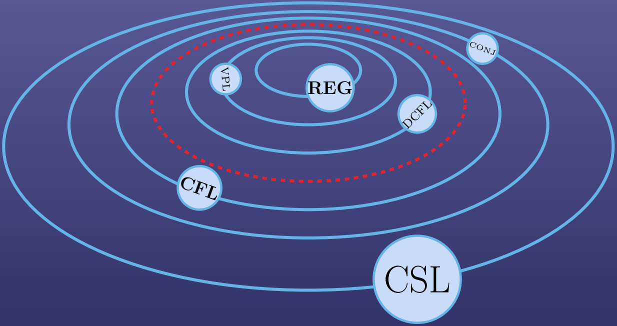
Bauman Moscow State University
Th. Computer Science Dept.

Finite State Machines and Regular Expressions

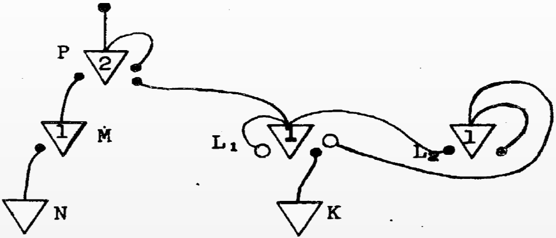


Antonina Nepeivoda
a_nevod@mail.ru

Lecture Outline

1. **Basic Notions**
2. **Closures and Determinisation**
   - ε-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;

 — 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 Σ, a regular expression is either a letter in Σ, ε, or a result of following operations, where r_1, r_2 are regular expressions:

- $r_1 \mid r_2$ — *union (alternation).* $\mathscr{L}(r_1 \mid r_2) = \mathscr{L}(r_1) \cup \mathscr{L}(r_2)$;
- $r_1\, r_2$ — *concatenation (sequencing).* $\mathscr{L}(r_1\, r_2) = \{\omega_1\omega_2 \mid \omega_1 \in \mathscr{L}(r_1)\ \&\ \omega_2 \in \mathscr{L}(r_2)\}$;
- $(r_1)^*$ — *iteration (0 or more concatenations of r_1 with itself);* $\mathscr{L}\big((r_1)^*\big) = \{\varepsilon\} \bigcup_{i=1}^{\infty} \mathscr{L}(r_1)$.

Syntactic Sugar

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

Regular Expressions by Kleene

👀 Academic Definition

Given alphabet Σ, a regular expression is either a letter in Σ, ε, or a result of following operations, where r_1, r_2 are regular expressions:

- $r_1 \mid r_2$ — *union (alternation).* $\mathscr{L}(r_1 \mid r_2) = \mathscr{L}(r_1) \cup \mathscr{L}(r_2)$;
- $r_1\, r_2$ — *concatenation (sequencing).*
  $\mathscr{L}(r_1\, r_2) = \left\{\omega_1\omega_2 \mid \omega_1 \in \mathscr{L}(r_1) \;\&\; \omega_2 \in \mathscr{L}(r_2)\right\}$;
- $(r_1)^*$ — *iteration (0 or more concatenations of r_1 with itself);*
  $\mathscr{L}\left((r_1)^*\right) = \{\varepsilon\} \bigcup_{i=1}^{\infty} \mathscr{L}(r_1)$.

Priorities: star > concatenation > union.

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

Terminological Clash

Academic regexes

- $|, \cdot, ^*$ (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 $\mathscr{A} = \langle Q, \Sigma, q_0, F, \delta \rangle$*, where:*

- Q — *state set;*
- Σ — *terminal alphabet;*
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ — *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

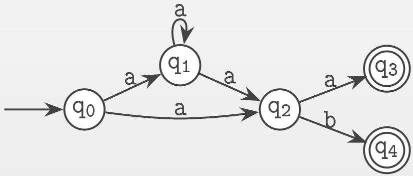
- Classical works (Kleene, Brzozowski): multiple NFA starting states are allowed.
- Modern formal language theory: the unique starting state in NFA is assumed.
- Equivalent (we can add an unique starting state with ε-transitions to the multiple states), but confusing (e.g. in Brzozowski minimisation).

