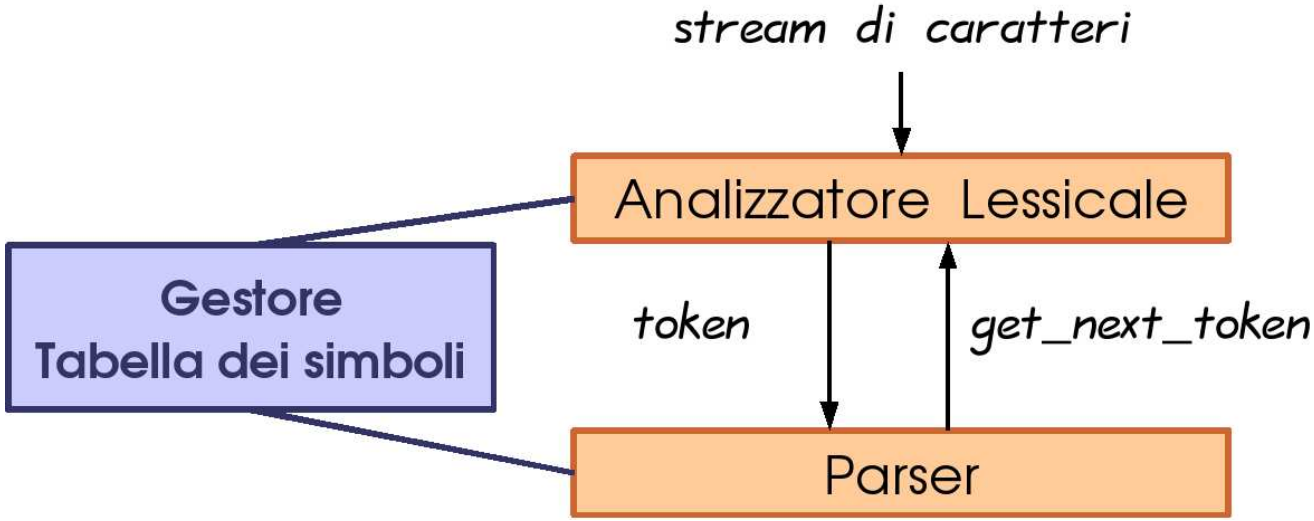
Analisi Lessicale

Maria Rita Di Berardini[1]

[1]Dipartimento di Matematica e Informatica
Universitá di Camerino
mariarita.diberardini@unicam.it

Ruolo dell'analisi lessicale

- Fornire alle fasi successive una sequenza di token a partire dallo stream di caratteri che rappresenta il programma che si vuole compilare
- Lo stream di token prodotto dall'analisi lessicale costituisce l'input delle fasi successive (analisi sintattica)



Ruolo dell'analisi lessicale

- Legge la sequenza di caratteri in input fino al riconoscimento di un nuovo token:
  - se il token corrisponde ad un nuovo identificatore, aggiunge un record alla tabella dei simboli
  - elimina spazi bianchi (blank, tabulazioni e newline) e commenti
  - se non viene riconosciuto alcun token, segnala un errore (errori lessicali)
  - associare i giusti numeri di riga agli errori
- Perchè separiamo la due fasi:
  - semplicità di progettazione: un parser che non deve tener conto di spazi bianchi e commenti è più semplice da scrivere
  - efficienza del processo di compilazione: l'uso di tecniche specifiche per le diverse fasi consente di migliorare le prestazioni del compilatore

Token, Pattern, Lessemi

- **Token**: definisce un insieme di sequenze di caratteri (stringhe) con significato comune
  - identificatori (**id**)
  - numeri (**num**)
  - parole chiave (**if**, **for**, **else**, ...)
  - operatori
- **Pattern**: regola che descrive l'insieme di stringhe associato ad un dato token
  - lettera seguita da cifre e/o lettere
  - una qualsiasi costante numerica
  - `if`, `for`, `else`, ...
  - `:=`, `+`, $\leq$
- **Lessema** (*lexeme*): sequenza di caratteri nel programma sorgente che fa "match" con il pattern di un certo token
  - pi, count, x,
  - 34, 3.1455,
  - $:=$, $+$, $\leq$

Token, Pattern & Lessemi

- Possiamo pensare ai lessemi che fanno match con il pattern di un dato token come ad delle sequenze di caratteri nel programma sorgente che possono essere trattate come un unica unità lessicale

Attributi dei token

- I token hanno degli attributi, che consentono di distinguere lessemi diversi che fanno match con lo stesso pattern
- Consideriamo il seguente statement `E := M*2;`
  - ⟨**id**, *puntatore all'entrata nella symbol table per E* ⟩
  - ⟨**assign_op**, ⟩
  - ⟨**id**, *puntatore all'entrata nella symbol table per M* ⟩
  - ⟨**mult_op,** ⟩
  - ⟨**num**, *numero intero 2*⟩

  in alternativa, avremmo potuto inserire il lessema 2 nella symbol table e associare all'attributo per num la corrispondente entrata

Problemi da risolvere

1. Specificare i pattern per i vari token del nostro linguaggio (soluzione espressioni regolari)
2. Stabilire se una certa sequenza di caratteri fa match o meno con un dato pattern (soluzione automi a stati finiti deterministici e non deterministici)
3. Come vedremo le espressioni regolari sono un formalismo che consente di descrivere dei linguaggi, mentre gli automi consentono di riconscere linguaggi
4. Breve panoramica su stringhe e linguaggi

Part I

Stringhe e Linguaggi

Stringhe

- Un alfabeto Σ è un insieme finito di simboli (UNICODE, ASCII, ...)
- Una stringa su un dato alfabeto Σ è una sequenza finita di simboli
- Data una stringa s usiamo $|s|$ per denotare la lunghezza (ossia, il numero di simboli) della stringa
- La stringa vuota (denota da ε) ha lunghezza zero
- Operazioni sulle stringhe:
  1. date due stringhe s_0 ed s_1 la concatenazione di s_0 ed s_1, scritta come $s_0 s_1$, è la stringa ottenuta attaccando s_1 in fondo ad s_0. Es. se $s_0 = boun$ e $s_1 = giorno$ allora $s_0 s_1 = boungiorno$
  2. Elevamento a potenza: data una stringa s ed un interno non negativo k

$$s^k = \begin{cases} \varepsilon & \text{se } n = 0 \\ ss^{k-1} & \text{altrimenti} \end{cases}$$

Esempio $ab^0 = \varepsilon, \quad ab^1 = ab, \quad ab^2 = abab, \quad ab^3 = ababab$

Linguaggi

- Un linguaggio è un insieme finito di stringhe su un dato alfabeto Σ
- Operazioni su linguaggi:
  1. Unione: $L_1 \cup L_2 = \{v \mid v \in L_1 \text{ oppure } v \in L_2\}$
  2. Concatenzione: $L_1 L_2 = \{v_1 v_2 \mid v_1 \in L_1, v2 \in L_2\}$
  3. Esponenziazione: $L^0 = \{\varepsilon\}$, $L^1 = L$, ..., $L^K = LL^{k-1}$
  4. Chiusura (o stella) di Kleene: $L^* = \bigcup_{k=0}^{\infty} L^k$
  5. Chiusura (o stella) positiva di Kleene: $L^+ = \bigcup_{k=1}^{\infty} L^k$

Esempi

Assumiamo $L = \{0, 1\}$

- $L^0 = \{\varepsilon\}$
- $L^1 = L$
- $L^2 = LL = \{0, 1\}\{0, 1\} = \{00, 01, 10, 11\}$
- $L^3 = LL^2 = \{0, 1\}\{00, 01, 10, 11\} = \{000, 001, 010, 011, 100, 101, 110, 111\}$
- In generale L^k è l'insieme di tutte le stringhe di lunghezza k
- L^* è l'unione di L^k con $k \geq 0$, è l'insieme di tuttu le stringhe di lunghezza maggiore o uguale a zero
- L^+ è l'unione di L^k con $k \geq 1$, è l'insieme di tuttu le stringhe di lunghezza maggiore o uguale ad uno (in generale, $L^* = L^+ \cup \{\varepsilon\}$)

Esempi

Assumiamo $L = \{A, \ldots, Z, a, \ldots, z\}$ (lettere) e $C = \{0, \ldots, 9\}$ (cifre)

- $L \cup C$ è il linguaggio delle lettere e delle cifre
- LC é l'insieme di tutte le stringhe di lunghezza 2 formate da una lettera seguita da una cifra
- L^4 è il linguaggio delle stringhe di lettere di lunghezza 4
- $L(L \cup C)^*$ è il linguaggio delle stringhe di lunghezza > 0 che iniziano con una lettera e sono composte da cifre e lettere
- C^+ il linguaggio delle stringhe di cifre di formate da almeno una cifra

Part II

Espressioni regolari

Espressioni Regolari

- Le espressioni regolari su un dato alfabeto Σ sono un formalismo per descrivere linguaggi
- Sia Σ un alfabeto finito. L'insieme delle espressioni regolari su Σ è costituito da tutte e sole le espressioni generate induttivamente come segue:

  1) ε è un'espressione regolare che **denota** il linguaggio $\{\varepsilon\}$[1]
  2) Se $a \in \Sigma$ allora a è un'espressione regolare che denota il linguaggio $\{a\}$

  Siano r_1 ed r_2 due espressioni regolari ed $L(r_1)$ ed $L(r_2)$ i linguggi denotati, rispettivamente, da r_1 ed r_2

  3) $(r_1) \mid (r_2)$ è un'espressione regolare che denota il linguaggio $L(r_1) \cup L(r_2)$
  4) $(r_1)(r_2)$ è un'espressione regolare che denota il linguaggio $L(r_1)L(r_2)$
  5) $(r_1)^*$ è un'espressione regolare che denota il linguaggio $L(r_1)^*$
  6) (r_1) è un'espressione regolare che denota il linguaggio $L(r_1)$

[1] **Nota**: Il linguaggio $\{\varepsilon\} \neq \emptyset$

Valutazione delle espressioni regolari

- Qual'è il valore di una data espressione regolare? Il valore di un'espressione regolare r è il linguaggio denotato da r, ossia $L(r)$
- Due espressioni regolari r ed s sono equivalenti ($r = s$) se e solo se $L(r) = L(s)$
- La valutazione di un'espressione regolare dipende dalle seguenti precedenze:
  1. L'operatore unario $*$
  2. La concatenazione
  3. L'operatore $|$
- Quindi, $r = a \mid b^*c = (a) \mid (b^*c) = (a) \mid ((b^*)(c)) = (a) \mid (((b)^*)(c))$
- Possiamo usare le parentesi per forzare le precedenze, ad esempio $a \mid b^* \neq (a \mid b)^*$

$a \mid b^* \neq (a \mid b)^*$

- $r_1 = a \mid b^* = (a) \mid (b^*) = (a) \mid ((b)^*)$
- $r_2 = (a \mid b)^* = ((a) \mid (b))^*$
- $L(r_1) = L(a) \cup L((b)^*) = L(a) \cup (L(b))^* = \{a\} \cup \{b\}^* = \{a\} \cup \{\varepsilon, b, bb, bbb, \ldots\} = \{a\} \cup \{b^n \mid n \geq 0\}$
- $L(r_2) = L(a \mid b)^* = \{a, b\}^* = \{a, b\}^0 \cup \{a, b\} \cup \{a, b\}^2 \cup \{a, b\}^3 \cup \ldots = \{\varepsilon\} \cup \{a, b\} \cup \{aa, ab, ba, bb\} \cup \{aaa, aab, aba, abb, baa, bab, bba, bbb\} \cup \ldots =$ l'insieme di tutte le stringhe formate dalla concatenazione di un numero qualsiasi di a e b in qualsiasi ordine
- Banalmente, $L(r_1) \subset L(r_2)$
- $aa, bba, ab, \ldots \in L(r_2)$ ma $aa, bba, ab, \ldots \notin L(r_1)$

Assiomi

Commutatività di $\mid$	$r \mid s = s \mid r$
Associatività di $\mid$	$r \mid (s \mid t) = (r \mid s) \mid t$
Associatività della concat.	$r(st) = (rs)t$
Distribituvità della concat. rispetto a $\mid$	$r(s \mid t) = rs \mid rt$, $(r \mid s)t = rt \mid st$
Elemento neutro della concat.	$r\varepsilon = \varepsilon r = r$
Relazione tra $\mid$ e *	$r^* = (r \mid \varepsilon)^*$
Idempotenza di *	$(r^*)^* = r^*$

Linguaggi regolari

- Un linguaggio denotato da un'espressione regolare è detto linguaggio regolare
- Alcuni esempi di espressioni/linguaggi regolari sull'alfabeto $\Sigma = \{a, b\}$

$r = a \mid b$ $\quad L(r) = L(a) \cup L(b) = \{a\} \cup \{b\} = \{a, b\}$

$r = (a \mid b)(a \mid b)$ $\quad L(r) = L(a \mid b)L(a \mid b) = \{a, b\}\{a, b\} = \{aa, ab, ba, bb\}$

$r = a^*$ $\quad L(r) = L(a)^* = \{a\}^* = \{\varepsilon, a, aa, aaa, \ldots\} = \{a^n \mid n \geq 0\}$

$r = r = a \mid b^*c$ $\quad L(r) = L(a) \cup L((b^*)c) = \{a\} \cup (L(b^*)L(c)) = \{a\} \cup (\{b^n \mid n \geq 0\}\{c\}) = \{a\} \cup \{b^n c \mid n \geq 0\}$

Definizioni regolari

- Proviamo a scrivere l'espressione regolare che definisce il token **id**
- Gli identificatori sono sequenze di cifre e/o lettere che iniziano con una lettera
- L'alfabeto $\Sigma = L \cup C = \{A, \ldots, Z, a, \ldots, z\} \cup \{0, \ldots, 9\}$
- L'espressione regolare $r_{\mathbf{id}}$ che definisce il token **id** è:

$$(A \mid \ldots \mid Z \mid a \mid \ldots \mid z)\big((A \mid \ldots \mid Z \mid a \mid \ldots \mid z) \mid (0 \mid \ldots \mid 9)\big)^*$$

