición 1 (Expresión Regular y Lenguaje descrito)** *Sea un alfabeto* $V = \{a_1, \ldots, a_k\}$ *y los operadores de* concatenación *(*$\circ$*, a veces se omite),* unión o alternancia $|$, clausura $*$ *y* paréntesis de agrupación ( ). *Definimos de forma recursiva una* ***expresión regular*** *y el* ***lenguaje descrito*** *por la* ER *con alfabeto* $V$ *como:*

***ER básicas (sin operadores):***

1. SINTAXIS*: la constante* $\varnothing$ *es una* ER.

   SEMÁNTICA*: el lenguaje descrito es* $L(\varnothing) = \varnothing$.

2. SINTAXIS*: la constante* $\lambda$ *es una* ER.

   SEMÁNTICA*: el lenguaje descrito es* $L(\lambda) = \{\lambda\}$.

3. SINTAXIS*: si* $a_i \in V$ *entonces la constante* $a_i$ *es una* ER.

   SEMÁNTICA*: el lenguaje descrito es* $L(a_i) = \{a_i\}$.

   *Ejemplo: siendo* $V = \{a, b\}$ *se tiene que* $a$ *y* $b$ *son* ER *y describen los lenguajes* $L(a) = \{a\}$ *y* $L(b) = \{b\}$*, respectivamente.*

***ER con operadores:***

4. SINTAXIS *(**regla de concatenación**): si* $R_1$ *y* $R_2$ *son* ER *entonces* $\boxed{R_1 \circ R_2 = R_1 R_2}$ *es una* ER.

   SEMÁNTICA*: el lenguaje descrito es* $L(R_1 \circ R_2) = L(R_1) \circ L(R_2)$.

   *Ejemplo: de que* $R_1 = a$ *y* $R_2 = b$ *son* ER *(por caso base 3) se deduce que* $ab$ *también es una* ER *y el lenguaje descrito es* $L(ab) = L(a) \circ L(b) = \{a\} \circ \{b\} = \{ab\}$*. Por otra parte, de que* $R_1 = ab$ *y* $R_2 = b$ *son* ER *se deduce que* $R_1 \circ R_2 = abb$ *es una* ER *y* $L(abb) = \{abb\}$.

5. SINTAXIS *(**regla de unión**): Si* $R_1$ *y* $R_2$ *son* ER *entonces* $\boxed{R_1|\, R_2}$ *es una* ER.

   SEMÁNTICA*: el lenguaje descrito es* $L(R_1|\, R_2) = L(R_1) \cup L(R_2)$

   *Ejemplo: de que* $R_1 = a$ *y* $R_2 = b$ *son* ER *se deduce que* $a|b$ *también es una* ER *y* $L(a|b) = L(a) \cup L(b) = \{a\} \cup \{b\} = \{a, b\}$*. De que* $R_1 = abb$ *y* $R_2 = ab$ *son* ER *se deduce que* $R_1|R_2 = abb|ab$ *es una* ER *y* $L(abb|ab) = L(abb) \cup L(ab) = \{abb, ab\}$.

6. SINTAXIS *(**regla de clausura**): si* $R$ *es una* ER *entonces* $\boxed{R^*}$ *es una* ER

   SEMÁNTICA*: el lenguaje descrito es* $L(R^*) = (L(R))^*$.

   *Ejemplo: de que* $R = a$ *es una* ER *se deduce que* $a^*$ *también es una* ER *y* $L(a^*) = (L(a))^* = \{a\}^* = \{a^i \mid i \geq 0\}$.

7. SINTAXIS *(**regla de agrupación**): si* $R$ *es una* ER *entonces* $\boxed{(R)}$ *es una* ER

   SEMÁNTICA*: el lenguaje descrito es* $L((R)) = (L(R))$.

   *Ejemplo: de que* $R = abb$ *es una* ER *se deduce que* $(abb)$ *es una* ER *y* $L((abb)) = (L(abb)) =$ $(\{abb\}) \overset{\text{suprimir ( ) no necesario aquí}}{=} \{abb\}$.

   *Por otra parte, al ser* $(abb)$ *y* $aa$ ER *entonces por la regla de concatenación se deduce que también lo es* $(abb)^* \circ aa$.
   *Se tiene que* $L((abb)^* \circ aa) = L((abb)^*) \circ L(aa) = (L(abb))^* \circ L(aa) = \{abb\}^* \circ \{aa\} = \{(abb)^i aa \mid i \geq 0\}$

**Nota:** Las ER básicas $\varnothing$, $\lambda$, $a_i$ tienen el mismo nombre que los que usamos habitualmente para denotar al conjunto vacío, a la cadena vacía y a un símbolo $a_i$ de un alfabeto, respectivamente, pero son conceptos distintos. Aquí son expresiones regulares constantes y cada una describe un lenguaje finito.
El operador de **alternancia o unión** '|' de expresiones regulares se llama '+' o incluso '$\cup$' en la sintaxis teórica que se introduce algunos libros de la bibliografía, por ej. en la sintaxis de ER de JFLAP se usa '+' en lugar de '|'. Preferimos no usar el operador '+' para la unión de ER porque genera confusión con el operador '+' de clausura positiva de lenguajes o con el operador '+' de concatenación de cadenas en algunos lenguajes de programación como Java.

***Reglas de precedencia y asociatividad de operadores***

Las reglas sintácticas indicadas en la Definición 1 para definir las expresiones regulares de forma recursiva son ambiguas, en el sentido de que cuando se prescinde de los paréntesis no queda claro cuál es el orden de aplicación de operadores a las subexpresiones. Eso implica que en principio se podría dividir la expresión regular de distintas formas en subexpresiones más simples y eso afectaría al lenguaje descrito.

**Ejemplo 1** *En la expresión regular $ab^*$ con alfabeto $\{a, b\}$ hay un operador de concatenación implícito (se omite '$\circ$' cuando se escribe una ER concreta) y hay un operador de clausura. Entonces, ¿debemos entender que la ER $ab^*$ es equivalente a la ER $a \circ (b^*) = a(b^*)$, que es del tipo $R_1 \circ R_2$ con $R_1 = a, R_2 = b^*$, o debe entenderse que es $(ab)^*$, que es del tipo $R^*$ con $R = ab$? Lo correcto es considerar que $ab^* = a(b^*)$, porque la clausura afecta a la $b$ y luego se concatena $a$ con $b^*$. Se da por supuesto que la clausura tienen mayor precedencia que la concatenación. Por tanto el lenguaje que describe $ab^*$ es $L(ab^*) = L(a) \circ L(b^*) = \{a\} \circ \{b\}^* = \{ab^i \mid i \geq 0\}$. La expresión regular $(ab)^*$ no equivale a $ab^*$ porque describe otro lenguaje distinto, que es:*

$$L((ab)^*) = (L(ab))^* = \{ab\}^* = \{(ab)^i \mid i \geq 0\}$$

La **regla de precedencia de operadores** se establecen para evitar la ambigüedad en el significado de una ER cuando se prescinde de los paréntesis de agrupación.
La precedencia y asociatividad de los operadores de expresiones regulares es la misma que la precedencia y asociatividad de los correspondientes operadores de lenguajes. Así pues, los operadores listados de mayor a menor precedencia son: ( ) $*$ $\circ$ |
Este convenio de precedencia de operadores es análogo al de las expresiones aritméticas con paréntesis, potencia, producto y suma (de mayor a menor precedencia).