Encoding into Grammars

Observation

- Transition $q_1 \xrightarrow{a} q_2$ can be seen as a rewriting rule $[q_1] \to a[q_2]$, 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 $[q_F] \to \varepsilon$.

Automaton:



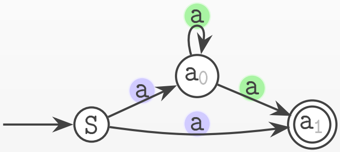
Grammar:

$$S \to \mathtt{a}[q_1] \qquad [q_2] \to \mathtt{a}[q_3]$$
$$S \to \mathtt{a}[q_2] \qquad [q_2] \to \mathtt{b}[q_4]$$
$$[q_1] \to \mathtt{a}[q_1] \qquad [q_3] \to \varepsilon$$
$$[q_1] \to \mathtt{a}[q_2] \qquad [q_4] \to \varepsilon$$

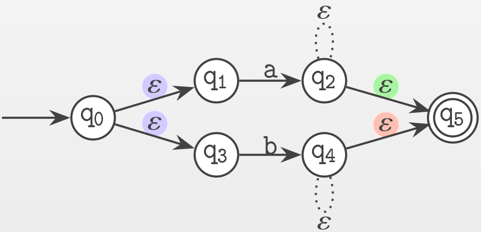
We rename the starting nonterminal $[q_0]$ to S, for uniformity.

Sources of Non-Determinism in an NFA

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



- ε-transitions (so-called silent actions).

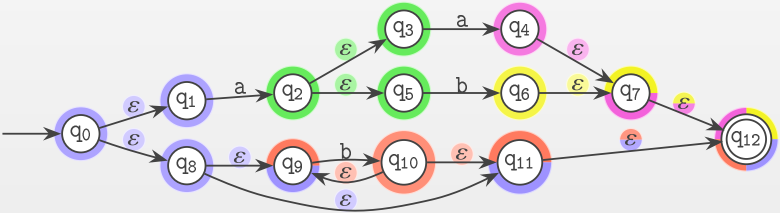


Closures

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

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

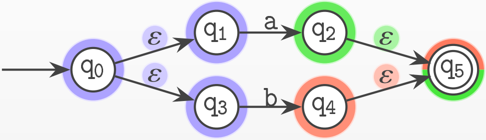
Special case: ε-closures: sets of states reachable via doing nothing.



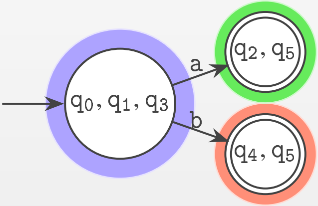
Given such closures, they can be considered as new «states».

Simple Example of ε-Removal

An NFA $\mathscr{A}$ with the ε-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.

ε-Closures and Chain Rules

- *Any transition $q_i \xrightarrow{\varepsilon} q_j$ corresponds to* ***a chain rule*** *$[q_i] \to [q_j]$ in the corresponding grammar G.*
- *state ε-closure is a closure set of the corresponding non-terminal N:*
  $C(N) = \left\{N_i \mid \exists N'_1, \dots N'_k \big(N \to N'_1 \,\&\, \cdots \,\&\, N'_k \to N_i\big)\right\}$
  I.e. $\langle N, N_i \rangle$ are pairs in ***a transitive closure*** *$\to_c^+$ of the relation $\to_c$:*
  $A_i \to_c A_j \Leftrightarrow (A_i \to A_j \in G)$.
- *Before removing all chain rules, for every $N' \in C(N)$ and a non-chain rule $N' \to \Phi$, we add the transition $N \to \Phi$ to the set of grammar rules. Exactly as in the ε-closure algorithm for NFA.*

Initial grammar:

$S \to Q_1$ $S \to Q_3$ $Q_1 \to \mathtt{a}Q_2$

$Q_3 \to \mathtt{b}Q_4$ $Q_2 \to Q_5$ $Q_4 \to Q_5$

$Q_5 \to \varepsilon$

After removing chain rules:

$S \to \mathtt{a}Q_2$ $S \to \mathtt{b}Q_4$

$Q_2 \to \varepsilon$ $Q_4 \to \varepsilon$

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

ω-Closures and Subset Construction

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

Subset Automaton Construction

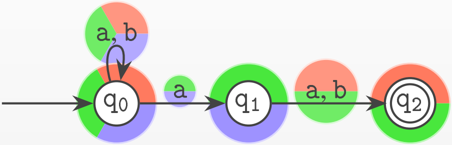
Let an ε-free NFA $\mathscr{A}$ be given. Its **subset automaton** $D(\mathscr{A})$ can be constructed as follows.