- Una semplificazione consiste nell'assegnare dei nomi ad espressioni regolari, nomi che possono essere usati nella definizione di altre espressioni
- Ad esempio:

$$\begin{array}{lcl} \mathbf{digit} & \rightarrow & 0 \mid \ldots \mid 9 \\ \mathbf{letter} & \rightarrow & A \mid \ldots \mid Z \mid a \mid \ldots \mid z \\ \mathbf{id} & \rightarrow & \mathbf{letter}\ (\mathbf{letter} \mid \mathbf{digit})^* \end{array}$$

- Espressioni regolari + nomi = Definizioni regolari

Definizioni regolari

- Le definizioni regolari introducono la possibilità di assegnare dei nomi ad espressioni regolari
- Una definizione regolare è una sequenza del tipo:

$$\begin{array}{lcl} d_1 & \longrightarrow & r_1 \\ d_2 & \longrightarrow & r_2 \\ & \ldots & \\ d_n & \longrightarrow & r_n \end{array}$$

  dove: $d_1, d_2, \ldots, d_n$ sono dei **nomi distinti** ed $r_1, r_2, \ldots, r_n$ sono delle espressioni regolari sull'alfabeto $\Sigma \cup \{d_1, d_2, \ldots, d_n\}$
- È possibile usare i nomi $d_1, d_2, \ldots, d_n$ nella definizione di espressioni regolari

Identificatori e Costanti numeriche

Scrivere una definizione regolare per identificatori e costanti numeriche in Pascal (interi 342, razionali 45.56, o con esponente 6.89, 123.89E-34)

digit	$\rightarrow$	$0 \mid \ldots \mid 9$	
letter	$\rightarrow$	$A \mid \ldots \mid Z \mid a \mid \ldots \mid z$	
id	$\rightarrow$	**letter** $(\textbf{letter} \mid \textbf{digit})^*$	
digits	$\rightarrow$	**digit digit***	123
opt_fraction	$\rightarrow$	**.digits** $\mid \varepsilon$	.89
opt_exponent	$\rightarrow$	$(E\ (+ \mid - \mid \varepsilon))$ **digits** $\mid \varepsilon$	E-34
num	$\rightarrow$	**digits opt_fraction opt_exponent**	123.89E-34

Alcune Shorthands

- Se r è un'espressione regolare, l'espressione $r?$ è una abbreviazione per l'espressione $r \mid \varepsilon$; se $L(r)$ è il linguaggio denotato da r, $r?$ denota il linguaggio $L(r) \cup \{\varepsilon\}$

digit	$\longrightarrow$	$0 \mid \ldots \mid 9$	
letter	$\longrightarrow$	$A \mid \ldots \mid Z \mid a \mid \ldots \mid z$	
id	$\longrightarrow$	**letter** (**letter** $\mid$ **digit**)*	
digits	$\longrightarrow$	**digit digit***	123
opt_fraction	$\longrightarrow$	(**.digits**)?	.89
opt_exponent	$\longrightarrow$	$(E\ (+ \mid -)?)?$ **digits**	E-34
num	$\longrightarrow$	**digits opt_fraction opt_exponent**	123.89E-34

Alcune Shorthands

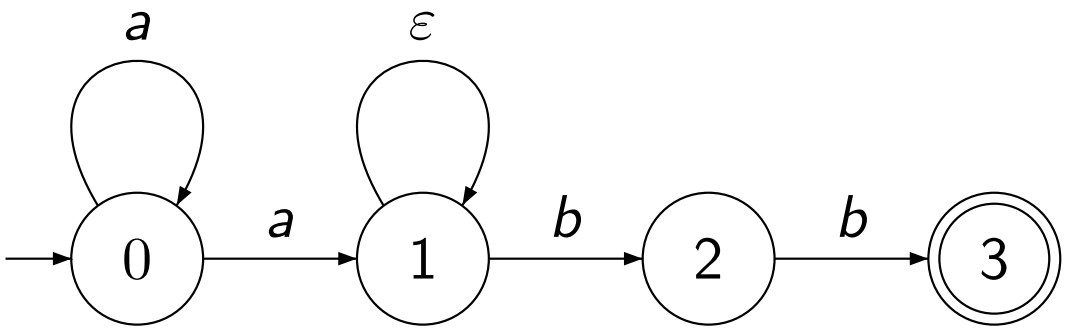
- La notazione $[abc]$, dove a, b e c sono dei simboli dell'alfabeto, denota l'espressione regolare $a \mid b \mid c$
- Classe di caratteri: $[a - z]$ denota l'espressione regolare $a \mid b \mid \ldots \mid z$

$$\begin{array}{lcl} \textbf{digit} & \longrightarrow & [0 - 9] \\ \textbf{letter} & \longrightarrow & [A - Za - z] \end{array}$$

Part III

Automi a Stati Finiti

Automi a stati finiti non deterministici: definizione



Automi a stati finiti non deterministici: definizione

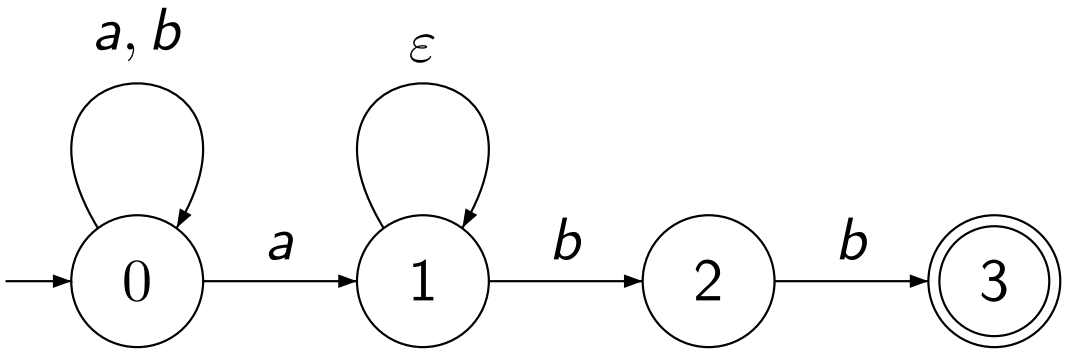
- Un automa a stati finiti non deterministico (NFA - Non-deterministic Finite Automaton) sull'alfabeto Σ è una tupla $\langle S, \Sigma, \mathbf{move}, s_0, F \rangle$ dove:
  1. S è un insieme finito di stati
  2. Σ è l'alfabeto dei simboli
  3. **move** è una funzione di transizione (prende in input uno stato, un simbolo dell'alfabeto o ε, e restituisce un insieme di possibili successori)

$$\mathbf{move} : (S \times (\Sigma \cup \{\varepsilon\})) \longmapsto \wp(S)^2$$

  4. $s_0 \in S$ è lo stato iniziale
  5. $F \subseteq S$ è l'insieme degli stati finali o di accettazione

[2] $\wp(S) = \{A \mid A \subseteq S\}$ è l'insieme delle parti (l'insieme dei sottoinsiemi) di S

Automi a stati finiti non deterministici: definizione



dove $S = \{0, 1, 2, 3\}$, $\Sigma = \{a, b\}$, $s_0 = 0 \in S$, $F = \{3\} \subseteq S$ e

$$\textbf{move}(0, a) = \{0, 1\} \quad \textbf{move}(0, b) = \{0\}$$
$$\textbf{move}(1, b) = \{2\} \quad \textbf{move}(1, a) = \emptyset \quad \textbf{move}(1, \varepsilon) = \{1\}$$
$$\textbf{move}(2, b) = \{3\} \quad \textbf{move}(2, a) = \emptyset$$
$$\textbf{move}(3, a) = \emptyset \quad \textbf{move}(3, b) = \emptyset$$

Cammini Etichettati

- Sia $N = \langle S, \Sigma, \mathbf{move}, s_0, F \rangle$ un NFA, un **cammino etichettato** di lunghezza $k \geq 0$ è una sequenza di $k+1$ stati della forma:

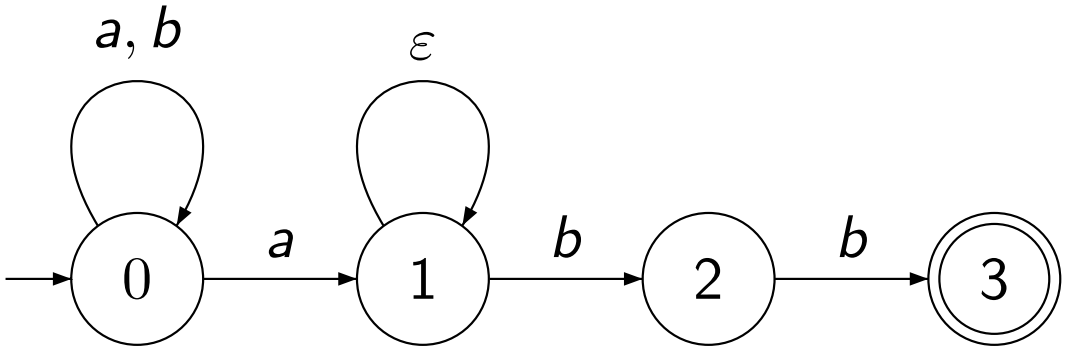
$$s_0 \xrightarrow{x_1} s_1 \xrightarrow{x_2} s_2 \ldots s_{k-1} \xrightarrow{x_k} s_k$$

dove

1. s_0 è lo stato iniziale
2. per ogni $j = 1, 2, \ldots, k$, $s_j \in move(s_{j-1}, x_j)$ (lo stato s_j è raggiugibile dalla stato s_{j-1} mediante una transizione etichettata con x_j)

- La stringa $x_1 x_2 \ldots x_k$ è detta **stringa associata al cammino**
- In particolare, se $k = 0$ il cammino è costituito dal solo stato iniziale e la corrispondente stringa associata è la stringa vuota (ε)

Cammini Etichetti



$0 \xrightarrow{a} 0 \xrightarrow{a} 1$ *aa*

$0 \xrightarrow{a} 0 \xrightarrow{a} 1 \xrightarrow{\varepsilon} 1$ *aa*

$0 \xrightarrow{a} 0 \xrightarrow{a} 1 \xrightarrow{b} 2$ *aab*

$0 \xrightarrow{a} 0 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{b} 3$ *aabb*

Linguaggio accettato

- Sia $N = \langle S, \Sigma, \mathbf{move}, s_0, F \rangle$ una stringa $\alpha \in \Sigma^*$ è accettata da N se e solo se esiste un cammino

$$s_0 \xrightarrow{x_1} s_1 \xrightarrow{x_2} s_2 \ldots s_{k-1} \xrightarrow{x_k} s_k$$

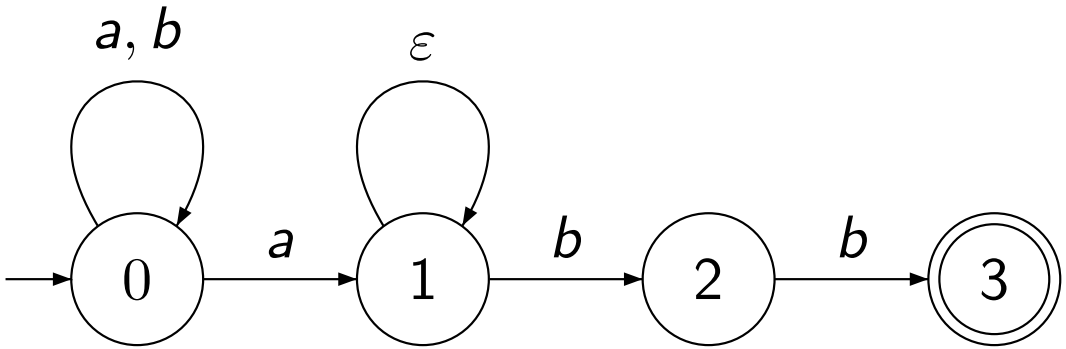
tale che

1. $\alpha = x_1 x_2 \ldots x_k$ è la stringa associata al cammino
2. $s_k \in F$ (è uno stato finale dell'automa)

- Il linguaggio accettato da N è l'insieme

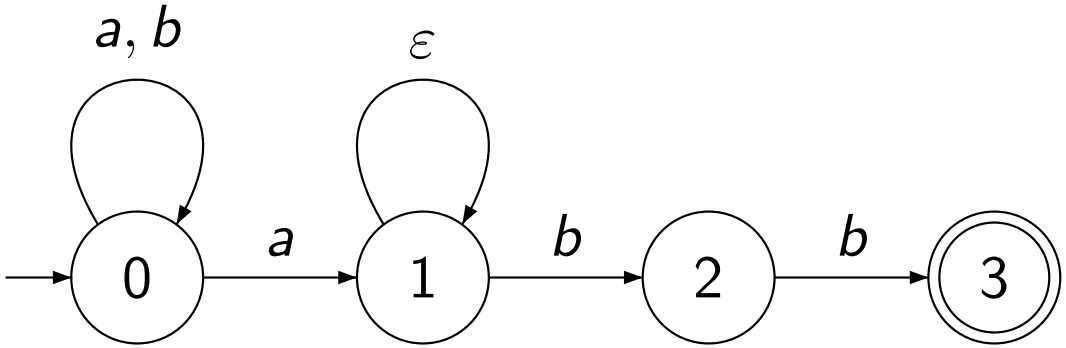
$$L(N) = \{\alpha \in \Sigma^* \mid \alpha \text{ è accettata da } N\}$$

Linguaggio accettato



$$L(N) = \{a, b\}^* abb$$

Algoritmo di riconoscimento



$\{0\}$	ababb	$move(0, a) = \{0, 1\}$
$\{0, 1\}$	babb	$move(0, b) = \{0\}, move(1, b) = \{2\}$
$\{0, 2\}$	abb	$move(0, a) = \{0, 1\}, move(2, a) = \emptyset$
$\{0, 1\}$	bb	$move(0, b) = \{0\}, move(1, b) = \{2\}$
$\{0, 2\}$	b	$move(0, b) = \{0\}, move(2, b) = \{3\}$
$\{0, 3\}$	ε	

