ChatGPT

Unidad 4: Autómatas Finitos No Deterministas

Definición y concepto de AFND

Un **Autómata Finito No Determinista (AFND)** es un modelo matemático similar al AFD (autómata finito determinista) pero que permite múltiples opciones de transición para un mismo estado y símbolo de entrada [1] [2]. Formalmente se define como una *quíntupla* **AFND = (Σ, Q, q_0, A, f)**, donde los componentes son análogos al AFD: Σ es el alfabeto de entrada; Q es un conjunto finito de estados; $q_0 \in Q$ es el estado inicial; $A \subseteq Q$ es el conjunto de estados de aceptación; y **f: Q × Σ → P(Q)** es la función de transición [3] [4]. A diferencia de un AFD (donde f(q,a) devuelve un único estado), en un AFND la transición desde un estado *q* con un símbolo *a* puede llevar a *un conjunto* de posibles estados [2]. Es decir, para cada estado y símbolo, f puede devolver *varios* destinos (incluso ninguno). Esto captura el concepto de *no determinismo*, permitiendo que el autómata "intente" simultáneamente diferentes caminos de procesamiento para la cadena de entrada.

El **no determinismo** no implica aleatoriedad, sino la existencia de múltiples caminos computacionales. Si el autómata tiene más de una transición posible para alguna combinación de estado y símbolo, se dice que su comportamiento es no determinista [5] [2]. En cada paso, el AFND *bifurca* su configuración en todas las transiciones posibles. El grado de bifurcación se mide mediante el **factor de ramificación**, definido como el número de posibles próximos estados: $FR = |f(q, a)|$ [6] [7]. Un FR > 1 indica bifurcación (varios caminos concurrentes). Aunque en la implementación real esto se interpreta como exploración de un árbol de posibilidades, conceptualmente podemos pensar que el autómata "explora" todas las rutas en paralelo.

Aceptación de cadenas: Un AFND acepta (reconoce) una cadena *α* si existe al menos *un* camino de estados que consume toda la cadena y termina en un estado de aceptación [8] [9]. En otras palabras, partiendo de la configuración inicial (q_0, α), el autómata puede *moverse* no determinísticamente siguiendo las transiciones definidas de manera que, tras leer todos los símbolos, alcance una configuración (q_f, λ) con $q\_f \in A$ (estado final aceptante) [8] [9]. Si al terminar de leer la entrada **ninguno** de los caminos posibles termina en estado de aceptación, entonces la cadena es rechazada [10] [11]. Formalmente: un AFND reconoce la cadena α si existe una secuencia de pasos $(q_0, \alpha) \vdash \ldots \vdash (q\_f, \lambda)$ con $q\_f \in A$ [12].

Funcionamiento y ejemplos de AFND

Dado que un AFND puede seguir múltiples transiciones, su ejecución sobre una cadena puede representarse como un **árbol de configuraciones**: cada nodo del árbol es una configuración *(estado, sufijo_de_entrada)*, y se ramifica cuando hay más de una transición posible para el mismo símbolo [7] [13]. El autómata explora *todos* los caminos simultáneamente; si **alguno** de ellos finaliza en estado de aceptación tras consumir la entrada completa, la cadena es aceptada [7] [13]. En el árbol de computación, esto significa que al menos una rama llega a un nodo cuyo estado es aceptante con cadena vacía restante. Por ejemplo, en la Figura 4.1 del apunte, se ilustra el árbol de configuraciones para la cadena β = 1011000 procesada por un cierto AFND: varias ramas se exploran y finalmente una de ellas alcanza el estado de aceptación *t*, señalando la aceptación de β [14] [15].

Ejemplo 1: Un caso sencillo de AFND es aquel que reconoce cadenas binarias terminadas en `100`. Esto puede describirse con la expresión regular (0+1)*100. Un diseño posible (no determinista) es el siguiente [16]:

- *Estados:* p (inicial), q, r, s (estado final aceptante).
- *Transiciones:* desde p, al leer `1`, el autómata puede quedarse en p **o** ir a q (bifurcación) [17] [18]; desde p con `0`, retorna a p. Luego q con `0` va a r, r con `0` va a s, y s es aceptante.
- Este AFND acepta cualquier cadena que termine en `100`. Por ejemplo, para la cadena α = `01100`, el procesamiento no determinista produce varias ramas; una de ellas lleva a p→q→r→s consumiendo `01100` y termina en *s*, indicando aceptación [19] [20]. Efectivamente, el AFND halló en la entrada el sufijo `100` requerido. Es más fácil diseñar este autómata usando no determinismo (bifurcando en p ante un `1`) que construir directamente un AFD equivalente [18].

