Use of regular expressions in programming:

→ design of compilers for programming languages

Elemental objects in a programming language, which are called tokens (for example variables names and constants) can be described with regular expressions. We get the syntax of a programming language this way. There exists an algorithm for recognizing regular expressions that has been implemented $\implies$ an automatic system generates the lexical analyzer that checks the input in a compiler.

→ eliminate redundancy in programming

The same regular expression can be generated in more than one way (obvious from the definition of a regular expression) $\implies$ there exists an equivalence relation on regular expressions and algorithms that check when two regular expressions are equivalent.

Theoretical importance of regular expressions

For the study of formal languages and grammars, the importance of regular expressions comes from the following theorem:

Theorem: A language is regular $\iff$ some regular expression describes it.

Sketch of proof: Recall the definition of a regular language as the language obtained in finitely many steps from finite subsets of words via union, concatenation or the Kleene star. We can construct a regular expression

from the definition of the regular language in question, and vice versa starting with a regular expression, we can define a finite sequence of L_i's such that each L_i is a finite set of words or is obtained from previous L_i's via union, concatenation or the Kleene star.

qed

Finally, we can state the complete characterization of regular languages:

Theorem: The following are equivalent:

(i) L is a regular language.

(ii) L is recognized by a (deterministic or non-deterministic) finite state acceptor.

(iii) L is produced by a regular grammar.

(iv) L is given by a regular expression.

Remark: It is possible to prove directly that (iv) $\iff$ (ii), but the construction is rather complicated. Instead, we sketched above the proof that (i) $\iff$ (iv), and we had previously stated that (i) $\iff$ (ii) $\iff$ (iii), so we now have that (i) $\iff$ (ii) $\iff$ (iii) $\iff$ (iv).

Example: Let $L = \{0^m 1^n \mid m, n \in \mathbb{N}, m \geq 0, n \geq 0\}$ be the regular language we considered before. We now give a regular expression for L: $L = 0^* \circ 1^*$. Recall we previously show this language is regular from the definition of a regular language, so solving this problem is a direct illustration of the implication (i) $\iff$ (iv).

8.6 The Pumping Lemma

Task: Understand another criterion for figuring out when a language is regular.

Let a finite set A be the alphabet, and let L be a language over A. Then $L \subset A^*$. We make the following two crucial observations:

1. If L is finite, then clearly there exists a finite state acceptor that recognizes $L \Rightarrow L$ is regular.
2. If $L = A^*$, then L is likewise regular. Here is why: Let $A = \{a_1, \ldots, a_n\}$. The acceptor

   with just one state i recognizes A^*.

Question: If L is infinite, but $L \subsetneqq A^*$, how can we tell whether L is regular?

Answer: The Myhill-Nerode Theorem would have us look at equivalence classes of words, but that analysis can be complicated at times. The Pumping Lemma provides another way of checking whether L is regular.

The Pumping Lemma: If L is a regular language, then there is a number p (the pumping length) where if w is any word in L of length at least p, then $w = xuy$ for words x, y, and u satisfying:

1. $u \neq \epsilon$ (i.e., $|u| > 0$, the length of u is positive);
2. $|xu| \leq p$;
3. $xu^n y \in L \ \forall\, n \geq 0$.

Remark: p can be taken to equal the number of states of a deterministic finite state acceptor that recognizes L (we know such a finite state acceptor exists because L is regular).

Sketch of proof: The name of the lemma comes from the fact that if L is regular, then all of its words can be pumped through a finite state acceptor that recognizes L. We assume this acceptor is deterministic and has p states. We will show the Pumping Lemma is a consequence of the Pigeonhole Principle we studied in the unit on functions. If a word w has length l, then the finite state acceptor must process l pieces of information ($w = a_1 a_2 \cdots a_l$, where $a_k \in A \ \forall\, k,\ 1 \leq k \leq l$) $\implies$ it passes through $l+1$ states starting with the initial state. In the hypotheses of the lemma, we assume $|w| = l \geq p$, but $p = \#$(states of the acceptor) $\implies$ the acceptor passes through $l + 1 \geq p + 1$ states to process w and therefore at least one state is repeated among the first $p + 1$. Let s_1, s_2, ..., s_{l+1} be the sequence of states. $|w| = l \geq p \implies s_i = s_j$ with $i < j \leq p + 1$. Now we set x to be the part of w that makes the acceptor pass through states s_1, s_2, ..., s_i, i.e., $x = a_1 a_2 \cdots a_{i-1}$ (the first $i - 1$ letters in w). We set u to be the part of w that makes the acceptor pass through states s_i, s_{i+1}, s_{i+2}, ..., s_j. In other words, $u = a_i a_{i+1} \cdots a_{j-1}$. Since $i < j$, $|u| \geq 1$ $\implies$ $u \neq \epsilon$. Finally, set y to be the part of w (the tail end) that makes the acceptor pass through states s_j, s_{j+1}, ..., s_{l+1}, i.e., $y = a_j a_{j+1} \cdots a_l$. Since $j \leq p + 1$, $j - 1 \leq p$, so $|xu| = |a_1 a_2 \cdots a_{j-1}| = j - 1 \leq p$ as needed. Furthermore, $s_i = s_j$, so at the beginning of u and at its end the acceptor is in the same state $s_i = s_j \implies xu^n y$ is accepted for every $n \geq 0 \implies xu^n y \in L$ as needed. We have obtained conditions (1)-(3).

qed

Applications of the Pumping Lemma

As a statement, the Pumping Lemma is the implication $P \to Q$ with P being the sentence "L is a regular language" and Q being the decomposition of every w, $|w| \geq p$ as $w = xuy$. We use the contrapositive $\neg Q \to \neg P$ (tautologically equivalent to $P \to Q$) as our criterion for detecting non-regular languages.

Examples: 1. $L = \{0^m 1^m \mid m \in \mathbb{N}, m \geq 0\}$ is not regular. Let $w = 0^m 1^m$. We cannot decompose w as $w = xuy$ because whatever we let u be, we get a contradiction to $xu^n y \in L \ \forall\, n \geq 0$. If $u \in 0^*$ (string of 0's),

$x \in 0^*$ and $y = 0^p 1^q$ (string of p 0's with $p \geq 0$ and q 1's). There are values of n for which $xu^n y \notin L$.
If $u \in 1^*$, we get a contradiction the same way.
If $u \in 0^*1^*$, $xu^2y \notin L$ for any x, y words!

2. $L = \{0^m \mid m \text{ is prime}\}$ is not regular.
Since $w = 0^m$, x, u, y can consist only of 0's, so then $x = 0^i$, $u = 0^j$, $y = 0^k$. If $xu^n y \in L \ \forall\, n \geq 0$, then $i + nj + k$ is prime $\forall\, n \geq 0$, which is impossible.
Set $n = i + 2j + k + 2$, then

$$\begin{aligned} i + nj + k &= i + (i + 2j + k + 2)j + k = i + ij + 2j^2 + jk + 2j + k \\ &= i(j+1) + 2j(j+1) + k(j+1) = (j+1)(i + 2j + k), \end{aligned}$$

where $|u| > 0$, so $j \geq 1$. Therefore, $n = (j+1)(i+2j+k)$ is not prime!

Practice at home: weitz.de/pump (on Edi Weitz's website)

The pumping game, an online game to help you understand the Pumping Lemma.