Automata and Regular Expressions

Algorithmics study and revision notes

Jaak Vilo

2026-08-22

Contents

1 Descriptions and machines for the same language

A regular expression describes a set of strings compositionally. A finite automaton recognises such a set by updating a finite state while reading input. The equivalence between these views is useful in both

directions: expressions are convenient to write, while automata make execution and complexity explicit.

This note concerns **classical regular expressions**. Practical libraries add syntax and execution policies that must be analysed separately.

2 Learning goals

After studying this note, you should be able to:

- define alphabets, words, languages, regular expressions, DFAs, and NFAs;
- distinguish the empty language from the empty word;
- simulate an ε-NFA;
- build a Thompson NFA from a parsed regular expression;
- determinise an NFA by subset construction and prove the construction correct;
- distinguish whole-string recognition from substring search;
- minimise a DFA by partition refinement; and
- explain why regex, NFA, and DFA define the same class of languages.

3 Prerequisites

You should know sets, functions, graphs, breadth-first search, and basic string terminology. The exact-matching note gives concrete examples of prefix automata and failure links, but it is not required here.

4 Words, languages, and regular expressions

An **alphabet** Σ is a finite set of symbols. A **word** is a finite sequence over Σ; ε is the unique word of length zero. The set of all finite words is Σ^*. A **language** is any subset of Σ^*, while $\emptyset$ is the language containing no words. Thus $\{\varepsilon\} \neq \emptyset$.

Classical regular expressions are defined recursively:

- $\emptyset$, ε, and every symbol $a \in \Sigma$ are expressions;
- if R and S are expressions, so are union $R|S$, concatenation RS, and Kleene star R^*.

Their languages satisfy

$$\begin{aligned} L(R|S) &= L(R) \cup L(S), \\ L(RS) &= \{xy : x \in L(R), y \in L(S)\}, \\ L(R^*) &= \bigcup_{k \geq 0} L(R)^k. \end{aligned}$$

Star binds more tightly than concatenation, which binds more tightly than union. Parenthesise whenever structure could be unclear. Operators such as $R^+ = RR^*$ and $R? = (R|\varepsilon)$, character classes, and bounded repetitions are regular syntactic sugar. Backreferences are not: they can describe nonregular dependencies.

5 Deterministic finite automata

A deterministic finite automaton (DFA) is a tuple

$$M = (Q, \Sigma, \delta, q_0, F),$$

where Q is a finite state set, $\delta : Q \times \Sigma \to Q$ is a total transition function, q_0 is the start state, and $F \subseteq Q$ is the accepting set. Extend δ from symbols to words by repeated application. Word w is accepted exactly when $\delta^*(q_0, w) \in F$.

An explicit transition table scans an n-symbol word in $\Theta(n)$ time while retaining one current state. The automaton itself can require $\Theta(|Q||\Sigma|)$ table entries; sparse maps trade space for lookup cost. A missing transition in a diagram means an edge to a nonaccepting dead state if the DFA is to remain total.

6 Nondeterministic finite automata

An ε-NFA has transition function

$$\Delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q).$$

It conceptually follows every possible path. This is not random choice: after each input prefix, the machine state is the set of all reachable NFA states. Define $E(S)$, the **epsilon closure** of set S, as all states reachable from S using only ε edges.

```
simulate_nfa(M, w):
    current = epsilon_closure({start})
    for symbol a in w:
        next = empty set
        for q in current:
            next = next union Delta(q, a)
        current = epsilon_closure(next)
    accept iff current intersects F
```

With adjacency lists, direct simulation takes $O(n(|Q| + |E|))$ worst-case time and $O(|Q|)$ active-state space. Bit sets can process many states per word. Acceptance requires at least one reachable accepting state, not every accepting state.

7 Thompson construction

First parse a regular expression into a syntax tree. Thompson construction recursively gives every subexpression a fragment with one entry and one exit:

- symbol a: entry $\xrightarrow{a}$ exit;
- ε: entry $\xrightarrow{\varepsilon}$ exit;
- $R|S$: a new entry epsilon-branches to both fragments, whose exits epsilon-join at a new exit;
- RS: connect R's exit to S's entry by epsilon;
- R^*: a new entry may bypass R or enter it, and R's exit may repeat or leave.

Every rule preserves the language described by its syntax-tree node. The construction produces $O(r)$ states and edges for an expression of size r.