También se considera la **regla de asociatividad por la izquierda** de la concatenación y la unión. De esta forma, si tenemos una expresión regular formada por concatenación de 3 subexpresiones del tipo $R_1 \circ R_2 \circ R_3$, se supone que equivale a la ER con paréntesis $(R_1 \circ R_2) \circ R_3$. También vale esta regla para la unión, esto es, si tenemos $R_1|R_2|R_3$, se supone que equivale a la ER con paréntesis $(R_1|R_2)|R_3$.

**Ejemplo 2** *La expresión regular $ab^*c|(ca)^*$ es equivalente a la expresión con paréntesis $((a\,(b^*)\,)c\,)|\,(ca)^*$, donde no hay ambigüedad respecto a qué subexpresiones se aplica cada operador. Sin embargo, la ER $((a\,(b^*)\,)c\,)|\,(ca)^*$ resulta menos legible que $ab^*c|(ca)^*$, por eso se prescinde de los paréntesis teniendo en cuenta la precedencia y asociatividad de operadores.*

## 1.2. Lenguajes Regulares

**Definición 2 (Lenguaje Regular)** *Un lenguaje $L_r$ es un LENGUAJE REGULAR si y sólo si existe una expresión regular $R$ que lo describe, es decir, se cumple que $L_r = L(R)$.*

*Llamamos* $\boxed{\mathcal{L}_{reg}}$ *a la clase o conjunto que contiene a todos los lenguajes regulares.*

**Ejemplo 3** *Considerando el alfabeto $V = \{a, b, c\}$ se tiene que los lenguajes básicos y finitos $\{a\}, \{b\}, \{c\}$ son lenguajes regulares. También es regular el lenguaje finito $\{abb, ac, aabb\}$, porque puede describirse mediante la expresión regular $abb|ac|aabb$.*
*Todos los lenguajes mostrados en el tema 2 y los vistos hasta ahora en este tema son regulares. Por ejemplo, podemos afirmar que $\{(ab)^i \mid i \geq 0\}$ es un lenguaje regular porque se describe mediante la expresión regular $(ab)^*$ (como se ha comprobado en el Ejemplo 1).*

***Uso adecuado de los operadores de expresiones regulares y lenguajes***

Los operadores de unión, concatenación y clausura de expresiones regulares son semejantes a los operadores de unión, concatenación y clausura de lenguajes, respectivamente. La analogía viene de que una ER que combina otra (u otras) mediante un operador describe al lenguaje que resulta de aplicar el operador análogo a los sublenguajes descritos por cada subexpresión regular, como establece la semántica para los distintos tipos de expresiones regulares:

$$L(R_1|\, R_2) = L(R_1) \cup L(R_2), \quad L(R_1 \circ R_2) = L(R_1) \circ L(R_2), \quad L(R^*) = (L(R))^*$$

Pero hay que entender que los operadores son distintos en el sentido de que aplicados a expresiones regulares construyen otra ER más compleja y aplicados a lenguajes construyen otro lenguaje. Por eso, a pesar de la semejanza entre operadores de ER y operadores de lenguajes, hay que **evitar usar incorrectamente la notación**.

**Ejemplo 4** *Dando por supuesto que partimos del alfabeto $V_{bin} = \{0, 1\}$ entonces es **incorrecto** escribir $\{0^*\} \cup \{1\} \cup \{0^*\}$ para referirnos al lenguaje $\{0\}^* \cup \{1\} \cup \{0\}^*$. La expresión $\{0^*\} \cup \{1\} \cup \{0^*\}$ no denota ni un lenguaje ni una expresión regular. No es una expresión regular porque en la sintaxis de las expresiones regulares no se usan las llaves de conjuntos ni el operador $\cup$. No es un lenguaje con alfabeto $\{0, 1\}$ porque $0^*$ en $\{0^*\}$ no es una cadena, sino una ER y los elementos de los lenguajes son cadenas, no expresiones regulares. También **es incorrecto** escribir $\{0\}^*|\{1\}|\{0\}^*$, porque se está usando el operador $|$ de ER para operar con lenguajes. Es **correcto** $(\lambda|b^*a)$ porque $\lambda$ es una expresión regular básica según la sintaxis, además de usarse en otro contexto para hacer referencia a la cadena vacía.*

# 2. Análisis de expresiones regulares

Vamos a ver cómo analizar una expresión regular para comprobar si es sintácticamente correcta y para averiguar cuál es el *lenguaje regular* que describe.

Para **analizar sintácticamente un ER** hay que comprobar si se ajusta a *las reglas sintácticas* (Definición 1).

En el tema 4 introduciremos el concepto de gramática y veremos cómo se puede usar una gramática para comprobar algorítmicamente si una ER está bien formada. De momento podemos hacerlo manualmente intentando construir un **árbol de análisis sintáctico** para la ER, que refleja la estructura de la misma mostrando cómo se divide la ER en subexpresiones regulares más simples combinadas mediante operadores.
En el árbol se parte de la ER que se quiere analizar y en cada nivel se subdivide la ER según un caso de la definición recursiva, hasta llegar a las hojas que sólo están etiquetadas con expresiones regulares básicas u operadores. Si no se puede construir el árbol de análisis para cierta expresión entonces la expresión es sintácticamente incorrecta, o simplemente no es un expresión regular. En otro caso, leyendo las hojas del árbol de izquierda a derecha podemos comprobar que coincide con la expresión regular.

**Ejemplo 5** *En la Figura 1 mostramos el árbol de análisis para la* ER $ab^*c|(ca)^*$*, donde se indica a qué caso o regla sintáctica se ajusta la expresión regular que etiqueta a cada nodo. Mediante el árbol podemos comprobar cómo se construye la* ER *a partir de subexpresiones más simples y operadores. La expresión regular* $ab^*c|(ca)^*$ *es sintácticamente correcta y podemos comprobarlo observando que al concatenar las etiquetas de las hojas de izquierda a derecha se obtiene* $ab^*c|(ca)^*$.
*Sin embargo, la expresión* $^*c|(ca)^*$ *no es una expresión regular bien formada porque no respeta las reglas sintácticas (el operador * debe ir precedido de una* ER*).*

