Teoría de la computación

Unidad 3. Expresiones y Lenguajes Regulares

Dr. Christian Millán

Unidad 3. Expresiones y Lenguajes Regulares

Contenido

1. Expresiones regulares
2. Autómatas finitos y expresiones regulares
3. Aplicaciones de expresiones regulares
4. Algebra para las expresiones regulares
5. Lema de bombeo
6. Propiedades de la clausura.

1. Expresiones Regulares

Como dato histórico las **Expresiones Regulares** (ER) fueron desarrolladas por Kleene en 1956. El poder de las ER radica en la sencillez de su notación, la cual se asemeja a una notación algebraica. Es decir una ER puede ser vista como una fórmula en un lenguaje especial.

> “Una expresión regular ER es un patrón que nos permite expresar a un lenguaje regular. Un ER tiene la ventaja de expresar los tipos de repeticiones infinitas que se llegan a dar en un lenguaje regular”.

1. Expresiones Regulares

Para la notación básica se utilizan los siguientes elementos:

- Elementos
- Contadores
- Secuencias
- Disyunción

2. Autómatas finitos y ER

Elementos

“A”,”B”: Caracteres pertenecientes al alfabeto.

( ): Paréntesis: Agrupación de elementos de la ER. La anidación de paréntesis indica el orden de los operadores. De hecho, los siguientes elementos son presentados en orden jerárquico descendente.

2. Autómatas finitos y ER

Contadores

* : A*: Estrella de Klenne: Cero o más ocurrencias del carácter o expresión anterior.

+: A+: Una o más ocurrencias del carácter o expresión anterior.

?: Cero o una ocurrencia del carácter.

2. Autómatas finitos y ER

Secuencias

AB: Indica una sola secuencia de los elementos AB.

Esta operación no utiliza ningún operador visible entre A y B; haciendo así más corta la representación de la expresión regular. Algunos autores utilizan caracteres como +, • ; sin embargo, en la práctica y sobre todo en programas lingüísticos, no se utiliza carácter alguno.

2. Autómatas finitos y ER

Disyunción

A | B: Teniendo la última precedencia, está el operador disyunción que indica por lo menos algunas de las dos opciones, ya sea A o B, pero no ambas.

2. Autómatas finitos y ER

Conversión de una expresión regular a un AFND-ε.

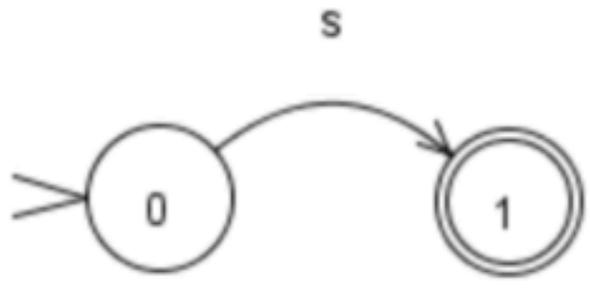
El siguiente algoritmo está detallado con base a los elementos de una ER antes mencionados. Para la construcción del autómata se hace uso de la transición vacía ε.

2. Autómatas finitos y ER

Algoritmo:
Entrada: Una expresión regular ER con un alfabeto Σ
Salida: Un AFND-ε

Analizar los elementos de la expresión regular
1. Para cada símbolo que pertenece a $s \in \Sigma$, incluyendo con esto a la palabra vacía ε, construir un subautómata para cada símbolo s. Para hacer esto:
   a) Se inserta su estado de inicio que no sea final
   b) Se inserta su estado final
   c) Se inserta una transición del estado de inicio al final con la etiqueta del símbolo s correspondiente.

La figura 3.4 muestra la forma que quedaría el subautómata correspondiente al símbolo s.