- q_0 becomes the starting state $\{q_0\}$ of $D(\mathscr{A})$.
- Given a state M in $D(\mathscr{A})$ 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_γ is non-empty and does not yet introduced as a state of $D(\mathscr{A})$, add it to set of states of $D(\mathscr{A})$.
- The final states of $D(\mathscr{A})$ are labelled with the sets containing at least one final state of $\mathscr{A}$.

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

Subset Automaton: a Simple Example

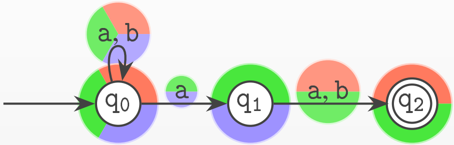
Let us consider the following NFA with γ-closures of its states:



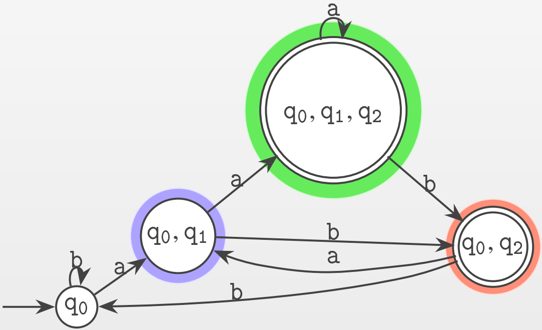
- $\mathtt{a}$-closure of starting state $\{q_0\}$ is $\{q_0, q_1\}$.
- $\mathtt{b}$-closure of starting state $\{q_0\}$ is the state $\{q_0\}$ itself.
- $\mathtt{a}$-closure of the state $\{q_0, q_1\}$ is $\{q_0, q_1, q_2\}$.
- $\mathtt{b}$-closure of the state $\{q_0, q_1\}$ is $\{q_0, q_2\}$.
- $\mathtt{a}$-closure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

Subset Automaton: a Simple Example

Let us consider the following NFA with γ-closures of its states:



Hence, its subset automaton is:



Deterministic Finite Automata

Definition

A deterministic finite automaton (DFA) is a tuple $\mathscr{A} = \langle Q, \Sigma, q_0, F, \delta \rangle$, *where:*

- Q *is a state set,* Σ *is a terminal alphabet;*
- δ *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, γ *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 $\mathscr{L}(\mathscr{A})$ *of DFA* $\mathscr{A}$ *is a set* $\left\{\omega \mid \exists q \in F(q_0 \xrightarrow{\omega} q)\right\}$, *i.e. there exists a final state that is* ω*-closure of* q_0.

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

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

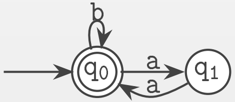
Traps and Trims

- **A trap state** is a state s.t. any its ω-closure is non-final.

Trim DFA

- For any q_i, γ 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.

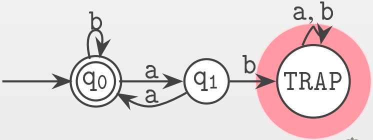
Trim DFA example



Complete DFA

- For any q_i, γ 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 $\Sigma = \{\mathtt{a}, \mathtt{b}\}$



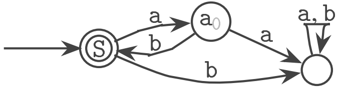
Complementation and Traps

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

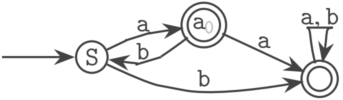
Initial trim DFA for $(\mathtt{ab})^*$

Invalid complement DFA recognising $\mathtt{a}(\mathtt{ba})^*$

Complete DFA for $(\mathtt{ab})^*$



Valid compement 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: $\mathscr{L}_1 \cap \mathscr{L}_2 = \overline{(\overline{\mathscr{L}_1} \cup \overline{\mathscr{L}_2})}$[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.

