Regular expressions: definition

- An algebraic equivalent to finite automata.
- We can build complex languages from simple languages using operations on languages.
- Let $\Sigma = \{a_1, \ldots, a_n\}$ be an alphabet. The simple languages over $\Sigma$ are
  - the empty language $\emptyset$, which contains no word.
  - for every symbol $a \in \Sigma$, the language $\{a\}$, which contains only the one-letter word "a".
- The regular operations on languages are $\cup$ (union), $\circ$ (concatenation), and $^*$ (iteration).
- An expression that applies regular operations to simple languages is called a regular expression (and the resulting language is a regular language; we will see later why...).
- $L(E)$ is the language defined by the regular expression $E$.

Formally, $R$ is a regular expression if $R$ is

1. $a$ for some $a$ in the alphabet $\Sigma$ (stands for a language $\{a\}$),
2. $\varepsilon$, standing for a language $\{\varepsilon\}$,
3. $\emptyset$, standing for the empty language,
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions,
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions,
6. $(R_1^*)$, where $R_1$ is a regular expression.

Notations

- When writing regular expressions, we use the following conventions:
  - For simple languages of the form $\{a\}$, we write $a$ (omitting braces).
  - Parentheses are omitted according to the rule that iteration binds stronger than concatenation, which binds stronger than union.
  - The concatenation symbol $\circ$ is often omitted.
  - We write $\Sigma$ for $a_1 \cup \ldots \cup a_n$.
  - We write $\varepsilon$ for $\emptyset$ (which is the language that contains only the empty word).
- For example, $01^* \cup \varepsilon$ stands for the expression $\{(\emptyset \circ (\{1\}^*)) \cup (\emptyset^*)\}$.

Examples of expressions

- $\Sigma * 00 \Sigma * \ldots$ the language of all words that contain the substring 000
- $(\Sigma \Sigma)^* \ldots$ the language of all words with an even number of letters
- $0^* 1^* 0^* \ldots$ the language of all words that contain an even number of 1’s

Note that concatenating the empty set to any set yields the empty set; $1^* \emptyset = \emptyset$
Equivalence with Finite Automata

- Regular expressions and finite automata are equivalent in their descriptive power.
- Any regular expression can be converted into a finite automaton that recognizes the language it describes, and vice versa.
- We will prove the following result

**Theorem.** A language is recognizable by a FA if and only if some regular expression describes it.

- This theorem has two directions. We state each direction as a separate lemma.

**Lemma 1.** If a language is described by a regular expression, then it is recognizable by a FA.

- We have a regular expression $R$ describing some language $A$.
- We show how to convert $R$ into an NFA recognizing $A$.
- We proved before that if an NFA recognizes $A$ then a DFA recognizes $A$.
- To convert $R$ into an NFA $N$, we consider the six cases in the formal definition of regular expression.

Proof of Lemma 1 (6 cases)

1. $R = a$ for some $a$ in $\Sigma$. Then $L(R) = \{a\}$, hence

   $$N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$$
   $$\delta(q_1, a) = \{q_2\}$$
   $$\delta(r, b) = \emptyset$$ for $r \neq q_1$ or $b \neq a$.

2. $R = \epsilon$. Then $L(R) = \{\epsilon\}$, hence

   $$N = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$$
   $$\delta(r, b) = \emptyset$$ for any $r$ and $b$.

3. $R = \emptyset$. Then $L(R) = \emptyset$, hence

   $$N = (\{q\}, \Sigma, \delta, q, \emptyset)$$
   $$\delta(r, b) = \emptyset$$ for any $r$ and $b$.

4. $R = R_1 \cup R_2$.
5. $R = R_1 \circ R_2$.
6. $R = R_1^*$.

- in these cases we use the constructions given in the proofs that the class of regular languages is closed under the regular operations.
- We construct the NFA for $R$ from NFAs for $R_1, R_2$, and the appropriate closure construction.
Example 1

Building an NFA from the regular expression \((ab \cup a)^*\)

Example 2

Building an NFA from the regular expression \((a \cup b)^* aba\)
Equivalence with Finite Automata

• We are working on the proof of the following result

**Theorem.** A language is regular if and only if some regular expression describes it.

• We have proved

**Lemma 1.** If a language is described by a regular expression, then it is regular.
  • For given regular expression \( R \) describing some language \( A \), we have shown how to convert \( R \) into an NFA recognizing \( A \).
  • Now we will prove the other direction

**Lemma 2.** If a language is regular then it is described by a regular expression.
  • For a given regular language \( A \), we need to write a regular expression \( R \), describing \( A \).
  • Since \( A \) is regular, it is accepted by a DFA.
  • We will describe a procedure for converting DFAs into equivalent regular expressions.
  • We will define a new type of finite automaton, *generalized* NFA (GNFA).
  • and show how to convert DFAs into GNFAs and then GNFAs into regular expression.