2. Autómatas finitos y ER

2. Para reconocer A^* se considera el subautómata A de la figura 3.5 más un nuevo estado 0 que será el inicial del autómata nuevo y un único estado final 1. Se unen mediante una transición vacía ε del estado inicial 0 al inicial 0A y también del estado inicial 0 al 1. También mediante una transición vacía se unen cada uno de los estados finales del subautómata A al estado final 1 y también de los estados finales del subautómata A al estado inlcial 0A de este subautómata. Cabe señalar que A es la representación de otra expresión regular compuesta de uno o varios elementos, incluyendo los paréntesis:

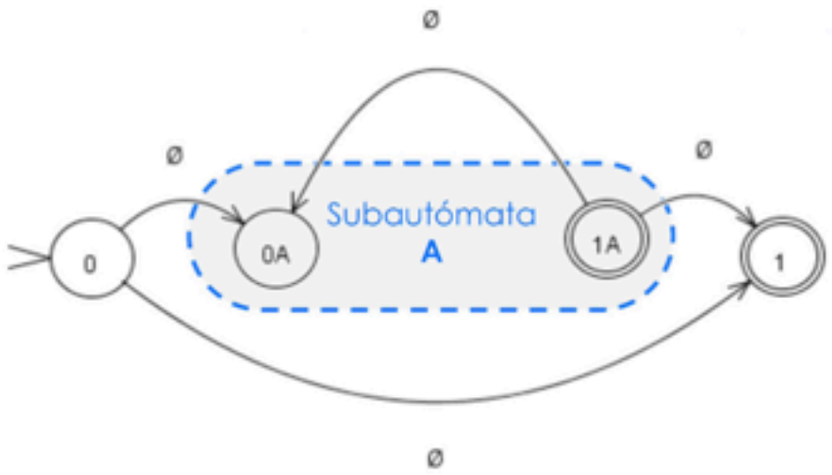


Figura 3.5 Subautómata correspondiente de la operación A^*.

2. Autómatas finitos y ER

Esta nueva gráfica de la figura 3.5 reconoce efectivamente a la potencia estrella A^*

3. Para reconocer A^+ se realiza el proceso análogo que A^*, excepto que no se lleva a cabo una transición con vacío del estado inicial 0 al estado final 1. Este proceso se ilustra en en la figura 3.6

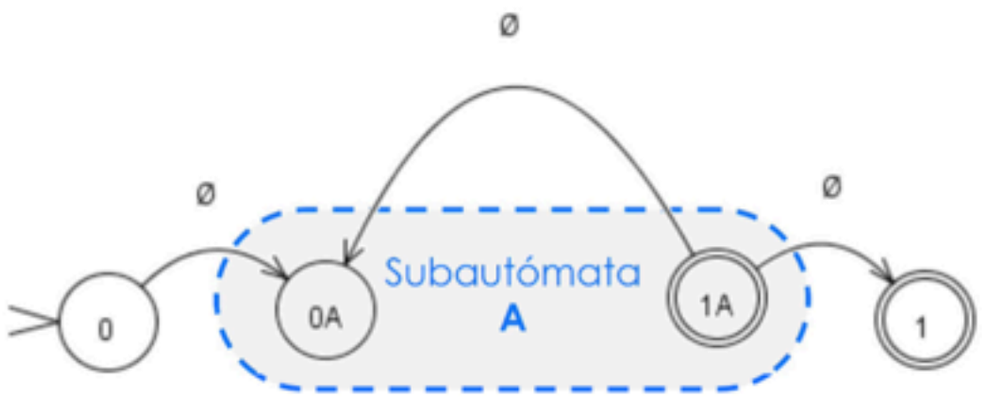


Figura 3.6 Subautómata correspondiente a la operación A^+.

