Tema 2

Análisis Léxico

Universidad Europea
LAUREATE INTERNATIONAL UNIVERSITIES

2019 - 2020 – Prof. Leopoldo Santos

Análisis léxico

- El Análisis Léxico
- Funciones del analizador léxico y ventajas
- Definiciones básicas
- ¿Cómo funciona el analizador léxico?
- Diseño de un analizador léxico
- Reconocimiento de identificadores
- Errores léxicos

Análisis Léxico

- **Léxico**.

  1. m. Diccionario de una lengua.

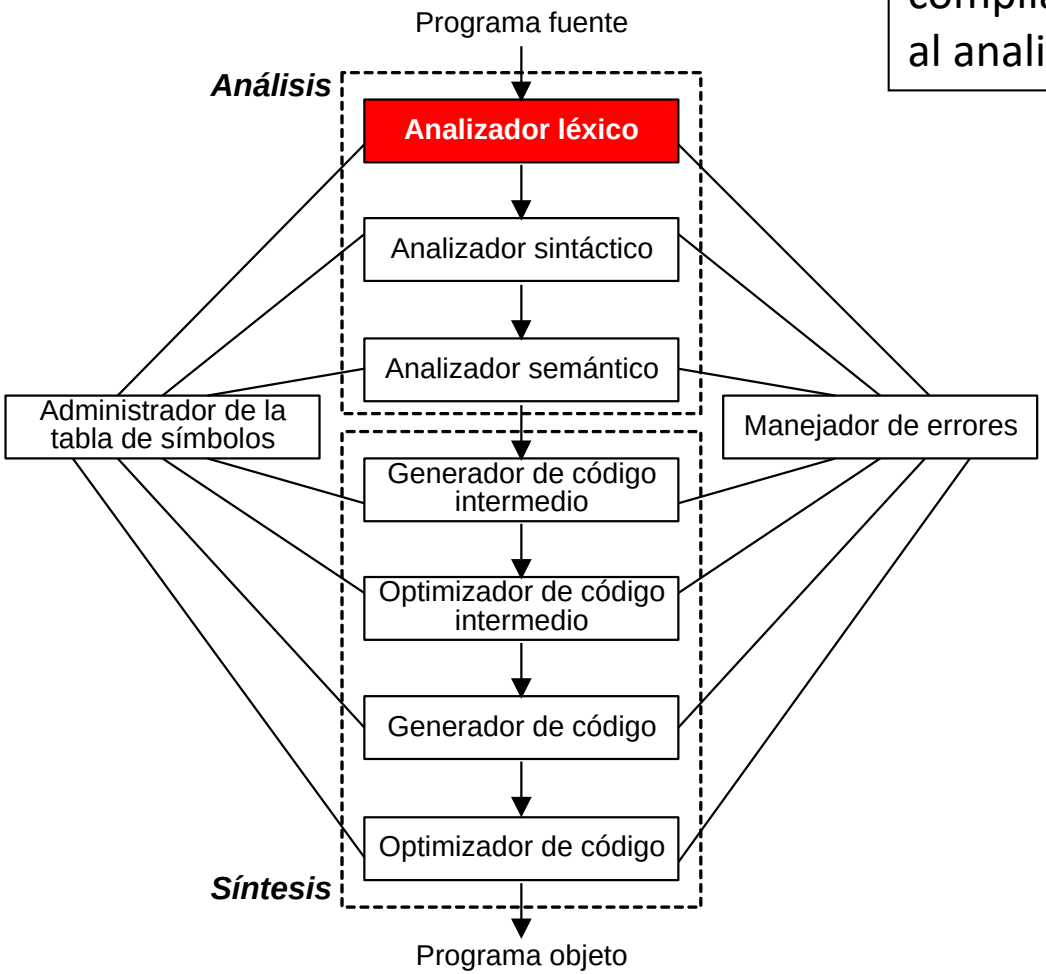
  2. m. Vocabulario, conjunto de las palabras de un idioma, o de las que pertenecen al uso de una región, a una actividad determinada, a un campo semántico dado, etc.

  Diccionario de la RAE

- Analizador léxico (*scanner*). Lee los caracteres de la entrada y elabora como salida una secuencia de componentes léxicos pertenecientes al LP.

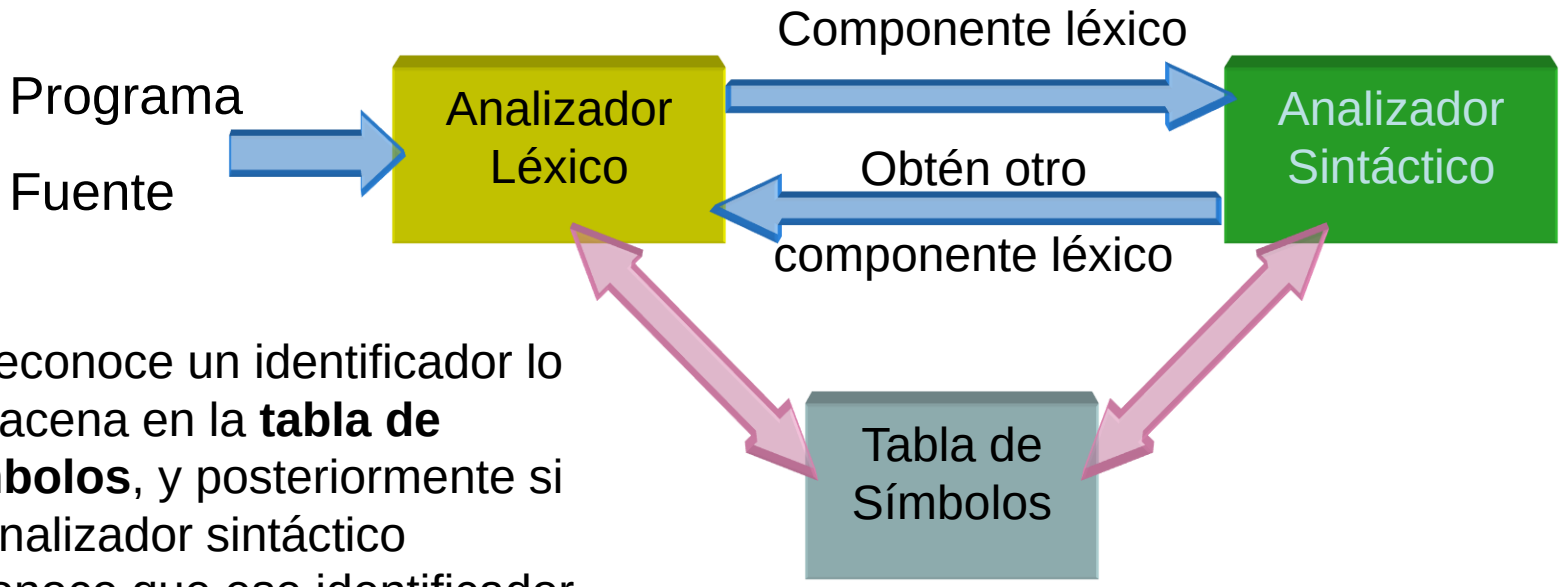
Análisis Léxico

Tiene como entrada el código fuente del lenguaje de programación que acepta el compilador y como salida, proporciona al analizador sintáctico los tokens.