Ejemplo 2: El *Ejemplo 4.1* del apunte teórico presenta un AFND que reconoce cadenas sobre {0,1} cuyo sufijo es `1000` (es decir, forma general α = (0+1)*1000)* [21]*. El autómata resultante (Figura 4.1) tiene un estado inicial p que* permite cualquier prefijo *de 0s y 1s (retornando a sí mismo con 0, y no determinísticamente quedándose o yendo a un siguiente estado con 1) seguido de una secuencia de estados q→r→s→t para asegurar la terminación en* `1000` [22] [23]*. En p, ante el símbolo* `1` *se activa la condición no determinista: f(p, 1) = {p, q}, es decir, leyendo* `1` *desde p el autómata puede tanto continuar en p (consumiendo el* `1` *como parte del prefijo arbitrario) como saltar al camino que conduce al patrón de sufijo* [24] [25]*. Esto le da flexibilidad para* "adivinar"* el inicio del sufijo. Por ejemplo, la cadena `1011000` es aceptada porque uno de los caminos posibles logra alinear los últimos cuatro símbolos con los estados q, r, s, t respectivamente [14] [15].

Ejemplo 3 (ejercicio): Un ejercicio típico consiste en completar la definición formal añadiendo un **estado de error** a un AFND. Por ejemplo, dado el AFND del ejemplo anterior (sufijo `1000`), se pide identificar las transiciones faltantes (condiciones de error) y agregarlas en la tabla de transición incorporando un estado extra E que represente toda situación no permitida [26] [27]. El estado E sería de no aceptación y absorbe cualquier símbolo una vez alcanzado (similar al sink en AFD). Este ejercicio ayuda a practicar la formalización completa de un AFND.

Autómatas con transiciones vacías (AFND-λ)

Una extensión del AFND permite **transiciones vacías**, también llamadas *transiciones λ* (lambda) o *transiciones espontáneas*. Estas son movimientos del autómata que no consumen ningún símbolo de entrada [28]. Para habilitar esto, se modifica la función de transición para incluir a λ como entrada posible. Formalmente, en un **AFND-λ** se tiene **$f: Q \times (\Sigma \cup \{\lambda\}) \to P(Q)$** [29] [30]. Una transición λ significa que el autómata puede pasar de un estado a otro *gratuitamente*, es decir, sin leer símbolo alguno [28] [31]. En los diagramas, suelen representarse con flechas etiquetadas "λ".

La presencia de λ-transiciones agrega flexibilidad al diseño. Por ejemplo, es útil para conectar sub-autómatas en construcciones más complejas o para permitir saltos opcionales. No obstante, también requiere adaptar el criterio de aceptación: ahora una cadena puede ser aceptada incluso si quedan transiciones λ por realizar después de consumir todos los símbolos. En efecto, para la función extendida **$f_e(q, \beta)$** que permite λ-movimientos, se considera la clausura λ (conjunto de estados alcanzables solo con transiciones vacías) en cada paso [32] [33]. Una cadena α será aceptada por un AFND-λ si partiendo de q_0 y leyendo α (permitiendo moverse por λ en cualquier momento), existe alguna secuencia de transiciones que termine en un estado de A [34] [35]. En símbolos: $(q_0, \alpha) \vdash \ldots \vdash (q\_f, \lambda)$ con $q\_f \in A$, considerando que los pasos pueden incluir movernos por λ sin consumir entrada.