[1] More straightforward intersection construction is given in Lecture IV.

What a DFA can Tell about its Language?

Given any DFA of a language $\mathscr{L}$, one can establish:

- whether $\mathscr{L}$ has forbidden prefixes — i.e. words that never can start any word from $\mathscr{L}$. Existence of forbidden prefixes is equivalent to existence of trap states in the DFA.
- whether $\mathscr{L}$ is finite — the finiteness holds iff there are no loops in the non-trap part of the DFA.

However, if a finite $\mathscr{L}$ contains at least two words ω_1, ω_2 s.t. $\forall \upsilon(\omega_1 \neq \omega_2\upsilon \ \& \ \omega_2 \neq \omega_1\upsilon)$, there is more than one trim DFA recognizing it. Moreover, if $\mathscr{L}$ is infinite, then there is infinite set of DFAs recognizing $\mathscr{L}$.

In Lecture III, we will give a construction of DFA that can be considered as an unique encoding of the regular language recognized by the DFA.

What a DFA can Tell about its Language?

If $\mathscr{L}$ is infinite, then for every $n \in \mathbb{N}$, there is an infinite set of the words with the length exceeding n: i.e. $\{\omega \mid |\omega| > n\}$ is infinite.

By the pigeonhole principle, given a DFA $\mathscr{A}$ with N states and any word $\omega = a_1 ... a_M$ of the length at least N recognized by $\mathscr{A}$, the sequence of states along the path $q_0 \xrightarrow{\omega} q_F$ contains at least one intermediate state q_i twice or more.

Consider $\underbrace{a_1 a_2 \dots a_{k-1}}_{\text{path from } q_0 \text{ to } q_i} \overbrace{a_k \dots a_{k+m}}^{\text{loop from } q_i \text{ to } q_i} \underbrace{a_{k+m+1} \dots a_M}_{\text{path from } q_i \text{ to } q_F} \in \mathscr{L}(\mathscr{A})$

Excluding or repeating the $a_k ... a_{k+m}$ substring, we still get words in $\mathscr{L}(\mathscr{A})$:

- $a_1 a_2 \dots a_{k-1} \overbrace{\cancel{a_k \dots a_{k+m}}}^{\text{no loop from } q_i \text{ to } q_i} a_{k+m+1} \dots a_M \in \mathscr{L}(\mathscr{A})$
- $a_1 a_2 \dots a_{k-1} \overbrace{(a_k \dots a_{k+m})^j}^{j \text{ loops from } q_i \text{ to } q_i} a_{k+m+1} \dots a_M \in \mathscr{L}(\mathscr{A})$

Moreover, again by pigeonhole principle, if q_i is the first repeated state along the trace, then $k + m \leq N$.

Pigeonhole Principle and Pumping

Let p be the number of states in an ε-free finite automaton $\mathscr{A}$ recognizing infinite language $\mathscr{L}$. Then any word ω s.t. $|\omega| \geq p$, can be decomposed as $\omega = \underbrace{\omega_1}_{\text{before any loop}} \overbrace{\omega_2}^{\text{inside a loop}} \omega_3$, moreover:

1. $|\omega_2| > 0$, since there are no ε-transitions.
2. $|\omega_1\omega_2| \leq p$, by the pigeonhole principle for the states of $\mathscr{A}$.
3. $\forall j \in \mathbb{N}\big(\omega_1\omega_2^j\omega_3 \in \mathscr{L}(\mathscr{A})\big)$, as the loop can be entered arbitrarily many times ($j = 0$ is also valid).

If $\mathscr{L}$ is regular, the minimal p value for which any $\omega \in \mathscr{L}$, s.t. $|\omega| \geq p$, admits the given decomposition satisfying 1, 2, 3, is called <u>pumping length</u> of the language $\mathscr{L}$.