2. Autómatas finitos y ER

4. Para el caso de reconocer el operador ? Se hace lo mismo que en paso 2, excepto que no se crea la transición de regreso de los estados finales del subautómata A al estado inicial, de esa manera solo se puede aceptar o no un elemento. Esto se ilustra en la siguiente gráfica (figura 3.7)

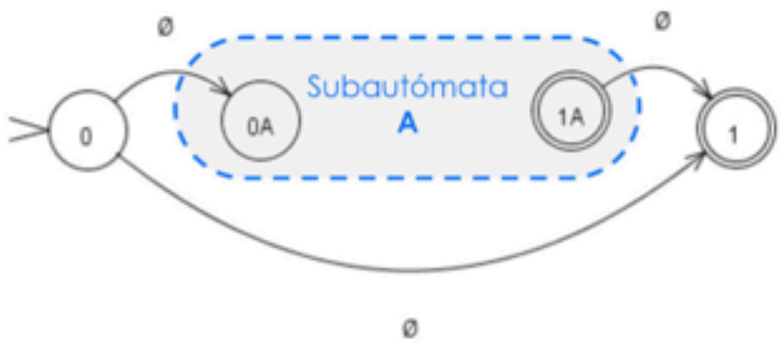


Figura 3.7 Subautómata correspondiente al operador ?.

2. Autómatas finitos y ER

5. Para reconocer AB se crea una transición vacía de cada uno de los estados finales del subautomáta A al estado inicial 0B; todos los estados finales del subautómata A son reemplazados por los estados no finales y no iniciales. Como se muestra en la figura 3.8

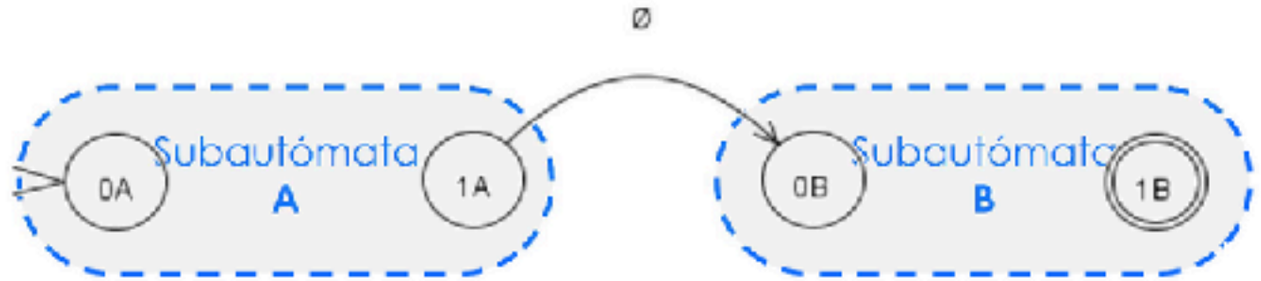


Figura 3.8 Construcción del subautómata correspondiente a la secuencia de dos subautómatas.

2. Autómatas finitos y ER

6. Una vez construidos tales subautómatas se busca en la expresión regular los elementos que estén sólo separados por el símbolo de disyunción "|". Para ambos elementos sus subautómatas correspondientes son unidos por un estado inicial y un final, como se muestra en la figura 3.9. Esto se hace de la siguiente manera:
a) Se inserta un estado inicial 0, que no sea final.
b) Desde el estado inicial 0 se insertan transiciones con vacío ε al estado inicial de cada subautómata implicado.
c) Los estados iniciales de los subautómatas son reemplazados por estados no iniciales y no finales.
d) Se inserta un estado final 1, que no sea inicial.
e) Los estado finales de los subautómatas son conectados con vacío al estado final 1.
f) A los subautómatas se les borra sus estados finales por estados que no sean finales ni iniciales.