Automi a stati finiti deterministici: definizione

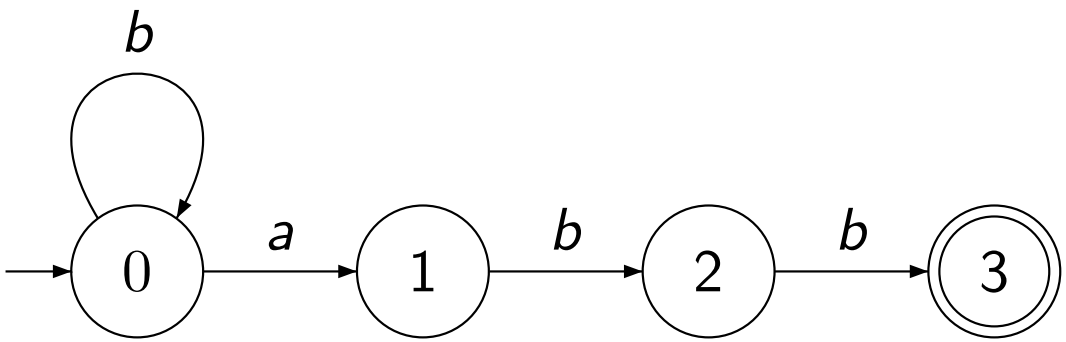
- Un automa a stati finiti deterministico (DFA - Deterministic Finite Automaton) sull'alfabeto Σ è una tupla $\langle S, \Sigma, \mathbf{move}, s_0, F \rangle$ dove:
  1. S è un insieme finito di stati
  2. Σ è l'alfabeto dei simboli
  3. **move** è una funzione di transizione

$$\mathbf{move} : (S \times \Sigma) \longmapsto \wp(S)$$

tale che per ogni stato $s \in S$ e per ogni simbolo dell'afabeto $x \in \Sigma$, move(s, x) è vuoto oppure contiene un solo elemento

  4. $s_0 \in S$ è lo stato iniziale
  5. $F \subseteq S$ è l'insieme degli stati finali o di accettazione

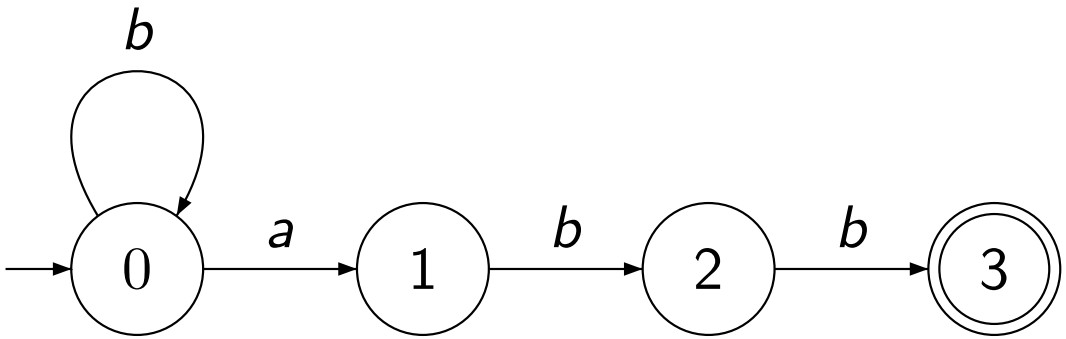
Automi a stati finiti deterministici: definizione



- Non ci sono più ε-transizioni
- Dato uno stato s ed un simbolo x, esiste al più una transizione uscente da s etichettata con x
- Di conseguenza, il cammino di accettazione di una data stringa w, se esiste, è **unico**

Algoritmo di riconoscimento

L'algoritmo di riconoscimento lo stesso, ma, ad ogni passo, l'insieme degli stati correnti contiene uno solo elemento



$\{0\}$	ababb	$move(0, a) = \{1\}$
$\{1\}$	babb	$move(1, b) = \{2\}$
$\{2\}$	abb	$move(2, a) = \emptyset$
$\emptyset$	bb	

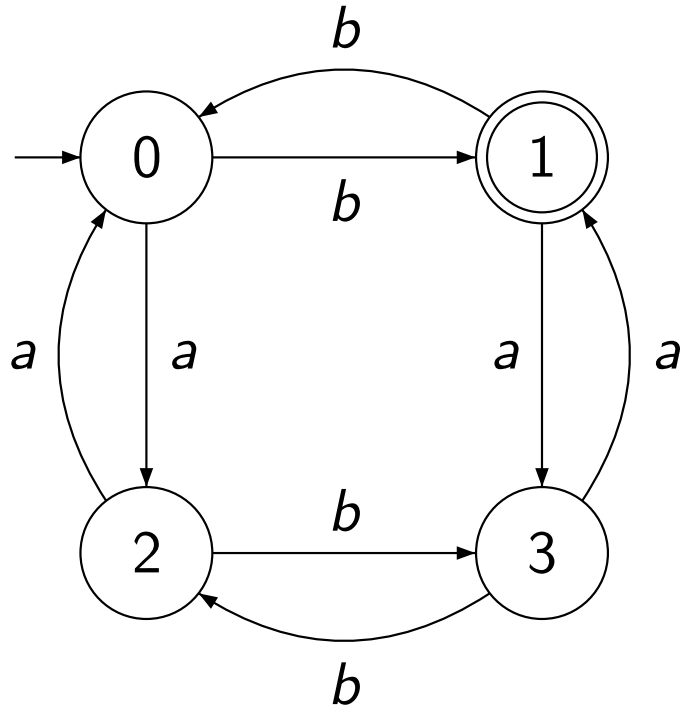
$\{0\}$	babb	$move(0, b) = \{0\}$
$\{0\}$	abb	$move(0, a) = \{1\}$
$\{1\}$	bb	$move(1, b) = \{2\}$
$\{2\}$	b	$move(2, b) = \{3\}$
$\{3\}$	ε	

Un esercizio

Costruire un DFA che accetti il linguaggio delle stringhe sull'alfabeto $\Sigma = \{a, b\}$ contenenti un numero pari di a e dispari di b

$S = \{0, 1, 2, 3\}$ dove i vari stati hanno il seguente significato:

- 0: numero pari di a e di b
- 1: numero pari di a e dispari di b
- 2: numero dispari di a e pari di b
- 3: numero dispari di a e di b



Automi deterministici vs. Automi Non-deterministici

- Per ogni NFA esiste un DFA equivalente (ossia, in grado di riconoscere lo stesso linguaggio) e viceversa
- Il non determinismo rende più semplice scrivere un automa e/o riconoscere il linguaggio accettato
- L'algoritmo di riconoscimento è più immediato nel caso di automi deterministici: il cammino accettante se c'è, è unico)
- Conversione di NFA in un equivalente DFA: costruzione degli insiemi; costruisce a partire da un NFA un DFA equivalente

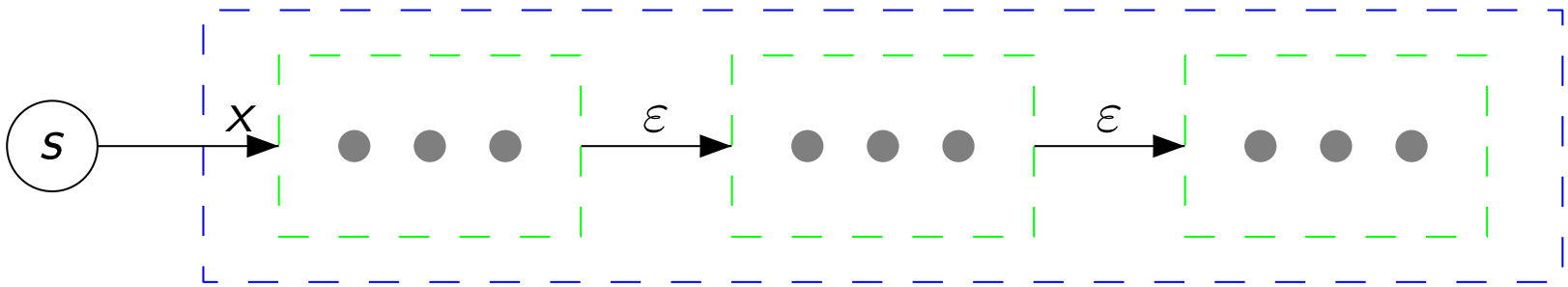
Costruzione dei sottoinsiemi

- **Input**: un NFA $N = \langle S, \Sigma, \mathbf{move}, s_0, F \rangle$
- **Output**: un DFA $D = \langle Dstates, \Sigma, \mathbf{Dmove}, s'_0, F' \rangle$ equivalente ad N e tale che $Dstates \subseteq \wp(S)$
- Alcune utili funzioni:
  1. ε-closure: $S \longmapsto \wp(S)$
     per ogni stato $s \in S$, ε-closure(s) contiene s più eventuali altri stati raggiungibili in N da s tramite sequenze di ε-transizioni
  2. ε-closure: $\wp(S) \longmapsto \wp(S)$
     per ogni insieme di stati $T \subseteq S$, ε-closure(T) $= \bigcup_{(s \in T)} \varepsilon$-closure$(s)$
     è una estensione della precedente funzione a insiemi di stati
  3. **move**: $(\wp(S) \times \Sigma) \longmapsto \wp(S)$
     per ogni $T \subseteq S$ ed per ogni $a \in \Sigma$, $\mathbf{move}(T, a) = \bigcup_{(s \in T)} move(s, a)$
     è una estensione della funzione **move** a insiemi di stati

Costruzione dei sottoinsiemi

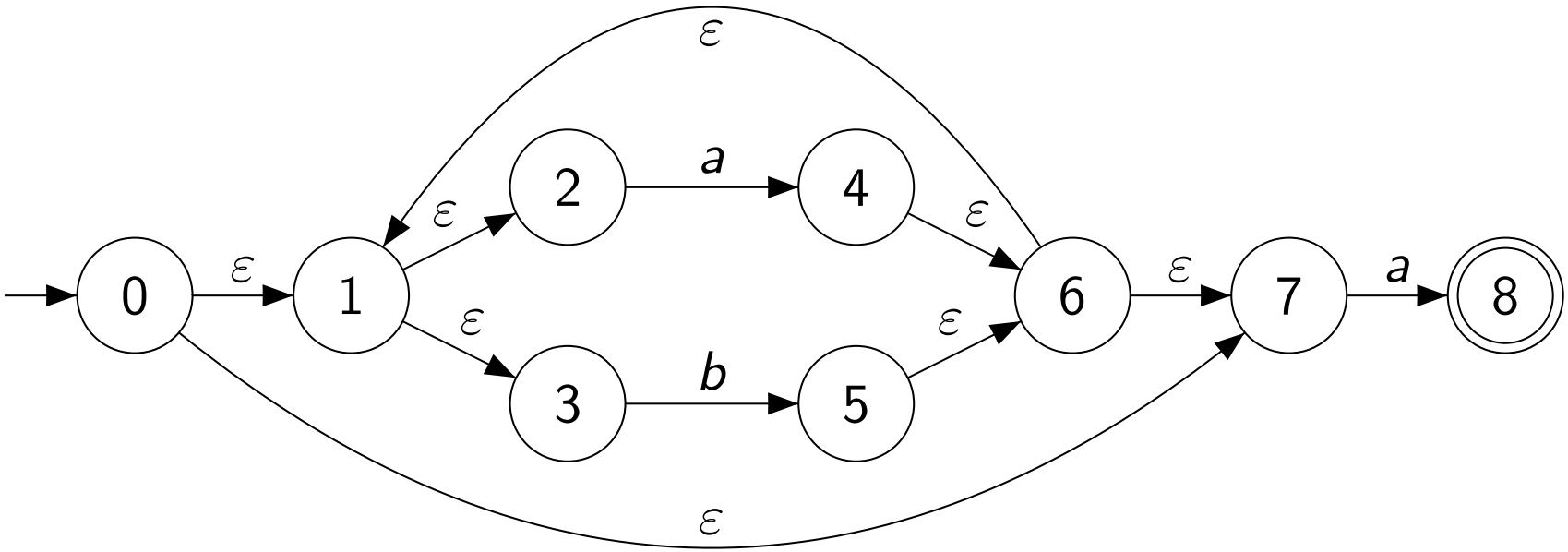
- **Input**: un NFA $N = \langle S, \Sigma, \textbf{move}, s_0, F \rangle$
- **Metodologia**:
  $Dstates = \{\varepsilon\text{-closure}(s_0)\}$ ed ε-closure(s_0) è non marcato
  while (esiste uno stato $T \in Dstates$ non marcato) **do begin**
  marca T;
  for each $x \in \Sigma$ **do begin**
  $U := \varepsilon\text{-closure}(\textbf{move}(T, x))$;
  if $(U \notin Dstates)$ **then**
  aggiungi U, non marcato, in $Dstates$;
  Dmove$(T, x) := U$;
  end;
  end;
- $s_0' = \varepsilon\text{-closure}(s_0)$
- $F' = \{T \in Dstates \mid T \cap F \neq emptyset\}$ l'insieme degli stati di D contenenti almeno uno stato finale di N

ε-closure(**move**(s, x))



Un esercizio

Determinare un automa deterministico equivalente al seguente NFA:



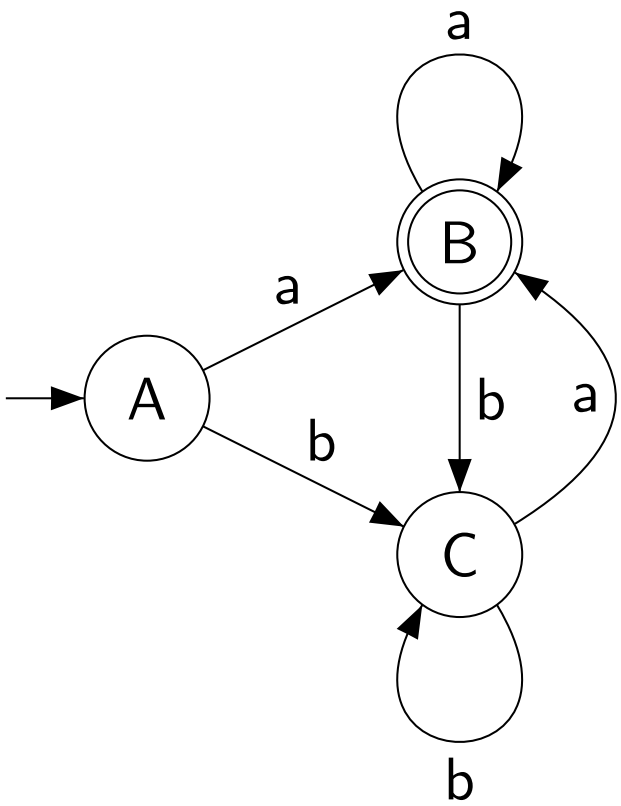
Prima la ε-closure

- Abbiamo bisogno di calcolare la ε-closure per tutti gli stati del nostro NFA
  - ε-closure$(0) = \{0, 1, 7, 2, 3\}$
  - ε-closure$(1) = \{1, 2, 3\}$
  - ε-closure$(4) = \{4, 6, 1, 7, 2, 3\}$
  - ε-closure$(5) = \{5, 6, 1, 7, 2, 3\}$
  - ε-closure$(6) = \{6, 1, 7, 2, 3\}$
  - ε-closure$(x) = \{x\}$ per ogni $x \in \{2, 3, 7, 8\}$ (questi stati non hanno ε-transizioni uscenti)

