Autómatas y Lenguajes

Expresiones regulares:

**Alfabeto** Σ: conjunto finito no vacío de símbolos.

Σab = {a, b}

Σaz = {a, b, c, ..., z}

Σ09 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

Σ01 = {0, 1}

**Palabra o cadena:** secuencia finita (posiblemente vacía) de símbolos de Σ.

a, casa, mesa, abacaba, 110101, 0234.

**λ:** cadena vacía (0 caracteres)

**Lenguaje L**: conjunto (posiblemente vacío o infinito) de palabras sobre Σ

Ejemplos: {casa, mesa, abacbbb}, todas las palabras del castellano, números naturales, todas las secuencias binarias, bytes (cadenas binarias de 8 bits).

Σ ∗ : el lenguaje universal sobre Σ, ej. todas las cadenas alfabéticas (de longitud arbitraria), todas las secuencias binarias.

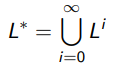
Se cumple ∅ ⊆ L ⊆ Σ ∗ para todo L.

**Concatenación de palabras**: Se pueden concatenar dos cadenas α y β, escrito α.β, aunque el punto puede omitirse. Ej. la concatenación de casa y mesa es casamesa. Nótese que α.λ = λ.α = α.

**Concatenación de lenguajes**: Dados dos lenguajes L1 y L2, se define su concatenación L1.L2 = {α.β | α ∈ L1 y β ∈ L2}.

Ejemplo: si L1 = {a, aa} y L2 = {λ, b, bb}, entonces, considerando todas las posibles concatenaciones (de forma análoga a un producto cartesiano), tenemos L1.L2 = {a, ab, abb, aa, aab, aabb}

**Clausura de un lenguaje L**: Se define L 0 = {λ}, y para i > 0, Li = L.Li-1 (es decir, L . . . i veces . . . L) La clausura L∗ de L es:



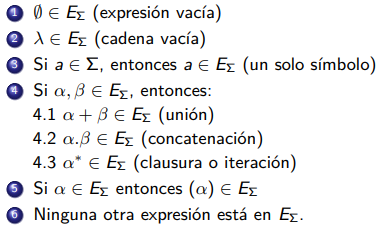
Por ejemplo, si L = {casa, mesa,silla}, entonces:

L0 = {λ}

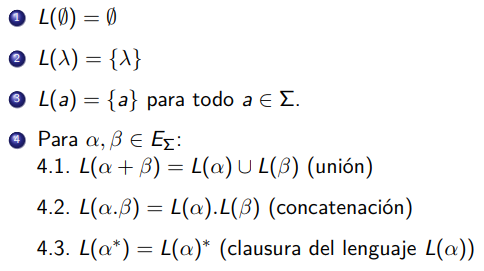
L1 = L.L 0 = L

L2 = L.L1 = L.L = {casacasa, casamesa, casasilla, mesacasa, mesamesa, mesasilla,sillacasa,sillamesa,sillasilla}

L3 = L.L2 = L.L.L = . . .

**Las expresiones regulares** son un formalismo conciso para definir lenguajes en el sentido de la sección anterior. El conjunto de expresiones regulares EΣ para un alfabeto dado Σ se define mediante las siguientes reglas:

Precedencia: ‘\*’ > ‘.’ > ‘+’, similar al de exponenciación, multiplicación y suma en aritmética. Por ejemplo, ((0(1 ∗ )) + 0) = 01∗ + 0.

Cada expresión regular r ∈ EΣ define un lenguaje L(r) ⊆ Σ ∗ sobre el alfabeto Σ, de acuerdo a las siguientes reglas:

No contiene la subcadena 110 (0 + 10) ∗1 ∗ 11

Contiene al menos dos 0s, pero no consecutivos 1 ∗ (01∗1) ∗ (011∗0)1 ∗ 12

Tiene al menos 3 números, y el tercero es 0 (0 + 1)(0 + 1)0(0 + 1) ∗ 13

El número de 0s es un múltiplo de 3 1 ∗ + (1 ∗01∗01∗01∗ ) ∗ 14

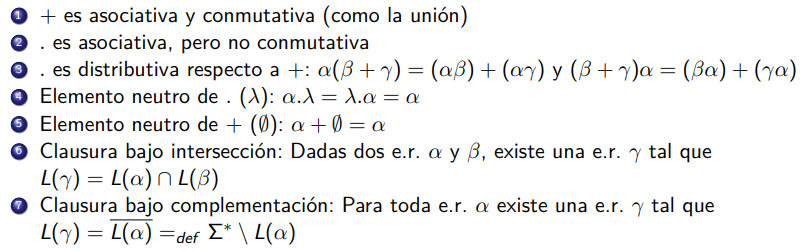
Empieza y termina con el mismo número 1(0 + 1) ∗1 + 0(0 + 1) ∗0