Ejemplo 1: Supongamos un AFND-λ con estado inicial p que tiene una transición **λ→ q**. Esto significa que el autómata puede comenzar en p y saltar espontáneamente al estado q antes de leer nada. Si, por ejemplo, desde q el autómata reconoce cierta subcadena específica, la presencia de la transición λ desde p permite hacer opcional o diferir el consumo de símbolos. Este patrón es aprovechado en construcciones de unión de lenguajes: p (nuevo inicio) tiene λ hacia dos estados iniciales de sub-autómatas distintos, de modo que el autómata resultante puede ir por el camino de uno u otro [36] [37]. Las transiciones λ también se usan para **concatenar** máquinas: p (inicio) λ→ (inicio de máquina 1); estados finales de máquina 1 λ→ (inicio de máquina 2); etc., sin necesidad de consumir símbolos en esos "enganches" [38] [39]. (Estos son precisamente los pasos utilizados en el *algoritmo de Thompson*, que veremos más adelante.)

Ejemplo 2: El *Ejercicio 6* propone un AFND-λ con Σ = {a, b}, estados {p, q, r, s}, estado inicial p y estado de aceptación r [40] [41]. En la tabla de transición dada, por ejemplo, p tiene transiciones f(p, a) = {q, r} y f(p, b) = {q} [42]. Aunque no se muestra explícitamente en el enunciado, podemos suponer que existen transiciones λ entre ciertos estados (el ejercicio sugiere calcular conjuntos T y T* de estados vinculados por λ) [43] [44]. Una de las ramas de procesamiento de este autómata para la cadena "babb" ilustra su operación: comenzando en p con "babb", tras leer b puede ir a q (porque p con b da {q}) y, mediante alguna secuencia de λ-movimientos internos, también alcanzar otros estados como r, etc., hasta eventualmente consumir toda la entrada. De hecho, en el ejercicio se indica que "babb" es aceptada, al igual que "bbbb", mientras que "bbbaa" no lo es [45]. Esto se comprueba construyendo el árbol de configuraciones (considerando transiciones λ aunque no se dibujen todas por simplicidad) y observando que al menos una rama termina en r con la palabra vacía [45].

Ejemplo 3 (ejercicio): Un problema común es **eliminar las transiciones λ** de un AFND-λ para convertirlo en un AFND (equivalente). Por ejemplo, el *Teorema 1* asegura que para cualquier AFND-λ existe un AFND sin λ que reconoce el mismo lenguaje [33] [34]. Como práctica, se puede tomar un AFND-λ dado (p. ej. el del Ejercicio 7: Σ={0,1}, Q={A,B,C,D}, con transiciones definidas incluyendo λ [46]) y aplicar el procedimiento de eliminación: (a) agregar transiciones directas que simulen los saltos λ intermedios, (b) si un estado aceptante es alcanzable por λ desde el inicial, marcar inicial como aceptante, (c) suprimir todas las columnas λ de la tabla [47] [48]. Tras este proceso, el ejercicio 7 guía a convertir el resultado (ya sin λ) a un AFD equivalente [49].

Equivalencia entre AFND y AFD (teorema de Rabin-Scott)

Una propiedad fundamental es que los AFND **no son más poderosos** que los AFD en cuanto a lenguajes reconocidos: todo lenguaje que puede reconocer un AFND también lo puede reconocer algún AFD equivalente [50] [51]. En 1959, Rabin y Scott demostraron precisamente que todo AFND (incluso con λ) reconoce un lenguaje **regular**, y para cada AFND existe un AFD que acepta el *mismo* conjunto de cadenas [52] [53]. Por lo tanto, AFND, AFND-λ y AFD son modelos diferentes pero *equivalentes* en términos de poder de reconocimiento (todos reconocen exactamente los lenguajes regulares).

La equivalencia se puede establecer constructivamente. Por un lado, dado un AFND-λ, podemos primero eliminar sus λ-transiciones (Teorema 1 mencionado) obteniendo un AFND equivalente sin λ [33] [34]. Luego, dado un AFND (sin λ), existe un procedimiento sistemático –conocido como **construcción de subconjuntos**– para obtener un AFD. Este procedimiento se resume en el *Teorema 2*: cada AFND $M = (\Sigma, Q, q_0, A, f)$ tiene un AFD equivalente $M' = (\Sigma, Q', q_0', A', f')$ donde los estados Q' pueden tomarse como los subconjuntos de Q [54] [55]. Intuitivamente, el AFD simula en un único estado *c* de Q' **todos** los estados en que podría estar el AFND en paralelo en cierto punto de la lectura [56] [57].

