

# INTRODUCTION TO LANGUAGE THEORY AND COMPILING

## *Basic definitions: alphabet, word, language*

**Definition 1.4** (Alphabet). An *alphabet* is a *finite* set of symbols. We will usually denote alphabets by  $\Sigma$ . ☹️

**Definition 1.6** (Word). A *word* on an alphabet  $\Sigma$  is a *finite* (and possibly empty) sequence of symbols from  $\Sigma$ . We use the symbol  $\varepsilon$  to denote the *empty word*, i.e., the empty sequence (that contains no symbol). ☹️

**Definition 1.8** (Language). A *language* on an alphabet  $\Sigma$  is a (possibly empty or infinite) set of words on  $\Sigma$ . ☹️

### The $\Sigma^*$ notation

Let  $\Sigma$  be an alphabet. Since any alphabet is a set, we can also regard  $\Sigma$  as a *language*, which contains only words of one character. Then, we can write  $\Sigma^*$ , which contains *all the words (including the empty one) that are made up of characters from  $\Sigma$* . This notation will be used very often in the rest of these notes.

## *Operations on words and languages*

**Definition 1.17** (Concatenation of two words). Given two words  $w = w_1 w_2 \cdots w_n$  and  $v = v_1 v_2 \cdots v_\ell$ , the *concatenation* of  $w$  and  $v$ , denoted  $w \cdot v$ , is the word:

$$w \cdot v = w_1 w_2 \cdots w_n v_1 v_2 \cdots v_\ell$$

By convention,  $\varepsilon \cdot w = w \cdot \varepsilon = w$ , for all words  $w$ . In particular  $\varepsilon \cdot \varepsilon = \varepsilon$ . ☹️

1. For all languages  $L$ , for all natural numbers  $n$ ,  $L^n$  is the language containing all words obtained by taking  $n$  words from  $L$  and concatenating them:

$$L^n = \{w_1 w_2 \cdots w_n \mid \text{for all } 1 \leq i \leq n : w_i \in L\}$$

2. For all languages  $L$ , the *Kleene closure* of  $L$ , denote  $L^*$  is the language containing all words made up of an arbitrary number of concatenations of words from  $L$ :

$$L^* = \{w_1 w_2 \cdots w_n \mid n \geq 0 \text{ and for all } 1 \leq i \leq n : w_i \in L\}$$

3. A variation on the Kleene closure is  $L^+$  which is the language containing all words made up of an arbitrary and *strictly positive* number of concatenations of words from  $L$ :

$$L^+ = \{w_1 w_2 \cdots w_n \mid n \geq 1 \text{ and for all } 1 \leq i \leq n : w_i \in L\}$$

## Regular languages

**Definition 2.1** (Regular languages). Let us fix an alphabet  $\Sigma$ . Then, a language  $L$  is regular iff:

1. either  $L = \emptyset$ ;
2. or  $L = \{\varepsilon\}$ ;
3. or  $L = \{a\}$  for some  $a \in \Sigma$ ;
4. or  $L = L_1 \cup L_2$ ;
5. or  $L = L_1 \cdot L_2$ ;
6. or  $L = L_1^*$

a finite  
number  
of times

where  $L_1$  and  $L_2$  are regular languages on  $\Sigma$ .



Using the same ideas, we can prove that all finite languages are regular.



# Regular expressions

**Definition 2.3** (Regular expressions). Given a finite alphabet  $\Sigma$ , the following are regular expressions on  $\Sigma$ :

1. The constant  $\emptyset$ . It denotes the language  $L(\emptyset) = \emptyset$ .
2. The constant  $\varepsilon$ . It denotes the language  $L(\varepsilon) = \{\varepsilon\}$ .
3. All constants  $a \in \Sigma$ . Each constant  $a \in \Sigma$  denotes the language  $L(a) = \{a\}$ .
4. All expressions of the form  $r_1 + r_2$ , where  $r_1$  and  $r_2$  are regular expressions on  $\Sigma$ . Each expression  $r_1 + r_2$  denotes the language  $L(r_1 + r_2) = L(r_1) \cup L(r_2)$ .
5. All expressions of the form  $r_1 \cdot r_2$ , where  $r_1$  and  $r_2$  are regular expressions on  $\Sigma$ . Each expression  $r_1 \cdot r_2$  denotes the language  $L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$ .
6. All expressions of the form  $r^*$ , where  $r$  is a regular expression on  $\Sigma$ . Each expression  $r^*$  denotes the language  $L(r^*) = (L(r))^*$ .

In addition, parenthesis are allowed in regular expressions to group sub-expressions (with their usual semantics).