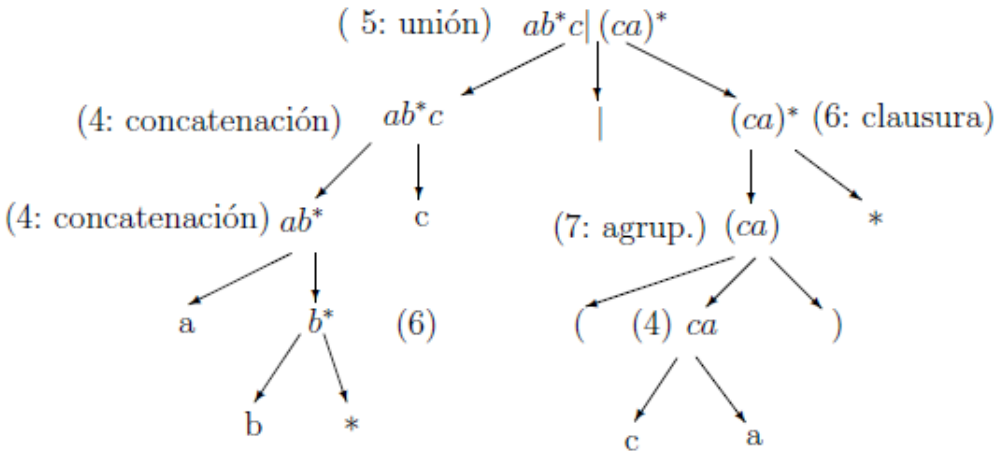



Figura 1: árbol de análisis de $ab^*c|(ca)^*$

Queremos ahora **analizar semánticamente una expresión regular** $R$ para saber qué tipo de cadenas describe, o lo que es lo mismo, para obtener el lenguaje $L(R)$. Este lenguaje se puede obtener a partir de $R$ *aplicando reglas semánticas* que definen al lenguaje descrito por $R$ (Definición 1) y teniendo en cuenta también las reglas de precedencia de operadores.

**Ejemplo 6** *Vamos a obtener* $L(ab^*c|(ca)^*)$*, como expresión que combina lenguajes básicos mediante operadores aplicando las reglas semánticas. Descomponemos la* ER $(ab^*c|(ca)^*)$ *en subexpresiones más simples (como en el árbol de la Figura 1) y se aplican las reglas semánticas para obtener el sublenguaje de cada subexpresión regular.*

1. *Para* $b^*$ *se tiene que* $L(b^*) = (L(b))^* = (\{b\})^* = \{b\}^*$
2. *Para* $ab^*$ *se tiene que* $L(ab^*) = L(a) \circ L(b^*) \overbrace{=}^{\text{(sustituyendo de 1.)}} \{a\} \circ \{b\}^*$
3. *Para* $ab^*c$ *se tiene que* $L(ab^*c) = L(ab^*) \circ L(c) \overbrace{=}^{\text{(de 2.)}} \{a\} \circ \{b\}^* \circ \{c\}$
4. *Para* $(ca)$ *se tiene que* $L((ca)) = (L(ca)) = (L(c) \circ L(a)) = (\{c\} \circ \{a\})$
5. *Para* $(ca)^*$ *se tiene que* $L((ca)^*) = L((ca))^* \overbrace{=}^{\text{(de 4.)}} (\{c\} \circ \{a\})^*$

*Finalmente* $ab^*c|(ca)^*$ *describe al lenguaje regular:*

$$L(ab^*c|(ca)^*) = L(ab^*c) \cup L((ca)^*) \overbrace{=}^{\text{(de 3. y 5.)}} \{a\} \circ \{b\}^* \circ \{c\} \cup (\{c\} \circ \{a\})^*$$

Una **manera simplificada de obtener L(R)** como expresión que combina lenguajes básicos mediante operadores de lenguajes es sustituyendo en la ER $R$ cada constante de alfabeto $a_i$ por el lenguaje que describe $\{a_i\}$, la constante $\lambda$ por el lenguaje $\{\lambda\}$ y el operador $|$ de unión de ER por el operador $\cup$ de unión de lenguajes.

**Ejemplo 7** *Según lo dicho anteriormente, la* ER $ab^*c|(ca)^*$ *la podemos traducir directamente a una expresión con lenguajes básicos y operadores de lenguajes:*

$$L(ab^*c|(ca)^*) = \{a\} \circ \{b\}^* \circ \{c\} \cup (\{c\} \circ \{a\})^*$$

### *De R a L(R) expresado de forma explícita (sin operadores)*

El lenguaje $L(ab^*c|(ca)^*) = \{a\} \circ \{b\}^* \circ \{c\} \cup (\{c\} \circ \{a\})^*$ podemos describirlo de forma explícita (definición por comprensión) resolviendo las operaciones de lenguajes. Para ello tenemos en cuenta que $\{b\}^*$ contiene cadenas formadas por cero o más $b's$ y $(\{c\}\{a\})^* = \{ca\}^*$ contiene cadenas formadas por repeticiones de la subcadena $ca$ (cero o más). Entonces:

$$\{a\} \circ \{b\}^* \circ \{c\} \cup (\{c\} \circ \{a\})^* = \{w \in \{a,b\}^* \mid w = ab^ic \vee w = (ca)^j, i,j \geq 0\}$$

La expresión regular $ab^*c|(ca)^*$ es una especificación formal concisa para el lenguaje regular $\{w \in \{a,b,c\}^* \mid w = ab^ic \vee w = (ca)^j, i,j \geq 0\}$. Este lenguaje, expresado informalmente, consiste en cadenas que empiezan por 'a' y terminan por 'c' y en medio un número arbitrario de b's (indicado por el patrón alternado $ab^*c$), o bien, cadenas que están formadas por cero o más concatenaciones de "ca" (indicado por el otro patrón alternado $(ca)^*$). Se tiene que $\lambda \in L(ab^*c|(ca)^*)$, porque considerando $j = 0$ en la descripción por comprensión se deduce que la cadena $(ca)^0 = \lambda$ pertenece a ese lenguaje.

Con la práctica se puede saber qué tipo de cadenas describe una ER de manera directa, aplicando los conocimientos sobre la sintaxis y semántica de las ER para *interpretar el significado* de una expresión regular. En adelante, siempre que aparezca una ER que consiste en una

cadena de constantes del alfabeto (como *abb*) entenderemos que describe a esa misma cadena de símbolos (la cadena "abb"), sin necesidad de dividir la ER como concatenación ER básicas. Para describir informalmente el tipo de cadenas que describe una ER, usualmente se traduce una concatenación en la ER como "seguido de", la alternancia se traduce como "o bien" y la clausura como "cero o más repeticiones/concatenaciones de...", "secuencia opcional de ..." o "número arbitrario de...".

**Ejemplo 8** *La expresión teclado|pantalla es una* ER *considerando el alfabeto de letras latinas y podemos afirmar directamente que* $L(teclado|pantalla) = \{teclado, pantalla\}$ *porque el patrón alterno teclado únicamente describe a la cadena "teclado" y el otro patrón pantalla sólo describe a la cadena "pantalla".*

**Ejemplo 9** *Sea el alfabeto* $V_{dig} = \{0, 1, \ldots, 9\}$*. La expresión regular*

$$R_p = (0|1|2|3|4|5|6|7|8|9)^*(0|2|4|6|8)$$

*describe a las cadenas que consisten en una secuencia opcional de dígitos de* 0 *a* 9 *seguida de los dígitos* 0 *o* 2 *o* 4 *o* 6 *o* 8*. Esta interpretación del significado de la* ER *es una* ***traducción literal*** *de lo descrito por la expresión regular a lenguaje natural. Pero en este caso el lenguaje* $L(R_p)$ *admite una* ***interpretación no literal*** *evidente: "es el conjunto de cadenas que representan números naturales pares". A este lenguaje lo llamábamos* $N_{par}$ *en el tema 1 y lo describíamos matemáticamente por comprensión como:*

$$N_{par} = \{xp \mid x \in V_{dig}^* \wedge p \in \{0, 2, 4, 6, 8\}\}$$

*Así que la expresión regular* $R_p$ *indicada más arriba es otra descripción formal y finita del lenguaje infinito* $N_{par}$*, porque* $L(R_p) = \{xp \mid x \in V_{dig}^* \wedge p \in \{0, 2, 4, 6, 8\}\}$*. También podemos decir que el lenguaje* $N_{par}$ *es un* ***lenguaje regular*** *porque se describe mediante la expresión regular* $(0|1|2|3|4|5|6|7|8|9)^*(0|2|4|6|8)$*.*