All Regular Languages Can Be Pumped

Ordinary Pumping Lemma for Regular Languages

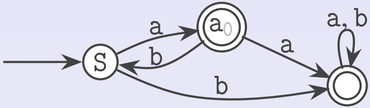
If $\mathscr{L}$ is a regular language, then there exists such a $p \in \mathbb{N}$, that

$$\forall \omega \in \mathscr{L}\Big(|\omega| \geq p \Rightarrow \exists \omega_1, \omega_2, \omega_3\big(\omega = \omega_1\omega_2\omega_3 \;\&\; |\omega_2| > 0 \;\&\; |\omega_1\omega_2| \leq p \;\&\; \forall j \in \mathbb{N}(\omega_1\omega_2^j\omega_3 \in \mathscr{L})\big)\Big).$$

- a non-regular language can have a finite (prefix) pumping length, e.g. $\{\omega\omega\upsilon \mid \omega \neq \varepsilon\}$;
- the pumping length can be less than the number of states in any ε-free NFA or DFA recognizing the language, e.g. as in $\{\omega \mid \omega \in \{a,b\}^* \;\&\; \omega$ starts and ends with the same letter$\}$.

Guessing NFA Language

Let us look at the complement NFA again:



We have guessed its language given by a regular expression:

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

We could use another one, e.g.:

$$\mathtt{b(a|b)^* \mid a(ba)^*} \underbrace{\mathtt{((a|bb)(a|b)^*)}\ \overbrace{\boxed{?}}^{\text{recall that this is the option operator}}}_{\text{equivalent to } \mathtt{((a|bb)(a|b)^*|}\varepsilon\mathtt{)}}$$

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:

$$\begin{array}{ll} S \rightarrow \mathtt{a}[q_1] & [q_2] \rightarrow \mathtt{a}[q_3] \\ S \rightarrow \mathtt{a}[q_2] & [q_2] \rightarrow \mathtt{b}[q_4] \\ [q_1] \rightarrow \mathtt{a}[q_1] & [q_3] \rightarrow \varepsilon \\ [q_1] \rightarrow \mathtt{a}[q_2] & [q_4] \rightarrow \varepsilon \end{array} \quad \longrightarrow \quad \begin{array}{l} S = \mathtt{a}[q_1] \mid \mathtt{a}[q_2] \\ [q_1] = \mathtt{a}[q_1] \mid \mathtt{a}[q_2] \\ [q_2] = \mathtt{a}[q_3] \mid \mathtt{b}[q_4] \\ [q_3] = \varepsilon \\ [q_4] = \varepsilon \end{array}$$

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

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

How to deal with self-referring rules as $[q_1] = \mathtt{a}[q_1]$?

Arden's Lemma

Theorem

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

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

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

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:

$$S \to aT \quad S \to aS$$

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

First, construct the system and substitute F:

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

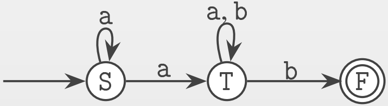
$$T = ((a \mid b)T) \mid b(\varepsilon)$$

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

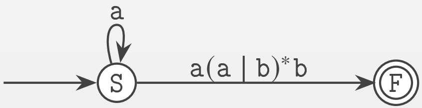
Let us construct the language of the grammar:

$$S \to aT \quad S \to aS$$

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

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 $\mathscr{L}(S)$ 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 ε-transition to q_0, and "the very final state" T, having ingoing ε-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 ρ 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[2].

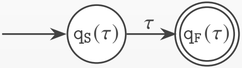
How can we program a conversion of a regular expression to an NFA recognising the same language?

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

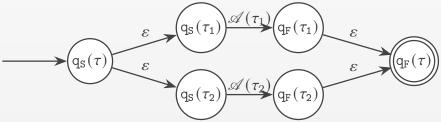
Construction-by-Definition: Thompson NFA

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

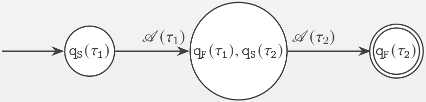
- $\tau = \gamma, \gamma \in \Sigma \Rightarrow \mathscr{A}(\tau)$ is



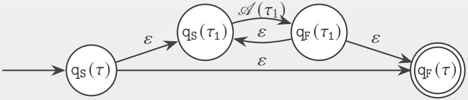
- $\tau = \tau_1 \mid \tau_2 \Rightarrow \mathscr{A}(\tau)$ is