**Theorem 2.1.** *For all regular languages  $L$ , there is a regular expression  $r$  s.t.  $L(r) = L$ . For all regular expressions  $r$ ,  $L(r)$  is a regular language.*

**Theorem 2.2** (Kleene's theorem). *For every regular language  $L$ , there is a DFA  $A$  such that  $L(A) = L$ . Conversely, for all  $\varepsilon$ -NFAs  $A$ ,  $L(A)$  is regular.*

# Finite automata

**Definition 2.5** (Finite automaton). A finite automaton is a tuple:

$$A = \langle Q, \Sigma, \delta, q_0, F \rangle$$

where:

1.  $Q$  is a finite set of states;
2.  $\Sigma$  is the (finite) input alphabet;
3.  $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \mapsto 2^Q$  is the transition function;
4.  $q_0 \in Q$  is the initial state;
5.  $F \subseteq Q$  is the set of accepting states.

**Definition 2.7** (Classes of finite state automata).

1. A *non-deterministic finite state automata with  $\varepsilon$ -transitions* ( $\varepsilon$ -NFA for short) is a finite state automaton, as in Definition 2.5.
2. A *non-deterministic finite state automaton* (NFA for short) is an  $\varepsilon$ -NFA  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$  s.t. for all  $q \in Q$ :  $\delta(q, \varepsilon) = \emptyset$ . In this case, we will henceforth assume that the signature of the transition function is  $\delta : Q \times \Sigma \mapsto 2^Q$ .
3. A *deterministic finite state automaton* (DFA for short) is an NFA  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$  s.t. for all  $q \in Q$ , for all  $a \in \Sigma$ :  $|\delta(q, a)| = 1$ . In this case, we will henceforth assume that the signature of the transition function is  $\delta : Q \times \Sigma \mapsto Q$ , and that the function is *complete*♣.




**Definition 2.10** (Accepted language of an  $\varepsilon$ -NFA). Let  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$  be an  $\varepsilon$ -NFA. Then, its *accepted language* is:

$$L(A) = \{w \in \Sigma^* \mid \exists q \in F \text{ s.t. } \langle q_0, w \rangle \vdash_A^* \langle q, \varepsilon \rangle\}$$

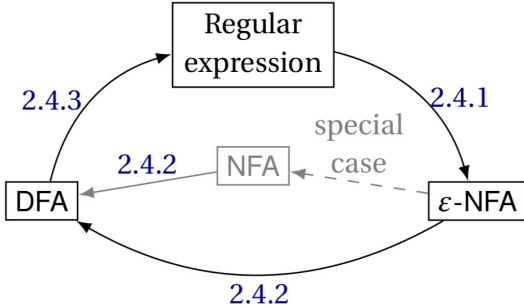


**Definition 2.8** (Configuration of an  $\varepsilon$ -NFA). A *configuration* of an  $\varepsilon$ -NFA  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$  is a pair  $\langle q, w \rangle \in Q \times \Sigma^*$ , where  $q$  is the *current state*, and  $w$  is the *input word suffix* that remains to be read.

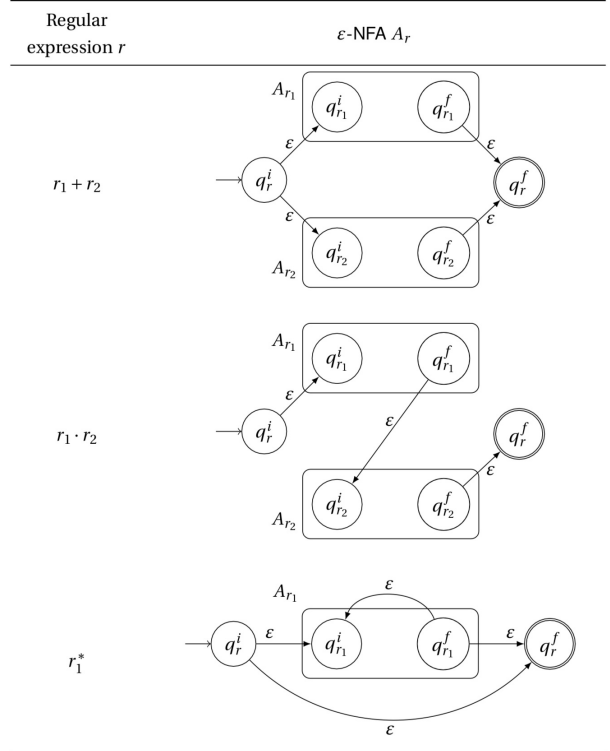
The *initial configuration* of  $A$  on the input word  $w$  is  $\langle q_0, w \rangle$ .

A configuration  $\langle q, w \rangle$  is *accepting* (or *final*) iff  $q \in F$  and  $w = \varepsilon$ . 