¿Qué es un token? Es una agrupación de caracteres reconocidos por el analizador léxico que constituyen los símbolos con los que se forman las sentencias del lenguaje. Lo que el analizador léxico devuelve al analizador sintáctico es el nombre de ese símbolo junto con el valor del atributo (si ese token lo necesita, ya que no todos los tokens llevan atributo, p. ej. una palabra reservada –if-)

Análisis Léxico



Si reconoce un identificador lo almacena en la **tabla de símbolos**, y posteriormente si el analizador sintáctico reconoce que ese identificador lleva asociada información tipo (entero, real, etc.) también añade información a la mencionada tabla

Sistema de gestión de errores: Se encarga de detectar símbolos que no pertenezcan a la gramática porque no encajen con ningún patrón, bien porque haya caracteres inválidos (Ej. @), o bien porque se escriban mal las palabras reservadas del lenguaje, los identificadores o los números (5.25 en lugar de 5,25)

Funciones del Análizador Léxico

- **Gestionar el fichero** que contiene el código fuente, entendiéndose por ello el abrirlo, leerlo y cerrarlo
- **Eliminar** comentarios, tabuladores, espacios en blanco, saltos de línea.
- **Relacionar** los errores con las líneas del programa
- **Expansión de macros**.
- **Inclusión de ficheros**.
- Reconocimiento de las **directivas de compilación**.
- **Introducir identificadores en la tabla de símbolos** (esta función es opcional, pudiendo realizarse también por parte del analizador sintáctico).

Funciones del Analizador Léxico

VENTAJAS:

- Se simplifica el diseño, puesto que hay una herramienta especializada en el tratamiento del fichero que contiene el código fuente.
- Aumenta la portabilidad del compilador, pudiendo tenerse versiones diferentes para distintos formatos del texto de código fuente (ASCII, EBCDID, …)
- Mejora la eficiencia al ser una herramienta especilizada en el tratamiento de caracteres.
- Detección de determinados errores fáciles de corregir a este nivel (5.25 por 5,25).

Definiciones básicas

- Componente léxico = token + atributos.
  - *token*: elemento sintáctico que describe el tipo de componente léxico (símbolos terminales de la gramática).
  - *patrón* (asociado a un token): conjunto de cadenas para las que se obtiene el mismo token.
  - *lexema*: secuencia de caracteres en el programa fuente con la que concuerda el patrón asociado a un token.
- Ejemplo: **if x1 > 35 then a:= b + x1**

token	**Lexemas**	**Patrón**
If, then	If, then	“if”, “then”
Mas, Mayor, Asigna	+, >, :=	“+”, “>”, “:=“
Identificador	a, b, x1	Letra seguida de letras y dígitos
Número	35	Secuencia de dígitos

Definiciones básicas

- La pregunta es: ¿Dónde parar?
  - Como hemos comentado, debe existir un compromiso entre esfuerzo y beneficios.
  - Por un lado, debemos asignar al analizador léxico una parte razonable del análisis.
  - El analizador léxico no debe ocuparse de analizar frases enteras.
  - Los elementos básicos se corresponden generalmente con los elementos básicos que maneja un programador:
    - Palabras reservadas, identificadores, literales, operadores…

Definiciones básicas

Ejemplo: Lenguaje JAVA

PROG	→	LISTA_CLASES
	|	λ
LISTA_CLASES	→	CLASE ; LISTA_CLASES
	|	CLASE ;
CLASE	→	MODIFICADOR class IDENT { CUERPO_CLASE }
MODIFICADOR	→	public
	|	private
CUERPO_CLASE	→	LISTA_VAR_FUNC
	|	λ
LISTA_VAR_FUNC	→	DECL_VAR ; LISTA_VAR_FUNC
	|	DECL_FUNC ; LISTA_VAR_FUNC
	|	DECL_VAR
	|	DECL_FUNC

Definiciones básicas

DECL_VAR	→	MODIFICADOR TIPO IDENT
	|	MODIFICADOR TIPO IDENT = VALOR
DECL_FUNC	→	MODIFICADOR TIPO_FUNC IDENT (LISTA_PARAM_OPC) { CUERPO_FUNC }
TIPO	→	int
	|	float
	|	IDENT
TIPO_FUNC	→	TIPO
	|	void
LISTA_PARAM_OPC	→	LISTA_PARAM
	|	λ
LISTA_PARAM	→	PARAM , LISTA_PARAM
	|	PARAM

Definiciones básicas

PARAM	→	TIPO IDENT
CUERPO_FUNC	→	LISTA_SENT
	|	λ
LISTA_SENT	→	SENT ; LISTA_SENT
	|	SENT ;
SENT	→	IDENT = NUMERO
	|	IDENT = IDENT
IDENT	→	LETRA
	|	LETRA RESTOIDENT
RESTOIDENT	→	ALFANUM
	|	ALFANUM RESTOIDENT
NUMERO	→	DIGITO
	|	DIGITO NUMERO

Definiciones básicas

ALFANUM	→	LETRA
	|	DIGITO
LETRA	→	a | b | ... | z | A | B | ... | Z
DIGITO	→	0 | 1 | ... | 9
VALOR	→	IDENT
	|	NUMERO

Expresiones Regulares

- Hay tres operaciones básicas con expresiones regulares
- **Selección entre alternativas**: Se denota por el **metacaracter |**.
  - Ej: a|b, significa que puede ser a ó b. Esta operación equivale a la unión, puesto que tanto a como b valdrían como lexemas para este patrón (obsérvese que utilizamos patrón y expresión regular de forma indistinta).
- **Concatenación**: Se construye poniendo un símbolo al lado del otro y no utiliza ningún metacaracter.
  - Ej: ab, significa que el lexema equivalente tiene que ser “ab”, sin alternativa posible.
- **Repetición**: También se la denomina cerradura de Kleene y se denota por el matacaracter *. Identifica una concatenación de símbolos incluyendo la cadena vacia, es decir “0 o mas instancias” del símbolo afectado.
  - Ej: a*, significa que los lexemas para este patrón podrían ser: λ, a, aa, aaa, aaaa, …

Expresiones Regulares

- **Una o mas repeticiones**: Se denota por el metacaracter **+**, también se le denomina cierre positivo. Esta operación indica que el símbolo afectado tendrá una o mas instancias.
  - Ej: a+, significa que los lexemas para esta expresión regular podrían ser: a, aa, aaa, aaaa, …
- **Cero o una instancia**: Se denota por el metacaracter **?**, y significa cero o una ocurrencia del símbolo afectado.
  - Ej: a?, significa que los léxemas que podrían valer son: λ ó a. Describe a un símbolo opcional
- **Intervalo de caracteres**: Para especificar un intervalo de caracteres todos ellos válidos, podríamos usar la alternativa o utilizar los corchetes y un guión.
  - Ej: [a-z], significa cualquier carácter de la a a la z, que también se podría haber especificado por a|b|c|d y así sucesivamente hasta la z, pero esta forma es mucho mas abreviada y evita errores. También se pueden incluir los intervalos múltiples [a-zA-Z], que representa todas las letras minúsculas y mayúsculas.

