Regular expressions are a notation for specifying (denoting) languages. *A regular expression* defines/denotes/specifies a langue (a set of strings).

Regular expressions constitute a recursively defined set:

Base cases	**Recursive cases**
$\emptyset$	$r \mid s$ (in the book as $r \cup s$)
ε	$r\ s$
$\mathtt{a} \in \Sigma$	$r*$

A language for which there exists a regular expression that generates it is called **regular**. We can talk of the set (or class) of **regular languages**.

Theorem (Lemma?) 2.3.1: The class of languages accepted by finite automata is closed under union, concatenation, Kleene star, complementation, and intersection.

Rewritten:

If L_1 and L_2 are in the set of languages accepted by DFAs/NFAs, then so are

$$L_1 \cup L_2 \qquad L_1 L_2 \qquad L_1 * \qquad \overline{L_1} \quad \text{and} \quad L_1 \cap L_2$$

Analyzed in terms of quantification:

$$\begin{array}{lllll} \forall\ L_1, L_2, & \text{if} & \exists\ M_1, M_2 & | & L(M_1) = L_1 \text{ and } L(M_2) = L_2 \\ & \text{then} & \exists\ M_3 & | & L(M_3) = L_1 \cup L_2 \text{ (etc)} \end{array}$$

Main result:

Theorem 2.3.2: A language L is regular iff $\exists\ M \in NFA$ such that $L(M) = L$.

Corollary:

Set of regular languages	=	Set of NFA languages	=	Set of DFA languages

Theorem 2.3.2: A language L is regular iff $\exists\ M \in NFA$ such that $L(M) = L$.

Proof (outline). *(⇒) Suppose t is a regular expression.*

Base cases. *Suppose* $t = \varepsilon$

Suppose $t = \emptyset$

Suppose $t = \mathtt{a} \in \Sigma$

Inductive cases. *Suppose* $t = r|s$ *We know by induction that there exist* M_1 *and* M_2 *such that* $L(M_1) = r$ *and* $L(M_2) = s$.

Theorem 2.3.2: A language L is regular iff $\exists\ M \in NFA$ such that $L(M) = L$.

Proof (outline) continued. *($\Leftarrow$) Suppose $M \in NFA$. [We need to construct a regular expression that generates the language that M accepts.]*

Label the states of M $q_1, q_2, \ldots q_n$ arbitrarily except that $s = q_1$.

Consider the set of state-transition paths from q_i to q_j that do not include any state q_x for $x > k$.

Let $R(i, j, k)$ be the set of strings that drive the machine from q_i to q_j without stopping at any state q_x for $x > k$.

For any q_i and q_j, show that $R(i, j, k)$ is regular by induction on k.

Hence $R(1, j, |K|)$ is regular for any $q_j \in F$. Therefore $L(M)$ is regular. $\Box$

News of the day: *Not all languages are regular.*

Non-constructive proof: The set of languages is uncountable, but the set of regular expressions is countable. Hence some languages can't be specified by a regular expression.

Theorem 2.4.1: Let L be a regular language. There is an integer $n \geq 1$ such that any string $w \in L$ with $|w| \geq n$ can be written as $w = xyz$ such that $y \neq \varepsilon$, $|xy| \leq n$, and $xy^i z \in L$ for each $i \geq 0$.

Theorem 2.4.1: Let L be a regular language. There is an integer $n \geq 1$ such that any string $w \in L$ with $|w| \geq n$ can be written as $w = xyz$ such that $y \neq \varepsilon$, $|xy| \leq n$, and $xy^i z \in L$ for each $i \geq 0$.

This is a *pumping theorem*:

Proof (sketch). *Let M be a DFA that accepts L. Suppose $w \in L$ and w is at least as long as the number of states in M.*

At least one state is repeated in the transition sequence, some $q_i = q_j$. Let $xyz = w$ where x is the prefix of w from s to q_i, y is the substring of w from q_i to q_j, and z the suffix of w from q_j to $f \in F$.

When the machine gets back to $q_i = q_j$, it could accept another copy of y—or it could have not had y in the input string at all.

Hence $\forall\ i, i \geq 0, xy^i z \in L$. $\square$