En la construcción, el **estado inicial** del AFD es el conjunto de estados alcanzables desde q_0 (incluyendo clausura λ si aplica) [58] [59]. Luego, iterativamente, para cada estado conjunto *C* y cada símbolo *a*, se calcula *f'(C, a)* = ∪_{q ∈ C} f(q, a) (y aplicando clausura λ si corresponde). Este resultado es otro subconjunto de Q, que pasa a ser un nuevo estado de Q' [56] [60]. Se continúa hasta que no aparezcan estados nuevos. Los estados de Q' que contengan al menos un estado original de A (aceptación) serán marcados como aceptantes en A' [61] [62]. Al finalizar, obtenemos un AFD (posiblemente con muchos estados, algunos inalcanzables o equivalentes) que reconoce el mismo lenguaje que el AFND original. Es común que haya que *minimizar* luego el AFD resultante, ya que la construcción suele introducir estados redundantes [63] [64].

Conversión de AFND a AFD (procedimiento de subconjuntos)

Para ilustrar la equivalencia, veamos cómo convertir concretamente un AFND en un AFD. Aplicaremos el *Caso 1* del Teorema 2 (AFND sin λ):

Paso 1: El estado inicial del nuevo AFD será $C_0 = \{q_0\}$, el subconjunto que contiene únicamente al estado inicial original [58].

Paso 2: A partir de C_0, calculamos sus transiciones. Ejemplo: si $C_0 = \{p\}$, entonces para cada símbolo *a* del alfabeto determinamos f(C_0, a) = f({p}, a) = f(p, a). En el *Ejemplo 4.3*, partiendo del AFND del sufijo 1000 mencionado antes (estados {p,q,r,s,t}), se obtuvo: - f'({p}, 0) = {p} (queda en p), - f'({p}, 1) = {p, q} (desde p con 1 iba a {p,q}) [65]. Así, C_0 on 0 devuelve {p} (que ya es C_0 de nuevo) y C_0 on 1 devuelve {p,q}, que se nombra como C_1 [66].

Paso 3: Repetir: calcular transiciones desde C_1. Siguiendo el ejemplo, si $C_1 = \{p,q\}$: - f'({p,q}, 0) = f(p,0) ∪ f(q,0) = {p} ∪ {r} = {p,r}, se define como nuevo estado C_2 [67] [68]. - f'({p,q}, 1) = f(p,1) ∪ f(q,1) = {p,q} ∪ Ø = {p,q}, que resulta ser C_1 nuevamente [67]. Continuando así: - Desde C_2 = {p,r}: f'({p,r}, 0) = {p,s} = C_3; f'(...,1) = {p,q} = C_1 [69]. - Desde C_3 = {p,s}: f'(...,0) = {p,t} = C_4; f'(...,1) = {p,q} = C_1 [69]. - Desde C_4 = {p,t}: f'(..., 0) = {p} = C_0; f'(...,1) = {p,q} = C_1 [69]. Llegado este punto, cualquier transición desde los estados existentes (C_0...C_4) lleva a un estado ya visto. Por lo tanto, la construcción termina.

Paso 4: Determinar estados de aceptación en el AFD. Un estado *C_i* de Q' será aceptante si contiene al menos un estado aceptante original [70] [71]. En nuestro ejemplo, el estado original de aceptación era {t}, por lo que todos los conjuntos que incluyan *t* serán marcados. Observamos que C_4 = {p,t} contiene t, así que C_4 es aceptante [72]. (También C_0, C_1, C_2, C_3 no lo contienen, así que no lo son). Finalmente, podemos renombrar los subconjuntos con etiquetas más simples (c_0, c_1, … c_4) y presentar la tabla de transición resultante [73] [74]. En este caso, el AFD obtenido tiene 5 estados c_0...c_4, siendo c_4 el único de aceptación [75] [76]. De hecho, resulta ser el mismo AFD que ya habíamos encontrado en la unidad anterior para ese lenguaje (sufijo "1000") [77].