Expresiones Regulares

- **Clases de caracteres**: Es como el intervalo pero sin el guión y sirve también para abreviar las alternativas.
  - Ej: [abc], equivale a a|b|c.
- **Cualquier carácter**: Se denota por un punto **.,** Sirve para expresar que cualquier carácter encaja con la expresión regular. Se suele utilizar al final de la especificación de un analizador léxico por si queremos hacer alguna acción para todo lo que no concuerde con los patrones definidos.
- **Cualquier carácter que no pertenezca a un conjunto**: Se denota por **^**, o tambien por la **tilde ~**, y significa cualquier carácter distinto a los que está afectando el metacaracter.
  - Ej: [^a], significa cualquier carácter que no sea a.

Especificación de los componentes léxicos: Expresiones regulares

- Ejemplos. Sea V = {a, b}:
  - a | b $\Rightarrow$ {a, b}.
  - (a | b)(a | b) $\Rightarrow$ {aa, ab, ba, bb}.
  - a^* $\Rightarrow$ {λ, a, aa, aaa, ... }.
  - $(a \mid b)^*$ $\Rightarrow$ {λ, a, b, aa, ab, ba, bb, aaa, ... }
  - $\Rightarrow$ todas las cadenas de a y b.

- Si dos expresiones regulares r y s representan el mismo lenguaje, se dice que r y s son equivalentes (r = s).
  - a | b = b| a.

Especificación de los componentes léxicos: Definiciones regulares

Ejemplo. Identificadores en C.

LETRA $\Rightarrow$ A | B | ... | Z | a | b | ... | z

DIGITO $\Rightarrow$ 0 | 1 | ... | 9

IDENTIFICADOR $\Rightarrow$ ({LETRA} | “_”) ({LETRA} | {DIGITO} | “_”)*

Especificación de los componentes léxicos: Definiciones regulares

- Ejemplo. Números reales sin signo en C (X.XE±X).

DIGITO $\Rightarrow$ 0 | 1 | ... | 9

DIGITOS $\Rightarrow$ {DIGITO} {DIGITO}*

FRACCION_OPTATIVA $\Rightarrow$. {DIGITOS} | λ

EXPONENTE_OPTATIVO $\Rightarrow$ E (+ | - | λ) {DIGITOS} | λ

NUMERO $\Rightarrow$ {DIGITOS} {FRACCION_OPTATIVA} {EXPONENTE_OPTATIVO}

Especificación de los componentes léxicos: Abreviaturas en la notación

- + $\Rightarrow$ Uno o más casos (cierre positivo).
  - $r^* = r^+ \mid \lambda$ $\qquad\qquad$ $r^+ = r\, r^*$
- ? $\Rightarrow$ Cero o un caso.
  - r? = r | λ

    FRACCION_OPTATIVA → (.DIGITOS)?
- Clases de caracteres.
  - [abc] = a | b | c
  - [a-z] = a | b | ... | z

    IDENTIFICADOR → [A-Za-z][A-Za-z0-9]*

Especificación de los componentes léxicos: Expresiones regulares

- Ejemplos:
  - Números binarios múltiplos de dos:
    $(0 \mid 1)^* \cdot 0$
  - Cadenas que contengan al menos una *b*.
    $.^*b.^*$
  - Cadenas de *a* y *b* sin *a* consecutivas.
    $b^*(abb^*)^*(a|\lambda)$
  - Cadenas de *a* y *b* con *a* consecutivas.
    $(a|b)^*aa(a|b)^*$

Especificación de los componentes léxicos: Definiciones regulares

– Ejemplo. Números reales con signo en C (±X.XE±X).

DIGITO $\Rightarrow$ [0-9]

NATURAL $\Rightarrow$ {DIGITO}$^{+}$

NAT_CON_SIGNO $\Rightarrow$ (+|-)? {NATURAL}

NUMERO $\Rightarrow$ {NAT_CON_SIGNO}? (“.” {NATURAL}) (E {NAT_CON_SIGNO})?

Reconocimiento de componentes léxicos

- Las ER permiten especificar componentes léxicos, pero se necesita un formalismo que se pueda implementar como un programa de ordenador.

- Un reconocedor de un lenguaje es un programa que toma como entrada una cadena x y responde “sí” si x está en el lenguaje, y “no”, si no lo está.

- **Usaremos un autómata finito como reconocedor para una expresión regular**.

Definiciones básicas

- Autómata Finito
  - Es un modelo de estados que realiza acciones de forma automática sobre una entrada para producir una salida.
  - Nos sirven para representar las Expresiones Regulares (ER)

  - Se pueden representar mediante:
    - Tabla de transiciones,
    - Diagrama de transiciones

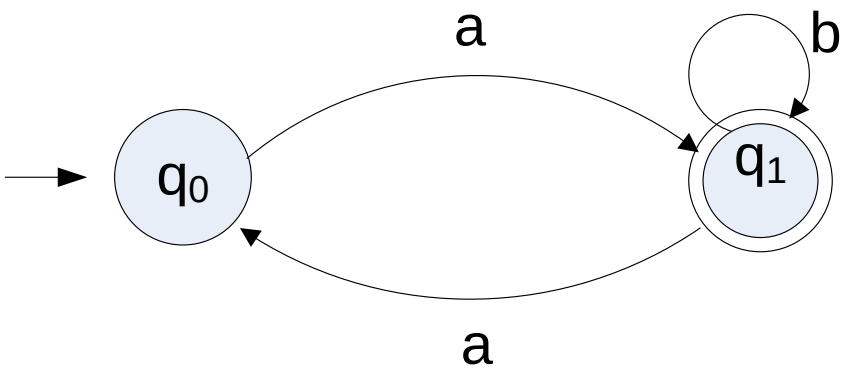
Diseño de un Analizador Léxico

- **Tabla de transiciones**:
  - En las filas colocamos los estados (q) que pertenecen al conjunto de estados Q
  - En las columnas estarán los símbolos de la entrada (e) y que pertenecen al alfabeto ∑
  - El estado inicial llevará el siguiente símbolo →
  - Los estados finales llevarán el siguiente símbolo *
  - En la posición (fila, columna) tendremos el estado que determina la función f(q, e)

f	a	b
-->q_0	q_1	q_1
*q_1	q0	q_1

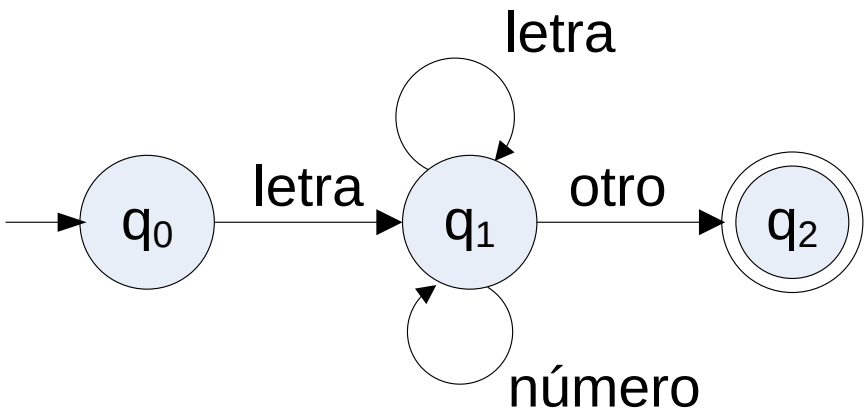
Diseño de un Analizador Léxico

- **Diagrama de transiciones**:
  - El estado inicial llevará el símbolo →
  - En los nodos se mostrarán los estados
  - Los arcos unirán los estados con el símbolo de la entrada
  - Los estados finales tendrán un doble círculo (equivalente al * en la tabla)