8 Worked construction: the expression `a*b`

For `a*b`, number a Thompson NFA as follows:

```
0 --epsilon--> 1,2
2 --a--> 3
3 --epsilon--> 1,2
1 --epsilon--> 4
4 --b--> 5
```

State 0 is initial and state 5 is accepting. The bypass edge permits zero `a`s; the return edge permits more; the final `b` is mandatory. This complete example accepts `b`, `ab`, and `aaab`, but rejects `a` and `ba`.

9 Subset construction

Each DFA state can represent one reachable NFA-state set. Start with $E(\{q_0\})$. From subset S on symbol a, take every a transition from S and epsilon-close the union. A subset is accepting exactly when it intersects the NFA accepting set.

```
determinise(NFA):
    start_set = epsilon_closure({NFA.start})
    queue = [start_set]
    while queue not empty:
        S = remove_front(queue)
        for a in alphabet:
            U = epsilon_closure(move(S, a))
            record transition S --a--> U
            if U is new: add U to queue
```

For the `a*b` NFA above, reachable subsets are:

DFA name	NFA states	on `a`	on `b`	accepting?
A	`{0,1,2,4}`	B	C	no
B	`{1,2,3,4}`	B	C	no
C	`{5}`	D	D	yes
D	`{}`	D	D	no

The central invariant is: after reading any prefix x, the DFA subset is exactly the NFA states reachable after reading x, including epsilon moves. It is true for ε by the start closure. One symbol step preserves it by the definition of `move` and closure, so induction proves language equivalence.

An r-state NFA has at most 2^r subsets. Only reachable subsets are constructed, but exponential growth is possible. Determinisation moves work from repeated set simulation into preprocessing and storage.

10 Recognition versus search in text

Recognition asks whether an entire word belongs to $L(R)$. Search asks which factors of a text belong. To report endpoints of nonempty matches with an NFA, inject a fresh start closure before each input symbol; handle zero-length matches separately when $\varepsilon \in L(R)$:

```
active = empty set
for position j and symbol a:
    active = epsilon_closure(move(active union start_closure, a))
    if active intersects F: report an occurrence ending at j
```

This simulates all starts without rescanning the text. To report start positions as well, active states must retain origin information; the extra work can be proportional to the number of live origins and reported matches. If $\varepsilon \in L(R)$, there is a zero-length match at every text boundary and that policy must be stated.

When several matches overlap, a library must also choose a **match policy**: report all endpoints, the leftmost-first alternative, the leftmost-longest alternative, or some non-overlapping sequence. Language recognition alone does not determine that policy, capture-group values, or whether offsets count bytes, code units, or Unicode scalar values.

Equivalently, recognition by $\Sigma^* R$ detects whether a match ends at the final position, and a streaming version can report accepting positions. It does not by itself recover every start.

Many programming-language regex engines use prioritised backtracking to implement capture groups and match policies. Backreferences can go beyond regular languages, and careless backtracking can take exponential time even for some regular-looking expressions. Classical NFA/DFA bounds apply only to the stated automaton model.

11 DFA minimisation

Two DFA states are equivalent when every possible continuation either leads both to acceptance or both to rejection. Equivalent states may be merged.

1. Remove states unreachable from the start.

2. Partition the rest into accepting and nonaccepting blocks.
3. Repeatedly split a block when two states have transitions, on some symbol, into different blocks.
4. Make each final block one state of the quotient DFA.

For the determinised `a*b` machine, A and B begin in the same nonaccepting block. On both symbols they enter the same blocks: `a` leads to B and `b` to accepting C. No continuation distinguishes them, so they merge. Dead state D remains separate because, for example, continuation `b` is accepted from A/B but not from D. The minimal complete DFA therefore has three states.

Partition refinement is correct because a split records a distinguishing first symbol followed by a previously known distinguishing continuation. When refinement stops, states in one block respond identically to every continuation. Efficient algorithms such as Hopcroft's run in $O(|\Sigma||Q| \log |Q|)$ time; simpler table-filling methods are often adequate for hand examples.