Ejemplo (conversión): En el ejercicio 3 se proponía diseñar un AFND para el lenguaje *(a+b)*ba(a+b)a (sobre Σ={a,b}) y luego convertirlo en AFD [78]. El AFND planteado tenía estados p, q, r, s, t con p inicial y t final [79]. Siguiendo el procedimiento: - C_0 = {p}. Desde C_0: con a → {p} (queda en p, C_0 mismo); con b → {p,q} = C_1 [80] [81]. - Desde C_1 = {p,q}: con a → {p,r} = C_2; con b → {p,q} = C_1 [82] [83]. - Desde C_2 = {p,r}: con a → {p,s} = C3; con b → {p,q,s} = C4 [84] [85]. Y así sucesivamente hasta generar todos los conjuntos posibles [84] [86]. Finalmente se identifican los estados de aceptación: aquí el original t ∈ A, por lo que cualquier C_i que contenga t será aceptante. En efecto, en la construcción aparecen C_5 = {p,t} y C_6 = {p,r,t}, ambos incluyen t, entonces C_5 y C_6 (renombrados) se marcan como aceptantes [87] [70]. El AFD resultante tiene 7 estados (C_0–C_6) antes de minimizar. Este ejemplo muestra cómo el AFD

equivalente puede tener hasta 2^n estados en el peor caso (si Q tiene n estados), aunque muchos pueden ser inalcanzables o combinados luego.

Ejemplo (AFND-λ a AFD): Para un AFND con transiciones λ, el proceso es similar pero incorporando la *clausura λ*. Es decir, el estado inicial del AFD será C_0 = ε-closure({q_0}) (todos los estados alcanzables desde q_0 por solo transiciones λ) [88] [89]. Luego para cada conjunto C y símbolo *a*, primero se ven todas las transiciones desde cualquier estado de C por *a*, y al resultado se le aplica también la clausura λ. En el *Ejercicio 7*, por ejemplo, se calculó T* (clausura λ global) para el AFND-λ dado, resultando C_0 = {A,B,D} (porque desde A se llega por λ a B y D) [90] [91]. Luego, f'(C_0, 0) = ε-closure(f(A,0) ∪ f(B,0) ∪ f(D,0)) = ε-closure({B,C} ∪ {B,D} ∪ {A,C}) = {A,B,C,D} = C_2 [92] [93], etc. Tras completar la tabla, se obtienen estados como C_1 = {B,C,D}, C_2 = {A,B,C,D}, C_3 = {C}, etc., con sus transiciones [93] [94]. Los estados aceptantes son aquellos que contienen al original C (estado final) –en este caso, varios conjuntos lo incluían, por lo que hubo múltiples aceptantes [95] [96]. Nuevamente, el AFD final podría no ser conexo y suele requerir minimización.

Relación entre autómatas finitos y gramáticas regulares

Existe un **isomorfismo** entre los autómatas finitos y las gramáticas regulares. En particular, para cada autómata finito determinista existe una gramática regular que genera el mismo lenguaje, y viceversa [97] [98]. Esto significa que las gramáticas regulares (lineales por la derecha) describen exactamente los lenguajes reconocibles por autómatas finitos.

De gramática a autómata: Dada una gramática regular *G = (Σ_T, Σ_N, S, P)*, podemos construir un AF que reconozca *L(G)* aplicando las siguientes reglas [99] [100]:

- El conjunto de símbolos terminales Σ_T de la gramática será el alfabeto de entrada Σ del autómata.
- El conjunto de símbolos no terminales Σ_N será el conjunto de estados Q del autómata. Es decir, cada no terminal se convierte en un estado.
- El símbolo inicial S de la gramática será el estado inicial q_0 del autómata.
- Por cada **producción** de la forma X := a Y (donde X,Y ∈ Σ_N y a ∈ Σ_T), se crea una **transición** en el autómata f(X, a) = Y [101] [99].
- Por cada producción de la forma X := a (terminal solitario que lleva a cadena terminal), se crea una transición desde X hacia un **nuevo estado de aceptación** A mediante el símbolo *a*: es decir, f(X, a) = A, donde A es un estado adicional que se marcará como aceptante [100] [102]. (Este nuevo estado A representa la "cesación" de derivación y corresponde al símbolo de terminación en la gramática, a menudo denotado explícitamente como un estado de aceptación sin transiciones salientes).

