Chapter 1

Basics of formal languages

1.2 Strings and languages

1.2.1 Strings

Alphabet: a collection of indivisible symbols, from which the strings are built. Typically finite. Usually denoted by the letter Σ.

Examples of alphabets:

- The Latin alphabet with a space: $\{\texttt{a}, \ldots, \texttt{z}, \texttt{␣}\}$.
- The alphabet of English words: $\Sigma = \{\texttt{a}, \texttt{aback}, \texttt{abaft}, \texttt{abandon}, \ldots, \texttt{zoom}\}$.
- The alphabet of the C++ programming language consists of 96 characters, including the following 91 graphical characters,

  ```
  a b c d e f g h i j k l m n o p q r s t u v w x y z
  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
  0 1 2 3 4 5 6 7 8 9
  _ { } [ ] # ( ) < > % : ; . ? * + - / ^ & | ~ ! = , \ " '
  ```

  as well as the space character, horizontal tab, vertical tab, form feed and new-line.
- The alphabet of C++ tokens.
- The binary alphabet $\{a, b\}$.
- The unary alphabet $\{a\}$.

In abstract examples, elements of Σ are typically denoted by lower-case Latin letters from the beginning of the alphabet $(a, b, \ldots)$.

A *string* over an alphabet Σ is a finite sequence $a_1 \ldots a_\ell$ of symbols from Σ, where $\ell \geqslant 0$ and $a_1, \ldots, a_\ell \in \Sigma$. The number of symbols in the string is known as its *length* and is denoted by $|a_1 \ldots a_\ell| = \ell$. The letters w, u, v, x, y and z are most commonly used to denote strings.

The unique string of length zero is denoted by ε.

Number of occurrences of a symbol a in a string w is denoted by $|w|_a$.

The set of all strings over an alphabet Σ is denoted by Σ^*.

The *concatenation* of two strings, u and v, is the string $u \cdot v = uv$.

Notation for concatenation of multiple strings: $\prod_{i=1}^{n} w_i = w_1 \ldots w_n$. Powers of concatenation: for every $n \geqslant 0$, $w^n = \prod_{i=1}^{n} w = w \cdot \ldots \cdot w$. Assume $w^0 = \varepsilon$ for every $w \in \Sigma^*$.

The reversal of a string $w \in \Sigma^*$, denoted by w^R, is formed of the same symbols, written backwards: $(a_1 \dots a_\ell)^R = a_\ell \dots a_1$. In particular, $\varepsilon^R = \varepsilon$. If $w = w^R$ for some string w, this string is called a *palindrome*.

A *homomorphism* between strings over alphabets Σ and Ω is a mapping $h\colon \Sigma^* \to \Omega^*$ that satisfies $h(u \cdot v) = h(u) \cdot h(v)$ for all $u, v \in \Sigma^*$. An *injective homomorphism*, for which $h(u) = h(v)$ implies $u = v$, is called a *code*. A code $h\colon \Sigma^* \to \Omega^*$ has a corresponding decoding function, $h^{-1}\colon \Omega^* \to \Sigma^*$.

Alternative terminology. Pure mathematicians, especially those in combinatorics, often call a string a *word*. Symbols may also be called *letters*. The empty string is occasionally denoted by λ, Λ or 1, where the latter notation means an algebraic identity.

1.2.2 Languages

Any subset of Σ^* is called a *language* over Σ. The set of all languages is thus the set of all subsets of Σ^*, denoted by 2^{Σ^*}. The letters L, K, M, N are typically used to denote languages.

The concatenation of two languages, K and L, is the language $\{uv \mid u \in K, v \in L\}$. The powers of concatenation are defined analogously to the case of strings:

$$L^n = \underbrace{L \cdot \ldots \cdot L}_{n \text{ times}} = \{ w_1 \dots w_n \mid w_i \in L\}.$$

In particular, $L^0 = \{\varepsilon\}$.

Since languages are sets, all Boolean set-theoretic operations are applicable to them. For a language L over an alphabet Σ, its *complement* is the language $\Sigma^* \setminus L$.