12 From an automaton back to a regular expression

State elimination completes the other direction of the equivalence. Label edges by regular expressions, adding union when several labels share endpoints. When removing state q, update every remaining pair (i, j) by

$$R_{ij} \leftarrow R_{ij} \mid R_{iq}(R_{qq})^* R_{qj}.$$

The added term represents paths that enter q, loop there any number of times, and leave. After all internal states are removed, the start-to-accept label is an equivalent regex. Removing the dead state from the minimal `a*b` DFA leaves an `a` loop followed by a `b` edge, yielding a^*b.

Together, Thompson construction, subset construction, and state elimination establish that regex, NFA, and DFA describe exactly the regular languages.

13 Time and representation trade-offs

Stage	Typical size/time	Important caveat
Thompson NFA	$O(r)$ states/edges	epsilon closure needed
NFA scan	$O(n(\|Q\| + \|E\|))$ direct	bit sets can reduce constants
Determinisation	up to $2^{\|Q\|}$ subsets	only reachable subsets built
DFA scan	$O(n)$	transition storage depends on alphabet
Minimisation	$O(\|\Sigma\|\|Q\| \log \|Q\|)$ efficient	remove unreachable states first

14 Connections

- KMP and Aho–Corasick are specialised finite automata with compact failure representations.
- Shift-And packs an active NFA-state set into machine-word bits.
- Approximate matching can layer automaton states by the number of errors used.
- Tries and suffix indexes also represent sets of prefixes or factors, but with different preprocessing targets.

15 Common mistakes

- Confusing ε with $\emptyset$.
- Treating NFA choices as probabilistic.
- Forgetting epsilon closure before the first symbol or after a move.
- Marking a DFA subset accepting only when all its NFA states accept.
- Constructing all 2^r subsets before checking reachability.
- Omitting the DFA dead state while still calling the transition function total.
- Applying finite-automaton guarantees to backreferences or a backtracking implementation without analysis.

16 Self-check

1. Give the language of `(a|b)*abb` and distinguish recognition from finding it inside a text.
2. Compute the epsilon closure of each singleton state in the `a*b` NFA.
3. Reconstruct every row of the subset table without looking at it.
4. State and prove the subset-construction invariant for one more symbol.
5. Why are DFA states A and B equivalent, but A and D distinguishable?
6. What extra information is needed to report every regex match start?

17 Revision summary

- Regular expressions define languages compositionally; finite automata recognise them operationally.
- Thompson construction gives a linear-size epsilon-NFA.
- Subset construction tracks exactly the NFA states reachable after each prefix.
- Determinisation can be exponential, but a fixed DFA scans linearly.
- Search differs from recognition because candidate starts must be introduced and possibly retained.
- Partition refinement merges states that no continuation can distinguish.
- State elimination converts finite automata back to regular expressions.

18 Implementations and hands-on exploration

- **Python automata:** `automata-lib` constructs and visualises explicit DFAs, NFAs, and GNFAs and supports regex-to-NFA and NFA-to-DFA conversion. Its regex syntax is a teaching-oriented regular subset, not the syntax or match policy of Python's `re` engine.
- **C++ regex engine:** RE2 searches with time linear in input length while working within a configurable memory budget, and fails gracefully if that budget is exhausted. It deliberately omits backreferences and look-around, and its leftmost match and capture semantics still need to be distinguished from whole-string recognition.
- **Rust regex engine:** `regex` gives a single `find` or `captures` search worst-case $O(mn)$ time for regex size m and byte length n by excluding backreferences and look-around. Iterating over all nonoverlapping matches can have a higher whole-iteration worst case. Unicode matching is enabled by default, while reported string spans are half-open byte offsets on UTF-8 boundaries.
- **C/C++ lexer generation:** re2c compiles regular rules to DFA-based source code. Lexer rules use longest match, then earliest rule on a tie; that policy is not the same as enumerating every regex match.

18.1 Small experiments

1. Use `automata-lib` to build an NFA for `a*b`, determinise and minimise it, and enumerate all words over `{a,b}` of length at most 4. Assert that the NFA and DFA accept exactly the same words and record each state count.
2. Time whole-string rejection of `a^n` by pattern `^(a|aa)*b$` in a backtracking engine and in RE2 or Rust `regex` as n grows. Keep syntax and anchors equivalent, and plot time against n instead of inferring complexity from one input.

19 Sources and further study

- Regular expressions and automata lecture deck, Jaak Vilo.
- (Cormen et al. 2022), finite automata and string-matching material in relevant editions.

References

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2022. *Introduction to Algorithms*. 4th ed. MIT Press.