### *Emparejamiento entre expresiones regulares y cadenas*

En inglés se usa el verbo *"match"* para indicar que hay coincidencia o emparejamiento entre una ER $R$ y una cadena $w$. En la práctica, comprobar si una cadena $w$ pertenece al lenguaje $L(R)$ supone comprobar si la cadena $w$ **"se ajusta al patrón"** dado por $R$, o al revés, si la expresión regular $R$ **"casa con"** o **"tiene coincidencia con"** la cadena $w$ (cuando $R$ casa con $w$ se dice en inglés $R$ *matches* $w$).

**Ejemplo 10** *¿Se cumple* $ac \in L(ab^*c|c^*)$*? La pregunta equivale a ¿se ajusta la cadena ac al patrón* $ab^*c|c^*$*? La respuesta es que* SÍ *porque el patrón dado por la* ER $ab^*c|c^*$ *consta a su vez de dos patrones alternados y hay que comprobar si ac casa con alguno de ellos. La cadena ac se ajusta al primer patrón* $ab^*c$ *porque este patrón indica que la secuencia de* $b's$ *es opcional. Por tanto podemos decir que la cadena ac se ajusta a la* ER *completa* $ab^*c|c^*$*, o también podemos decir que la* ER $ab^*c|c^*$ *tiene una coincidencia con la cadena ac. En inglés se dice que* $ab^*c|c^*$ *matches ac.*
*Formalmente puede afirmarse que* $ac \in L(ab^*c|c^*)$ *porque* $L(ab^*c|c^*) = L(ab^*c) \cup L(c^*)$ *y* $ac \in L(ab^*c)$*. Esto último es cierto porque* $L(ab^*c) = \{ab^nc \mid n \geq 0\}$ *y haciendo* $n = 0$ *se tiene claramente que* $ac \in \{ab^nc \mid n \geq 0\}$*.*

*También ocurre que $ab^*c|c^*$ casa con la cadena vacía, porque el patrón alternado $c^*$ tiene coincidencia con $\lambda$, ya que describe cadenas que consisten en una secuencia opcional de c's, de la forma $c^n, n \geq 0$ y $c^0 = \lambda$. Sin embargo, $ab^*c|c^*$ no casa con la cadena abb porque no es una cadena de c's y tampoco se ajusta al patrón $ab^*c$ porque le falta una c al final. Entonces $abb \notin L(ab^*c|c^*)$.*

# 3. Construcción de expresiones regulares

El problema que surge habitualmente en las aplicaciones de las expresiones regulares es el de obtener o construir una expresión regular para describir cadenas según la especificación de cierto lenguaje, en otras palabras, construir una expresión regular para describir el formato de cierto tipo de cadenas. Construir una expresión regular es un proceso importante en aplicaciones de ***regular pattern matching***, que son aquellas en las que se resuelven problemas de validación de formato, sustitución, búsqueda, extracción o análisis de cadenas usando un patrón dado por una expresión regular.

Para que una expresión regular **R sea correcta** para describir cadenas según cierta especificación (cadenas que forman un lenguaje $L_r$) tenemos que comprobar que se cumplen las dos condiciones siguientes:

1. **Que R no sea demasiado estricta**. Esto quiere decir que $R$ debe tener coincidencia con todas las cadenas consideradas válidas según la especificación (pertenecen a $L_r$). Formalmente, debe cumplirse $L_r \subseteq L(R)$.
2. **Que R no sea demasiado general**. Esto quiere decir que $R$ no debe tener coincidencia con cadenas consideradas incorrectas según la especificación (no hay *matching* con cadenas fuera de $L_r$). Formalmente, debe cumplirse $L(R) \subseteq L_r$.

**Ejemplo 11** *Un caso muy sencillo es obtener una* ER *para el lenguaje universal sobre cierto alfabeto, como $V = \{a, b, c\}$. Para describir a la unión de todos los símbolos usamos la* ER *$(a|b|c)$ y para describir cualquier cadena de símbolos de $V$ y $\lambda$ se toma la clausura. Por tanto la expresión regular $(a|b|c)^*$ describe el lenguaje universal $\{a, b, c\}^*$.*
*Podemos decir que el lenguaje universal $\{a, b, c\}^*$ es un lenguaje regular y en general* ***el lenguaje universal sobre cualquier alfabeto es un lenguaje regular****.*

**Ejemplo 12** *Supongamos que queremos describir el formato de los números decimales sin signo que tienen parte entera y parte fraccionaria obligatoria. Esas cadenas forman el lenguaje $L_{ef}$, que se trató en el tema anterior y se diseñó un $AFD$ para aceptar cadenas de ese lenguaje. Ahora obtenemos una* ER *que lo describe y con eso probamos también que el lenguaje $L_{ef}$ es regular.*

- *¿Se puede describir $L_{ef}$ con esta expresión regular?*

$$(0|1|2|3|4|5|6|7|8|9)^*.(0|1|2|3|4|5|6|7|8|9)^*$$

***NO,*** *el patrón de la* ER *es demasiado general: tiene coincidencia con cadenas como "123." (sin parte fracionaria) o ".7" (sin parte entera) o incluso con "."*

- *En el patrón de la* ER *se debe forzar a que aparezca al menos un dígito a la derecha e izquierda del punto decimal. Podemos hacerlo de la siguiente forma y le damos nombre a la* ER *que describe el lenguaje $L_{ef}$ para usarla luego en otra expresión:*

$$R_{ef} = (0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^*.(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^*$$

**Ejemplo 13** *Supongamos ahora que necesitamos describir a las cadenas de números con parte entera y fraccionaria que llevan opcionalmente el signo negativo '-'. Al lenguaje que contiene todas esas cadenas lo llamamos $L_{n?ef}$. Ya tenemos la* ER *$R_{ef}$ del ejemplo anterior para los números con parte entera y fraccionaria sin signo. Necesitamos otra* ER *para permitir que aparezca el signo negativo antes del número y el patrón de la* ER *sería el mismo solo que poniendo delante el símbolo '$-$', esto es, sería $R_{nef} = -R_{ef}$ y sustituyendo $R_{ef}$ queda:*

$$R_{nef} = -(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^*.(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^*$$

*Finalmente, como lo que se pretende es describir a todas las cadenas de $L_{n?ef}$ tendríamos que considerar la unión de esas dos expresiones regulares, porque la unión de* ER *describe la unión de los dos sublenguajes de $L_{n?ef}$, el de los números sin signo y el de los números con signo negativo. Entonces $R_{n?ef} = R_{ef}|R_{nef}$ es la expresión que buscamos y sustituyendo nombres de* ER *por su valor tenemos:*

$$R_{n?ef} = (0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^*.(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^* \mid -(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^*.(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^*$$