The set Σ^* forms a semiring with concatenation as product and union as sum, with identity $\{\varepsilon\}$ and with zero $\varnothing$. That is, concatenation is associative ($(KL)M = K(LM)$ for all $K, L, M \in \Sigma^*$) and it is distributive over union: $K(L \cup M) = KL \cup KM$ and $(K \cup L)M = KM \cup LM$. Concatenation of languages is not commutative.

Example 1.1. *The set of English words is a formal language over the alphabet $\{a, \dots, z, '\}$. It is finite.*

Example 1.2. *The set of well-formed English sentences is a formal language over the alphabet $\{a, \dots, z, '\}$, expanded with a few punctuation signs. It is infinite.*

Example 1.3. *The set of well-formed programs in C++ is a formal language over the 96-symbol alphabet defined above. It is infinite.*

Example 1.4. *The language $\{ a^n b^n \mid n \geqslant 0\}$.*

Example 1.5. *Let $\Sigma = \{a, b\}$ and let the symbols a and b represent a left bracket and a right bracket, respectively. Then the language of well-nested parentheses $D = \{\varepsilon, ab, aabb, abab, aabbab, abaabb, \dots\}$ is known as the Dyck language. It is named in honour of Walther von Dyck.*

The *iteration* of a language, also known as *Kleene star*:

$$L^* = \bigcup_{n \geqslant 0} L^n = \{ w_1 \dots w_n \mid n \geqslant 0,\ w_i \in L\}.$$

Define $L^+ = L \cdot L^*$. Note that $\varnothing^* = \{\varepsilon\}$ and $\varnothing^+ = \varnothing$.

Example 1.6. *Let L be the set of well-formed English sentences. Then, L^+ is the set of well-formed English texts.*

Example 1.7. *Let D be the Dyck language. Then, $D^* = D$.*

Reversal of a language $L^R = \{ w^R \mid w \in L \}$.

Example 1.8. *Let L be the set of well-formed English sentences. Then, $L \cap L^R$ is the set of English palindromes.*

The right- and left-quotient of languages:

$$K \cdot L^{-1} = \{ u \mid \exists v \in L : uv \in K \}$$
$$L^{-1} \cdot K = \{ v \mid \exists u \in L : uv \in K \}$$

NB: $K \cdot L^{-1}$ and $L^{-1} \cdot K$ should be regarded as indivisible symbols for functions of two language arguments. There is no such thing as "L^{-1}".

Homomorphic images of languages: let $h\colon \Sigma^* \to \Omega^*$ be a *homomorphism* between strings over alphabets Σ and Ω, and define the function $h\colon 2^{\Sigma^*} \to 2^{\Omega^*}$ by $h(L) = \{ h(w) \mid w \in L \}$. Also, regardless of whether h is a code or not, one can define the set of pre-images of h as a function $h^{-1}\colon 2^{\Gamma^*} \to 2^{\Sigma^*}$ with $h^{-1}(K) = \{ w \mid h(w) \in K \}$ for every $K \subseteq \Gamma^*$.

1.3 Finite automata and regular expressions

Finite automaton: the simplest model of computation. Useful, in particular, for the simplest kind of parsing, breaking a strings into elementary substrings (tokens), basically a text into words.

1.3.1 Deterministic finite automata (DFA)

A model of computation with finite memory. Each finite automaton has a fixed amount of internal states, and has to use them to process input strings of unbounded length. The strings are read from left to right, symbol by symbol, by a reading head that can see one input symbol at a time. The automaton begins its computation on an input string in a certain initial state q_0, with the reading head scanning the leftmost symbol of the input string. At each step of the computation, the automaton determines its next state on the basis of its current state and the symbol currently scanned, and moves to the next input symbol to the right, entering the chosen new state. Once the rightmost symbol is processed, the resulting state of the automaton is used to determine whether the string is considered accepted or rejected.