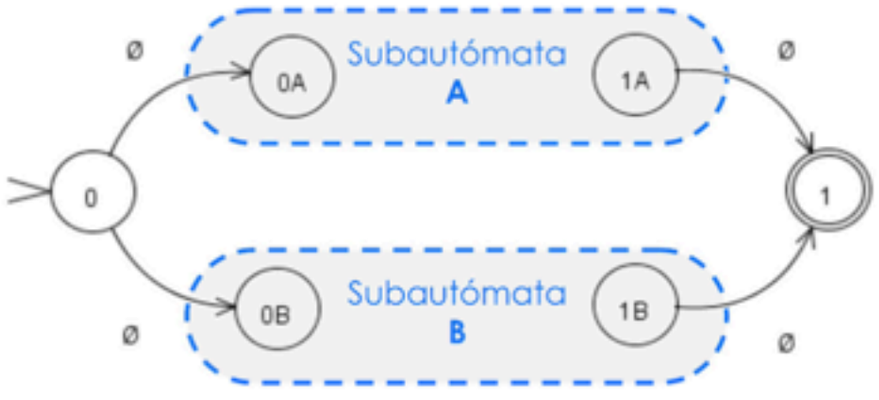


Figura 3.9 Construcción del subautómata correspondiente a la operación de disyunción.

Mientras no se haya analizado todas las partes constituyentes de la ER regresar al paso 2.

3. Aplicaciones de ER

Una ER proporciona una imagen del patrón que deseamos reconocer es el medio que emplean las aplicaciones que permiten realizar búsquedas de patrones de textos.

Las expresiones regulares a continuación se compilan entre bastidores para obtener autómatas deterministas o no deterministas, que luego se simulan para generar un programa que reconoce los patrones de los textos.

3. Aplicaciones de ER

Expresiones regulares en UNIX

La Notación extendida de UNIX proporciona características extendidas, en particular la capacidad de nombrar o hacer referencia a cadenas previas que presentan correspondencias con un patrón (lenguajes regulares).

3. Aplicaciones de ER

Expresiones regulares en UNIX

Clases de caracteres para representar conjuntos de caracteres largos. Las reglas para estas clases de caracteres son:

- El símbolo . (punto) quiere decir "cualquier carácter"
- La secuencia $[a_1 a_2 \cdots a_k]$ representa la expresión regular $a_1 + a_2 + \cdots + a_k$. Por ejemplo `[<>=!]`
- Entre los corchetes podemos escribir un rango de la forma $x - y$ para especificar caracteres de x hasta y en la secuencia ASCII. Por ejemplo, los dígitos se pueden expresar con [0-9], las letras mayúsculas [A-Z] y el conjunto de todas las letras y dígitos como [A-Za-Z0-9], si se desea incluir el signo menos agregar al final para evitar confusiones [-+.0-9]. Los corchetes u otros signos que tenga un significado especial en las expresiones regulares UNIX pueden presentarse como caracteres precediendolos de una barra invertida (\).

3. Aplicaciones de ER

Expresiones regulares en UNIX

- Notaciones especiales:
  - [:digit:] Representa los diez dígitos [0-9]
  - [:alpha:] Representa cualquier carácter alfabético [A-Za-z]
  - [:alnum:] Representa los dígitos y letras [A-Za-z0-9]
- El operador | representa la unión
- El operador ? significa "cero o uno de"
- El operador + significa "uno o más de"
- El operador {n} significa "n copias de"

Los paréntesis agrupan subexpresiones y aplican la misma precedencia de operadores vistas anteriormente.

3. Aplicaciones de ER

Análisis léxico

Una de las aplicaciones de las ER son el analizador léxico de un compilador. Este componente explora el código fuente de un programa y reconoce todas las unidades sintácticas, aquellas suncadenas de caracteres que forman agrupaciones lógicas. Las palabras clave y los identificadores son ejemplo habituales.

El comando UNI lex y su versión GNU flex, aceptan como entrada una lista de expresiones regulares, escritas en el estilo UNIX, seguidas cada una de ellas por una sección de código entre corchetes que indica lo que el analizador léxico hará cuando encuentre una instancia de una unidad sintáctica. Tal herramienta se le conoce como generador de analizadores léxicos.