## Equivalence between automata and regular expressions



Regular expression $r$	$\varepsilon$ -NFA $A_r$
$\emptyset$	
$\varepsilon$	
$a \in \Sigma$	



**Definition 2.13** ( $\varepsilon$ -closure). Let  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$  be an  $\varepsilon$ -NFA. For all  $i \in \mathbb{N}$ , let  $\varepsilon\text{closure}^i(q)$  be defined as follows:

$$\varepsilon\text{closure}^i(q) = \begin{cases} \{q\} & \text{if } i = 0 \\ \delta(\varepsilon\text{closure}^{i-1}(q), \varepsilon) \cup \varepsilon\text{closure}^{i-1}(q) & \text{otherwise} \end{cases}$$

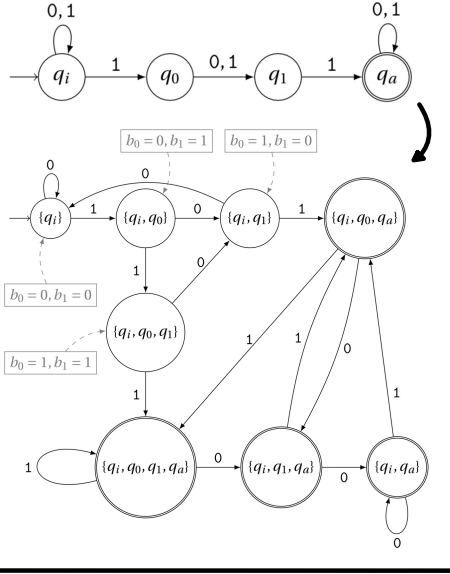
Then, for all  $q \in Q$ :  $\varepsilon\text{closure}(q) = \varepsilon\text{closure}^K(q)$ , where  $K$  is the least value s.t.  $\varepsilon\text{closure}^K(q) = \varepsilon\text{closure}^{K+1}(q)$ .

### Determinisation of $\varepsilon$ -NFAs

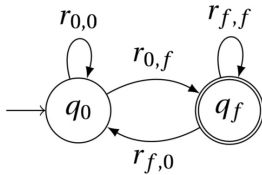
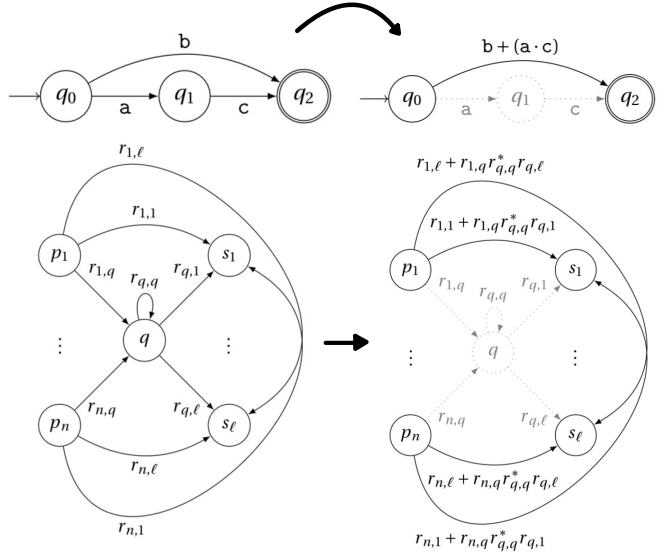
Given an  $\varepsilon$ -NFA  $A = \langle Q^A, \Sigma, \delta^A, q_0^A, F^A \rangle$ , we build the DFA  $D = \langle Q^D, \Sigma, \delta^D, q_0^D, F^D \rangle$  as follows:

1.  $Q^D = 2^{Q^A}$
2.  $q_0^D = \varepsilon\text{closure}(q_0^A)$
3.  $F^D = \{S \in Q^D \mid S \cap F^A \neq \emptyset\}$
4. for all  $S \in Q^D$ , for all  $a \in \Sigma$ :  $\delta^D(S, a) = \varepsilon\text{closure}(\delta^A(S, a))$

1. As expected, the set of states of the DFA is the set of subsets of the  $\varepsilon$ -NFA's states.
2. The initial state of the DFA is the set of states the NFA can reach from its own initial state  $q_0^A$  by reading only  $\varepsilon$ -labeled transitions. Thus,  $q_0^D$  is the set of states in which the  $\varepsilon$ -NFA can be *before reading any letter*.
3. A state of the DFA is accepting iff it contains at least one accepting state of the  $\varepsilon$ -NFA. This is coherent with the intuition that at least one execution of the  $\varepsilon$ -NFA must accept for the word to be accepted.
4. The transition function consists in: first reading a letter, then following as many  $\varepsilon$ -labeled transitions as possible.



### From $\varepsilon$ -NFAs to regular expressions



$$\rightarrow (r_{0,0} + r_{0,f} \cdot r_{f,f}^* \cdot r_{f,0})^* \cdot r_{0,f} \cdot r_{f,f}^*$$