Definition 1.1 (Kleene [1956])**.** *A* deterministic finite automaton *(DFA) is a quintuple $A = (\Sigma, Q, q_0, \delta, F)$, where*

- *Σ is an input alphabet;*
- *Q is a finite nonempty set of states;*
- *$q_0 \in Q$ is the initial state;*
- *$\delta\colon Q \times \Sigma \to Q$ is the transition function;*
- *$F \subseteq Q$ is the set of accepting states (also called final states).*

The transition function is extended to $\delta : Q \times \Sigma^* \to Q$ as follows: $\delta(q, \varepsilon) = q$, $\delta(q, aw) = \delta(\delta(q, a), w)$. The language recognized by a DFA is defined as $L(A) = \{ w \in \Sigma^* \mid \delta(q_0, w) \in F \}$.

Example 1.9. *The following DFA over the alphabet $\Sigma = \{a, b\}$, presented in Figure 1.1, recognizes the language $\{ w \mid w \in \{a,b\}^*, |w|_a \equiv 2 \pmod 3 \}$. It uses the set of states $Q = \{q_0, q_1, q_2\}$, where q_0 is the initial state, and has the transition function given in the table below.*

δ	a	b
q_0	q_1	q_0
q_1	q_2	q_1
q_2	q_0	q_2

The set of accepting states is $F = \{q_2\}$.

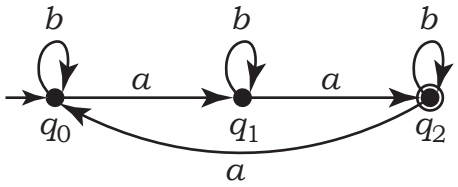


Figure 1.1: The DFA from Example 1.9, which counts modulo 3.

1.3.2 Nondeterministic finite automata (NFA)

Nondeterminism: may have multiple actions listed. Accordingly, multiple computations. Accepts a string if any of these computations is accepting.

Definition 1.2 (Rabin and Scott [1])**.** *A* nondeterministic finite automaton *(NFA) is a quintuple $B = (\Sigma, Q, Q_0, \delta, F)$, where Σ, Q, F are as in a DFA,*

- *there is a set of initial states $Q_0 \subseteq Q$, and*
- *and the transition function is defined as $\delta \colon Q \times \Sigma \to 2^Q$.*

The language recognized by an NFA is defined as

$$L(B) = \{ a_1 \dots a_n \mid a_i \in \Sigma,\ \exists r_0, \dots, r_n \in Q :\ r_0 = q_0,\ r_i \in \delta(r_{i-1}, a_i),\ r_n \in F \}.$$

A DFA can be regarded as a special case of an NFA with $|\delta(q, a)| = 1$ for all q and a.

Example 1.10. *The language of all strings over the alphabet $\Sigma = \{a, b\}$ that have the third last symbol equal to a is recognized by an NFA with the set of states $Q = \{q_0, q_1, q_2, q_3\}$, where q_0 is the initial states, and with the following transition table:*

δ	a	b
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	$\{q_2\}$
q_2	$\{q_3\}$	$\{q_3\}$
q_3	$\varnothing$	$\varnothing$

The set of accepting states is $F = \{q_3\}$.

This NFA operates by looping in the state q_0 until it eventually *guesses* that the next symbol a is the third last symbol of the input, and accordingly moves to the state q_1. The transitions in the states q_1, q_2, q_3 verify the guess by counting the remaining symbols and accepting if there are exactly two symbols after the guessed instance of a.

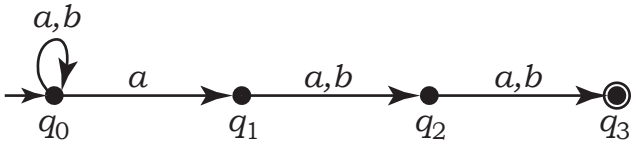


Figure 1.2: The NFA from Example 1.10, which guesses the third last symbol and then verifies the guess.