Poi *Dstates* e *Dmove*

- Lo stato iniziale del DFA è $A = \varepsilon$-closure(0)$= \{0, 1, 7, 2, 3\}$
- $Dmove(A, a) = \varepsilon$-closure(move(A,a)) $= \varepsilon$-closure$(\{8, 4\}) =$ ε-closure(8) $\cup$ ε-closure(4)$= \{8, 4, 6, 1, 7, 2, 3\} =$ B
- $Dmove(A, b) = \varepsilon$-closure(move(A,b)) $= \varepsilon$-closure$(\{5\}) =$ $\{5, 6, 1, 7, 2, 3\} =$ C
- $Dmove(B, a) = \varepsilon$-closure(move(B,a)) $= \varepsilon$-closure$(\{8, 4\}) =$ B
- $Dmove(B, b) = \varepsilon$-closure(move(B,b)) $= \varepsilon$-closure$(\{5\}) =$ C
- $Dmove(C, a) = \varepsilon$-closure(move(C,a)) $= \varepsilon$-closure$(\{8, 4\}) =$ B
- $Dmove(C, b) = \varepsilon$-closure(move(C,b)) $= \varepsilon$-closure$(\{5\}) =$ C
- Il DFA ha $\{A, B, C\}$ come insieme degli stati; l'unico finale è B la cui intersezione con l'insieme degli stati finali di N (ossia $\{8\}$) è diversa dall'insieme vuoto

L'automa deterministico equivalente è:



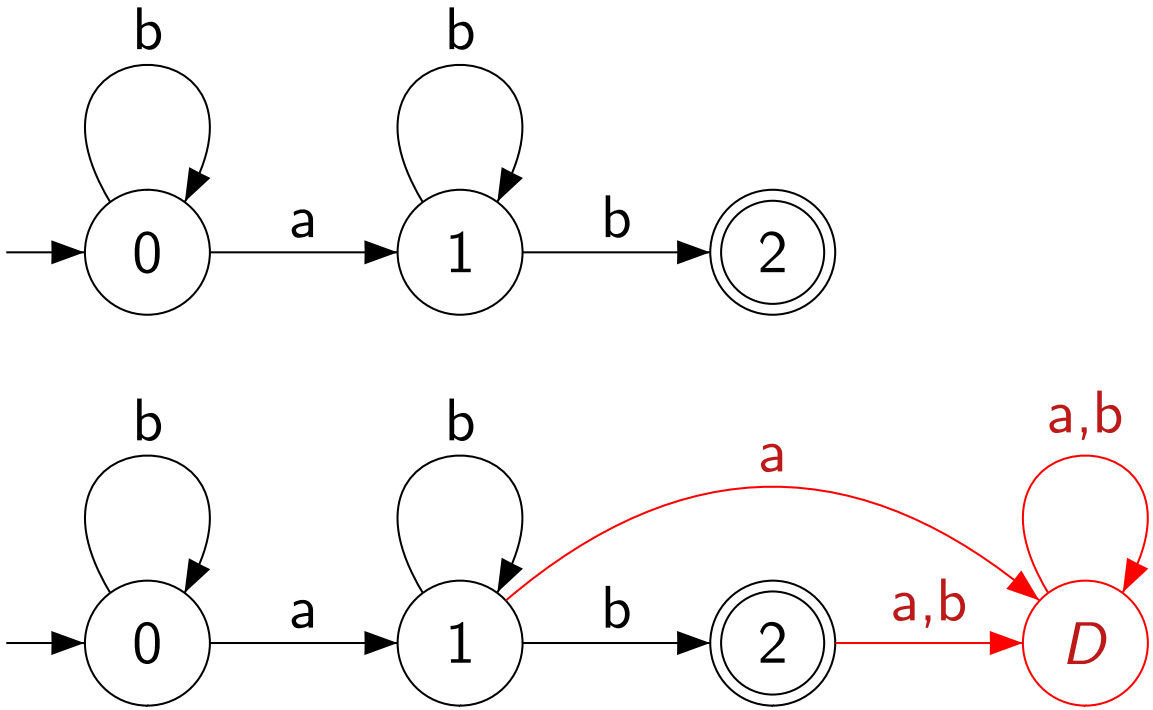
Mininizzazione di un DFA

- Tipicamente, un DFA ha un numero di stati maggiore rispetto al NFA equivalente
- L'algoritmo di minimizzazione prende in input un DFA M e restituisce un DFA M' equivalente ma con un numero minimo di stati
- L'automa minino è unico a meno di ridenominazioni degli stati
- Vincolo: può essere applicato ad automi la cui funzione di transizione non restituisce mai l'insieme vuoto

$$\forall s \in S, x \in \Sigma, \ \textbf{move}(s, x) \neq \emptyset$$

- Aggiungiamo un nodo, detto **nodo pozzo** o **dead state** che raccoglie tutte le transizioni "mancanti"

Dead state



Stati indistinguibili

Siano $M = \langle S, \Sigma, \mathbf{move}, s_0, F \rangle$ un DFA ed s e $t \in S$

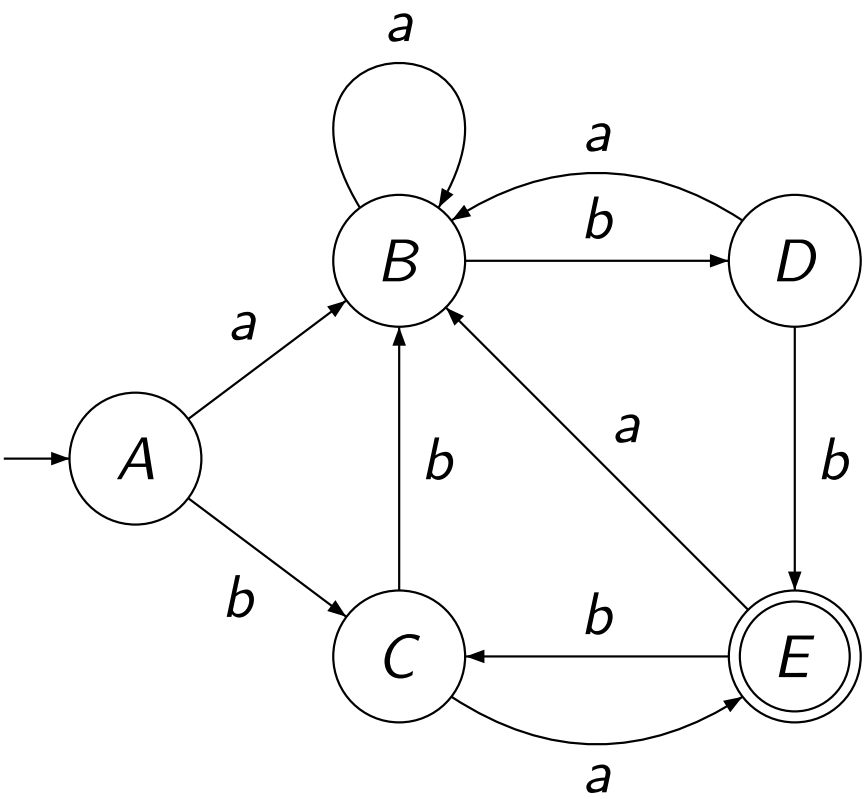
- Sia **move**: $(S \times \Sigma^*) \longmapsto \wp(S)$ l'estenzione della funzione di transizione **move** a stringhe in Σ^*
- Dato $s \in S$ e $w \in \Sigma^*$, $\mathbf{move}(s, w)$ è definita per induzione sulla lunghezza della stringa w:

  $\mathbf{move}(s, \varepsilon) = s,$

  $\mathbf{move}(s, xw) = move(move(s, x), w)$
- Diciamo che gli stati s e t sono indistinguibili se per ogni $w \in \Sigma^*$ $\mathbf{move}(s, w) \in F$ e $\mathbf{move}(t, w) \in F$
- Diciamo che gli stati s e t sono distinguibili se esiste una stringa $w \in \Sigma^*$ tale che
  - $\mathbf{move}(s, w) \in F$ ma $\mathbf{move}(t, w) \notin F$, oppure
  - $\mathbf{move}(t, w) \in F$ ma $\mathbf{move}(s, w) \notin F$

  In questo caso, diciamo che la stringa w distingue gli stati s e t

Stati indistinguibili



- ε distingue stati finali e stati non finali; possiamo partizionare l'insieme degli stati in due sottoinsiemi $S \setminus F$ ed F
- L'insieme $S \setminus F = \{A, B, C, D\}$ contiene due stati A e B distinti dalla stringa bb:

$$A \xrightarrow{b} C \xrightarrow{b} B \notin F,$$

$$B \xrightarrow{b} D \xrightarrow{b} E \in F$$

Partizionamento per stati indistinguibili

- Sia $M = \langle S, \Sigma, \textbf{move}, s_0, F \rangle$ un DFA
- L'algoritmo di minimizzazione cerca una partizione $\Pi = \{S_1, \ldots, S_n\}$ dell'insieme degli stati S tale che stati appartenenti ad insiemi della partizione siano indistinguibili, ossia:

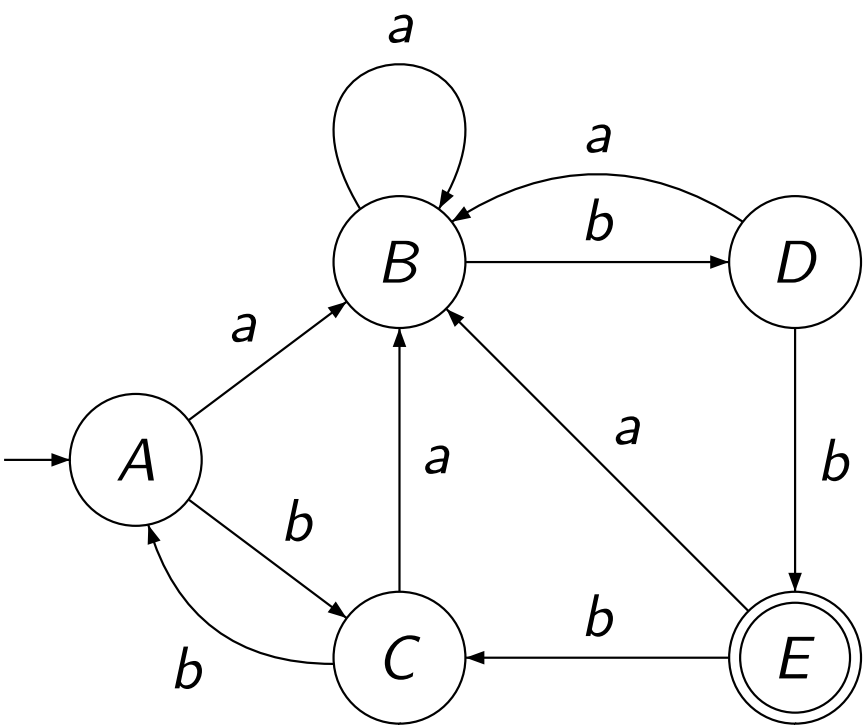
$$\forall i = 1, \ldots, n \text{ e } \forall s, t \in S_i,\ s \text{ e } t \text{ sono indistinguibili } (1)$$

- Partizione iniziale: $\Pi_0 = \{S \setminus F, F\}$
- Questa parzione, in generale troppo grossolana, viene raffinata per passi successivi fino ad ottenere una partizione finale Π_{finale} che soddisfa la condizione (1)

Passo di raffinamento

- Sia $\Pi = \{S_1, \ldots, S_n\}$ la parzione corrente
- Sia $i = 1, \ldots, n$, $s, t \in S_i$ ed $x \in \Sigma$ tali che $\mathbf{move}(s, x) = s' \in S_j$ e $\mathbf{move}(t, x) = t' \in S_k$ con $j \neq k$
- Gli stati s' e t' sono distinguibili ed esiste una stringa w tale che $\mathbf{move}(s', w) \in F$ ed $\mathbf{move}(t', w) \notin F$ (o viceversa)
- Allora, la stringa xw distingue s da t:
  $\mathbf{move}(s, xw) = \mathbf{move}(\mathbf{move}(s, x), w) = \mathbf{move}(s', w) \in F$ ma
  $\mathbf{move}(t, xw) = \mathbf{move}(\mathbf{move}(t, x), w) = \mathbf{move}(t', w) \notin F$
  (o viceversa)
- S_i contiene stati distinguibili e deve essere ulteriormente partizionato