Reconocimiento de Identificadores

- La forma de representar mediante expresiones regulares cualquier letra mayúscula o minúscula es: **[a-z][A-Z]** y le denominamos ***letra***. La forma de representar un número cualquiera es **[0-9]** y le denominamos ***número***. Para finalizar se define como **[otro]** cualquier otro símbolo, indicando que ya ha terminado de definirse el identificador. Esto lo representamos mediante un diagrama de transiciones de la siguiente manera



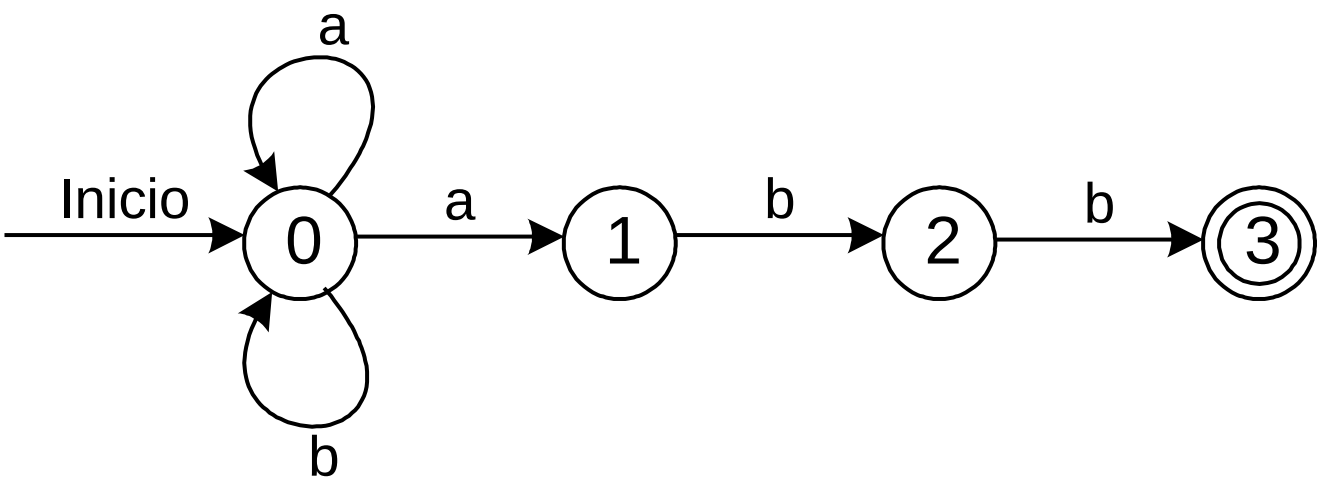
Reconocimiento de componentes léxicos
Autómatas Finitos

- La implementación más sencilla para el AF es una tabla de transiciones.
  - Una fila para cada estado.
  - Una columna por cada símbolo de entrada y λ si es necesario.
  - La entrada para la fila i y el símbolo a es el conjunto de estados que puede ser alcanzado por una transición del estado i con la entrada a.
- Ejemplo: **(a|b)*abb**.

Estado	**Símbolo de entrada**	
	a	**b**
0	{0,1}	{0}
1	-	{2}
2	-	{3}

Reconocimiento de componentes léxicos
Autómatas Finitos

- Ejemplo: **(a|b)*abb**.
  - S = {0, 1, 2, 3}
  - V = {a, b}
  - s_o = 0
  - F = {3}



Reconocimiento de componentes léxicos
Autómatas Finitos

- Un autómata finito se suele representar con un grafo dirigido, y está formado por:
  - Un conjunto de estados S.
  - Un conjunto de símbolos de entrada (alfabeto o vocabulario) V.
  - Un conjunto de transiciones (o movimientos) de un estado a otro, etiquetados con caracteres en V.
  - Un estado inicial s_0.
  - Un conjunto de estados finales (o de aceptación) F.

Reconocimiento de componentes léxicos
Autómatas Finitos

- Tipos:
  - **Autómata Finito Determinista** (AFD). No puede tener varias transiciones salientes con el mismo símbolo.
  - **Autómata Finito No determinista** (AFND). Puede tener una o ninguna transiciones salientes con el mismo símbolo.

- Generalmente, el análisis mediante un AFD es más rápido, pero tiene más estados.

Autómatas Finitos No deterministas

- Un AFND se distingue por:
  - Puede tener varias transiciones de salida con el mismo símbolo.
  - Puede tener transiciones etiquetadas con la cadena vacía (λ). Estas transiciones no consumen ningún símbolo de la entrada.

- Un AFND acepta una cadena de entrada x si y sólo si:
  - Hay algún camino en el grafo de transiciones desde el estado inicial a algún estado final.
  - Las etiquetas de las aristas a lo largo de dicho camino deletrean x.

Conceptos de AFND

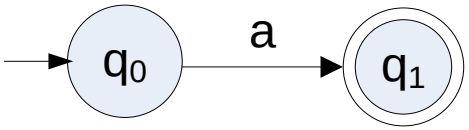
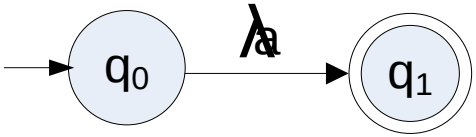
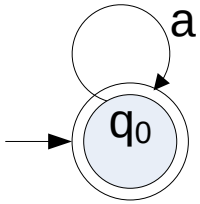
- **Lenguaje asociado a un AFND**: Es el conjunto de palabras que le hacen transitar desde el estado inicial a algún estado final, utilizando para ello la función **f.**
- **Equivalencia entre AFD y AFND**: Para cada AFD existe un AFND equivalente y viceversa. De hecho los AFD son un caso particular de los AFND. En la práctica el AFD tiene casi el mismo número de estados que el AFND, aunque normalmente tiene mas transiciones
- **Transformación de AFND en AFD**: De todo AFND se puede obtener un AFD, de tal forma que el lenguaje que reconozca el AFD sea equivalente al lenguaje reconocido por el AFND. Este concepto es importante puesto que es la base del siguiente tema, donde se convertirá el AFND obtenido a partir de una expresión regular en el AFD equivalente.