1.3.3 The subset construction

Lemma 1.1 (The subset construction, Rabin and Scott [1])**.** *Let $B = (\Sigma, Q, Q_0, \delta, F)$ be an NFA. Then the DFA $A = (\Sigma, 2^Q, Q_0, \delta', F')$ with $\delta'(s,a) = \bigcup_{q \in s} \delta(q,a)$ for all $s \subseteq Q$ and $a \in \Sigma$ and with $F' = \{\, s \mid s \subseteq Q,\ s \cap F \neq \varnothing \}$ recognizes the same language as B.*

Idea of the proof. Upon reading a string w, A computes the set of states reached in all possible computations of B on w. □

Example 1.11. *The subset construction applied to the NFA from Example 1.10 produces a DFA shown in Figure 1.3.*

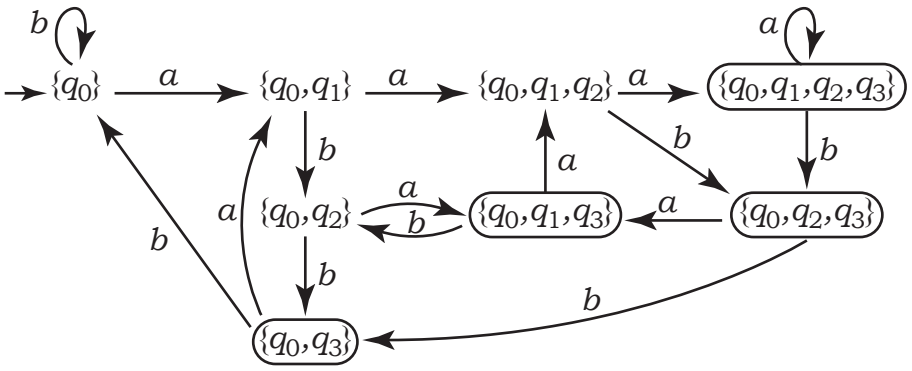


Figure 1.3: A DFA simulating the NFA from Example 1.10, obtained by the subset constructions (all states containing q_3 are accepting).

Theorem 1.1. *DFA and NFA recognize the same family of languages.*

(called the regular languages)

1.3.4 Nondeterministic finite automata with ε transitions

NFAs with ε transitions, known as ε-NFAs, have a transition function

$$\delta \colon Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$$

An ε-NFA C recognizes the following language:

$$L(C) = \{\, u_1 \dots u_n \mid u_i \in \Sigma \cup \{\varepsilon\},\ \exists r_0, \dots, r_n \in Q :\ r_0 = q_0,\ r_i \in \delta(r_{i-1}, u_i),\ r_n \in F \}.$$

Lemma 1.2. *Let $C = (\Sigma, Q, q_0, \delta, F)$ be an ε-NFA. For every state $q \in Q$, let $Closure(q) \subseteq Q$ be the set of states reachable from q by zero or more ε-transitions. Then the NFA $B = (\Sigma, Q, q_0, \delta', F')$ with $\delta'(q,a) = \{\, \delta(q', a) \mid q' \in Closure(q) \}$ for all $q \in Q$ and $a \in \Sigma$ and with $F' = \{\, q \mid Closure(q) \cap F \neq \varnothing \}$ recognizes the same language as C.*

Idea of the proof. The transition function δ' simulates a sequence of ε-transitions followed by a transition by an input symbol. The set of accepting states F' represents acceptance in all states, from which one can reach an accepting state by a sequence of ε-transitions. □

1.3.5 Regular expressions

Regular expressions are expressions over constant languages $\varnothing$, $\{\varepsilon\}$ and $\{a\}$ (for all $a \in \Sigma$), connected using union, concatenation and iteration (the *Kleene star* operator).

More formally, regular expressions can be defined as follows.

Definition 1.3 (Kleene [1956]). • *$\varnothing$ is a regular expression;*

- *for every symbol $a \in \Sigma$, a is a regular expression;*
- *if α and β are regular expressions, then so are $\alpha \mid \beta$, $\alpha \cdot \beta$ and α^*.*

Language defined by an expression: $L(\varnothing) = \varnothing$, $L(a) = \{a\}$, $L(\alpha \mid \beta) = L(\alpha) \cup L(\beta)$, $L(\alpha \cdot \beta) = L(\alpha) \cdot L(\beta)$, $L(\alpha^*) = L(\alpha)^*$.