3. Aplicaciones de ER

Búsqueda de patrones en texto

Las ER resultan valiosas para describir búsquedas de patrones, el problema general para el que la tecnología basada en expresiones regulares ha resultado ser útil es la descripción de clases de patrones de texto definidos vagamente. La vaguedad de la descripción prácticamente garantiza que no definiremos correctamente el patrón a la primera, quizá puede que nunca lleguemos a obtener exactamente la descripción correcta. Utilizando la notación de expresiones regulares, será fácil describir el patrón en un nivel alto, con poco esfuerzo, y modificar la descripción rápidamente cuando las cosas no funciones. Un "compilador" para expresiones regulares es útil para convertir las expresiones que hemos escrito en código ejecutable.

Ejemplo de una colección de páginas web:

- Extraer direcciones de calles
- Correos
- Clasificar negocios

4. Algebra para las ER

Asociatividad y conmutatividad

Lo conmutatividad es la propiedad de un operardor que establece que se puede cambiar el orden de sus operandos y obtener el mismo resultado. La ley de la asociatividad permite reagrupar los operandos cuando el operador se aplica dos veces. Las leyes que se cumplen para las expresiones regulares son:

- $L + M = M + L$. Esta ley, la ley de la **conmutatividad de la unió**n, establece que podemos efectuar la unión de dos lenguajes en cualquier orden.
- $(L + M) + N = L + (M + N)$. Esta ley, la ley **asociativa para la unión,** establece que podemos efectuar la unión de tres lenguajes bien calculando la unión de los dos primeros, o bien la unión de los dos últimos. Observe que con la conmutatividad de la unión, es posible obtener la unión de cualquier colección de lenguajes en cualquier orden y agrupamiento, y el resultado siempre será el mismo. Intuitivamente, una cadena pertenece a $L_1 \cup L_2 \cdots \cup L_k$ si y sólo si pertenece a uno o más de los L_i.
- $(LM)N = L(MN)$. Esta ley, la ley asociativa para la concatenación, establece que podemos concatenar rtes lenguajes concatenando primero los dos primeros o bien los dos últimos.

La ley que establece que $LM = ML$, es decir, que la concatenación es conmutativa es falsa.

4. Algebra para las ER

Elemento identidad y elemento nulo

- $\emptyset + L = L + \emptyset = L$. Esta ley establece que $\emptyset$ es elemento identidad para la unión.
- $\varepsilon L = L\varepsilon = L$. Esta ley establece que ε es el elemento identidad para la concatenación.
- $\emptyset L = L\emptyset = \emptyset$. Esta ley establece que $\emptyset$ es el elemento nulo de la concatenación.

Leyes distributivas

Considerar que la concatenación no es conmutativa.

- $L(M + N) = LM + LN$. Esta ley distributiva por la izquierda de la concatenación respecto de la unión.
- $(M + N)L = ML + NL$. Esta ley distributiva por la derecha de la concatenación respecto de la unión.

4. Algebra para las ER

Ley de idempotencia

- $L + L = L$. Esta ley de idempotencia para la unión, que establece que si tomamos la unión de dos expresiones idénticas, podemos reemplazar por una copia de la expresión.

Leyes relativas a las clausuras

- $(L^*)^* = L^*$. Esta ley dice que clausurar una expresión que ya está clausurada no modifica el lenguaje.
- $\emptyset^* = \varepsilon$. La clausura de $\emptyset$ sólo contiene la cadena ε.
- $\varepsilon^* = \varepsilon$. Es fácil comprobar que la única cadena que se puede formar concatenando cualquier número de copias de la cadena vacía es la propia cadena vacía.
- $L^+ = LL^* = L^*L.$
- $L^* = L^+ + \varepsilon$
- $L? = \varepsilon + L$

5. Lema de bombeo

🛠️ **Teorema de bombeo para lenguajes regulares**

Sea L un lenguaje regular. Existe entonces una constante n (que depende de L) tal que para toda cadena w perteneciente a L con $|w| \geq n$, podemos descomponer w en tres cadenas, $w = xyz$, tales que:
1. $y \neq \varepsilon$
2. $|xy| \leq n$
3. Para todo $k \geq 0$, la cadena $xy^k z$ también pertenece a L.