Conceptos de AFND

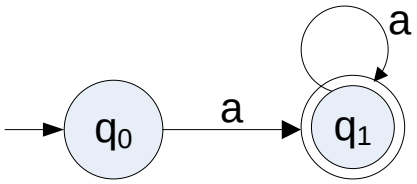
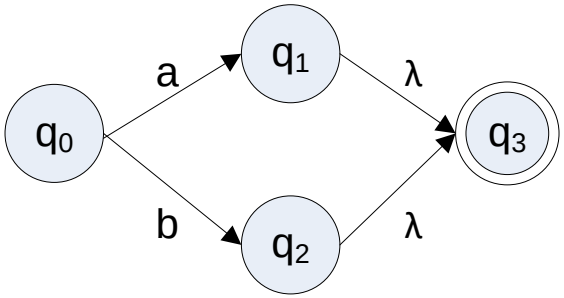
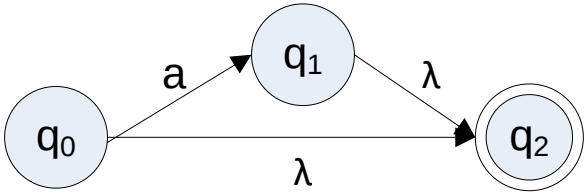
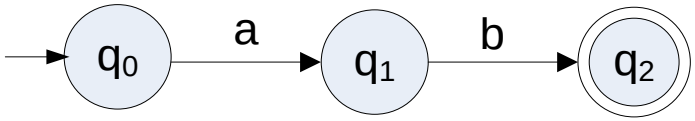
- **Transformación de una Expresión Regular en un Autómata Finito**: A partir de una expresión regular se puede obtener el autómata finito (determinista o no), capaz de reconocer el lenguaje que dicha expresión representa y viceversa, es decir, que a partir de un autómata finito, se puede obtener la expresión regular que este autómata representa.
- Para poder realizar esta transformación, **necesitamos conocer las equivalencias** entre las ER básicas y los autómatas finitos que las representan.
- **Equivalencias entre Expresiones Regulares básicas y autómatas finitos**: Vamos a ver las expresiones regulares básicas: repetición en sus distintas variantes, alternativa y concatenación y su equivalente AF.

Conceptos de AFND

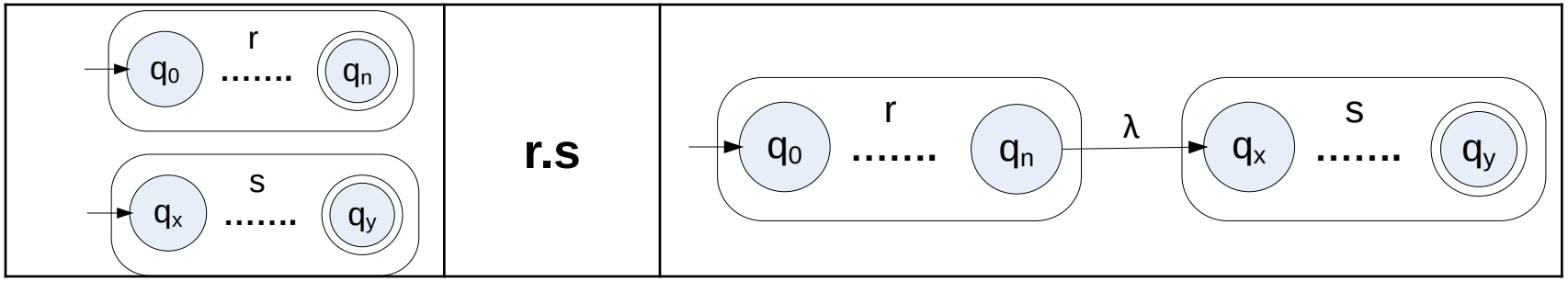
- **Equivalencias entre Expresiones Regulares básicas y autómatas finitos**: Vamos a ver las expresiones regulares básicas: repetición en sus distintas variantes, alternativa y concatenación y su equivalente AF.

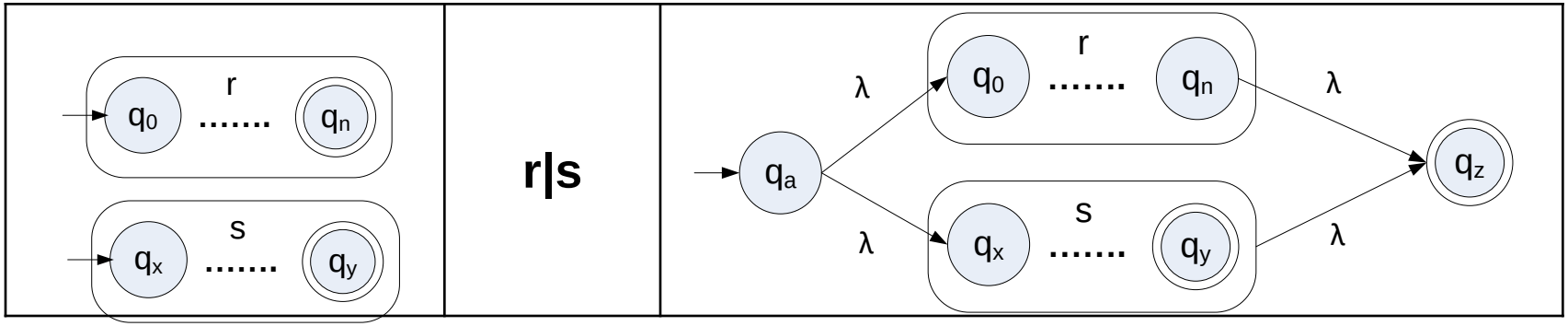
ER = a	
ER = λ	
ER = a* (cero o mas repeticiones de a)	

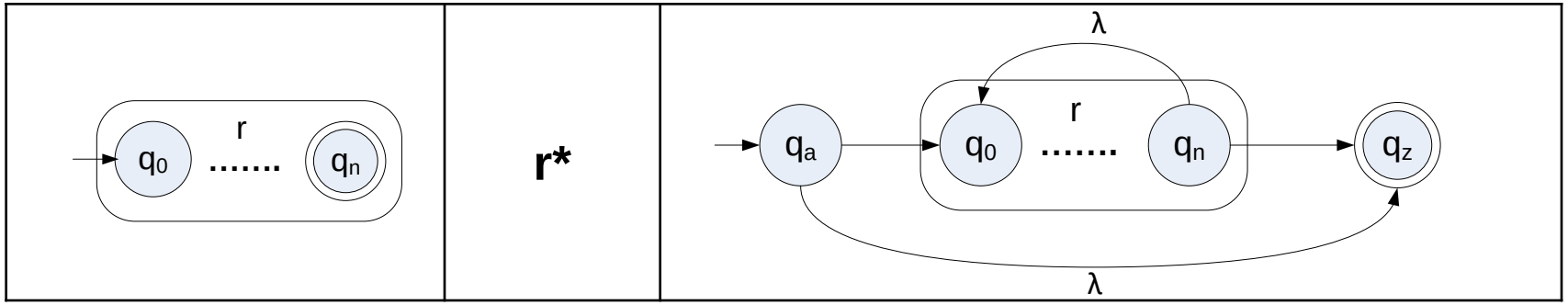
Conceptos de AFND

ER = a+ = aa* (una o mas repeticiones de a)	
ER = a|b (Alternativa)	
ER = a? = a|λ (cero o una instancia de a)	
ER = ab (Concatenación)	