- $\tau = \tau_1 \tau_2 \Rightarrow \mathscr{A}(\tau)$ is

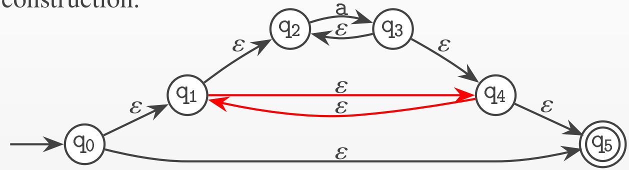


- $\tau = {\tau_1}^* \Rightarrow \mathscr{A}(\tau)$ is



High Ambiguity of Thompson NFA

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



Thompson's parsing algorithm constructs all closures dynamically: given a string to parse ω, its configurations are $\langle \omega_2, Q(\omega_1) \rangle$, where $Q(\omega_1)$ is a ω_1-closure set of the ε-closed starting state $\{q_0\}$, and $\omega = \omega_1\omega_2$.

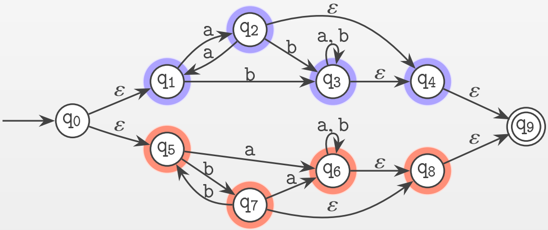
An example: parse trace by `aa` *of the NFA for* $(\mathtt{a}^*)^*$.

Step	*0*	*1*	*2*
ω_2	`aa`	`a`	ε
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 $\sim((\mathtt{aa})^*) \mid \sim((\mathtt{bb})^*)$.

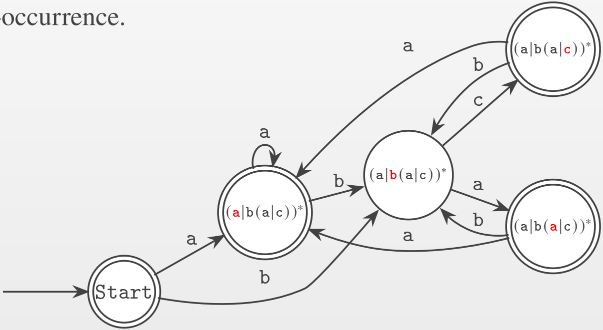


Complementation submodules are highlighted. States $\mathtt{q}_4$ and $\mathtt{q}_8$ are introduced in order to make final states of the submodules unique.

Term-Driven Parsing by Regex

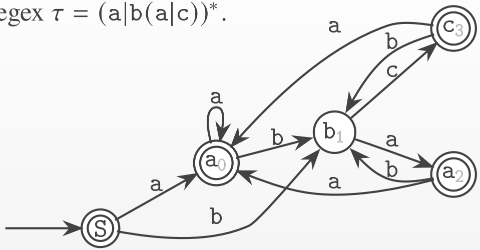
Let us consider regex $\tau = (\texttt{a}|\texttt{b}(\texttt{a}|\texttt{c}))^*$.
Although Thompson NFA for τ is non-deterministic, we can always determine parse trace according to τ as follows:

- 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 $\tau = (\mathtt{a}|\mathtt{b}(\mathtt{a}|\mathtt{c}))^*$.



Distinct occurrences of same letter $\mathtt{a}$ in τ correspond to distinct parse positions, hence, the parse path considers *linearised* regex terms, marked by their positions. Hence, $\mathtt{Linearize}((\mathtt{a} \mid \mathtt{b}(\mathtt{a}|\mathtt{c}))^*)$ is $(\mathtt{a}_0 \mid \mathtt{b}_1(\mathtt{a}_2|\mathtt{c}_3))^*$. Now the natural parsing position in $(\mathtt{a}_0 \mid \mathtt{b}_1(\mathtt{a}_2|\mathtt{c}_3))^*$ is determined by the corresponding linearised letter read last.

Given $\tau \in \mathcal{RE}$, its linearisation $\mathtt{Linearize}(\tau)$ can be constructed by subscripting letters in τ 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 τ

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

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.