Siguiendo este procedimiento, por ejemplo, la gramática:
G1: Σ_T = {0,1}, Σ_N = {D, E}, S = D, P = { D := 0E | 1, E := 0 | 1E | 1 }
genera un lenguaje que puede ser reconocido por el AFND [103] [104]:
AF_9 = (Σ={0,1}, Q={D, E, F}, q_0=D, A={F}, f)
Transiciones: f(D,0)=E; f(D,1)=F; f(E,0)=F; f(E,1)=E; f(E,1)=F.
(Aquí F es el nuevo estado de aceptación introducido, ya que tanto D como E tenían producciones que terminaban la derivación con un terminal) [105] [106]. Es fácil verificar que este autómata reconoce el mismo lenguaje que genera la gramática (por ejemplo, D→1 corresponde a la cadena "1" aceptada moviéndose D -1→ F). Los ejercicios 9, 10 y 11 de la guía proponen justamente realizar estas construcciones con distintas gramáticas regulares [107] [108].

De autómata a gramática: Dado un AFD (autómata determinista) podemos derivar una gramática regular que genere su lenguaje reconocido. Sea AFD = (Σ_E, Q, q_0, A, f). El procedimiento inverso es [109] [110]:

- El alfabeto de entrada Σ_E del autómata será el conjunto de terminales Σ_T de la gramática.
- El conjunto de estados Q del autómata será el conjunto de no terminales Σ_N de la gramática. Cada estado se convierte en un símbolo no terminal. El axioma de la gramática será el símbolo correspondiente al estado inicial q_0 [111] [112].
- Por cada transición f(X, a) = Y en el autómata (con X,Y ∈ Q y a ∈ Σ_E), se crea una producción *X := aY* en la gramática [109].
- Para cada transición que vaya a un estado de aceptación en el AFD, digamos f(X, a) = F con F ∈ A (F es estado aceptante), se agregarán **dos producciones**: *X := aF* (como antes) **y** *X := a* (producción que termina en ese terminal) [109] [113]. Esta segunda regla refleja que desde X, consumir *a* puede llevar al autómata a finalizar en aceptación (equivalente a derivar una cadena terminada en *a*). En la gramática resultante, típicamente no se incluye el símbolo correspondiente al estado de aceptación como no terminal en el cuerpo (o se incluye pero se permite terminar allí con una producción *F := λ*).

Por ejemplo, dado el siguiente AFD (Ejercicio 12):
AFD_{12} = (Σ={0,1}, Q={A, B, C, F}, q_0=A, A={F}, f) [114] [115] – *ver figura 4.32* –, la gramática correspondiente sería:
G_{12}: Σ_T = {0,1}, Σ_N = {A, B, C, F}, S = A,
P_{12} = { A := 1A | 0B, B := 0B | 1C, C := 1F | 1 | 0B, F := 0B | 1A } [114] [115].
Aquí obsérvese cómo, por ejemplo, la transición A -1→ A produce la regla A := 1A; la transición C -1→ F (que va a aceptacion) produce C := 1F **y** C := 1 (esta última correspondientemente listada en P_{12}) [114] [115].
Siguiendo estas reglas, la gramática generará exactamente las cadenas aceptadas por el AFD dado. Los ejercicios 12 y 13 proponen practicar este mecanismo con distintos autómatas [116] [117].

En síntesis, las gramáticas regulares y los autómatas finitos son dos formalizaciones equivalentes de los lenguajes regulares: podemos pasar de una a otro de manera mecánica, preservando el lenguaje reconocido o generado.

Expresiones regulares y algoritmo de Thompson

Las **expresiones regulares (ER)** constituyen otra forma de describir lenguajes regulares. Existe un método algorítmico (debido a Ken Thompson) que, dada una ER, construye automáticamente un AFND-λ que reconoce el mismo lenguaje denotado por la expresión [118] [119]. Este procedimiento es fundamental para aplicaciones como los analizadores léxicos, ya que permite convertir patrones regulares en máquinas reconocedoras.

El **algoritmo de Thompson** se basa en descomponer la expresión regular en sus componentes básicos y combinarlos usando autómatas parciales conectados por transiciones λ [120] [121]. Las reglas principales son:

- **Casos base:** Para la ER ∅ (lenguaje vacío) se construye un autómata con un estado inicial y *ninguna* cadena aceptada (por ejemplo, un autómata con estado de aceptación aislado sin forma de ser alcanzado, o con conjunto de aceptación vacío) [122] [123]. Para la ER λ (que denota solo la cadena vacía), se construye un AFND que *acepta λ* (por ejemplo, marcando el estado inicial también como final, o con una transición λ del inicial a un nuevo final) [124] [125]. Para una ER

consistente en un símbolo literal *a* (∈ Σ), se construye un autómata con una transición *a* desde el estado inicial a un estado final [126].