Es decir, siempre podemos hallar una cadena vacía y no demasiado alejada del principio de w que pueda "bombearse"; es decir, si se repite y cualquier número de veces, o se borra (el caso en que $k = 0$), la cadena resultante también pertenece al lenguaje L.

5. Lema de bombeo

Demostración

Supongamos que L es regular. Entonces $L = L(A)$ para algún AFD A. Supongamos que A tienen n estados. Consideremos ahora cualquier cadena w de longitud n o mayor por ejemplo $w = a_1 a_2 \cdots a_m$, donde $m \geq n$ y cada a_i es un símbolo de entrada. Para $i = 0, 1, \ldots, n$ definimos el estado p_i como $\hat{\delta}(q_0, a_1 a_2 \cdots a_i)$, donde δ es la función de transición de A y q_0 es el estado inicial de A. Es decir, p_i es el estado en que se encuentra A después de leer los primeros i símbolos de w. Observe que $p_0 = q_0$.

Por el principio del juego de las sillas, no es posible que los $n + 1$ p_i para $i = 0, 1, \ldots, n$ sean diferentes, ya que sólo existen n estados distintos. Por lo tanto, podemos determinar dos enteros distintos i y j, como $0 \leq i < j \leq n$, tales que $p_i = p_j$. Ahora podemos descomponer $w = xyz$ como sigue:

1. $x = a_1 a_2 \ldots a_i$
2. $y = a_{i+1} a_{i+2} \ldots a_j$
3. $z = a_{j+1} a_{j+2} \cdots a_m$

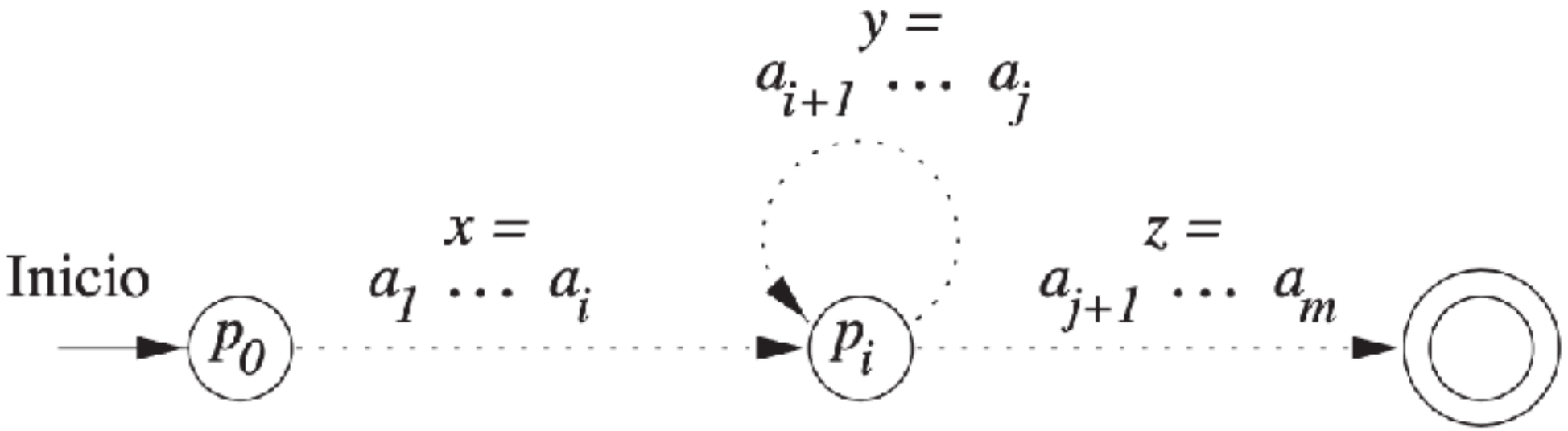
5. Lema de bombeo

Es decir, x nos lleva a p_i una vez; y nos lleva desde p_i hasta p_i de nuevo (ya que p_i también es p_j) y z es el resto de w. Las relaciones entre cadenas y los estados se muestran en la figura 4.1. Observe que x puede estar vacía en el caso de que $i = 0$. También z puede estar vacía si $j = n = m$. Sin embargo, y no puede estar vacía, ya que i es estrictamente menor que j.