Diagramas de Thomson

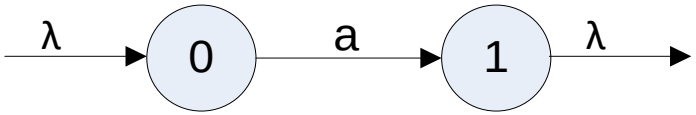




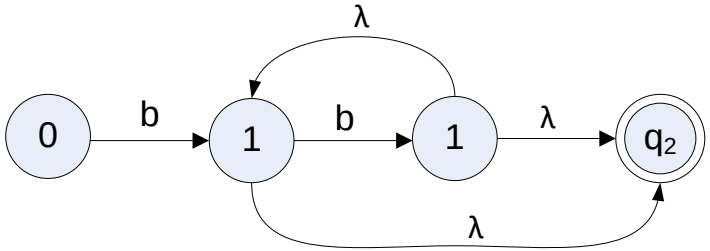


Conversión de una ER en un AFN

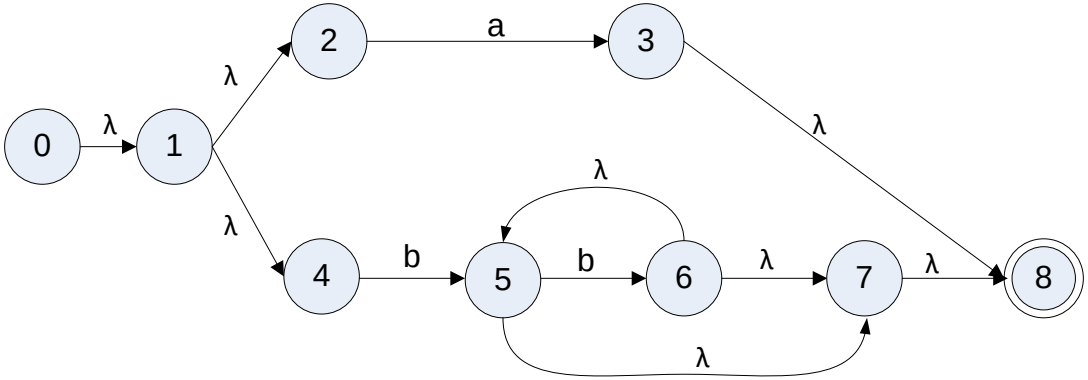
- Ejemplo: Construir el AFN para la cadena **(a|b+)b?**
- **Paso 1:** Hacemos la construcción de Thomson para **a.**



- **Paso 2**: Obtenemos hacemos la construcción de Thomson de **b+** (recordemos que es igual que bb*).

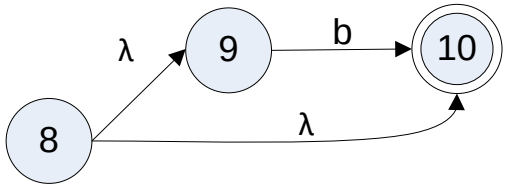


- **Paso 3**: Realizamos la alternativa **a|b+**

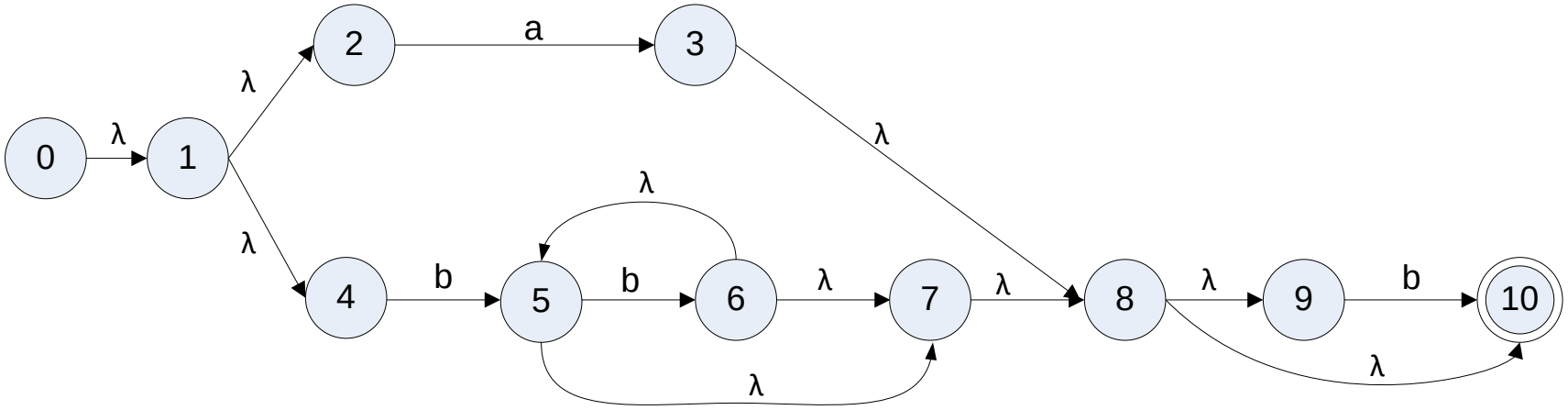


Conversión de una ER en un AFN

- **Paso 4**: Realizamos **b?,** que indica cero o una instancia de b



- **Paso 5**: Lo unimos todo para obtener **(a|b+)b?**



Autómatas Finitos Deterministas

- Un AFD es un caso particular de AFN que cumple:
  - No existen estados con λ-transiciones.
  - Para cada estado *s* y símbolo de entrada *a*, existe una única arista de salida etiquetada como *a*.

Autómatas Finitos Deterministas

- Podemos simular un AFD con el siguiente pseudocódigo:

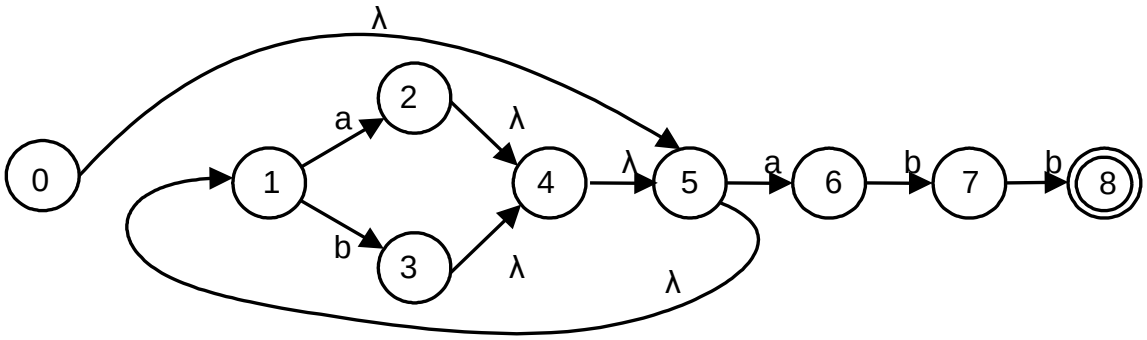
```
s := s0;
c := siguiente_carácter();
mientras c≠eof hacer
        s := mueve(s, c);
        c := siguiente_carácter();
fin
si s está en F entonces
        devolver “sí”
si no
        devolver “no”;
```

Autómatas Finitos Deterministas
Conversión de un AFN en un AFD

- Implementar un AFD es más sencillo que un AFN.
  - En cada estado del autómata, existe una única transición posible para un símbolo determinado.
  - No existen λ-transiciones.
    $\Rightarrow$ No hay que probar varias posibilidades en cada estado.