- **Union (E + F):** Se toma un autómata para E (con su q_0^E y q_A^E) y otro para F (q_0^F, q_A^F). Se crea un **nuevo estado inicial** q_0, desde el cual se agrega una transición λ a q_0^E y otra λ a q_0^F [127] [36]. También se crea un **nuevo estado final** q_A, al cual se conectan con λ todos los antiguos estados de aceptación q_A^E y q_A^F [128] [129]. Se eliminan las marcas de aceptación de q_A^E y q_A^F (ya no serán finales, porque ahora q_A concentrará la aceptación) [130] [129]. El resultado es un AFND-λ que en esencia corre en paralelo ambos sub-autómatas E y F; cualquier cadena aceptada por E o por F hará que el nuevo autómata llegue al estado q_A mediante alguna de las ramas λ, por lo que reconoce L(E) ∪ L(F) [37] [131].

- **Concatenación (E · F o EF):** Se toma el autómata de E y el de F. Se crea un nuevo estado inicial q_0, con una transición λ hacia el antiguo q_0^E [38] [39]. Luego, desde **cada** estado de aceptación antiguo de E, se agrega una transición λ hacia el antiguo estado inicial de F [39] [132]. Se quitan las marcas de aceptación de los estados finales de E (porque ya no terminan la cadena, ahora pueden continuar en F). Se crea un nuevo estado final q_A, y desde cada estado de aceptación antiguo de F se conecta una λ hacia q_A [39] [133]. Así el autómata resultante primero procesa una cadena válida para E y luego, vía λ, enlaza con el procesamiento de una cadena válida para F. Reconocerá exactamente la concatenación L(E)·L(F) [38] [39].

- **Cerradura de Kleene (E*):** Se construye un nuevo estado inicial q_0, con una transición λ hacia el antiguo q_0^E, y un nuevo estado final q_A. Se agrega una transición λ de q_0 directamente a q_A (esto permite aceptar la cadena vacía, perteneciente a L(E*)) [28]. Además, desde cada estado de aceptación antiguo de E se añaden transiciones λ tanto hacia q_A (para terminar la cadena) como de regreso al antiguo q_0^E (para repetir E nuevamente) [134] [120]. De este modo, el autómata puede ejecutar E las veces que quiera (incluyendo cero veces).

Siguiendo estas reglas, el algoritmo de Thompson construye un AFND-λ complejo paso a paso. Por ejemplo, si queremos un AFND-λ para la ER `(0+1)*100` (cadenas binarias terminadas en 100), podríamos: construir primero un sub-AFND-λ para `(0+1)*` (que esencialmente consiste en un bucle con dos ramas λ hacia 0 y 1 y retornos para la cerradura), luego otro para `100` (concatenación de los literales `1`, `0`, `0`), y finalmente concatenarlos. En la práctica, Thompson define formalmente cada paso. Los ejercicios propuestos (ej. ejercicio 14) piden aplicar este método a expresiones regulares concretas para obtener sus AFND-λ, luego derivar el AFD equivalente y finalmente minimizarlo [135] [136]. Esto refuerza la idea de que **todas** las representaciones vistas (ER, AFND-λ, AFND, AFD, gramática regular) describen los mismos lenguajes, y podemos pasar de una a otra.

Aplicación: análisis léxico (compiladores)

Los autómatas finitos deterministas, a menudo obtenidos a partir de ER mediante Thompson y minimización, se utilizan en herramientas de análisis léxico para reconocer patrones en el código fuente. Un **analizador léxico** (scanner) convierte la secuencia de caracteres de un programa en una secuencia de *tokens* identificando lexemas que coinciden con expresiones regulares (por ejemplo, identificadores, números, palabras clave).

En la práctica, se definen ER para cada categoría léxica y luego se construyen autómatas (generalmente AFD mínimos) que "tokenizan" la entrada. Por ejemplo, en el *Ejemplo 3.9* del capítulo anterior se mostró un analizador léxico que reconoce constantes numéricas (enteros, reales, notación científica) usando un AFD diseñado a mano [137] [138]. Ahora, con las técnicas de esta unidad, podríamos lograr lo mismo

definiendo una gramática regular o ER para los números y luego derivando el autómata correspondiente. De hecho, el apunte propone una gramática para números (que incluye dígitos, punto decimal, signo, exponente, etc.) y guía la construcción del autómata equivalente paso a paso [139] [140].