Si el autómata A recibe la entrada $xy^k z$ para cualquier $k \geq 0$. Si $k = 0$, entonces el autómata va desde el estado inicial q_0 (que también es p_0 hasta p_i para la entrada x. Puesto que p_i también es p_j, al leer la entrada z, A tiene que ir desde p_i hasta el estado de aceptación mostrado en la Figura 4.1. Por tanto, A acepta xz.

Si $k > 0$, entonces A va desde q_0 hasta p_i para la entrada x, va en círculo desde p_i hasta p_i k veces para la entrada y^k, y lyego pasa al estado de aceptación para la entrada z. Por lo tanto, para cualquier $k \geq 0$, A también acepta $xy^k z$; es decir, $xy^k z$ pertenece a L.

5. Lema de bombeo



5. Lema de bombeo

🛠️ **El lema de bombeo como un juego de adversarios**

El lema de bombeo implica cuatro identificadores diferentes: "**para todos** los lenguajes L **existe** n tal que **para toda** w perteneciente a L con $|w| \geq n$ **existe** xyz igual a w tal que $\cdots$ ". Podemos interpretar la aplicación del lema de bombeo como un juego en el que:

1. El jugador 1 selecciona el lenguaje L para demostrar que no es regular.
2. El jugador 2 selecciona n, pero no revela al jugador 1 lo que vale n; el jugador 1 debe plantear el juego para todos los n posibles.
3. El jugador 1 selecciona w, que puede depender de n y cuya longitud tienen que ser al menos igual a n.
4. El jugador 2 divide w en x, y y z, teniendo en cuenta las restricciones que se estipulan en el lema de bombeo; $y \neq \epsilon$ y $|xy| \leq n$. De nuevo, el jugador 2 no dice al jugador 1 qué valores tiene x, y y z, auque debe respetar las restricciones.
5. El jugador 1 "gana" eligiendo un valor de k, que puede ser una función de $n, x, y,$ y z, tal que $xy^k z$ no pertenezca a L.

6. Propiedades de la clausura

Principales propiedades de clausura de los lenguajes regulares:

- La **unión** de dos lenguajes regulares es regular
- La **intersección** de dos lenguajes regulares es regular
- El **complementario** de un lenguaje regular es regular
- La **diferencia** de dos lenguajes regulares es regular
- La **reflexión** de un lenguaje regulare es regular
- La **clausura** (*) de un lenguajes regular es regular
- La **concatenación** de dos lenguajes regulares es regular
- Un **homomorfismo** (sustitución de símbolos por cadenas) de un lenguaje regular es regular
- El homomorfismo inverso de un lenguaje regular es regular

6. Propiedades de la clausura

Clausura de lenguajes para las operaciones booleana

1. Sean L y M lenguajes del alfabeto Σ. Luego $L \cup M$ es un lenguaje que contiene todas las cadenas que pertenece a L, a M o a ambos.
2. Sean L y M lenguajes del alfabeto Σ. Luego $L \cap M$ es un lenguaje que contiene todas las cadenas que pertenece a L, como a M.
3. Sean L un lenguaje del alfabeto Σ. Entonces $\overline{L}$, es el lenguaje complementario de L, es el conjunto de todas las cadenas pertenecientes a Σ^* que no pertenecen a L.

6. Propiedades de la clausura

Clausura para la complementación

Si a partir de un AFD se construye otro que acepte el lenguaje complementario. Por lo tanto partiendo de una expresión regular, podríamos encontrar otra para el lenguaje complementario de la forma siguiente:

1. Convertir la expresión regular en un $AFN - \varepsilon$
2. Convertir el $AFN - \varepsilon$ en un AFD
3. Complementar los estados de aceptación de dicho AFD
4. Convertir el AFD complementado en una expresión regular.