The default precedence of operations is the following: the star has the highest precedence, followed by the concatenation, while the precedence of the union is the lowest. For instance, $(a \cup bc^*)d$ should be read as $(a \cup (b(c^*)))d$.

Further redundant notation: $\varepsilon = \varnothing^*$, $\alpha^+ = \alpha^*\alpha$.

Example 1.12 (cf. Example 1.10). *The set of all strings over the alphabet $\Sigma = \{a, b\}$ with the third last symbol equal to a is defined by the regular expression $(a \mid b)^*a(a \mid b)(a \mid b)$.*

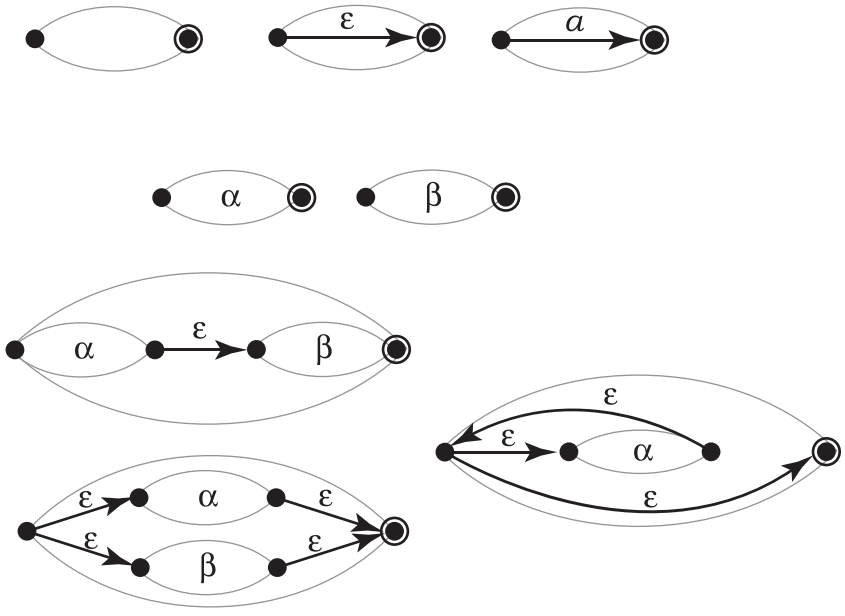
Theorem 1.2 (Kleene's theorem [1956]). *A language is recognized by a DFA if and only if it is defined by a regular expression.*

The set of languages representable by these equivalent formalisms is known as *the family of regular languages*, and any such language is called a *regular language*.

Lemma 1.3. *For every regular expression, there exists an ε-NFA recognizing the language defined by this expression.*

Sketch of a proof. The construction of an ε-NFA equivalent to a given regular expression is done inductively on the structure of the expression. The constructed ε-NFA shall have a unique initial state and a unique accepting state.

The below figure gives ε-NFAs for the three base cases of regular expressions, as well as the three cases of the induction step (where α and β are smaller regular expressions, for which ε-NFAs exist by the induction hypothesis). □



Conversely, every finite automaton can be transformed to an equivalent regular expression.

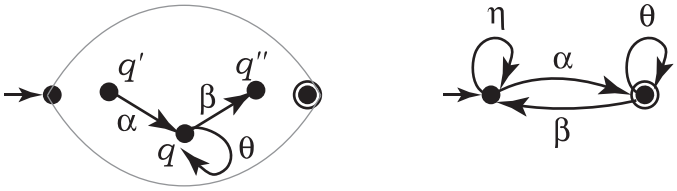
Lemma 1.4. *For every ε-NFA there exists an equivalent regular expression.*

Idea of the proof. Without loss of generality, assume that the given ε-NFA has a unique accepting state, which is different from the initial state (one can use ε-transitions to ensure that).

