Construcción Directa de AFD a partir de `(a|b)*`

Procedimiento Manual — Diseño de Lenguajes de Programación

Universidad del Valle de Guatemala

1. Objetivo

Convertir la expresión regular `(a|b)*` a un Autómata Finito Determinista (AFD) mediante el algoritmo de **construcción directa** basado en *followpos* (Aho et al., Sección 3.9.5). Se presenta cada paso del procedimiento realizado a mano y se valida el resultado final.

2. Paso 1: Expresión regular aumentada

Se agrega el marcador de fin `#` concatenado a la expresión original:

$$r' = (\texttt{a} \mid \texttt{b})^* \cdot \texttt{\#}$$

Esto permite identificar los estados de aceptación del AFD: serán aquellos cuyo conjunto de posiciones contenga la posición asignada a `#`.

3. Paso 2: Árbol sintáctico con posiciones

Se construye el árbol sintáctico de r' y se asignan posiciones numéricas a cada hoja que representa un símbolo del alfabeto o el marcador de fin.

a=0 =0 =0 =0

Las posiciones asignadas son:

Posición	Símbolo
1	`a`
2	`b`
3	`#` (marcador de fin)

4. Paso 3: Cálculo de *nullable*, *firstpos* y *lastpos*

Se calculan las tres propiedades para cada nodo del árbol, de las hojas hacia la raíz, aplicando las reglas del algoritmo.

Reglas utilizadas

- **Hoja con símbolo c en posición i:** nullable = false, firstpos = $\{i\}$, lastpos = $\{i\}$.
- **Unión** $(c_1 \mid c_2)$: nullable = nullable(c_1) $\vee$ nullable(c_2), firstpos = firstpos(c_1) $\cup$ firstpos(c_2), lastpos = lastpos(c_1) $\cup$ lastpos(c_2).
- **Concatenación** $(c_1 \cdot c_2)$: nullable = nullable(c_1) $\wedge$ nullable(c_2).

  $$\text{firstpos} = \begin{cases} \text{firstpos}(c_1) \cup \text{firstpos}(c_2) & \text{si nullable}(c_1) \\ \text{firstpos}(c_1) & \text{en otro caso} \end{cases}$$

  $$\text{lastpos} = \begin{cases} \text{lastpos}(c_1) \cup \text{lastpos}(c_2) & \text{si nullable}(c_2) \\ \text{lastpos}(c_2) & \text{en otro caso} \end{cases}$$
- **Cerradura de Kleene** (c_1^*): nullable = true, firstpos = firstpos(c_1), lastpos = lastpos(c_1).

Resultado por nodo

Nodo	**nullable**	**firstpos**	**lastpos**
`a` (pos 1)	false	$\{1\}$	$\{1\}$
`b` (pos 2)	false	$\{2\}$	$\{2\}$
`a` | `b`	false	$\{1,2\}$	$\{1,2\}$
(`a` | `b`)*	true	$\{1,2\}$	$\{1,2\}$
`#` (pos 3)	false	$\{3\}$	$\{3\}$
$\cdot$ (raíz)	false	$\{1,2,3\}$	$\{3\}$

Nota sobre la raíz ($\cdot$, concatenación):

firstpos(raíz) = firstpos($*$) $\cup$ firstpos(`#`) porque nullable($*$) = true, entonces:

$$\text{firstpos(raíz)} = \{1,2\} \cup \{3\} = \{1,2,3\}$$

lastpos(raíz) = lastpos(`#`) = $\{3\}$ porque nullable(`#`) = false.

5. Paso 4: Cálculo de *followpos*

Se aplican las dos reglas de *followpos*:

Regla de concatenación (nodo raíz $\cdot$)

Para cada posición i en lastpos(hijo izquierdo), agregar firstpos(hijo derecho) a followpos(i).