Algoritmo di minimizzazione



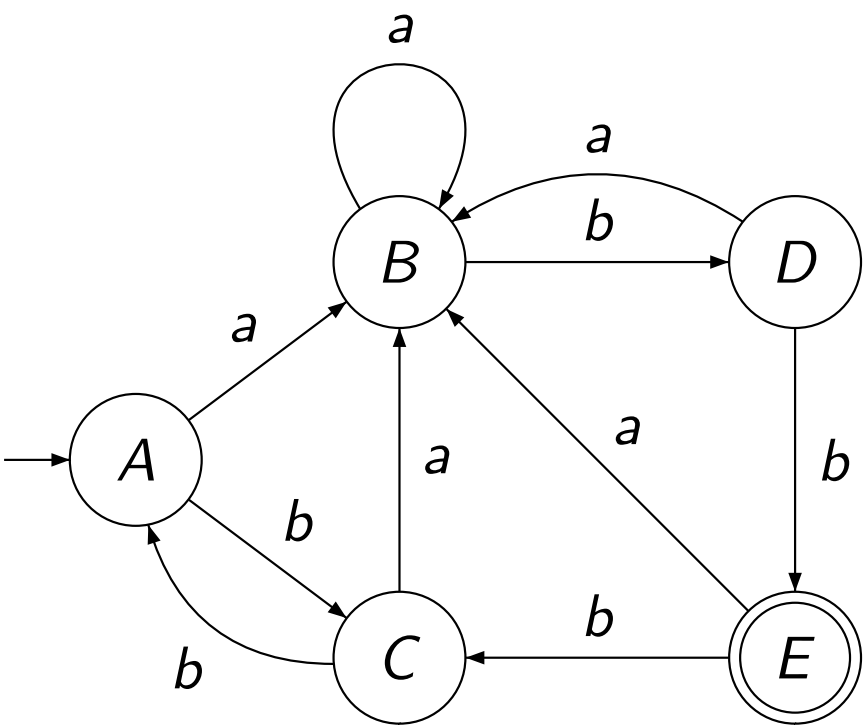
$$\Pi_0 = \{S \setminus F = \{A, B, C, D\}, F = \{E\}\}$$

$$\textbf{move}(D, b) = E \in F,$$
$$\textbf{move}(A, b) = C \in S \setminus F$$

$S \setminus F$ viene partizionato in due sottoinsiemi $\{D\}$ e $G_1 = \{A, B, C\}$

otteniamo una nuova partizione
$\Pi_1 = \{G_1, \{D\}\{E\}\}$

Algoritmo di minimizzazione



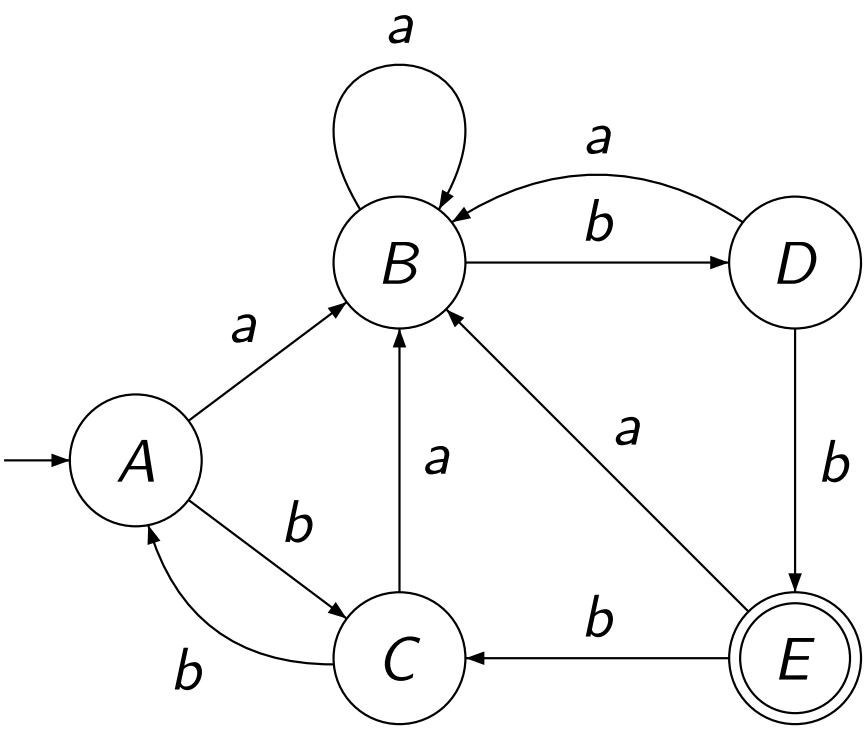
$$\Pi_1 = \{G_1 = \{A, B, C\}, \{D\}, \{E\}\}$$

$\mathbf{move}(A, b) = C \in G_1$,
$\mathbf{move}(B, b) = D \in \{D\}$

G_1 viene partizionato in due sottoinsiemi $\{B\}$ e $G_2 = \{A, C\}$

otteniamo una nuova partizione
$\Pi_2 = \{G_2, \{B\}, \{D\}, \{E\}\}$

Algoritmo di minimizzazione



$$\Pi_2 = \{G_2 = \{A, C\}, \{B\}, \{D\}, \{E\}\}$$

$$\textbf{move}(A, b) = C \in G_2,$$
$$\textbf{move}(C, b) = A \in G_2$$

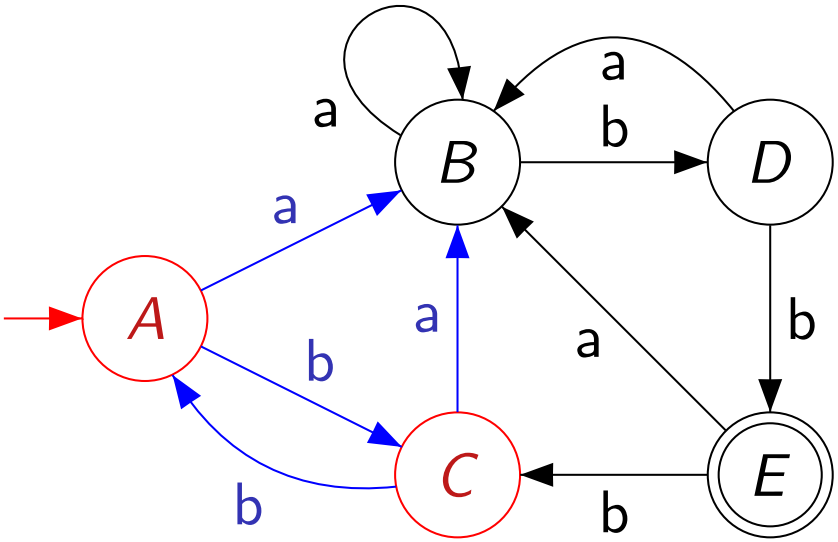
$$\textbf{move}(A, a) = B \in \{B\}$$
$$\textbf{move}(C, a) = B \in \{B\}$$

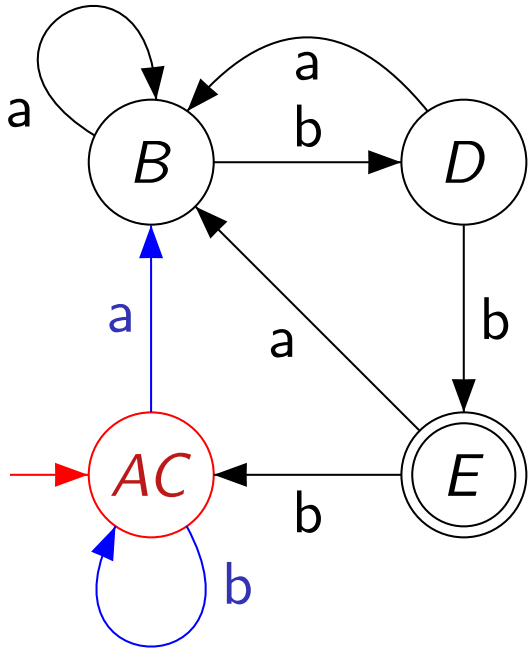
G_2 non può essere ulteriormente partizionato

$$\Pi_{finale} = \{G_2, \{B\}, \{D\}, \{E\}\}$$

Algoritmo di minimizzazione

- $\Pi_{finale} = \{G_2, \{B\}, \{D\}, \{E\}\}$
- L'automa minimo viene costruito scegliendo uno stato per ogni gruppo in Π_{finale} e modificando opportunamente la funzione di transizione





Da espressioni regolari a NFA

Definizione di pattern

Reg_Expr

Algoritmo di Thompson

Riconoscimento di lessemi

NFA

costr. dei sottoinsiemi

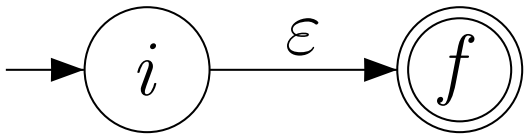
DFA

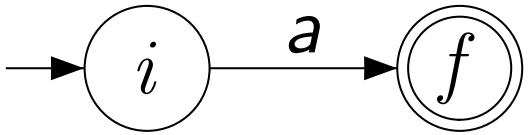
Algoritmo di Thompson

- data un'espressione regolare r costruisce un NFA N che riconosce il linguaggio $L(r)$
- L'automa N soddisfa le seguenti proprietà:
  1) ha un solo stato finale
  2) non esiste alcuna transizione uscente dallo stato finale
  3) non esiste alcuna transizione entrante nello stato iniziale
- il procedimento è definito per induzione sulla struttura di r

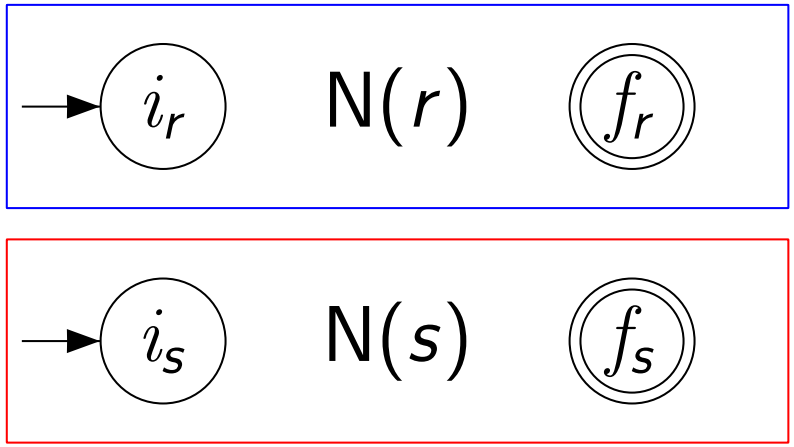
Algoritmo di Thompson: casi base

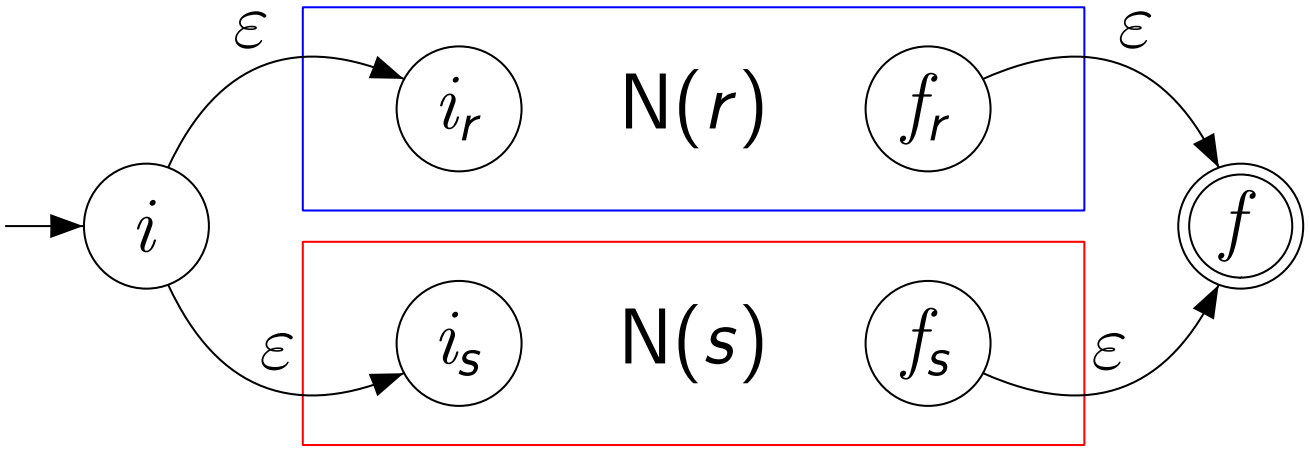
- Assumiamo $r = \varepsilon$ oppure $r = a$