Consider a more expressive model than an ε-NFA, in which every transition is labelled with a regular expression. This can be defined as a quintuple $C = (\Sigma, Q, q_0, R, q_{acc})$, where R is a finite set of triples (q, α, q'), with $q, q' \in Q$ and with a regular expression α, and every such triple is a transition upon reading any string in $L(\alpha)$. The language defined by such a model is

$$L(C) = \{\, u_1 \dots u_n | u_i \in \Sigma^*, \exists r_0, \dots, r_n \in Q : r_0 = q_0, r_n = q_{acc}, (q_{i-1}, \alpha_i, q_i) \in R \text{ and } u_i \in L(\alpha_i) \text{ for all } i\}.$$

The proof is by constructing a sequence of RE-NFAs, in which all states, except the initial and the accepting states, are eliminated one by one, at the expense of increasing the size of regular expressions used on the transitions.



Consider the left picture: in order to remove the state q, for each incoming transition labelled by α and for each outgoing transition labelled by β, a new transition from q' to q'' labelled with the regular expression $\alpha\theta^*\beta$ is added. These new transitions handle all uses of the state q in all computations, and then q can be removed.

If there are two transitions between any two states, labelled by α and β, they are replaced by a single transition labelled by the regular expression $\alpha \mid \beta$.

Eventually all states except the initial and the accepting ones will be eliminated, and the automaton will be as in the right figure. Then it is equivalent to the regular expression $\eta^*\alpha(\theta \cup \beta\eta^*\alpha)^*$. □

Example 1.13. *Consider the DFA from Example 1.9. At the first step of the transformation, the state q_1 is eliminated, and the path from q_0 to q_1 is replaced with the regular expresion b^*ab^*; the resulting RE-NFA is given in Figure 1.4. It is then transformed to the regular expression $b^*ab^*a(b \mid ab^*ab^*a)^*$.*

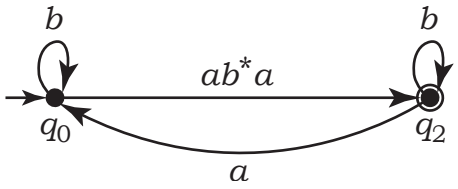


Figure 1.4: Transforming the DFA from Example 1.9 to a regular expression.

1.3.6 Closure properties

Let $\mathcal{L} \subseteq 2^{\Sigma^*}$ be a family of languages, let $f : 2^{\Sigma^*} \times \dots \times 2^{\Sigma^*} \to 2^{\Sigma^*}$ be an n-argument function on languages. Then, $\mathcal{L}$ is said to be *closed* under f if the value of f on any arguments from $\mathcal{L}$ belongs to $\mathcal{L}$ as well, that is, that $L_1, \dots, L_n \in \mathcal{L}$ implies $f(L_1, \dots, L_n) \in \mathcal{L}$.

Theorem 1.3. *The family of regular languages are closed under all Boolean operations, concatenation and star.*

Union, concatenation, star: direct by regular expressions.

Complementation: through DFA as follows.

Lemma 1.5. *For every DFA $A = (\Sigma, Q, q_0, \delta, F)$, the DFA $A = (\Sigma, Q, q_0, \delta, Q \setminus F)$ recognizes the complement of $L(A)$.*

Note that this construction does not work for NFAs. In fact, examples of n-state NFAs are known, whose complement requires an NFA of 2^n states.

Closure under intersection: using de Morgan's law $K \cap L = \overline{\overline{K} \cup \overline{L}}$ and the closure of regular languages under complementation and union.

Alternatively, one can use a direct construction of a DFA which simulates two given DFA and accepts their intersection:

Lemma 1.6 (The direct product construction)**.** *For every DFA $A = (\Sigma, Q, q_0, \delta, F)$ and $B = (\Sigma, R, r_0, \eta, F')$, the DFA $(\Sigma, Q \times R, (q_0, r_0), \theta, F \times F')$, in which $\theta((q, r), a) = (\delta(q, a), \eta(r, a))$ recognizes the intersection of $L(A)$ and $L(B)$.*

The same construction works for NFAs.