Autómatas Finitos Deterministas
Conversión de un AFN en un AFD

- Ejemplo AFN para $(a|b)^*$ abb.
  - En lugar de adivinar qué λ-transición debemos tomar, diremos que el AFN puede tomar cualquiera, y formamos un estado conjunto: {0, 5, 1} (cierre-λ({0})).
  - Ahora tomamos la transición para el carácter a. Desde el estado 1, podemos alcanzar el 2; y desde el estado 5, podemos alcanzar el 6. Así, tenemos el estado {2, 6}.
  - Si computamos el cierre-λ({2,6}), obtenemos {1, 2, 4, 5, 6}.



Autómatas Finitos Deterministas
Conversión de un AFN en un AFD

- Definición formal de cierre-λ.
  - Sea transición(s, c) el conjunto de todos los estados del AFN alcanzables desde s mediante transiciones directas etiquetadas con c.
  - Para un conjunto de estados S, cierre-λ(S) es el conjunto de estados que pueden ser alcanzados desde S sin consumir ningún símbolo de entrada.

Autómatas Finitos Deterministas
Conversión de un AFN en un AFD

- Cálculo de cierre-λ(S).

```
meter todos los elementos de S en una pila
inicializar cierre-λ(S) a S
mientras la pila no esté vacía hacer
    sacar s, el elemento tope de la pila
    para cada estado t alcanzable desde s mediante λ hacer
        si t no está en cierre-λ(S) entonces
            añadir t a cierre-λ(S)
            meter t en la pila
    fin
fin
```

Autómatas Finitos Deterministas
Conversión de un AFN en un AFD

- Algoritmo para convertir un AFN en AFD.
  - Entrada: AFN N.
  - Salida: AFD D.
  - Método: Se construye una tabla de transiciones tranD para D, de manera que tranD simula “en paralelo” todos los posibles movimientos que se pueden dar en N ante una determinada cadena de entrada.
  - Además del cierre-λ, se utiliza la operación mueve(T, a), que dará como resultado un conjunto de estados del AFN hacia los cuales existe una transición con el símbolo a desde algún estado s $\in$ T.
  - Tomamos 0 como el estado inicial del AFN.

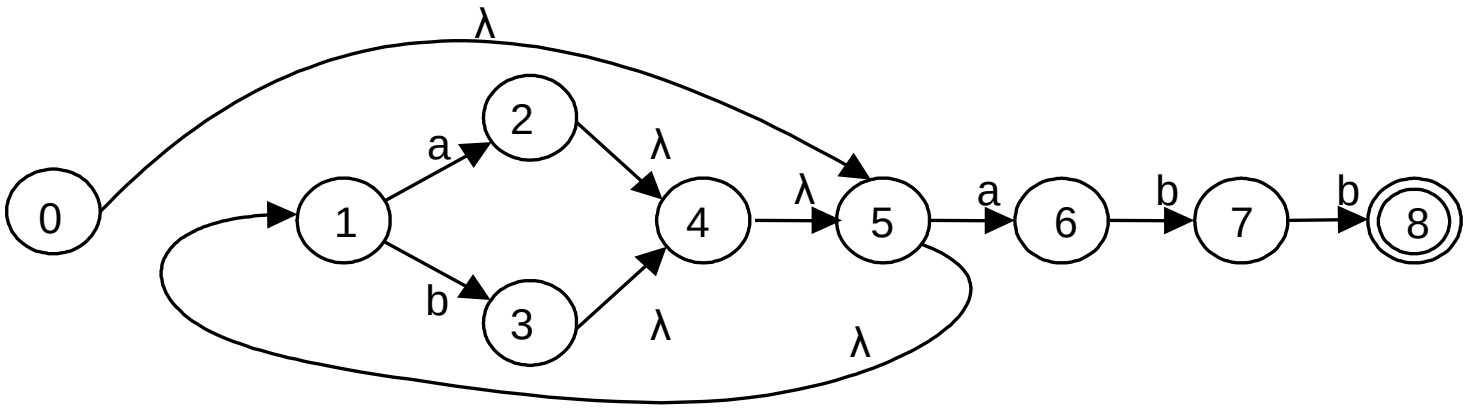
Autómatas Finitos Deterministas
Conversión de un AFN en un AFD

– Pseudocódigo:

```
construir estado U := cierre-λ(0)
añadir U a estadosD como estado no marcado
mientras haya un estado no marcado T en estadosD hacer
        marcar T
        para cada símbolo de entrada a hacer
                U := cierre-λ(mueve(T, a))
                si U no está en estadosD entonces
                        añadir U a estadosD como estado no marcado
                tranD[T, a] := U
        fin
fin
```

Autómatas Finitos Deterministas
Conversión de un AFN en un AFD

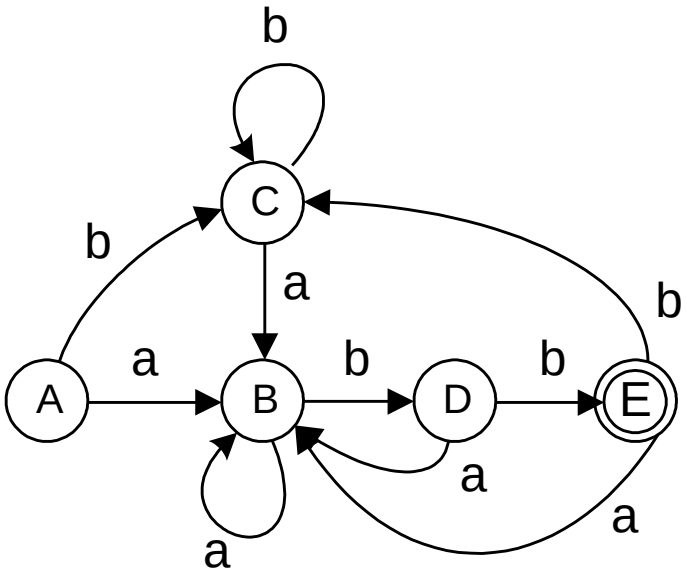
- Ejemplo AFN para $(a|b)^*$ abb.
  - Estado de inicio del AFD es A = cierre-λ({0}) = {0, 1, 5}



Autómatas Finitos Deterministas
Conversión de un AFN en un AFD

- Ejemplo AFN para $(a|b)^*$ abb.
  - Estado de inicio del AFD es A = cierre-λ({0}) = {0, 1, 5}

Estado	Símbolo de Entrada	
	a	**b**
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C



Autómatas Finitos Deterministas
Minimización de un AFD

- Minimización de estados de un AFD.
  - Todo conjunto regular es reconocido por un AFD con el mínimo número de estados que es único.
  - Algoritmo.
    1. Se construye una partición inicial P del conjunto de estados con dos grupos: Estados de aceptación F y estados de no-aceptación S-F.