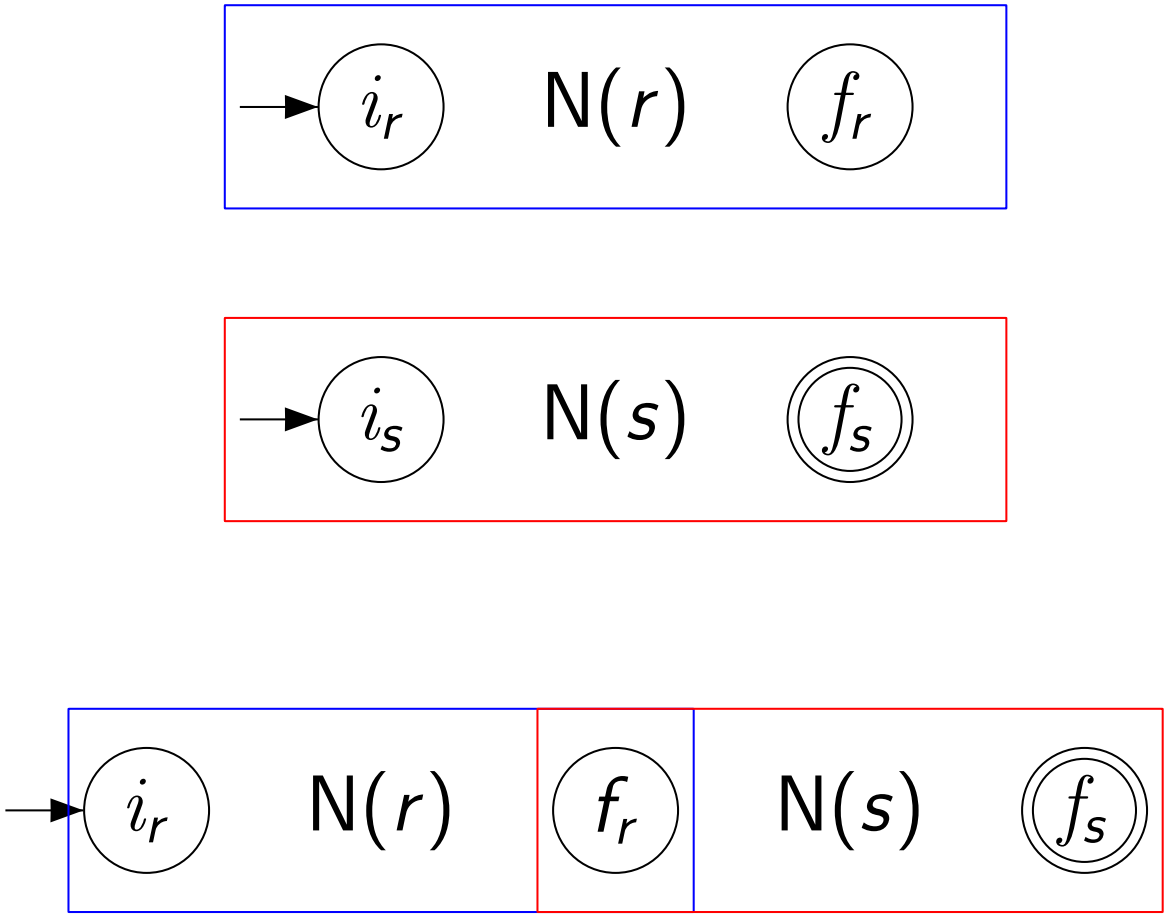


Algoritmo di Thompson: $r \mid s$

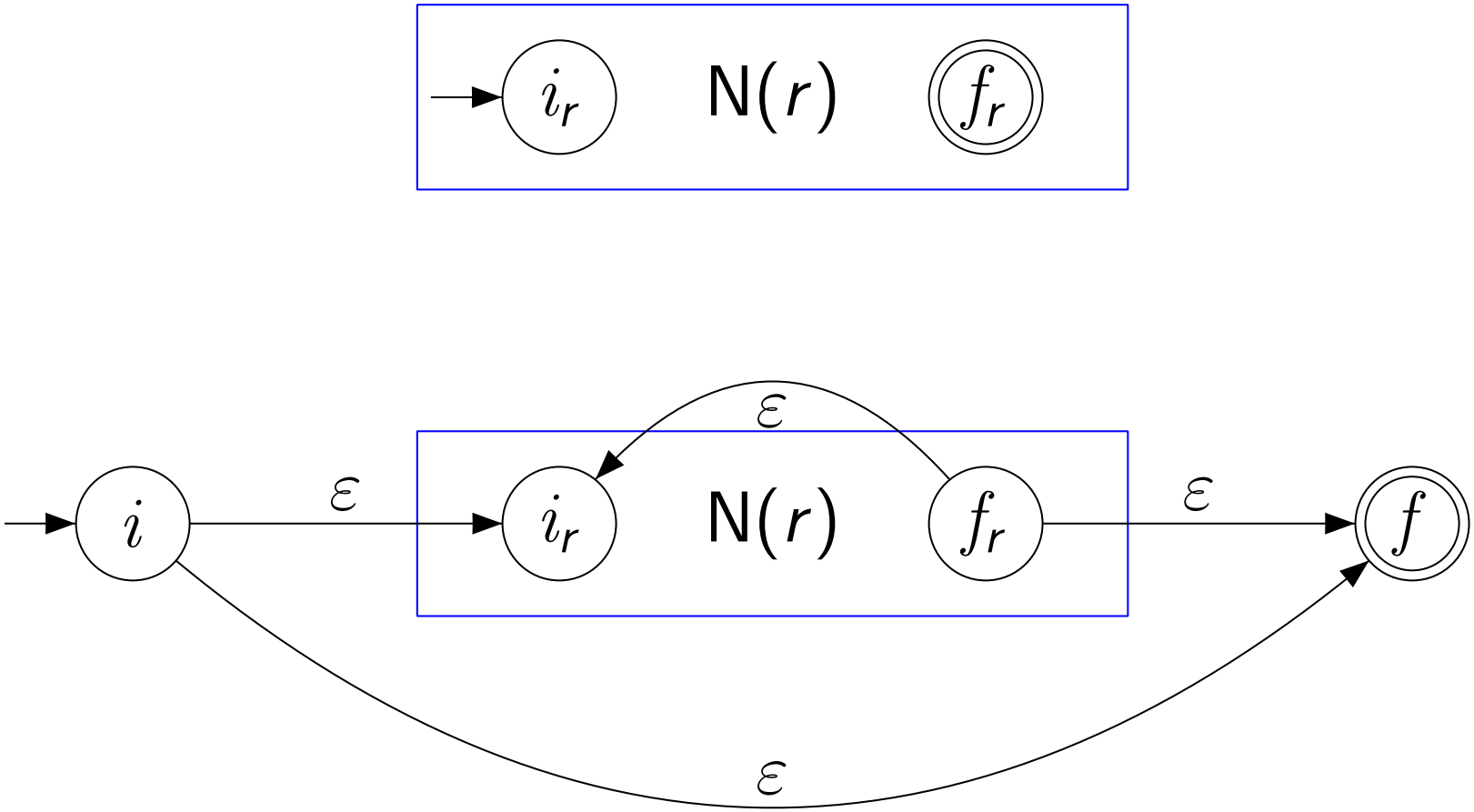




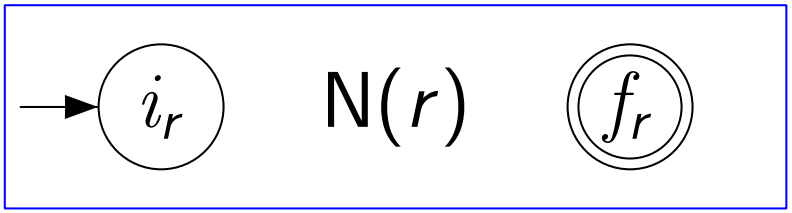
Algoritmo di Thompson: *rs*



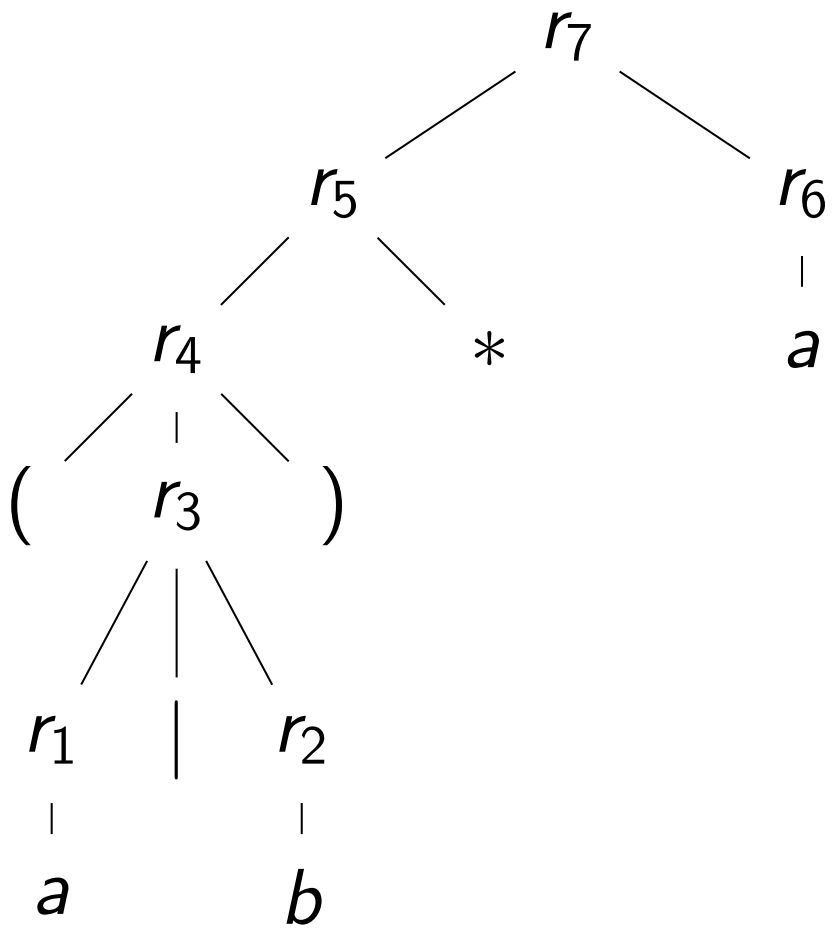
Algoritmo di Thompson: r^*



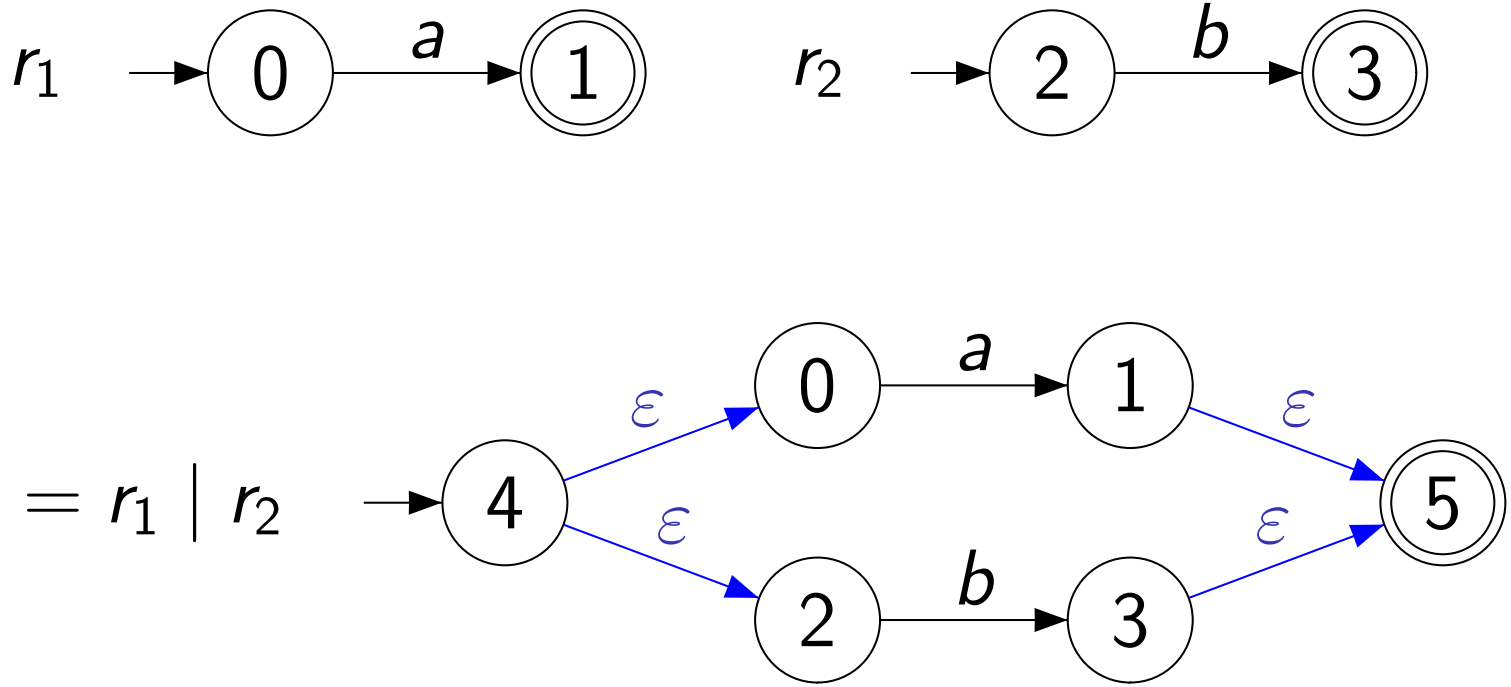
Algoritmo di Thompson: (r)



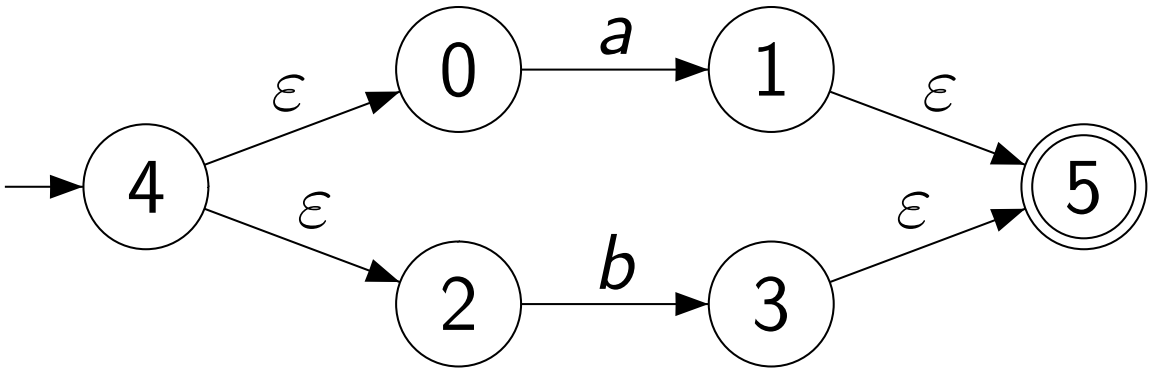
Automa associato all'espressione $r = (a \mid b)^* a$