- lastpos($*$) $= \{1, 2\}$, firstpos(#) $= \{3\}$
- followpos(1) $\mathrel{+}= \{3\}$
- followpos(2) $\mathrel{+}= \{3\}$

Regla de cerradura de Kleene (nodo $*$)

Para cada posición i en lastpos(hijo), agregar firstpos(hijo) a followpos(i).

- lastpos(`a|b`) $= \{1, 2\}$, firstpos(`a|b`) $= \{1, 2\}$
- followpos(1) $\mathrel{+}= \{1, 2\}$
- followpos(2) $\mathrel{+}= \{1, 2\}$

Tabla *followpos* resultante

Posición	Símbolo	followpos
1	`a`	$\{1, 2, 3\}$
2	`b`	$\{1, 2, 3\}$
3	`#`	$\varnothing$

6. Paso 5: Construcción del AFD

Estado inicial

El estado inicial es firstpos(raíz) $= \{1, 2, 3\}$.

Se define como estado $A = \{1, 2, 3\}$.

Como la posición 3 (#) está contenida en A, A **es un estado de aceptación**.

Transiciones desde A

Para cada símbolo del alfabeto $\Sigma = \{\texttt{a}, \texttt{b}\}$:

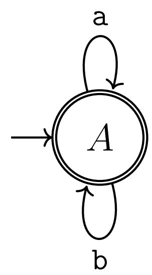
- **Símbolo `a`:** posiciones en A con símbolo `a` $= \{1\}$.
  $\bigcup_{i\in\{1\}}$ followpos(i) = followpos(1) $= \{1, 2, 3\} = A$
  Transición: $A \xrightarrow{\texttt{a}} A$
- **Símbolo `b`:** posiciones en A con símbolo `b` $= \{2\}$.
  $\bigcup_{i\in\{2\}}$ followpos(i) = followpos(2) $= \{1, 2, 3\} = A$
  Transición: $A \xrightarrow{\texttt{b}} A$

No se generan estados nuevos. El algoritmo termina.

Tabla de transiciones

Estado	`a`	`b`	Aceptación
$\to A$	A	A	Sí

7. Paso 6: AFD resultante



El AFD tiene un único estado $A = \{1, 2, 3\}$ que es simultáneamente el estado inicial y el estado de aceptación. Ambas transiciones (`a` y `b`) regresan al mismo estado.

8. Validación

Se verifica que el AFD reconoce exactamente el lenguaje $L = (\mathtt{a} \mid \mathtt{b})^*$, es decir, cualquier cadena formada por `a` y `b`, incluyendo la cadena vacía ε.

Cadena	**Recorrido**	**Resultado**
ε (vacía)	Se queda en A (aceptación)	ACEPTADA
`a`	$A \to A$ (aceptación)	ACEPTADA
`b`	$A \to A$ (aceptación)	ACEPTADA
`abba`	$A \to A \to A \to A \to A$ (aceptación)	ACEPTADA
`c`	No hay transición con `c`	RECHAZADA

Todos los resultados son consistentes con el lenguaje $(\mathtt{a} \mid \mathtt{b})^*$:

- La cerradura de Kleene acepta la cadena vacía ε. ✓
- Cualquier combinación de `a` y `b` es aceptada. ✓
- Símbolos fuera del alfabeto $\{\mathtt{a}, \mathtt{b}\}$ son rechazados. ✓

9. Conclusión

La construcción directa de $(\mathtt{a}|\mathtt{b})^*$ produce un AFD de un solo estado, lo cual es el resultado mínimo posible. Esto tiene sentido intuitivo: la expresión acepta cualquier cadena sobre $\{\mathtt{a}, \mathtt{b}\}$ (incluyendo ε), por lo que no es necesario distinguir entre estados — toda cadena válida es aceptada desde el inicio.

Referencias

- Aho, Lam, Sethi, Ullman. *Compilers: Principles, Techniques, and Tools* (2nd ed.), Sección 3.9.5.
- Hopcroft, J.E. *An $n \log n$ algorithm for minimizing states in a finite automaton*, 1971.