Autómatas Finitos Deterministas
Minimización de un AFD

- Minimización de estados de un AFD.
  - Algoritmo (cont.).
    2. Aplicar el siguiente procedimiento para construir una nueva partición P_{nueva}.

para cada grupo G de P **hacer**

crear partición de G en subgrupos que cumplan que dos estados s y t de G estén en el mismo subgrupo sii para todos los símbolos de entrada a, s y t tienen transiciones en a hacia estados del mismo grupo de P;

sustituir G en P_{nueva} por el conj. de todos los subgrupos formados;

fin

Autómatas Finitos Deterministas
Minimización de un AFD

- Minimización de estados de un AFD.
  - Algoritmo (cont.).

3. Si $P_{nueva} = P$, hacer $P_{final} := P$ y continuar con el paso 4. Si no, hacer $P := P_{nueva}$ y continuar con el paso 2.
4. Escoger en cada grupo de la partición un estado representante, que formará parte del AFD mínimo.

   Construir la tabla de transiciones utilizando los estados representantes.

   El estado inicial del AFD mínimo es el representante del grupo que contiene el estado s0 del AFD inicial.

   Los estados finales son los representantes que están en F.

Autómatas Finitos Deterministas
Minimización de un AFD

- Minimización de estados de un AFD.
  - Algoritmo (cont.).

5. Eliminar todos los estados inactivos (estados de no aceptación que tiene trancisiciones hacia él mismo con todos los símbolos de entrada)

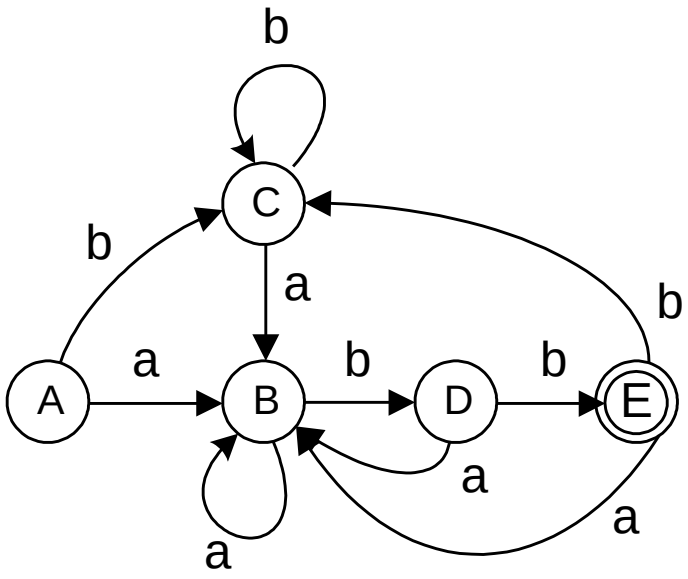
   Eliminar estados inalcanzables desde el estado inicial.

   Todas las transiciones a estos estados quedan indefinidas.

Autómatas Finitos Deterministas
Minimización de un AFD

- Ejemplo de mimimización de AFD para $(a|b)^*$ abb.
  - Inicialmente, P = (ABCD) (E).

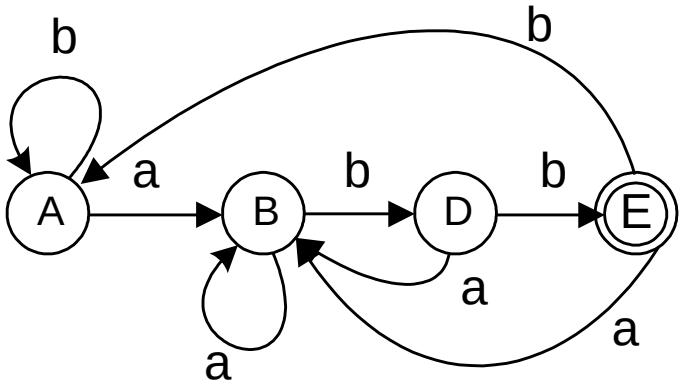
Estado	Símbolo de Entrada	
	a	**b**
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C



Autómatas Finitos Deterministas
Minimización de un AFD

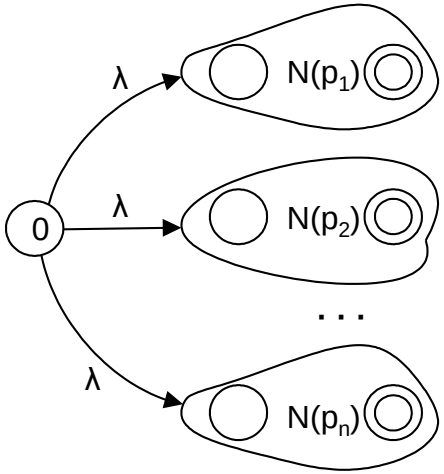
- Ejemplo de mimimización de AFD para $(a|b)^*$ abb.
  - Inicialmente, P = (ABCD) (E).

Estado	Símbolo de Entrada	
	a	**b**
A	B	A
B	B	D
D	B	E
E	B	A



Reconocedores y Analizadores Léxicos

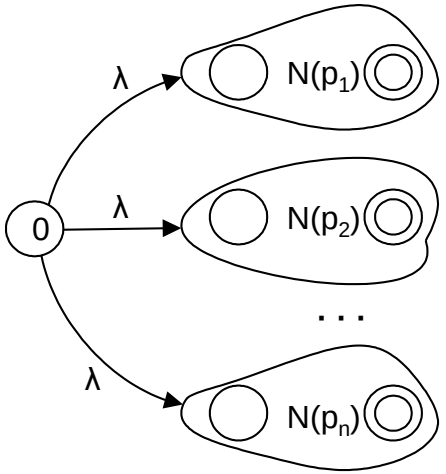
- La especificación de un analizador léxico está formada por el conjunto de patrones que tiene que reconocer y la acción asociada a cada patrón.
- Concordancia de patrones (AFN).
  - AFN para cada p_i con nuevo estado s_0 y transiciones λ
  - Cada p_i puede ser un AFN.



Reconocedores y Analizadores Léxicos

– Simulación AFN.

S := cierre-λ(0);
a := siguiente_carácter();
mientras a ≠ eof **hacer**
    S := cierre-λ(mueve(S,a));
    a := siguiente_carácter();
si ∃s ∈ S / s ∈ F **entonces**
    devolver “sí”;
si no devolver “no”;



Reconocedores y Analizadores Léxicos

- En ocasiones, habrá concordancia para más de un patrón.
  - El reconocedor elegirá el lexema más largo que haya concordado.
  - Si hay dos o más patrones que concuerdan con el lexema más largo, se elige el primer patrón de la lista (Prioridad!!!).

- Durante el proceso de reconocimiento, se debe llevar un registro de la última vez que se alcanzó un estado final con dos o más patrones.