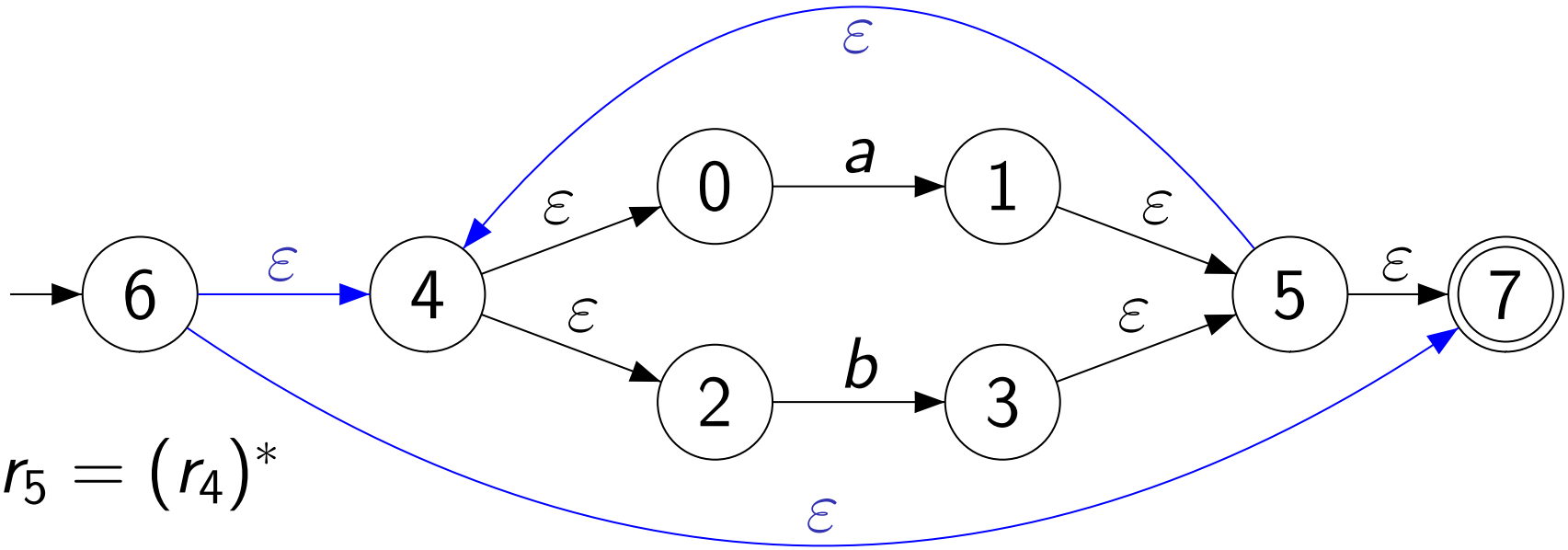
Automa associato all'espressione $r = (a \mid b)^* a$



Automa associato all'espressione $r = (a \mid b)^* a$

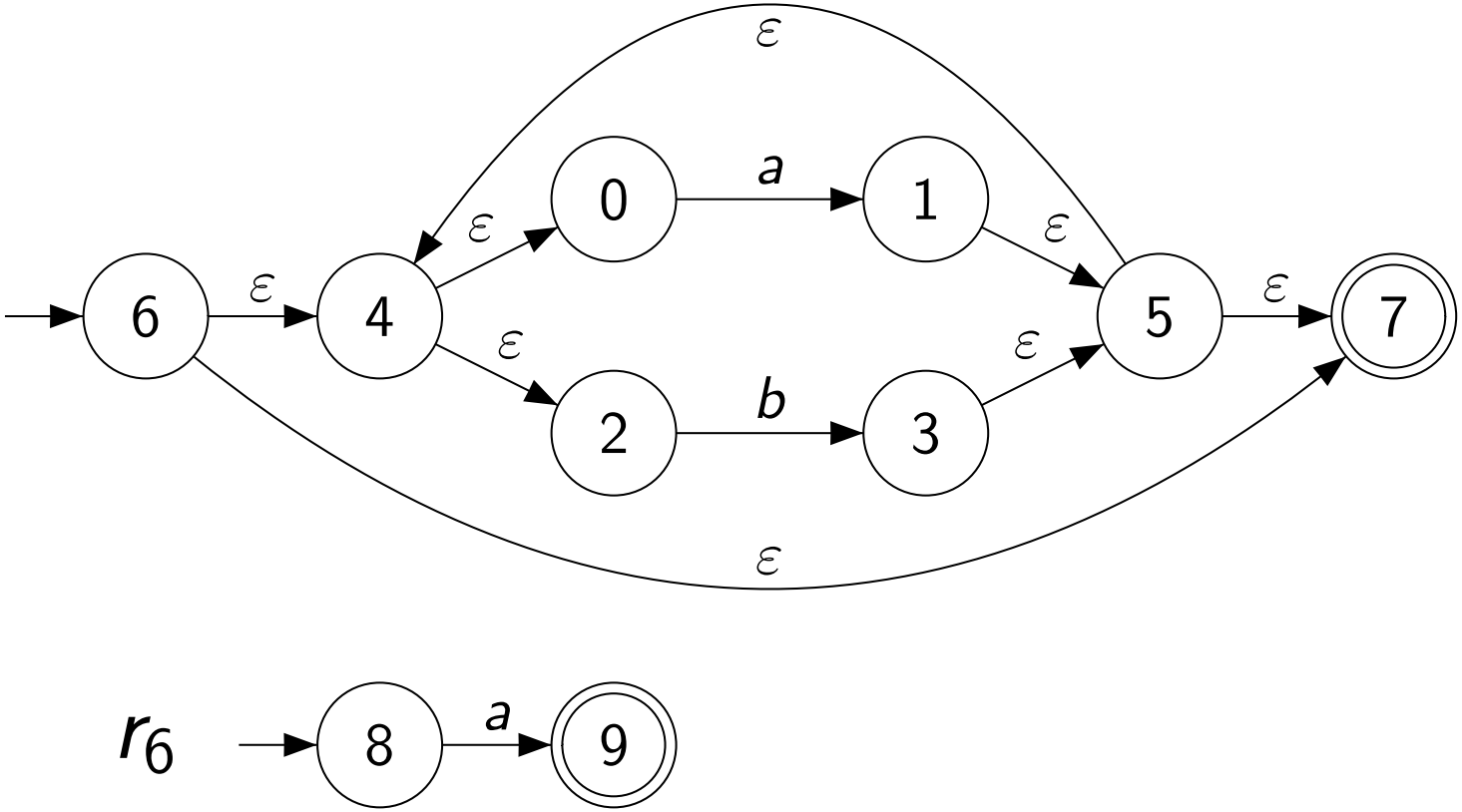


$r_4 = (r_3)$



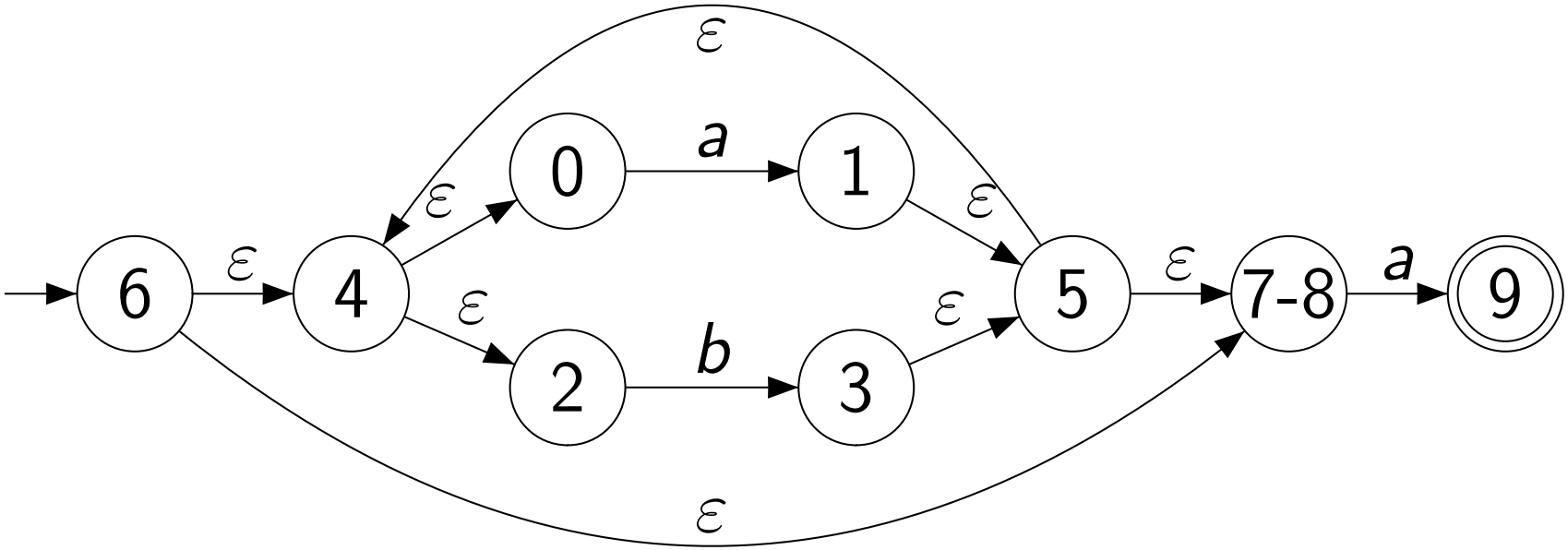
$r_5 = (r_4)^*$

Automa associato all'espressione $r = (a \mid b)^* a$



Automa associato all'espressione $r = (a \mid b)^* a$

$$r_7 = r_5 r_6$$



Part IV

Generatori Automatici

Generatori Automatici di Analizzatori Lessicali

- Un analizzatore lessicale riconosce i lessemi che si susseguono nel programma sorgente individuando tutti quelli che fanno match con i vari token del linguaggio (pattern matching)
- **Input**: la definizione di un certo numero di pattern $p_1, p_2, \ldots, p_n$

$$\begin{array}{ll} p_1 & \{a_1\} \\ p_2 & \{a_2\} \\ \ldots & \\ p_n & \{a_n\} \end{array}$$

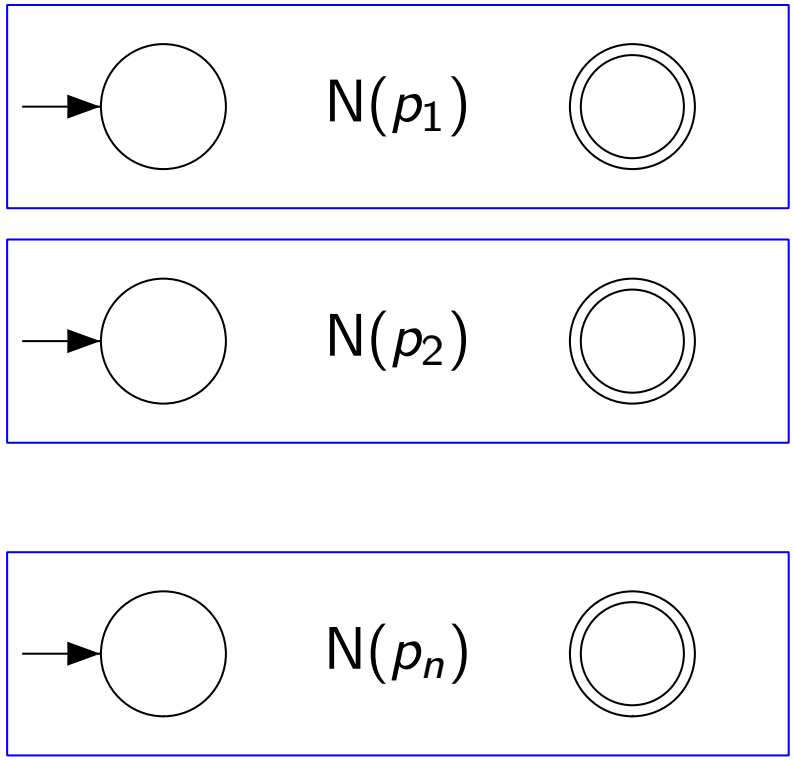
  dove $p_1, p_2, \ldots, p_n$ sono delle espressioni/definizioni regolari
- Ad ogni pattern viene associata un azione eseguita al momento in cui si individua un lessema che fa match con quel dato token (es: istalla un identificatore nella symbol table)

Generatori Automatici

- **Output**: un programma che legge una sequenza di caratteri (il nostro programma in input) ed implementa un ciclo in cui, ad ogni passo viene individuata **la sequenza più lunga** che fa match con uno dei pattern p_i
- Nel caso in cui la sequenza faccia match con due patterns distinti viene scelto in pattern più in alto nella definizione (ad esempio p_2 e non p_3)
- Una volta individuato un pattern p_i viene eseguita l'azione a_i ad esso associata
- Il ciclo termina se:
  - non ci sono più caratteri da esaminare (fine del programma) oppure
  - si è verificato un errore

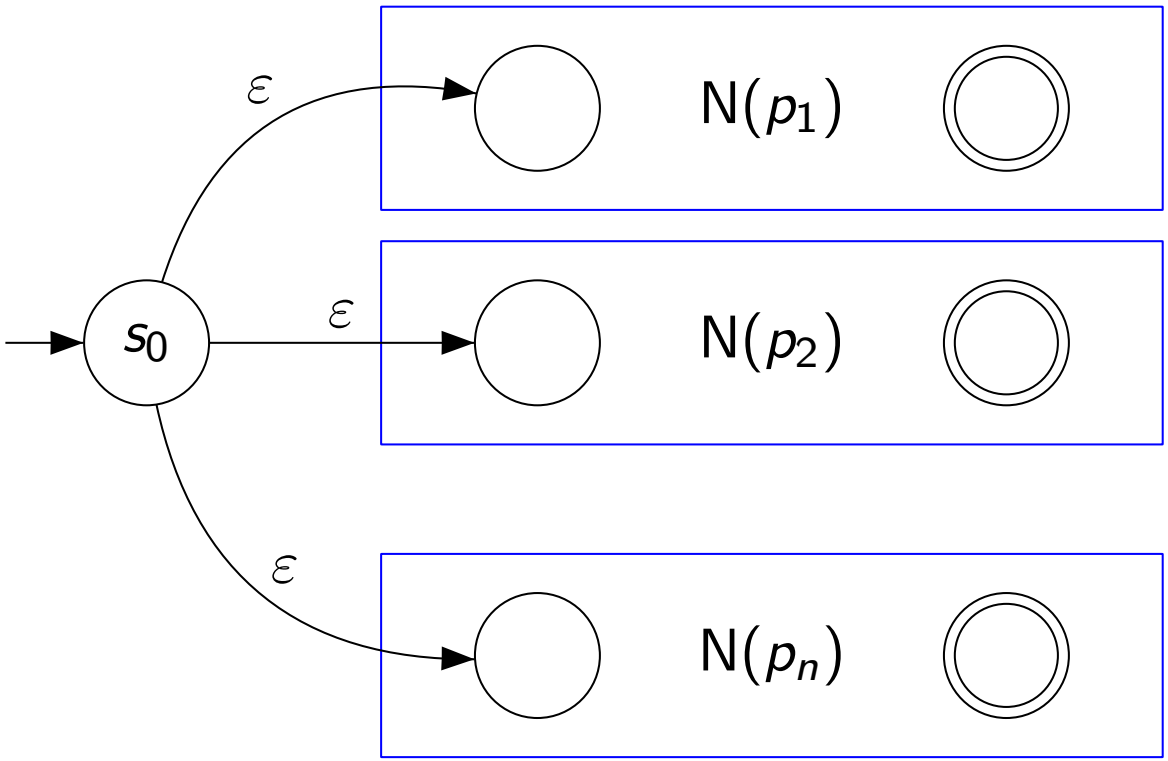
Generatori Automatici basati su NFA

- **Passo 1**: convertire i pattern $p_1, p_2, \ldots, p_n$ in NFA ottenendo così n automi non deterministici $N(p_1), N(p_2), \ldots, N(p_n)$