*Hay una forma de acortar la escritura de expresiones regulares en el caso de tener muchos símbolos en el alfabeto y es mediante una* ***sintaxis extendida*** *para expresiones regulares como la que se usa en aplicaciones de Unix, Linux o Windows. Esta sintaxis extiende la notación o* ***sintaxis teórica*** *que hemos introducido y se verá en clases de prácticas. De momento, sólo diremos que una* ER *para el lenguaje $L_{n?ef}$ con sintaxis extendida sería:*

$$\text{-?[0-9]}+|\backslash.\text{[0-9]}+$$

*El operador ? indica que la* ER *que lo precede es opcional (signo - en este caso). El rango entre corchetes* [0-9] *es una* ER *que tienen coincidencia con cualquier dígito y \. es para describir el punto decimal, que debe ser "escapado" porque el punto sin la barra es otra* ER *distinta. Por último, el operador + es el operador de clausura positiva (una o más repeticiones). Para las clases y exámenes de teoría nos limitaremos a la sintaxis teórica que es más simple y es suficiente para introducir los conocimientos teóricos elementales sobre expresiones y lenguajes regulares.*

**Ejemplo 14** *Queremos obtener una expresión regular para describir al lenguaje $B_{alt}$ que contiene todas las cadenas con alfabeto $V_{bin} = \{0, 1\}$ que no tienen dos ceros o dos unos consecutivos, que es lo mismo que decir que los unos y ceros están alternados. Se entiende que $\lambda, 0, 1 \in B_{alt}$. Como este lenguaje es algo complejo seguimos la estrategia de* ***considerar los distintos casos*** *en las cadenas de $B_{alt}$ (distintos sublenguajes) y obtenemos una subexpresión regular para cada caso. Luego consideramos la unión de las esas subexpresiones, que describe*

*la unión de los sublenguajes de cada caso. De esta manera* ***descomponemos un lenguaje complejo en varios más simples*** *y combinamos* ER *para los sublenguajes, como técnica tipo divide y vencerás.*

> *Caso 1: cadena vacía o cadena de ceros y unos alternos que empieza por* 1 *y termina con* 0*. Se trata de cadenas del tipo* $(10)^n$ *con* $n \geq 0$*. Ese subconjunto del total de cadenas de* $B_{alt}$ *se describe con la expresión regular* $(\mathbf{10})^*$*.*
>
> *Caso 2: cadena vacía o cadena de ceros y unos alternos que empieza por* 0 *y termina con* 1*. Son del tipo* $(01)^n$ *con* $n \geq 0$ *y se describen con la expresión regular* $(\mathbf{01})^*$*. Cuando la cadena es* $\lambda$ *ese caso está ya incluido en el caso 1, pero si se incluye aquí al poner* $(01)^*$ *no se está haciendo nada incorrecto, porque al final vamos a considerar unión de* ER*, que describe la unión de lenguajes de cada subexpresión regular.*
>
> *Caso 3: cadena de ceros y unos alternos que empiezan por* 0 *y termina con* 0*. Usamos para ese subconjunto la* ER $\mathbf{0}(\mathbf{10})^*$*, que además tiene una coincidencia con* 0*, que es el caso más sencillo de este tipo de cadenas.*
>
> *Caso 4: cadena de ceros y unos alternos que empieza por* 1 *y termina con* 1*. Se describe con* $\mathbf{1}(\mathbf{01})^*$*, que además incluye al caso más simple* 1*.*

*Concluimos que podemos describir al lenguaje* $B_{alt}$ *con la unión de las expresiones regulares obtenidas en los casos anteriores, esto es, con la expresión regular*

$$(01)^*|(10)^*|0(10)^*|1(01)^*$$

*Esta* ER ***es correcta para describir el lenguaje*** $B_{alt}$ *porque todas las cadenas consideradas válidas (de ceros y unos alternos) se ajustan al patrón descrito en alguna de las subexpresiones de la* ER *y por otro lado la* ER *no permite coincidencias con cadenas incorrectas (aparecen dos ceros o dos unos seguidos).*

# 4. Propiedades de las expresiones regulares

**Definición 3** *Decimos que dos expresiones regulares* $R_1$ *y* $R_2$ *son* ***equivalentes****, y lo denotamos como* $R_1 = R_2$*, si y sólo si describen el mismo lenguaje, es decir,*

$$\boxed{R_1 = R_2 \Leftrightarrow L(R_1) = L(R_2)}$$

Para **demostrar que dos expresiones regulares son equivalentes** se pueden aplicar algunas **propiedades de las expresiones regulares**, como las siguientes:

| | |
|---|---|
| 1 : $R_1 \mid (R_2 \mid R_3) = (R_1 \mid R_2) \mid R_3$ [asociativa–unión] | 9 : $\lambda^* = \lambda$ |
| 2 : $R_1 \mid R_2 = R_2 \mid R_1$ [conmutativa–unión] | 10 : $\varnothing^* = \lambda$ |
| 3 : $R_1 \circ \lambda = \lambda \circ R_1 = R_1$ [identidad] | 11 : $R_1 \circ R_1^* = R_1^* \circ R_1$ |
| 4 : $R_1 \circ \varnothing = \varnothing \circ R_1 = \varnothing$ [anulación] | 12 : $R_1^* = (R_1^*)^*$ |
| 5 : $R_1 \circ (R_2 \circ R_3) = (R_1 \circ R_2) \circ R_3$ [asociativa–concat.] | 13 : $R_1^* = \lambda \mid R_1 \circ R_1^*$ |
| 6 : $R_1 \circ (R_2 \mid R_3) = R_1 \circ R_2 \mid R_1 \circ R_3$ [distributiva derecha] | 14 : $(R_1 \mid R_2)^* = (R_1^* \circ R_2^*)^*$ |
| 7 : $(R_2 \mid R_3) \circ R_1 = R_2 \circ R_1 \mid R_3 \circ R_1$ [distributiva izq.] | 15 : $(R_1 \mid R_2)^* = (R_1^* \circ R_2)^* \circ R_1^*$ |
| 8 : $L(R_1) \subseteq L(R_2) \Rightarrow R_1 \mid R_2 = R_2$ [regla de eliminación] | 16 : $R_1 \circ (R_2 \circ R_1)^* = (R_1 \circ R_2)^* \circ R_1$ |

Cada propiedad es una igualdad que indica que las ER de ambos miembros son equivalentes. Para demostrar que esa igualdad es cierta habría que demostrar que se cumple la igualdad de los lenguajes descritos por las ER de ambos miembros, teniendo en cuenta que las letras $R_1, R_2, R_3$ son variables que representan a tres ER arbitrarias. Las propiedades 1 a 8 son heredadas de las propiedades de las operaciones de unión y concatenación de lenguajes y el resto son propiedades que tienen que ver con la clausura de lenguajes.