Longitud impar (0 + 1)((0 + 1)(0 + 1))∗ 16

Empieza con 0 y tiene longitud impar, o empieza con 1 y tiene longitud par 0((0 + 1)(0 + 1))∗ + 1(0 + 1)((0 + 1)(0 + 1))∗ 17

Longitud al menos 1 y como mucho 3 (0 + 1) + (0 + 1)(0 + 1) + (0 + 1)(0 + 1)(0 + 1)

Dos expresiones regulares α y β son equivalentes si y solo si L(α) = L(β). Abusando de la notación, escribimos α = β



Nota: la longitud de γ en los dos últimos puntos puede ser exponencialmente mayor que la de las expresiones de partida

Ejemplos

¿Qué lenguaje define una e.r. dada? La respuesta a veces depende de una demostración formal.

Ejemplo 1: (0 + 1) ∗00(0 + 1) ∗ : cadenas de bits con dos ceros consecutivos, “obviamente”. Ejemplo 2: el complemento del lenguaje anterior, las cadenas de bits que no contienen dos ceros consecutivos:

¿Qué lenguaje define una e.r. dada? La respuesta a veces depende de una demostración formal.

Ejemplo 1: (0 + 1) ∗00(0 + 1) ∗ : cadenas de bits con dos ceros consecutivos, “obviamente”. Ejemplo 2: el complemento del lenguaje anterior, las cadenas de bits que no contienen dos ceros consecutivos: (01 + 1) ∗ (0 + λ)

¿Qué lenguaje define una e.r. dada? La respuesta a veces depende de una demostración formal.

Ejemplo 1: (0 + 1) ∗00(0 + 1) ∗ : cadenas de bits con dos ceros consecutivos, “obviamente”. Ejemplo 2: el complemento del lenguaje anterior, las cadenas de bits que no contienen dos ceros consecutivos: (01 + 1) ∗ (0 + λ)

Intuitivamente, podemos ver que en L((01 + 1) ∗ ) todo 0 va seguido de un 1, por tanto no hay dos 0’s consecutivos, pero por otro lado esta subexpresión no permite un 0 terminal. Esta posibilidad es la que se obtiene concatenando (0 + λ) a la expresión anterior.

¿Podemos formalizar esta intuición?

1 Demostrar que ningún s ∈ L((01 + 1) ∗ (0 + λ)) contiene dos 0s consecutivos.

2 Demostrar que todo s ∈ Σ ∗ 01 que no contenga dos 0s consecutivos pertenece a L((01 + 1) ∗ (0 + λ)).

“Son todas las que están y están todas las que son”

“Son todas las que están”

Hipótesis inductiva (HI): (01 + 1) i no contiene dos 0’s consecutivos y no termina en 0.

Obvio para i = 0.

Para i > 0, asumir que (HI) se cumple para todo s ∈ L((01 + 1) i−1 ) ⇒ (HI) se cumple para s.01 y s.1, dado que s no termina en 0 ⇒ (HI) se cumple también para L((01 + 1) i ).

Puesto que (HI) se cumple para todo i, se cumple para L((01 + 1) ∗ ).

Consideremos ahora las cadenas de L((01 + 1) ∗ (0 + λ)). Estas tienen la forma s.0 o s.λ, para algún s ∈ L((01 + 1) ∗ ). Pero puesto que s no contiene dos 0’s consecutivos ni termina en 0, tampoco s.0 o s.λ contienen 0’s consecutivos, como queríamos demostrar.

“Están todas las que son”

Si *s* no contiene dos 0’s consecutivos, todo 0, excepto si ocurre en la última posición, va seguido de un 1.

Por tanto, s puede particionarse en cadenas de 1’s seguidas de 01 seguidas de 1’s, etc., posiblemente con un 0 final. Por ejemplo, 11101101111101110 se particiona en 111-01-1-01-1111-01-111-0.

Así, *s* sin 0 final está en L((01 + 1) i ), donde i es la suma de todos los 1’s de las subcadenas de 1’s más el número de subcadenas 01. Luego s, tenga o no un 0 final, está en L.

Autómatas finitos:

Un autómata es un modelo matemático de un dispositivo computacional.

Estudiaremos varios tipos de autómatas desde el punto de vista de su capacidad de cómputo

En concreto, de su capacidad para decidir si una cadena pertenece a un lenguaje, es decir, para el problema de reconocimiento de palabras.

El autómata consta de un conjunto de “estados” y “consume”, como input, una cadena de símbolos, que le hacen transitar entre esos estados.

Ejemplo: un interruptor tiene dos estados, encendido y apagado. Podemos suponer que al pulsar el interruptor se consume el símbolo p (pulsar), y tras una secuencia de pulsaciones, el autómata nos dirá cuál es el estado final.