Generatori Automatici basati su NFA

- **Passo 2:** Costruire un unico NFA a partire dagli n automi costruiti nel passo precedente come segue:



Generatori Automatici basati su NFA

- Ad ogni iterazione l'algoritmo legge il corrente input e si "muove" lungo l'automa fino a quando non arriva ad uno stato finale
- Il determinismo viene simulato mantenendo un insieme di stati attuali (come nella costruzione dei sottoinsiemi)
- Usiamo una variabile `p` per mantenere il puntatore all'ultimo carattere dell'ultimo lessema individuato ed una variabile `npattern` per memorizzare il numero corrispondente all'ultimo pattern riconosciuto
- Quando incontriamo uno o più stati finali:
  - è stato individuato un nuovo possibile lessema (compreso tra `*(p+1)` e l'ultimo carattere letto); aggiorniamo i valori di `p` ed `npattern`
  - la corrente iterazione riprende a partire dallo stesso insieme di stati attuali e dal carattere successivo all'ultimo lessema individuato

Generatori Automatici basati su NFA

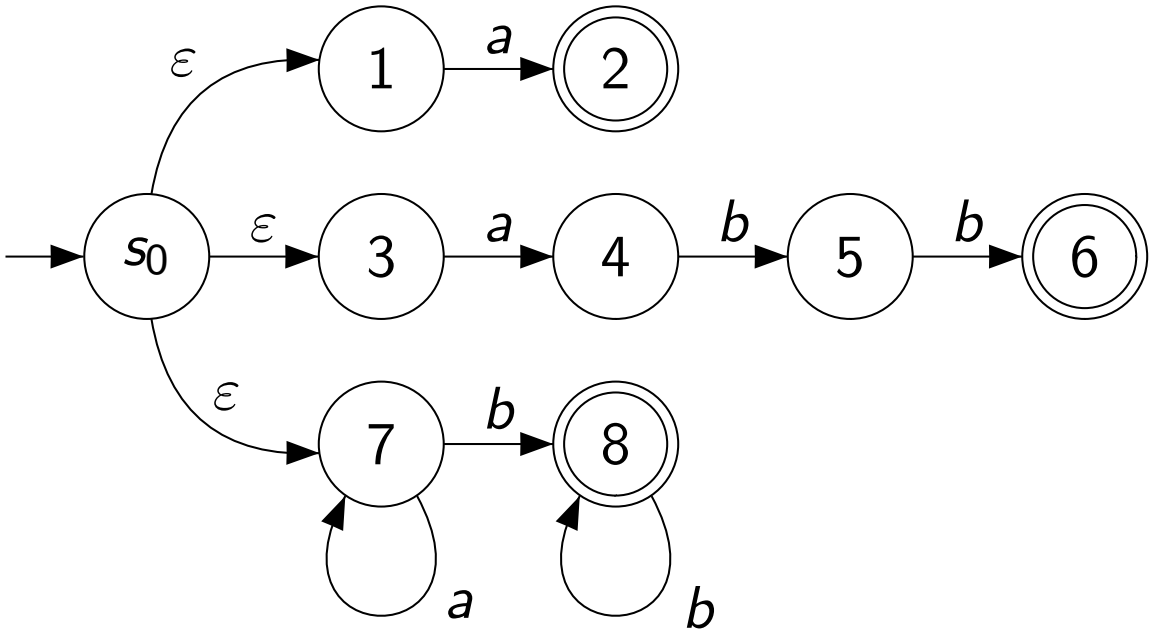
- La procedura va avanti finchè non è più possibile fare altre mosse (l'input è finito o non è possibile fare alcuna transizione)
- L'algoritmo annuncia che è stato riconosciuto un nuovo token e che il lessema corrispondente va dal primo carattere considerato nell'attuale interazione al carattere puntato da p
- Il ciclo ricomincia facendo ripartire l'automa dal suo stato iniziale e sull'input il cui primo carattere è quello successivo all'ultimo lessema riconosciuto (successivo a quello puntato da p)

Un esempio

- Consideriamo la seguente definizione di tre pattern

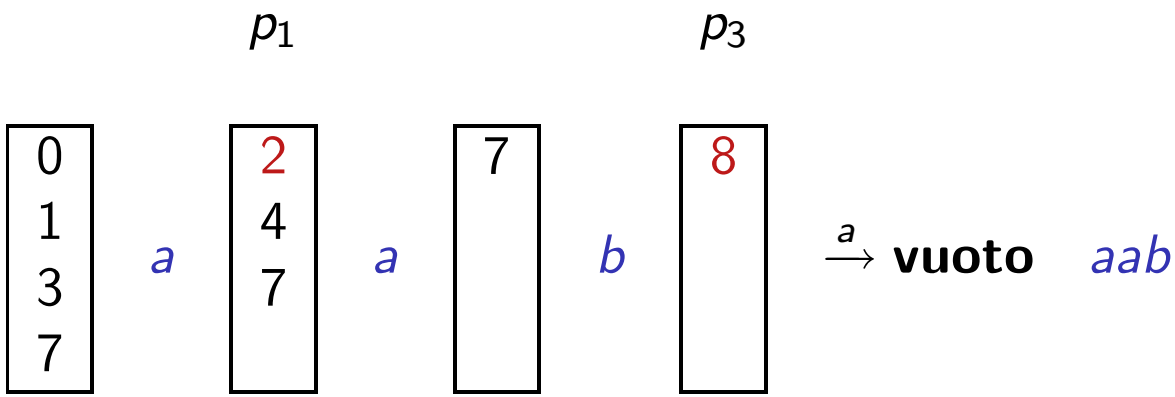
$$\begin{array}{ll} a & \{\} \\ abb & \{\} \\ a^* b^+ & \{\} \end{array}$$

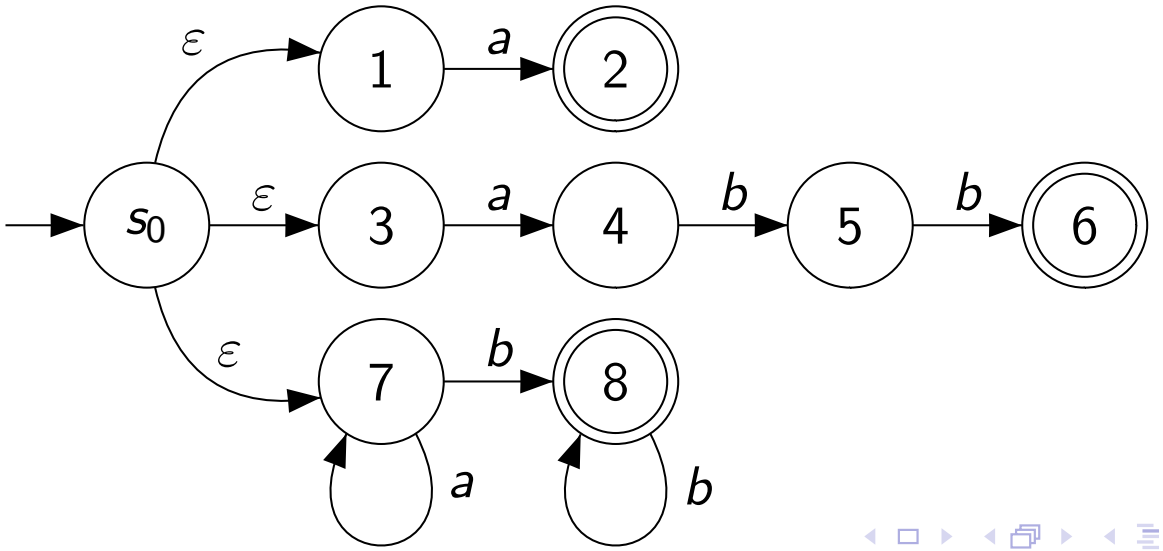
- Dobbiamo costruire gli NFA per ciascun di essi e poi metterli insieme



Un esempio

Simuliamo l'algoritmo di pattern matching in corrispondenza di *aaba*: lo stato iniziale è dato da ε-closure(0)$= \{0, 1, 3, 7\}$



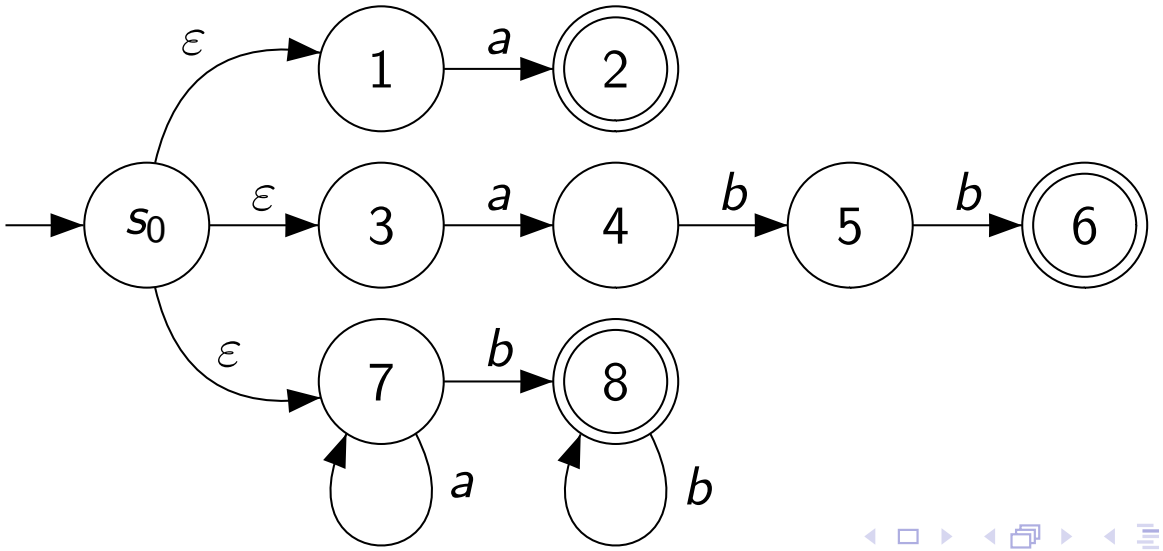


Un esempio

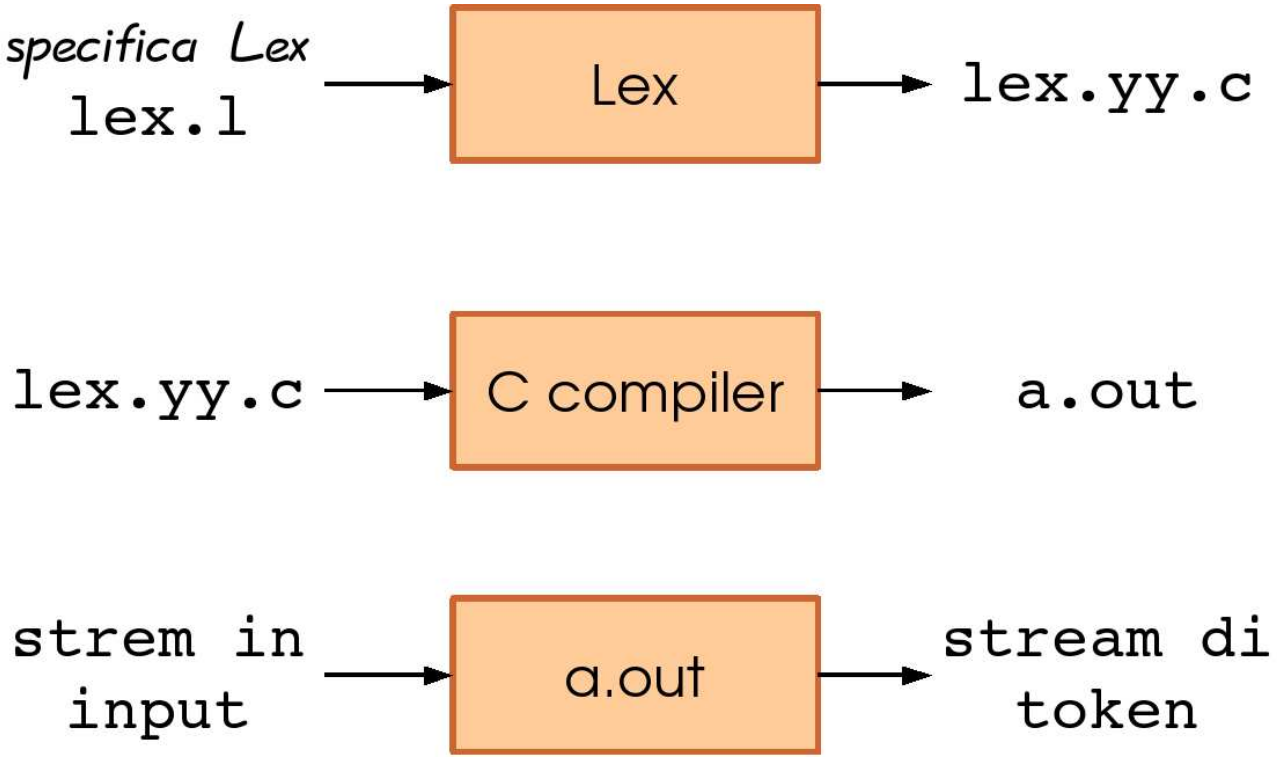
Resta da analizzare la porzione di input a: lo stato iniziale è ancora da ε-closure(0)$= \{0, 1, 3, 7\}$

p_1

0 1 3 7	a	2 4 7	**fine input**	a



Creare un analizzatore lessicale con LEX



Specifiche Lex - struttura generale

sezione dichiarazioni	dichiarazioni di costanti, variabili globali e definizioni regolari
%%	
transition rules	insieme di pattern ed azioni associate
%%	
procedure ausiliarie	necessarie per implementare le azioni associate ai pattern