---

Generalized Non-deterministic Finite Automata

• *Generalized non-deterministic finite automata* are simply NFAs wherein the transition arrows may have any regular expressions as labels, instead of only members of the alphabet or \( \varepsilon \).

\[
\begin{align*}
&\text{Q}_{\text{start}} \xrightarrow{ab^*} a^* \xrightarrow{(aa)^*} b^* \xrightarrow{ab} \xrightarrow{ab \cup ba} \text{Q}_{\text{accept}}
\end{align*}
\]

For convenience we require that GNFAs always have a form that meets the following conditions.

• the start state has arrows going to every other state but no ingoing arrows.
• there is only one accepting state. It has ingoing arrows from every other state but no outgoing arrows.
• moreover, the start state is not the same as the accept state.
• except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.
**Formal definition of GNFAs**

- A GNFA is a 5-tuple \((Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})\), where
  1. \(Q\) is the finite set of states,
  2. \(\Sigma\) is the input alphabet,
  3. \(\delta : (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \rightarrow \mathbb{R}\) is the transition function,
  4. \(q_{\text{start}}\) is the start state, and
  5. \(q_{\text{accept}}\) is the accept state.

- A GNFA accepts a string \(w \in \Sigma^*\) if \(w = w_1, w_2, \ldots, w_n\), where each \(w_i\) is in \(\Sigma^*\) and a sequence of states \(r_0, r_1, r_2, \ldots, r_n\) exists such that
  1. \(r_0 = q_{\text{start}}, r_n = q_{\text{accept}}\)
  2. For each \(i\), we have \(w_i \in L(R_i)\), where \(R_i = \delta(r_{i-1}, r_i)\); in other words, \(R_i\) is the expression on the arrow from \(r_{i-1}\) to \(r_i\).

**From DFAs to GNFAs**

- Add a new state with an \(\epsilon\) arrow to the old start state, a new accept state with \(\epsilon\) arrows from the old accept states.
- If any arrows have multiple labels (or if there are multiple arrows going between the same two states in the same direction) replace each with a single arrow whose label is the union of the previous labels.
- Add arrows labeled \(\varnothing\) between states that had no arrows.

---

**From GNFAs to Regular Expressions.**

**Convert(G)**

1. Let \(k\) be the number of states of GNFA \(G\).
2. If \(k=2\), then \(G\) must consist of a start state, an accept state, and a single arrow connecting them and labeled with a regular expression \(R\). Return the expression \(R\).
3. If \(k>2\), select any state \(q \in Q\) different from start and accept states and let \(G'\) be the GNFA \((Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})\), where
   \[Q' = Q - \{q\}\],
   And for any \(q_i \in Q' - \{q_{\text{accept}}\}\) and any \(q_j \in Q' - \{q_{\text{start}}\}\) let
   \[\delta'(q_i, q_j) = (R_i)(R_j)^* (R_i) \cup (R_j),\]
   \[\text{for } R_i = \delta(q_i, q_j), R_j = \delta(q_i, q_j), R_3 = \delta(q_i, q_j), R_4 = \delta(q_i, q_j).\]
4. Compute \(\text{Convert}(G')\) and return this value.

**Claim.** For any GNFA \(G\), \(G'\) is equivalent to \(G\).
Proof of Claim.

Claim. For any GNFA $G$, $G'$ is equivalent to $G$.

- We show that $G$ and $G'$ recognize the same language.
- Suppose $G$ accepts an input $w$
  - then there exists a sequence of states $s.t.$
    
    \[ q_{\text{start}} \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow \ldots \rightarrow q_{\text{accept}}, \]
    
    \[ w_i \in L(R_i), \quad w = w_1 w_2 \ldots w_k \]
  - if none of them is $q_r$, then $G'$ accepts $w$
    since each of the new regular expressions labeling arrows of $G'$ contains the old reg. expression as a part of union
  - if $q_r$ does appear, removing each sequence of consecutive $q_r$ states forms an accepting path in $G'$.
    the states $q_i$ and $q_j$ bracketing a sequence have a new regular expression on the arrow between them that describes all strings taking $q_i$ to $q_j$ via $q_r$ on $G'$.
- So, $G'$ accepts $w$.
- Suppose $G'$ accepts $w$
  - as each arrow between any states $q_i$ and $q_j$ in $G'$ describes the collection of strings taking $q_i$ to $q_j$ in $G$, either directly or via $q_r$, $G$ must also accept $w$.