1.3.7 Limitations of regular languages

Example 1.14. *The language $\{ a^n b^n \mid n \geqslant 0 \}$ is not regular.*

Proof. Suppose there is a DFA A for this language and consider the states $q_i = \delta(q_0, a^i)$ reached by the DFA after reading each string a^i with $i \geqslant 0$, Since there are finitely many states in the DFA, eventually the sequence of states $q_0, \ldots, q_i, \ldots$ will repeat itself: that is, there are numbers $0 \leqslant i < j$ with $q_i = q_j$. Since $a^i b^i \in L(A)$, the state $\delta(q_i, b^i)$ must be in F, and accordingly $\delta(q_j, b^i) \in F$. Then the string $a^j b^i$ is accepted by A as well, which is a contradiction. □

This reasoning extends to the following general method for proving that particular languages are not regular.

Lemma 1.7 (The pumping lemma)**.** *For every regular language $L \subseteq \Sigma^*$ there exists a constant $p \geqslant 1$, such that for every string $w \in L$ with $|w| \geqslant p$ there exists a factorization $w = xyz$, where y is nonempty and $|xy| \leqslant p$, such that $xy^k z \in L$ for all $i \geqslant 0$.*

Proof. Let $A = (\Sigma, Q, q_0, \delta, F)$ be a DFA recognizing L and define $p = |Q|$. Consider an arbirary string $w \in L$ of length at least p and let $r_0, r_1, \ldots, r_{|w|} \in Q$ be the computation of A on w, so that each state r_i is reached after reading i symbols of w. Since there more states that $|Q|$ in this sequence, there must be a pair of identical states $r_i = r_j$ with $0 \leqslant i < j \leqslant |w|$.

Denote the first i symbols of w by x and the last $|w| - j$ symbols of w by z, and let y be the middle part of w, so that $w = xyz$ and $|y| = j - i$. Let $q = r_i = r_j$. Then, $\delta(q_0, x) = q$, $\delta(q, y) = q$ and $\delta(q, z) = r_{|w|} \in F$. Each string of the form $xy^k z$ is then accepted by A by visiting the state q after reading each prefix xy^i. □

Example 1.15. *The language $L = \{ ww^R \mid w \in \{a, b\}^* \} = \{\varepsilon, aa, bb, aaaa, abba, baab, bbbb, aaaaaa, \ldots\}$ is not regular.*

Proof. Suppose it is, and let p be the constant given by the pumping lemma. Then for the string $a^p bba^p \in L$ the lemma gives a factorization $a^p bba^p = xyz$ with $|xy| \leqslant p$ and $|y| \geqslant 1$. Let $x = a^k$, $y = a^\ell$ and $z = a^{p-k-\ell} bba^p$. Then the lemma, in particular, asserts that $xz = a^{p-\ell} bba^p \notin L$ should be in L, which is a contradiction. □

Example 1.16. *The hyper-text markup language (HTML) is not regular.*

Idea of a proof. Consider that the string

$$\texttt{<html><body>}\underbrace{\texttt{
}\ldots\texttt{
}}_{i\geqslant 1}\underbrace{\texttt{
}\ldots\texttt{
}}_{j\geqslant 1}\texttt{</body></html>}$$

is a valid HTML document if and only if $i = j$, and the language $\{\, a^n b^n \mid n \geqslant 0\}$ is not regular (Example 1.14). □

Exercises

1.3.1. Prove that the language $\{\, a^m b^n \mid m \neq n\}$ is not regular.

1.3.2. Prove that the regular languages are closed under $\sqrt{L} = \{\, w \mid ww \in L\}$.

1.3.3. Prove that the regular languages are closed under the cyclic shift.

Bibliography

[1956] S. C. Kleene, "Representation of events in nerve nets and finite automata", in: C. Shannon, J. McCarthy (Eds.), *Automata Studies*, 1956.

[1] M. O. Rabin, D. Scott, "Finite automata and their decision problems", *IBM Journal of Research and Development*, 3:2 (1959), 114–125.