**Ejemplo 15** *Para demostrar que la propiedad* $10 : \varnothing^* = \lambda$ *es válida, basta probar que* $L(\varnothing^*) = L(\lambda)$*. Y esta igualdad es cierta, ya que teniendo en cuenta la definición de lenguaje descrito por una* ER *tenemos:*

$$L(\varnothing^*) = (L(\varnothing))^* = \varnothing^* = \bigcup_{n=0}^{\infty} \varnothing^n = \varnothing^0 = \{\lambda\} = L(\lambda)$$

**Ejemplo 16** *La propiedad asociativa de la concatenación* $R_1 \circ (R_2 \circ R_3) = (R_1 \circ R_2) \circ R_3$ *se cumple porque* $L(R_1 \circ (R_2 \circ R_3)) = L((R_1 \circ R_2) \circ R_3)$*, ya que la propiedad asociativa se cumple para la concatenación de lenguajes.*

*Usando esta propiedad podemos afirmar que* $a(b^*c) = (ab^*)c$*, ya que esa igualdad se obtiene sustituyendo en la propiedad general la variable* $R_1$ *por* $a$*,* $R_2$ *por* $b^*$ *y* $R_3$ *por* $c$*. La equivalencia* $a(b^*c) = (ab^*)c$ *indica que el lenguaje descrito por ambas expresiones regulares es el mismo. Por eso se puede prescindir de los paréntesis y escribir simplemente* $ab^*c$ *como expresión equivalente a* $(ab^*)c$ *(operador concatenación asociado por la izquierda) o a* $a(b^*c)$ *(operador concatenación asociado por la derecha).*

Las propiedades de las expresiones regulares son usadas en resultados teóricos y además **son útiles para simplificar** una ER reduciendo el número de operadores y acortando la expresión.

**Ejemplo 17** *La propiedad distributiva derecha/izquierda permite "sacar prefijo/sufijo común" en una* ER *con subexpresiones alternas de manera análoga a como se hace en una expresión aritmética con un factor común en una suma de términos.*

*Recordamos la expresión regular larga para describir los números decimales con parte entera y fraccionaria y signo negativo opcional:*

$$\overbrace{(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^*.(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^*}^{R_1} \mid$$
$$\overbrace{-}^{R_3}\overbrace{(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^*.(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^*}^{R_1}$$

*Considerando que $R_1 = \lambda \circ R_1$ y llamando $R_2$ a $\lambda$, comprobamos que la* ER *anterior es de la forma $R_2R_1|R_3R_1$. La expresión $R_1$ aparece como sufijo común de los dos términos que van alternados en la expresión regular. Entonces podemos aplicar la propiedad distributiva para cambiar $R_2R_1|R_3R_1$ por $(R_2|R_3)\,R_1$ y sustituyendo las variables por su valor concreto de* ER *tenemos que la expresión regular queda reducida como:*

$$(\overbrace{\lambda}^{R_2} \mid \overbrace{-}^{R_3})\ \overbrace{(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^*.(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^*}^{R_1}$$

*Si cualquier otra expresión regular tiene subexpresiones alternadas que comparten un prefijo o sufijo siempre podemos aplicar la propiedad distributiva para reducirla "sacando prefijo/sufijo común". Ej.- la* ER *$a^*bcc^*|a^*baa$ podemos reducirla a $a^*b(cc^*|aa)$ aplicando la propiedad distributiva sacando el prefijo común $a^*b$.*

**Ejemplo 18** *¿Se puede simplificar la expresión regular $(0^*1)^*0|10$? Sí que se puede, por la propiedad distributiva tenemos que $(0^*1)^*0|10 = ((0^*1)^*|1)0$. Ahora nos damos cuenta que en la subexpresión $(0^*1)^*|1$, la* ER *alterna 1 sólo describe a la cadena "1" y esa cadena se ajusta al patrón de la otra subexpresión $(0^*1)^*$ alterna, puesto que los ceros son opcionales. Como $L(1) \subseteq L((0^*1)^*)$ se puede aplicar la regla de eliminación para cambiar $(0^*1)^*|1$ por $(0^*1)^*$, eliminando la* ER *1 que resulta redundante. Finalmente la expresión regular de partida $(0^*1)^*0|10$ queda reducida a $(0^*1)^*0$.*

# 5. Aplicaciones de expresiones regulares

Las expresiones regulares se usan como formalismo descriptivo en la resolución de problemas de ***regular pattern matching***, que son los relacionados con el procesamiento de cadenas que siguen un patrón regular (búsqueda o extracción de datos, sustitución o conversión de formato, validación o análisis de cadenas de patrón regular, etc.). Suelen ser introducidas por un usuario en un editor de texto o como argumento de un comando del sistema operativo Unix/Linux para hacer búsqueda de cadenas de patrón regular, o puede ser usada por un programador en formularios para validar el formato de datos de entrada a un programa, etc.

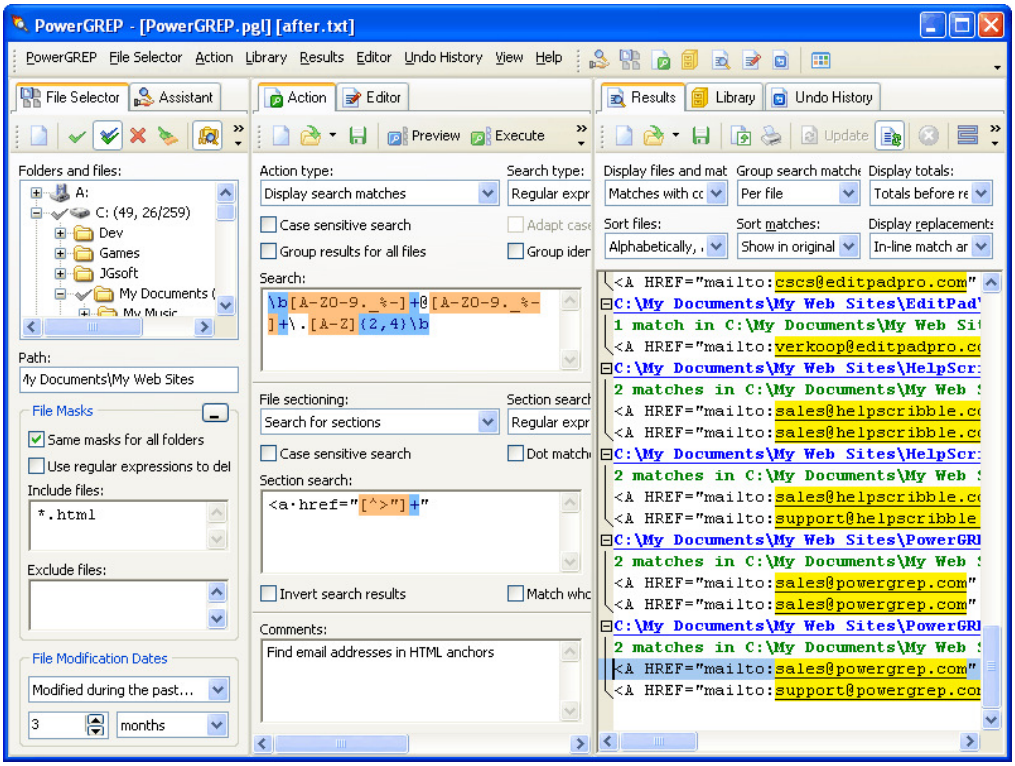
## 5.1. Herramientas que manejan expresiones regulares

Son muchas las herramientas que llevan integrado un motor de expresiones regulares (*regex engine*) que permite manejar expresiones regulares. Algunas de estas herramientas son:

- **Editores avanzados de texto.** Los editores de texto de uso común sólo permiten buscar una cadena fija y sustituir por otra cadena fija. Los editores avanzados, útiles sobre todo para programadores, permiten hacer búsquedas y sustituciones usando expresiones regulares. Por ejemplo: *Emacs, Vi, Vim, Kate,...* (en Linux), *EditPad Pro, TextPad,...* (en Windows).
- **Buscadores en sistemas de ficheros**. La herramienta de búsqueda que aparece en el escritorio de Windows es bastante limitada porque sólo se permiten cadenas fijas o con

pequeñas variaciones (con *wildcards*). Una de las primeras herramientas de búsqueda con expresiones regulares es el programa *grep/egrep*. Es una utilidad del shell de unix/linux que permite buscar en un directorio cadenas que se ajusten a un patrón dado por una expresión regular, con diversas opciones para imprimir el resultado de la búsqueda por los distintos ficheros del directorio. Existe una versión para Windows llamada *PowerGrep* que incorpora un interfaz de usuario bastante completo, semejante al de un editor de textos.

Ej.- En el gráfico siguiente se muestra una captura de pantalla de una búsqueda que consiste en lo siguiente: "direcciones de correo electrónico que aparezcan dentro de enlaces html en ficheros con extensión .html. del directorio 'My documents\My Web Sites' y en los subdirectorios".




- **Lenguajes de consulta en bases de datos.** Algunas bases de datos como *Oracle* o *MySQL* permiten el uso de expresiones regulares, aunque con restricciones, en sentencias del lenguaje de consulta SQL para seleccionar columnas de una tabla, para extraer parte de una columna, modificarla con *search-and-replace*, etc.

- **Lenguajes de programación.** Son muchos los lenguajes de programación que incluyen un *regex engine* implementado en paquetes que usan ER para facilitar los procesos de búsqueda, sustitución, extracción o validación de cadenas. Mediante estas funciones se pueden ahorrar decenas de líneas de código, que serían necesarios si el procesamiento de cadenas se hiciera "a mano". Algunos de estos lenguajes son: *Java, Javascript, PHP, Visual Basic, Perl, C#, VB.NET*, etc.

- **Herramientas de análisis léxico.** Un tipo de análisis de cadenas de patrón regular es el que se lleva a cabo en compiladores/intérpretes para comprobar si un fichero de código fuente contiene errores léxicos y agrupar los caracteres individuales en unidades léxicas

(*tokens*, ver sección siguiente). Hay herramientas que, a partir de una lista de expresiones regulares, generan el código de una función de análisis léxico, que sirve para recorrer un fichero filtrando los tokens, por ejemplo *lex, flex* (genera código C), *jlex* (para Java), etc.

## 5.2. Analizadores léxicos

Una de las aplicaciones más importantes de las expresiones regulares y los autómatas finitos está en la *construcción de analizadores léxicos* para compiladores o intérpretes de lenguajes de programación, de lenguajes de formateo de textos, lenguajes de comandos en herramientas matemáticas, y en general se aplican en cualquier proceso complejo que requiera traducir un **código fuente** en un lenguaje a **código objeto** en otro lenguaje. El proceso de desarrollo de un compilador es una tarea complicada que no podría llevarse a cabo con la eficiencia y fiabilidad necesaria sin las técnicas que proporciona la Teoría de Autómatas y Lenguajes Formales. No vamos a entrar en detalle sobre el diseño e implementación de un compilador, pues es materia de la asignatura de Compiladores.

El proceso de traducción de código fuente a código objeto se lleva a cabo en varias etapas y las que están más directamente relacionadas con los conceptos que se introducen en esta asignatura son la fase de análisis léxico y sintáctico. El **analizador sintáctico** es un módulo del traductor que se encarga esencialmente de comprobar si el código fuente es sintácticamente correcto según una gramática que describe la sintaxis del lenguaje del código fuente. Pero como el código fuente no es más que una secuencia de caracteres que individualmente no tienen significado, se necesita otro módulo que es el **analizador léxico** que se encarga de filtrar el código fuente, eliminando lo innecesario (como comentarios y espacios), eventualmente detectando caracteres extraños (error léxico) y agrupando los caracteres individuales en *elementos léxicos* con significado propio que se conocen como *tokens*. Un esquema de caja negra del analizador léxico es el siguiente:


Por ejemplo, en un lenguaje de programación, los tokens constituyen los elementos básicos que forman los programas: identificadores de variables o funciones, palabras reservadas, constantes numéricas o de string, símbolos de operadores, símbolo de terminación de sentencia, de inicio o fin de bloque, etc. Estos tokens son como "supersímbolos" del lenguaje y son los que interesan al analizador sintáctico para comprobar la sintaxis antes de que el traductor lleve a cabo finalmente la generación de código objeto.

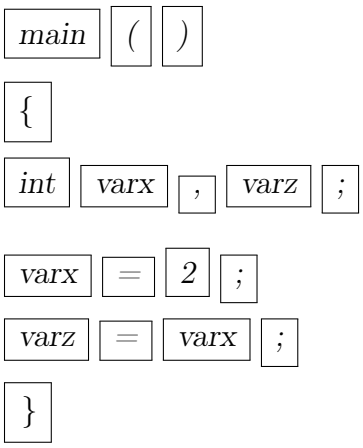
**Ejemplo 19** *Un programa fuente en C como:*

```
main ()
{
int varx, varz;

/* Asigna 2 a varz
```

```
y esto es un comentario en varias líneas */
varx = 2;
varz = varx;
}
```

*queda aceptado sin errores léxicos, agrupado en tokens y sin comentarios como:*




**Los tokens se pueden describir mediante expresiones regulares y, como veremos más adelante, se pueden aceptar mediante autómatas finitos**.

El proceso que se sigue para la **implementación del analizador léxico** puede resumirse en los siguientes pasos:

1. **Identificar los tokens** del lenguaje. Hay tokens que se corresponden con distintas cadenas (*lexemas*), por ejemplo, el token ID representa a un identificador y sus lexemas suelen ser cadenas de letras o dígitos que empiezan por una letra.

2. **Describir cada token con una expresión regular**. Por ejemplo, si se establece que un identificador comienza por una letra que opcionalmente va seguida de más letras o dígitos, se usa una expresión regular como:

   $$(a|\ldots|z|A|\ldots|Z)(a|\ldots|z|A|\ldots|Z|0|\ldots|9)^*$$

   que con sintaxis extendida tipo Unix se expresa como $[a\text{-}zA\text{-}Z][a\text{-}zA\text{-}Z0\text{-}9]^*$.

3. Considerar la **unión de las expresiones regulares** de todos los tokens y a partir de ahí **se diseña un** $AFD$, a ser posible mínimo, para que acepte cualquier cadena que se ajuste al patrón de cualquier token.