Ejemplo: Supongamos que queremos reconocer números (positivos o negativos, enteros o decimales, con notación científica opcional). Podemos escribir una gramática regular para describir ese lenguaje (con no terminales representando partes como enteros, fracción, exponente, etc.) [141] [142]. A partir de ella, aplicando las reglas vistas, obtenemos un autómata finito. En el apunte se construye dicho **autómata lexicográfico** que reconoce cadenas como "-277" o "3.1416" o "0.87E-5", etc., a través de un proceso sistemático: cada regla de la gramática se transforma en transiciones, incluyendo un nuevo estado final *Z* que representa haber leído un separador (espacio o fin de token) [143] [144]. El resultado es un AFD que puede integrarse en un compilador para identificar números en el código fuente.

Este ejemplo demuestra cómo los conceptos de AFND/AFD y sus conversiones tienen aplicación directa. En compiladores, normalmente se combinan múltiples autómatas (uno por token) en un único gran autómata mediante uniones (lo cual se maneja con transiciones λ de un nuevo inicio a los autómatas de cada token) [36]. Luego se determina el AFD mínimo para eficiencia. Así, el análisis léxico se fundamenta en la teoría de lenguajes regulares y autómatas finitos, que garantiza un reconocimiento de patrones rápido y correcto.

Ejercicios sugeridos: Para consolidar estos temas, se recomiendan ejercicios como: - Diseñar diversos AFND y AFND-λ para lenguajes dados (por ejemplo, palíndromos de cierto tipo, patrones con prefijo/sufijo específico, etc.) y luego convertirlos en AFD equivalentes. Esto ejercita la construcción no determinista y la aplicación del algoritmo de subconjuntos. - Tomar expresiones regulares de complejidad creciente (con varias combinaciones de unión, concatenación y cierre) y aplicar manualmente el algoritmo de Thompson para obtener el AFND-λ. Por ejemplo, derivar el AFND-λ para ER como `(ab+ba)*(bb|aa)` y comprobar que el AFD resultante (post-conversión y minimización) coincide con el lenguaje esperado. - Construir gramáticas regulares para lenguajes sencillos y obtener de ellas el autómata, y viceversa. Un caso interesante es definir la gramática de números o identificadores (letras y dígitos) y producir el AFD que los reconoce, simulando el trabajo de un analizador léxico.

En resumen, la Unidad 4 abarcó los **autómatas finitos no deterministas** y sus variaciones, mostrando cómo brindan una forma poderosa de especificar lenguajes regulares de forma flexible. Aprendimos los métodos para traducir AFND a AFD (y eliminar λ si es necesario), así como las correspondencias formales con las gramáticas regulares y las expresiones regulares. Estos conceptos no solo son teóricamente importantes, sino que también sustentan herramientas prácticas como los analizadores léxicos en la construcción de compiladores, consolidando así la utilidad de los lenguajes formales en la computación [137] [145]. Todas estas técnicas proporcionan un **apunte de repaso exhaustivo** para preparar el final de la materia, cubriendo la totalidad de los temas de la Unidad 4 tal como fueron desarrollados en clase y en la bibliografía.

---

1 6 14 15 21 22 23 24 25 32 33 34 35 36 37 38 39 47 48 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 72 73 74 75 76 77 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 137 138 139 140 141 142 143 144 145 SSL_U4_ApunteTeórico.pdf
file://file-PwXt9oxd3a7aGUqmBAm1Dj

2 3 4 5 7 8 9 10 11 12 13 16 17 18 19 20 28 29 30 31 97 98 SSL_U4_filminas.pdf
file://file-TChehjx4ohy1v4MCm1dHij

26 27 40 41 42 43 44 45 46 49 70 71 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 Ejercicios de AFND.pdf

file://file-URjDKkq8qKGQPxXGXrzXeE

114 115 116 117 135 136 SSL_U4_GuiaDeEjercicios.pdf

file://file-Bz4VyBoXJHNUbgKm5hkfnh