Lesson 2: Regular expressions

Theme: Regular expressions as alternative description of regular languages.

In the following we fix an alphabet $\Sigma$. Regular expressions (over $\Sigma$) are expressions built inductively as follows.

- $\emptyset$ is a regular expression.
- $a$ is a regular expression, for every symbol $a \in \Sigma$.
- If $e_1, e_2$ are regular expressions, then so are $(e_1 \cdot e_2)$ and $(e_1 \cup e_2)$.
- If $e$ is a regular expression, then so is $(e)^*$.

A regular expression $e$ over $\Sigma$ defines a language, denoted by $L(e)$, over the same alphabet as follows.

- If $e$ is $\emptyset$, then $L(e) = \emptyset$.
- If $e$ is $a$, for some symbol $a \in \Sigma$, then $L(e) = \{a\}$.
- If $e$ is of the form $(e_1 \cdot e_2)$, where $e_1$ and $e_2$ are regular expressions, then $L(e) = L(e_1) \cdot L(e_2)$.
- If $e$ is of the form $(e_1 \cup e_2)$, where $e_1$ and $e_2$ are regular expressions, then $L(e) = L(e_1) \cup L(e_2)$.
- If $e$ is of the form $(e_1)^*$, where $e_1$ is a regular expression, then $L(e) = L(e_1)^*$.

Usually, we omit writing $\cdot$ in $(e_1 \cdot e_2)$, and instead, we simply write $(e_1 e_2)$. Also, when there is no ambiguity, we will omit writing the brackets and simply write $e_1 e_2$ and $e_1^*$, instead of $(e_1 e_2)$ and $(e_1)^*$.

The following theorem states that the class of languages defined by regular expressions is exactly the class of regular languages.

**Theorem 2.1** Regular expressions define precisely the class of regular languages. More formally, it can be stated as follows.

- For every regular expression $e$ over $\Sigma$, $L(e)$ is a regular language.
- For every NFA $A$, there is a regular expression $e$ such that $L(e) = L(A)$.

**Proof.** We first prove the first item. The proof is by induction on the regex $e$. The base case is when $e$ is either $\emptyset$ or $a \in \Sigma$.

- When $e$ is $\emptyset$, then $L(e) = \emptyset$.
- When $e$ is $a$, for some symbol $a \in \Sigma$, then $L(e) = \{a\}$.

One can easily construct an NFA $A$ that accepts nothing.

We can construct an NFA $A$, that has only two states $p$ and $q$, $p$ is the initial state and there is only one accepting state $q$ and $\delta$ contains only one transition $(p, a, q)$.

For the induction step, we will prove the case where $e$ is either of the form $\alpha \cdot \beta$, $\alpha \cup \beta$ or $\alpha^*$.

By the induction hypothesis, there are NFA $A_1$ and $A_2$ that accept the languages $L(\alpha)$ and $L(\beta)$, respectively. By Remark 1.4 and Theorem 1.8 (in Lesson 1), there are NFAs for all the languages $L(\alpha \cdot \beta)$, $L(\alpha \cup \beta)$ and $L(\alpha^*)$. 

13
We now prove the second item. Let $A = (\Sigma, Q, q_0, F, \delta)$ be an NFA. Without loss of generality, we assume that $Q = \{1, \ldots, n\}$. For every $1 \leq i, j \leq n$ and $0 \leq k \leq n$, define the language $L(i, j, k)$ as follows.

$$L(i, j, k) := \left\{ w \in \Sigma^* \middle| \text{there is a run of } A \text{ on } w \text{ from state } i \text{ to state } j \text{ without passing any states } \geq k + 1 \right\}$$

That is, if $w \in L(i, j, k)$, there is a run of $A$ on $w$ from state $i$ to $j$ without passing through the states $k+1, \ldots, n$.

We will first prove the following claim.

**Claim 1** For every $1 \leq i, j \leq n$ and $0 \leq k \leq n$, define the language $L(i, j, k)$, there is a regex $e$ such that $L(e) = L(i, j, k)$.

**Proof.** The proof is by induction on $k$. The base case is $k = 0$. For every $1 \leq i, j \leq n$, consider the set of symbols $\Gamma_{i,j} = \{a \mid (i, a, j) \in \delta\}$.

- If $\Gamma_{i,j} = \emptyset$, then $L(i, j, 0) = \emptyset$. The desired regex $e$ is:
  $$e = \left\{ \emptyset \text{ if } i \neq j, \emptyset^* \text{ if } i = j \right\}$$

- If $\Gamma_{i,j} \neq \emptyset$, assume $\Gamma_{i,j} = \{a_1, \ldots, a_t\}$. The desired regex $e$ is:
  $$e = \left\{ a_1 \cup \cdots \cup a_t \text{ if } i \neq j, a_1 \cup \cdots \cup a_t \cup \emptyset^* \text{ if } i = j \right\}$$

For the induction hypothesis, we assume that the claim holds for $k$. For the induction step, we will prove it for $k + 1$. Note the following identity:

$$L(i, j, k + 1) = L(i, j, k) \cup \left( L(i, k + 1, k) \cdot L(k + 1, k + 1, k)^* \cdot L(k + 1, j, k) \right)$$

By the induction hypothesis, there are regexes that define each of $L(i, j, k)$, $L(i, k + 1, k)$, $L(k + 1, k + 1, k)$, and $L(k + 1, j, k)$. By the definition of regex, there is regex that define $L(i, j, k + 1)$.

To complete our proof, note that:

$$L(A) = \bigcup_{q_f \in F} L(q_0, q_f, n)$$

By the claim above, for each $L(q_0, q_f, n)$, there is a regex that defines it. Taking the union of all of them, we have a regex for $L(A)$.

**Corollary 2.2** Let $L$ be a language. The following are equivalent.

- $L$ is accepted by a DFA.
- $L$ is accepted by an NFA.
- $L$ is defined by a regular expression.
Remark 2.3 The term regular expressions are commonly abbreviated as regex. In most literatures and websites, the term “regex” are used more often than “regular expression.” Due to its widespread applications, many modern programming languages now include libraries for regex. The following are some of them.

- Java: [https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html](https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html)
- Python: [https://docs.python.org/2/library/re.html](https://docs.python.org/2/library/re.html)