4. **Usar la tabla de transición del** $AFD$ **para implementar la función del analizador léxico**. Esta función, en lugar de procesar el fichero con el código fuente completo para producir la secuencia de tokens, se suele diseñar como función que es llamada por el analizador sintáctico cada vez que éste solicita un nuevo token en el proceso de comprobación de sintaxis. Cada vez que se llama al analizador léxico, éste va leyendo carácter a carácter el fichero de entrada a partir de la posición del fichero en que quedó en la llamada anterior. Va cambiando de estado según los caracteres que lee consultando la tabla de

transición del $AFD$. Si alcanza un estado final entonces devuelve el *código de token* que ha aceptado (y el lexema o cadena que casa con la ER del token, si es necesario). Si acaba en un estado de error retorna un código de error léxico.
Cuando hay muchas expresiones regulares, todo este proceso se hace con una herramienta, como $flex$, que genera el código de la función del analizador léxico basada en el $AFD$ a partir de las expresiones regulares de los tokens.

# 6. Preguntas de evaluación

Aparte de los 25 ejemplos de los apuntes, que refuerzan los conocimientos teóricos, en esta sección tenemos preguntas que sirven también como auto-evaluación de los conocimientos teóricos. Abarcan problemas de razonamiento o demostración, problemas de aplicación algoritmos, métodos o definiciones y preguntas tipo test. Muchas de estas preguntas han aparecido en exámenes de cursos anteriores.

**Nota:**

- Los problemas de **mayor dificultad** se señalan con **(!)**.
- La etiqueta **JFLAP** delante de un ejercicio indica que es un problema con diagramas de transición editados en un fichero de JFLAP, que se encuentra en el Aula Virtual.

## 6.1. Problemas resueltos

1. Simplifica la ER $a|a\,(b|aa)\,(b^*aa)^*\,b^*|a\,(aa|b)^*$ de manera que sólo tenga un operador de clausura e indica en cada paso la propiedad que se usa.

   **Solución:** aplicando las propiedades de las expresiones regulares podemos obtener una ER equivalente:

   $$a \;\mid\; a\,(b|aa) \underbrace{(b^*aa)^*\,b^*}_{(R_1|R_2)^*=\left(R_1^*R_2\right)^*R_1^*} \;\mid\; a\,(aa|b)^* = \underbrace{a|a\,(b|aa)\,(b|aa)^*}_{R_1(R_2|R_3)=R_1R_2|R_1R_3} \;\mid\; a\,(aa|b)^* =$$

   $$a(\underbrace{\lambda|\,(b|aa)\,(b|aa)^*}_{R_1^*=\lambda|R_1R_1^*}) \;\mid\; a\,(aa|b)^* = \underbrace{a\,(b|aa)^*\,|a\,(aa|b)^*}_{\text{(regla de eliminación y prop. conmutativa)}} = a\,(aa|b)^*$$

2. Expresa el lenguaje descrito por la ER $(b(aa)^*|c)^*(d|\lambda)$ como expresión que combina lenguajes básicos (descritos por expresiones regulares básicas, sin operadores) mediante operadores regulares:

   **Solución:**

   $$L((b(aa)^*|c)^*(d|\lambda)) = (\{b\} \circ (\{a\} \circ \{a\})^* \cup \{c\})^*(\{d\} \cup \{\lambda\})$$

3. Sea el lenguaje $L_{1ao1b}$ formado por cadenas de alfabeto $V = \{a, b, c\}$ que contienen al menos una 'a' o al menos una 'b'. Se pide demostrar que este lenguaje es regular obteniendo una ER que lo describa.

   **Solución:** si nos damos cuenta de que $L_{1ao1b} = L_a \cup L_b$ donde $L_a$ contiene las cadenas que tienen al menos una 'a' y $L_b$ las que tienen al menos una 'b', es fácil obtener una ER para cada sublenguaje y considerar la unión de las expresiones regulares.
   Es sencillo comprobar que $R_a = (a|b|c)^*a(a|b|c)^*$ y $R_b = (a|b|c)^*b(a|b|c)^*$ describen a $L_a$ y $L_b$, respectivamente. Luego $R_a|R_b = (a|b|c)^*a(a|b|c)^* \mid (a|b|c)^*b(a|b|c)^*$ describe a $L_{1ao1b}$ y por tanto se puede afirmar que este lenguaje es regular.

4. **(!)** Demuestra que el lenguaje $L_{0i1} = \{w \in \{0,1\}^* \mid ceros(w) = unos(w)\}$ no es regular es partiendo del hecho de que el lenguaje $L_s = \{0^k 1^k \mid k \geq 0\}$ no es regular y **usando las propiedades de cierre de los lenguajes regulares**.

   **Solución:** Partiendo de que $L_s$ no es regular probamos que $L_{0i1}$ no es regular por reducción al absurdo. HIPÓTESIS: supongamos que $L_{0i1}$ es regular. Observamos que $L_s = L_{0i1} \cap L(0^*1^*)$. Como $L(0^*1^*)$ es regular, porque es un lenguaje descrito por una ER, entonces al ser $L_{0i1}$ regular (hipótesis) debería serlo también el lenguaje $L_s$, porque la clase de lenguajes regulares es cerrada bajo intersección. Pero como hemos partido de que $L_s$ no es regular entonces la hipótesis de que $L_{0i1}$ es regular es falsa. Luego $L_{0i1}$ no es regular.

## 6.2. Problemas propuestos

1. Expresa el lenguaje descrito por la ER $(a|ba)^*((bb)^*|aab|\lambda)$ como expresión que combina lenguajes básicos mediante operadores regulares. Se recuerda que los lenguajes básicos son los descritos por expresiones regulares básicas (sin operadores).

2. **(!)** Demuestra la equivalencia $(b|ab^*a)^*ab^* = b^*a(b|ab^*a)^*$ aplicando las propiedades de las expresiones regulares.

3. Expresa con palabras de la forma más concisa y significativa posible (no una "traducción literal" de la ER) el lenguaje que describen las siguientes expresiones regulares.

   *a)* $(00|01|10|11)^*$

   *b)* $(0|1)^*11$

   *c)* **(!)** $a^*(ba^*)^*bba(b^*a^*)^*$ (sugerencia: usar propiedades para simplificar la ER)

4. (JFLAP) Obtén expresiones regulares para los siguientes lenguajes:

   *a)* $L_a$: cadenas de alfabeto $\{a, b, c\}$ que contengan al menos la subcadena $a$ y al menos una $b$.

   *b)* $L_b = \{c\} \circ \{ab^i \mid i > 0\}^*$.

   *c)* $L_c$: cadenas de ceros y unos cuyo segundo símbolo sea 1.

   *d)* $(L_b \cup L_c)^*$.

   *e)* **(!)** Cadenas de a's/b's que tienen como mucho un par de b's consecutivas.

## 6.3. Preguntas tipo test

1. Señala la opción verdadera. Dada la ER $(a(cd)^*b)^* \,|(cd)^*$

   *a)* Es equivalente a la expresión regular $(a(cd)^*b)^*$.

   *b)* Tiene una coincidencia con la cadena "acdcdb".

   *c)* Las cadenas de menor longitud que se ajustan al patrón son "ab" y "cd".

2. Señala la opción <u>verdadera</u>. Sean $R_1$ y $R_2$ expresiones regulares cumpliendo $L(R_1) \subseteq L(R_2)$, entonces

   *a*) $R_1 \circ R_2 = R_2 \circ R_1$

   *b*) $R_1|R_2 = R_1$

   *c*) $R_2 \circ \varnothing^* = R_1|R_2$