Teorema. Si L es un lenguaje regular con el alfabeto Σ, entonces $\overline{L} = \Sigma^* - L$ también es un lenguaje regular.

📌 **Demostración**
Sea $L = L(A)$ para un AFD $A = (Q, \Sigma, \delta, q_o, F)$. Entonces, $\overline{L} = L(B)$, donde B es el AFD $(Q, \Sigma, \delta, q_o, Q - F)$. Es decir, B es exactamente como A pero los estados de aceptación de A tienen que ser estados de no aceptación de B, y viceversa. Entonces w pertenece a $L(B)$ si y sólo si $\hat{\delta}(q_0, w) \in$ a $Q - F$, lo que ocurre si y sólo si $w \notin L(A)$.

6. Propiedades de la clausura

Clausura para la intersección

Se puede obtener la intersección de los lenguajes L y M mediante la siguiente identidad:

$$L \cap M = \overline{\overline{L} \cup \overline{M}}$$

Se puede realizar una construcción directa de un AFD para la intersección de dos lenguaje regulares. Utiliza AFD en paralelo.

Teorema. Si L y M son lenguajes regulares, entonces también lo será $L \cap M$.

📌 **Demostración**

Sean L y M los lenguajes de los autómatas $A_L = (Q_L, \Sigma, \delta_L, q_L, F_L)$ y $A_M = (Q_;, \Sigma, \delta_M, q_M, F_M)$. Los alfabetos de ambos autómatas son idénticos. La construcción de el producto de dos autómatas es funcional tanto para AFD como para AFND. Supongamos que A_L y A_M son AFD para simplificar la explicación.

Para $L \cap M$ construiremos un autómata A que simule tanto a A_L como a A_M.

Formalmente definimos::

$A = (Q_L \times Q_M, \Sigma, \delta, (q_L, q_M), F_L \times F_M)$

donde $\delta((p, q), a) = (\delta_L(p, a), \delta_M(q, a))$.

6. Propiedades de la clausura

Clausura para la diferencia

Teorema. Si L y M son lenguajes regulares, entonces también lo es $L \cap M$.

📌 **Demostración**

$L - M = L \cap \overline{M}$. Si consideramos que $\overline{M}$ es un lenguaje regular y por Teorema sabemos que $L \cap \overline{M}$ es regular. Po r lo tanto, $L - M$ es regular.

6. Propiedades de la clausura

Reflexión

Teorema. Si L es un lenguaje regular L^R también lo es.

📌 **Demostración**

Supongamos que L está definido mediante la expresión regular E. La demostración se hace por inducción estructural sobre el tamaño de E. Demostremos que existe otra expresión regular E^R tal que $L(E^R) = (L(E))^R$, es decir, el lenguaje de E^R es la reflexión del lenguaje de E.

6. Propiedades de la clausura

Homomorfismo

Teorema. Si L es un lenguaje regular con alfabeto Σ y h es el homomorfismo sobre Σ, entonces $h(L)$ también es regular.

📌 **Demostración**

Sea $L = L(R)$ para una expresión regular R. En general, si E es una expresión regular con símbolos de pertenecientes a Σ, hagamos que $h(E)$ sea la expresión que obtenemos reemplazando por $h(a)$ cada símbolo a de Σ que pertenece a E. Vamos a ver qué $h(R)$ define al lenguaje $h(L)$.

La demostración se hace fácilmente por inducción estructural estableciendo que cuando tomamos una sub-expresión de E de R y le aplicamos h obtenemos $h(E)$, el lenguaje de $h(E)$ es lo mismo lenguaje que obtenemos si aplicamos h al lenguaje $L(E)$.

Formalmente $L(h(E)) = h(L(E))$.