Diagrama

Descripción generada automáticamente

**Mirar representación en diapositivas**

Un autómata finito es una colección (finita) de estados unidos por transiciones.

Hay un estado distinguido, el estado inicial.

Otros estados son distinguidos como estados de aceptación.

El autómata procesa la cadena de entrada empezando por el estado inicial y siguiendo las transiciones indicadas en cada caso para el símbolo que se está leyendo.

Si el autómata, tras procesar todo el input, termina en un estado de aceptación, acepta la entrada.

Si no, la rechaza.

Importante: un autómata no acepta una cadena solo por “pasar” por un estado de aceptación. Solo la acepta si termina en un estado de aceptación, tras haber procesado (o “consumido”) toda la cadena

**Mirar representación en diapositivas**

El lenguaje de un autómata es el conjunto de cadenas que acepta

Si D es un autómata que procesa símbolos del alfabeto Σ, se define su lenguaje

L(D): L(D) = {s ∈ Σ\* | D acepta s}

Autómatas Finitos Deterministas (AFDs) ⇒ Necesitamos una definición formal de autómata

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Imagen de la pantalla de un celular de un mensaje en letras blancas

Descripción generada automáticamente con confianza baja

Autómatas finitos no deterministas:

Determinismo y no determinismo

• En los AFDs, para cada estado y símbolo existe exactamente una transición definida, que determina el estado siguiente de forma unívoca al leer el símbolo en ese estado.

• En cada estado de la computación solo hay una elección posible sobre qué paso tomar.

Determinismo y no determinismo

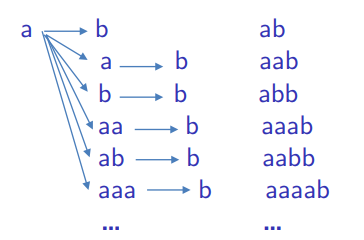
• En los AFDs, para cada estado y símbolo existe exactamente una transición definida, que determina el estado siguiente de forma unívoca al leer el símbolo en ese estado.

• En cada estado de la computación solo hay una elección posible sobre qué paso tomar.

Determinismo y no determinismo

• A menudo, sin embargo, es conveniente pensar en diferentes opciones posibles en cada estado de la computación.

• Ejemplo: Lenguaje a(a+b)\*b de cadenas que empiezan con a y terminan con b

Generar una cadena del lenguaje implica múltiples opciones: 

Verificar si una cadena pertenece al lenguaje también puede verse así.

• En a(a+b)\*b, al leer una b, ¿es una b terminal o no?

• En a(a+b)\*b(c+d), al leer una b, ¿es esta el principio del sufijo b(c+d) o no?

Determinismo y no determinismo

• Un modelo de computación se llama no determinista si pueden existir múltiples opciones que se pueden tomar en un estado de la computación

• El dispositivo de computación acepta si alguna secuencia de elecciones lleva a un estado de aceptación.

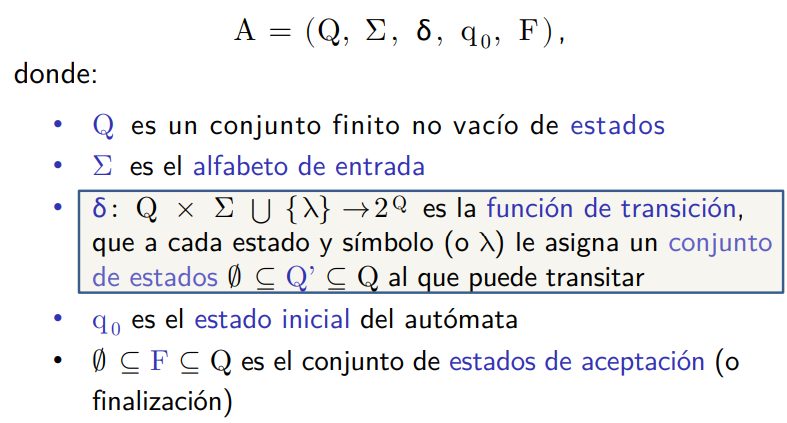
**AFNs**: Autómatas Finitos No Deterministas Muy similares a los AFDs, pero representan un cambio fundamental en la forma de pensar sobre la computación

• Se permiten un conjunto de transiciones para cada estado y símbolo dado.

• Este conjunto puede ser vacío

• Se permiten también “transiciones λ”, que se pueden hacer sin leer ningún símbolo (“leyendo λ”, por así decir).

Formalmente, un Autómata Finito No Determinista (AFN) es un tuplo:



Recordad: la principal diferencia con los autómatas deterministas es que ahora se puede transitar a uno cualquiera de un conjunto de estados al leer un símbolo, sin que esté predeterminado un único estado.

Este conjunto de estados representa alternativas de posibles ejecuciones del autómata ante una misma cadena.

Ejemplos: