



**Bauman Moscow State University**  
**Th. Computer Science Dept.**

# Finite State Machines and Regular Expressions



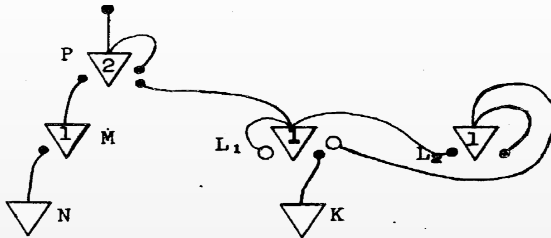
*Antonina Nepeivoda*  
*a\_nevod@mail.ru*

# Lecture Outline

- 1 **Basic Notions**
- 2 **Closures and Determinisation**
  - $\varepsilon$ -Removal by Closure
  - Subset Construction and Determinisation
- 3 **From NFA to Regular Expressions**
  - Solving Language Equations
  - State Eliminating Method
- 4 **From Regular Expressions to NFA**
  - Thompson NFA
  - Glushkov NFA



# Reminder: Neural Networks by McCulloch–Pitts



- — excitatory signal;
- — inhibitory signal;
- ▽ — an input neuron;

▽ $k$  — an inner neuron firing whenever none of the inhibitory signals and at least  $k$  of excitatory signals fire.

Naturally imitate: disjunction, conjunction, negation, iteration, concatenation.



# Regular Expressions by Kleene

## ☹☹ Academic Definition

Given alphabet  $\Sigma$ , a regular expression is either a letter in  $\Sigma$ ,  $\varepsilon$ , or a result of following operations, where  $r_1, r_2$  are regular expressions:

- $r_1 \mid r_2$  — union (alternation).  $\mathcal{L}(r_1 \mid r_2) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2)$ ;
- $r_1 r_2$  — concatenation (sequencing).  
 $\mathcal{L}(r_1 r_2) = \{\omega_1 \omega_2 \mid \omega_1 \in \mathcal{L}(r_1) \ \& \ \omega_2 \in \mathcal{L}(r_2)\}$ ;
- $(r_1)^*$  — iteration (0 or more concatenations of  $r_1$  with itself);

$$\mathcal{L}((r_1)^*) = \{\varepsilon\} \bigcup_{i=1}^{\infty} \mathcal{L}(r_1).$$

## Syntactic Sugar

- $r^+$  — positive iteration (shortcut for  $r r^*$ );
- $r?$  — option (shortcut for  $(r \mid \varepsilon)$ ).



# Regular Expressions by Kleene

## ☹☹ Academic Definition

Given alphabet  $\Sigma$ , a regular expression is either a letter in  $\Sigma$ ,  $\varepsilon$ , or a result of following operations, where  $r_1, r_2$  are regular expressions:

- $r_1 \mid r_2$  — union (alternation).  $\mathcal{L}(r_1 \mid r_2) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2)$ ;
- $r_1 r_2$  — concatenation (sequencing).  
 $\mathcal{L}(r_1 r_2) = \{\omega_1 \omega_2 \mid \omega_1 \in \mathcal{L}(r_1) \ \& \ \omega_2 \in \mathcal{L}(r_2)\}$ ;
- $(r_1)^*$  — iteration (0 or more concatenations of  $r_1$  with itself);

$$\mathcal{L}((r_1)^*) = \{\varepsilon\} \bigcup_{i=1}^{\infty} \mathcal{L}(r_1).$$

Priorities: star > concatenation > union.

$$ab^* \mid c^*d \Leftrightarrow \left(a(b^*)\right) \mid \left((c^*)d\right).$$



# Terminological Clash

## *Academic regexes*

- |, ., \* (sometimes +, ?) operations;
- define regular languages;
- studied in university courses (compilers & formal languages)

## *REGEX (extended regexes)*

- lookaheads, backreferences, etc;
- define non-context-free languages;
- used in practice (PCRE2 standart).

- Almost identical names are used for completely different (although related) notions.



# Occam Razor: Non-Deterministic Finite Automata

Only excitatory signals are left on there, and all inner neurons fire whenever there is at least one input signal.

## ☹☹ Definition

*A non-deterministic finite automaton (NFA) is a tuple*

*$\mathcal{A} = \langle Q, \Sigma, q_0, F, \delta \rangle$ , where:*

- *$Q$  — state set;*
- *$\Sigma$  — terminal alphabet;*
- *$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$  — transition rules;*
- *$q_0 \in Q$  — starting state;*
- *$F \subseteq Q$  — final states.*

Sometimes we use notation:

$\langle q_1, a, q_2 \rangle \in \delta \Leftrightarrow \langle q_1, a, M \rangle \in \delta \ \& \ q_2 \in M.$

Or, usually, simply:  $q_1 \xrightarrow{a} q_2.$



# Asymmetry of NFA Definition

- Classical works (Kleene, Brzozowski): multiple NFA starting states are allowed.
- Modern formal language theory: the unique starting state in NFA is assumed.
- Equivalent (we can add an unique starting state with  $\varepsilon$ -transitions to the multiple states), but confusing (e.g. in Brzozowski minimisation).



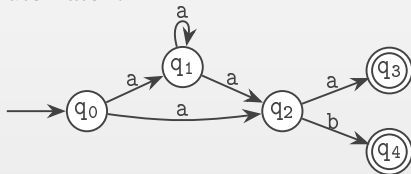


# Encoding into Grammars

## Observation

- Transition  $q_1 \xrightarrow{a} q_2$  can be seen as a rewriting rule  $[q_1] \rightarrow a[q_2]$ , assuming that  $[q_i]$  are some intermediate constructors, while  $a \in \Sigma$  is a terminal constructor.
- In order to model computation termination, for every final state  $q_F$ , we can add the rewriting rule  $[q_F] \rightarrow \varepsilon$ .

Automaton:



Grammar:

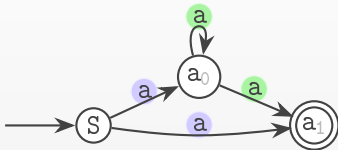
$S \rightarrow a[q_1]$	$[q_2] \rightarrow a[q_3]$
$S \rightarrow a[q_2]$	$[q_2] \rightarrow b[q_4]$
$[q_1] \rightarrow a[q_1]$	$[q_3] \rightarrow \varepsilon$
$[q_1] \rightarrow a[q_2]$	$[q_4] \rightarrow \varepsilon$

We rename the starting nonterminal  $[q_0]$  to  $S$ , for uniformity.

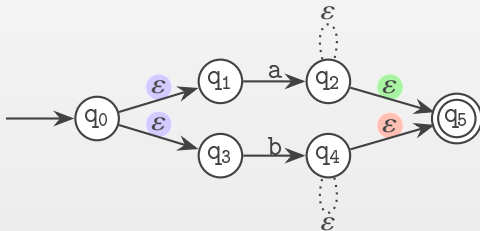


# Sources of Non-Determinism in an NFA

- Transition sets wrt (with respect to) a letter  $\gamma \in \Sigma$  that are not singletons.



- $\varepsilon$ -transitions (so-called silent actions).

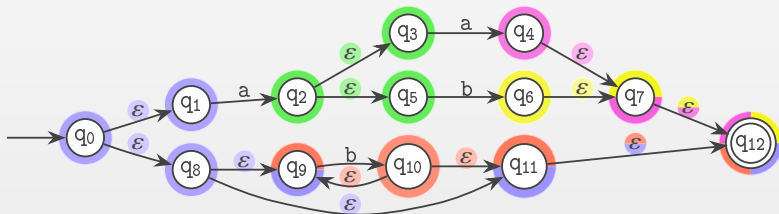


# Closures

Given  $\omega \in \Sigma^*$ , a  $\omega$ -closure of a state  $q$  in NFA  $\mathcal{A}$  is a set of states reachable from  $q$  by the action  $\omega$ .

We say that  $\omega$  is in the language of the NFA  $\mathcal{A}$  ( $\omega \in \mathcal{L}(\mathcal{A})$ )  
 $\Leftrightarrow$   $\omega$ -closure of the starting state of  $\mathcal{A}$  contains a final state.

Special case:  $\varepsilon$ -closures: sets of states reachable via doing nothing.

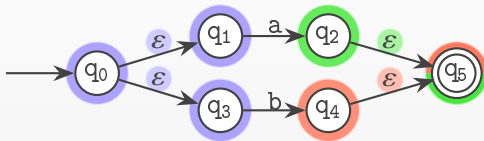


Given such closures, they can be considered as new «states».

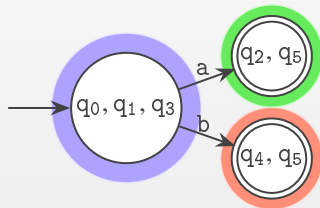


# Simple Example of $\varepsilon$ -Removal

An NFA  $\mathcal{A}$  with the  $\varepsilon$ -closures of its states being highlighted:



The closures are then merged into single states, and given a transition from  $q_i \xrightarrow{\gamma} q_j$ , where  $q_i$  belongs to closure  $M(q_i)$ , and  $q_j$  to  $M(q_j)$ , transition  $M(q_i) \xrightarrow{\gamma} M(q_j)$  is added.



A closure is marked as a final  $\Leftrightarrow$  it contains at least one final state.



## $\varepsilon$ -Closures and Chain Rules

- Any transition  $q_i \xrightarrow{\varepsilon} q_j$  corresponds to **a chain rule**  $[q_i] \rightarrow [q_j]$  in the corresponding grammar  $G$ .
- state  $\varepsilon$ -closure is a closure set of the corresponding non-terminal  $N$ :  
$$C(N) = \left\{ N_i \mid \exists N'_1, \dots, N'_k (N \rightarrow N'_1 \ \& \ \dots \ \& \ N'_k \rightarrow N_i) \right\}$$

I.e.  $\langle N, N_i \rangle$  are pairs in **a transitive closure**  $\rightarrow_c^+$  of the relation  $\rightarrow_c$ :  
 $A_i \rightarrow_c A_j \Leftrightarrow (A_i \rightarrow A_j \in G)$ .
- Before removing all chain rules, for every  $N' \in C(N)$  and a non-chain rule  $N' \rightarrow \Phi$ , we add the transition  $N \rightarrow \Phi$  to the set of grammar rules. Exactly as in the  $\varepsilon$ -closure algorithm for NFA.

Initial grammar:

$S \rightarrow Q_1$      $S \rightarrow Q_3$      $Q_1 \rightarrow aQ_2$   
 $Q_3 \rightarrow bQ_4$      $Q_2 \rightarrow Q_5$      $Q_4 \rightarrow Q_5$   
 $Q_5 \rightarrow \varepsilon$

After removing chain rules:

$S \rightarrow aQ_2$      $S \rightarrow bQ_4$   
 $Q_2 \rightarrow \varepsilon$      $Q_4 \rightarrow \varepsilon$

Note: unreachable non-terminals  $Q_1, Q_3, Q_5$  are deleted from the resulting grammar.



# $\omega$ -Closures and Subset Construction

The closure sets wrt transitions by non- $\varepsilon$  actions can be also merged in similar sense.

---

## *Subset Automaton Construction*

---

Let an  $\varepsilon$ -free NFA  $\mathcal{A}$  be given. Its **subset automaton**  $D(\mathcal{A})$  can be constructed as follows.

- $q_0$  becomes the starting state  $\{q_0\}$  of  $D(\mathcal{A})$ .
- Given a state  $M$  in  $D(\mathcal{A})$  and  $\gamma \in \Sigma$ , construct a closure set  $M_\gamma = \{q_i \mid \exists q_j \in M (q_j \xrightarrow{\gamma} q_i)\}$ . If  $M_\gamma$  is non-empty and does not yet introduced as a state of  $D(\mathcal{A})$ , add it to set of states of  $D(\mathcal{A})$ .
- The final states of  $D(\mathcal{A})$  are labelled with the sets containing at least one final state of  $\mathcal{A}$ .

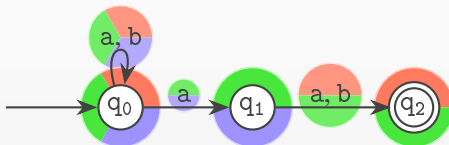
---

In fact, the states of  $D(\mathcal{A})$  are  $\omega$ -closures of  $\mathcal{A}$ -states, where  $\omega \in \Sigma^*$ .



# Subset Automaton: a Simple Example

Let us consider the following NFA with  $\gamma$ -closures of its states:

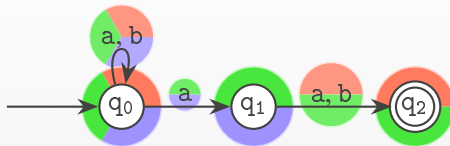


- a-closure of starting state  $\{q_0\}$  is  $\{q_0, q_1\}$ .
- b-closure of starting state  $\{q_0\}$  is the state  $\{q_0\}$  itself.
- a-closure of the state  $\{q_0, q_1\}$  is  $\{q_0, q_1, q_2\}$ .
- b-closure of the state  $\{q_0, q_1\}$  is  $\{q_0, q_2\}$ .
- a-closure of the state  $\{q_0, q_1, q_2\}$  is  $\{q_0, q_1, q_2\}$  itself, while b-closure is the state  $\{q_0, q_2\}$ .
- a-closure of the state  $\{q_0, q_2\}$  is  $\{q_0, q_1\}$ , while b-closure is  $\{q_0\}$ .

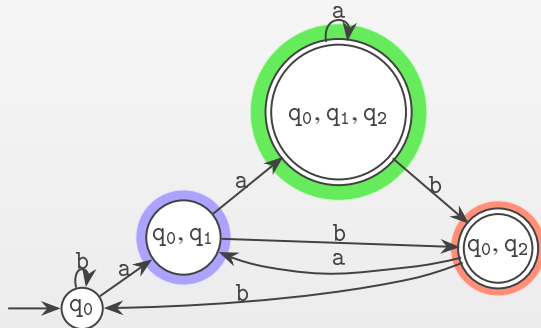


# Subset Automaton: a Simple Example

Let us consider the following NFA with  $\gamma$ -closures of its states:



Hence, its subset automaton is:





# Deterministic Finite Automata

## 👁👁 Definition

A deterministic finite automaton (DFA) is a tuple

$\mathcal{A} = \langle Q, \Sigma, q_0, F, \delta \rangle$ , where:

- $Q$  is a state set,  $\Sigma$  is a terminal alphabet;
- $\delta$  is a transition set  $\langle q_i, \gamma, q_j \rangle$ , where  $q_i, q_j \in Q$ ,  $\gamma \in \Sigma$ , and for any  $q_i$ ,  $\gamma$  there is at most one  $q_j$  such that  $q_i \xrightarrow{\gamma} q_j \in \delta$ ;
- $q_0 \in Q$  is a starting state,  $F \subseteq Q$  is a set of final states.

Language  $\mathcal{L}(\mathcal{A})$  of DFA  $\mathcal{A}$  is a set  $\{\omega \mid \exists q \in F (q_0 \xrightarrow{\omega} q)\}$ , i.e. there exists a final state that is  $\omega$ -closure of  $q_0$ .

By construction, the subset automaton has no non-determinism in the transition set:

- $\varepsilon$ -transitions are eliminated in the preliminary  $\varepsilon$ -free NFA;
- the non-singleton transition sets are processed in the subset construction.



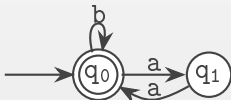
# Traps and Trims

- A **trap state** is a state s.t. any its  $\omega$ -closure is non-final.

## *Trim DFA*

- For any  $q_i, \gamma$  there is **at most one**  $q_j$  s.t.  $q_i \xrightarrow{\gamma} q_j \in \delta$ ;
- is naturally constructed via subset technique;
- default in RoFL course, useful for most operations.

### Trim DFA example

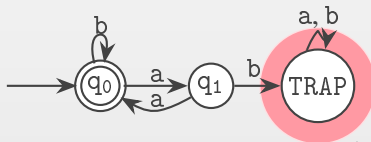


## *Complete DFA*

- For any  $q_i, \gamma$  there is **exactly one**  $q_j$  s.t.  $q_i \xrightarrow{\gamma} q_j \in \delta$ ;
- usually requires introducing **trap (sink) states**;
- useful for constructing complementation.

### DFA with the trap state

for  $\Sigma = \{a, b\}$



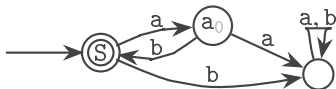
# Complementation and Traps

- By switching finality of all states in DFA  $\mathcal{A}$ , we can construct a DFA  $\mathcal{A}'$  accepting exactly the set of words that are rejected by the initial DFA, i.e.  $\mathcal{L}(\mathcal{A}') = \Sigma^* \setminus \mathcal{L}(\mathcal{A})$ .
- The language complementation requires complete DFA.

Initial trim DFA for  $(ab)^*$



Complete DFA for  $(ab)^*$

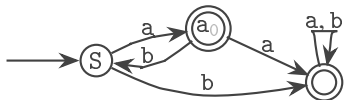


Invalid complement DFA

recognising  $a(ba)^*$



Valid complement DFA



- Without a trap state, complementation operation loses words starting with  $b$ , or containing either  $aa$  or  $bb$ .



# Back to Neural Nets

- Regular languages are closed under concatenation and union (trivially).
- Regular languages are closed under complementation (via subset construction and switching finality)  $\Rightarrow$  inhibitory signals can be modelled.

## 👁👁 Theorem

*Regular languages are closed under all boolean operations.*

*Proof:*  $\mathcal{L}_1 \cap \mathcal{L}_2 = \overline{(\overline{\mathcal{L}_1} \cup \overline{\mathcal{L}_2})}^1$ . Hence,  $k$ -signal neurons can be modelled as well.

More regular languages properties: subword-closed, subsequence-closed, closed wrt morphic images and inverse morphic images.

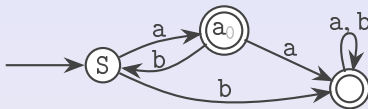
---

<sup>1</sup>More straightforward intersection construction is given in Lecture IV.



# Guessing NFA Language

Let us look at the complement NFA again:



We have guessed its language given by a regular expression:

$$a(ba)^* \mid (a|b)^*(aa|bb)(a|b)^* \mid b(a|b)^*$$

We could use another one, e.g.:

recall that this is the option operator

$$b(a|b)^* \mid a(ba)^* \underbrace{((a|bb)(a|b)^*)}_{\text{equivalent to } ((a|bb)(a|b)^* | \varepsilon)} \quad ?$$

How can we construct such expressions algorithmically rather than barely guess?



## One More Encoding: Equations

Sometimes it is convenient to gather all the right-hand sides of the rules with a same left-hand side together. Then, if we replace  $\rightarrow$  by  $=$  sign, we get an equation system determining non-terminal languages:

$$\begin{array}{llll} S \rightarrow a[q_1] & [q_2] \rightarrow a[q_3] & & S = a[q_1] \mid a[q_2] \\ S \rightarrow a[q_2] & [q_2] \rightarrow b[q_4] & \rightarrow & [q_1] = a[q_1] \mid a[q_2] \\ [q_1] \rightarrow a[q_1] & [q_3] \rightarrow \varepsilon & & [q_2] = a[q_3] \mid b[q_4] \\ [q_1] \rightarrow a[q_2] & [q_4] \rightarrow \varepsilon & & [q_3] = \varepsilon \\ & & & [q_4] = \varepsilon \end{array}$$

If there is no rule part  $[q_1] = a[q_1]$ , these languages could be found by exhaustive substitutions of the right-hand sides.

E.g.  $\mathcal{L}([q_3]) = \mathcal{L}([q_4]) = \{\varepsilon\}$ , while  
 $\mathcal{L}([q_2]) = \{a\mathcal{L}([q_3])\} \cup \{b\mathcal{L}([q_4])\} = \{a, b\}$ .

---

*How to deal with self-referring rules as  $[q_1] = a[q_1]$ ?*



# Arden's Lemma

## 👁👁 Theorem

If a language  $\mathcal{L}$  satisfies the equation  $\mathcal{L} = \mathcal{L}_1\mathcal{L} \cup \mathcal{L}_2$ , where  $\varepsilon \notin \mathcal{L}_1$ , then  $\mathcal{L} = \mathcal{L}_1^*\mathcal{L}_2$ .

*Proof:* Let us consider arbitrary  $\omega \in \mathcal{L}$ .

- If  $\omega \in \mathcal{L}_2$ , then the statement trivially holds.
- Otherwise,  $\exists \omega_1 \in \mathcal{L}_1, \omega' \in \mathcal{L} (\omega = \omega_1\omega')$ . The suffix  $\omega'$  also belongs to  $\mathcal{L}_1\mathcal{L} \cup \mathcal{L}_2$ , and  $|\omega'| < |\omega|$ , since  $\omega_1 \neq \varepsilon$ . Now we can repeat the same reasoning for  $\omega'$ , and due to finiteness of  $|\omega|$  and well-foundedness of  $(\mathbb{N}, <)$  we will eventually get  $\omega' \in \mathcal{L}_2$ .  $\square$

---

*Arden's lemma allows one to solve the equation systems in Gaussian style, via non-terminal elimination + substitution, assuming there are no chain rules in the grammar.*



## Equation Solving Example

Let us construct the language of the grammar:

$$S \rightarrow aT \quad S \rightarrow aS$$

$$T \rightarrow aT \quad T \rightarrow bT \quad T \rightarrow bF \quad F \rightarrow \varepsilon$$

First, construct the system and substitute  $F$ :

$$S = (aS) \mid (aT)$$

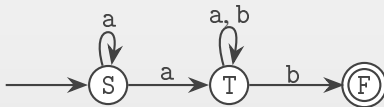
$$T = ((a \mid b)T) \mid b(\varepsilon)$$

Solve the second equation:  $T = (a \mid b)^*b$

Then substitute the solution:  $S = (aS) \mid (a(a \mid b)^*b)$ .

The resulting language is:  $S = a^*a(a \mid b)^*b$

The NFA that corresponds to the grammar is given below:





## Equation Solving Example

Let us construct the language of the grammar:

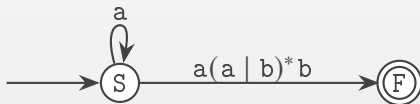
$$\begin{aligned} S &\rightarrow aT & S &\rightarrow aS \\ T &\rightarrow aT & T &\rightarrow bT & T &\rightarrow bF & F &\rightarrow \varepsilon \end{aligned}$$

The resulting language is:  $S = a^*a(a \mid b)^*b$

---

The NFA that corresponds to the grammar is given below .

After solving  $T$ -based equation and substituting  $F$  value, in fact we again constructed an NFA, whose transitions are marked with regexes.



If we assume that  $S$  is preceded by the “very starting state”  $S'$ , then  $\mathcal{L}(S)$  can be also considered as a transition in the NFA containing only  $S'$  and  $F$  states.



# Finding NFA Language

The extended NFAs allow one to use transitions marked with regexes.

---

## *State Exclusion Method*

---

- For the sake of uniformity, we introduce “the very starting state”  $S$ , having  $\varepsilon$ -transition to  $q_0$ , and “the very final state”  $T$ , having ingoing  $\varepsilon$ -transitions from  $q \in F$ . All the states except  $S$  and  $T$  are now ordinary.
- In order to exclude the state  $q$  s.t.  $q \xrightarrow{\tau} q$ , for all pairs  $q_A, q_B$ , where  $q_A \xrightarrow{\Phi} q$ ,  $q \xrightarrow{\Psi} q_B$ , add the transition  $q_A \xrightarrow{\Phi(\tau)^*\Psi} q_B$ , then we can delete  $q$ .
- When only  $S$  and  $T$  are left, where  $S \xrightarrow{\rho} T$ , the expression  $\rho$  is the regex equivalent to the NFA.



# From NFA to Regex: There and Back Again

- Given an NFA, it can be modified to DFA  $\Rightarrow$  linear-time parsing is straightforward.
- Given a regex, there is no known technique to make it deterministic<sup>2</sup>.

---

*How can we program a conversion of a regular expression to an NFA recognising the same language?*

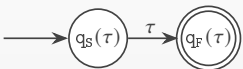
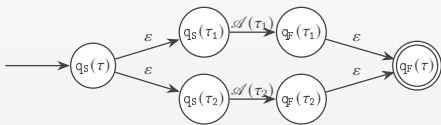
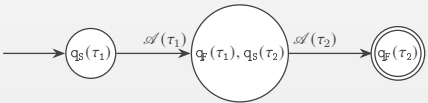
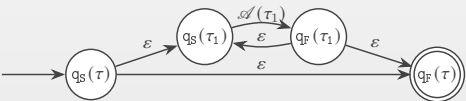
---

<sup>2</sup>Some regular languages never can be expressed by those, e.g.  $\{\omega \mid \omega[|\omega| - 2] = a\}$



# Construction-by-Definition: Thompson NFA

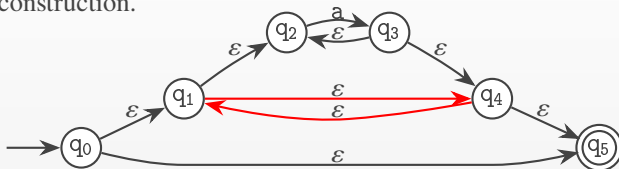
Any regular expression  $\tau$  has a recursive structure. Let us use this structure to model the corresponding NFA.

- $\tau = \gamma, \gamma \in \Sigma \Rightarrow \mathcal{A}(\tau)$  is
 
- $\tau = \tau_1 \mid \tau_2 \Rightarrow \mathcal{A}(\tau)$  is
 
- $\tau = \tau_1 \tau_2 \Rightarrow \mathcal{A}(\tau)$  is
 
- $\tau = \tau_1^* \Rightarrow \mathcal{A}(\tau)$  is
 



# High Ambiguity of Thompson NFA

The Thompson NFA for expression  $(a^*)^*$  contains a loop following silent actions  $\Rightarrow$  DFS-based parsing is to be augmented by an analogue of  $\varepsilon$ -closure construction.



Thompson's parsing algorithm constructs all closures dynamically: given a string to parse  $\omega$ , its configurations are  $\langle \omega_2, Q(\omega_1) \rangle$ , where  $Q(\omega_1)$  is a  $\omega_1$ -closure set of the  $\varepsilon$ -closed starting state  $\{q_0\}$ , and  $\omega = \omega_1 \omega_2$ .

*An example: parse trace by aa of the NFA for  $(a^*)^*$ .*

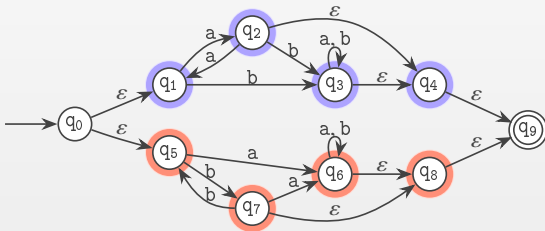
Step	0	1	2
$\omega_2$	aa	a	$\varepsilon$
Closure	$\{q_0, q_1, q_2, q_4, q_5\}$	$\{q_3, q_2, q_4, q_1, q_5\}$	$\{q_3, q_2, q_4, q_1, q_5\}$



# High Flexibility of Thompson NFA

- Follows regex structure precisely, can be divided to modules.
- Works fine for any module having exactly one final state  $\Rightarrow$  *extended NFA* construction is available for free.

Let us introduce the complementation operation  $\sim$ , implemented by determinisation and switching finality, and construct an extended Thompson NFA for  $\sim((aa)^*) \mid \sim((bb)^*)$ .



Complementation submodules are highlighted. States  $q_4$  and  $q_8$  are introduced in order to make final states of the submodules unique.

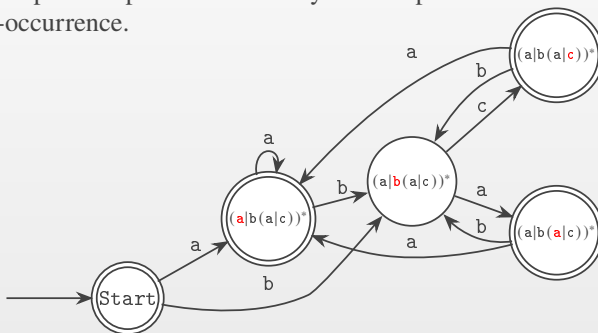


# Term-Driven Parsing by Regex

Let us consider regex  $\tau = (a|b(a|c))^*$ .

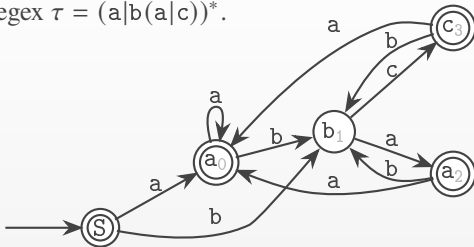
Although Thompson NFA for  $\tau$  is non-deterministic, we can always determine parse trace according to  $\tau$  as follows:

- if a string starts by a, then we follow the first alternative under the iteration and repeat the iteration;
- if a string starts by b, then we follow the second alternative and choose the next parse step deterministically with respect to the letter following the b-occurrence.



# Term-Driven Parsing by Regex

Let us consider regex  $\tau = (a|b(a|c))^*$ .



Distinct occurrences of same letter  $a$  in  $\tau$  correspond to distinct parse positions, hence, the parse path considers *linearised* regex terms, marked by their positions. Hence,  $\text{Linearize}((a | b(a|c))^*)$  is  $(a_0 | b_1(a_2|c_3))^*$ . Now the natural parsing position in  $(a_0 | b_1(a_2|c_3))^*$  is determined by the corresponding linearised letter read last.

*Given  $\tau \in \mathcal{RE}$ , its linearisation  $\text{Linearize}(\tau)$  can be constructed by subscripting letters in  $\tau$  with their positions (counting from 0).*





# Position NFA aka Glushkov NFA

We are ready to construct an NFA for the term-driven parsing.

---

## *Glushkov NFA for Regular Expression $\tau$*

---

- Construct  $\tau' = \text{Linearize}(\tau)$ .
  - Construct sets  $\text{First}(\tau')$  and  $\text{Last}(\tau')$  — linearised letters that can start and end the expression  $\tau'$ .
  - Construct follow-set  $\text{Follow}(\tau')$  of pairs  $\langle \gamma_i, \gamma_j \rangle$  of linearised letters s.t.  $\gamma_j$  can follow  $\gamma_i$  in  $\tau'$ .
  - Add the starting state  $S$  and states labelled by letters of  $\tau'$ . Make  $S$  final, if  $\tau$  accepts  $\varepsilon$ ; make all the states from  $\text{Last}(\tau')$  final.
  - Given  $\gamma_i \in \text{First}(\tau')$ , add a transition  $S \xrightarrow{\gamma} [\gamma_i]$  to  $\delta$  of  $\text{Glushkov}(\tau)$ .
  - Given  $\langle \alpha_i, \beta_j \rangle \in \text{Follow}(\tau')$ , add a transition  $[\alpha_i] \xrightarrow{\beta} [\beta_j]$  to  $\delta$  of  $\text{Glushkov}(\tau)$ .
- 



# History of Glushkov Automaton

1960s–1980s

Introduced by V.M. Glushkov in 1961.  
Till 1990s, mainly of theoretical interest.

1990s

Formalisation of SGML unambiguity notion  
in terms of Glushkov NFA (Wood&Bruggemann).

00s

Glushkov NFA is proved to generate (or be generated by)  
well-known NFA models by equivalence or simulation relations.  
Breaking fast implementation of Glushkov-NFA-based approach  
in RE2 library (still 20× faster than DFA-based Go regex library)



# History of Glushkov Automaton

1990s

Formalisation of SGML unambiguity notion in terms of Glushkov NFA (Wood&Bruggemann).

00s

Glushkov NFA is proved to generate (or be generated by) well-known NFA models by equivalence or simulation relations. Breaking fast implementation of Glushkov-NFA-based approach in RE2 library (still 20× faster than DFA-based Go regex library)

10-20s

“The Mother of All Automata” (Broda, 2017)  
Development of Glushkov models for extended set of regex operations.

