Class of Regular Languages is closed under $\cup$, $\circ$, $\ast$

Thanks to the equivalence of DFAs and NFAs, a language is regular iff an NFA accepts it. The characterization in terms of NFAs is very convenient to prove closure under $\cup$, $\circ$, $\ast$ operations.

**Theorem** If $A$, $B$ are regular languages, then so are $A \cup B$, $A \circ B$, $A^\ast$.

**Proof** Let $N_A$, $N_B$ be NFAs that accept $A$ and $B$ respectively. We'll construct an NFA $N$ that accepts $A \cup B$, $A \circ B$, or $A^\ast$ (as the case may be).

![NFA diagrams](image-url)
NFA for $A \cup B$

N has a new start state $\text{Start}$ and $\varepsilon$-moves from it to the start states $q, r$ of $N_A, N_B$ respectively.

NFA for $A \cap B$

Would this work?

N?

Idea is to let $q$ be the start state of $N$ and introduce $\varepsilon$-moves from...
accept states of $N_A$ to $r$. This works, but we must turn the accept states of $N_A$ into non-accept states of $N$ (why?). The final construction is:

$$N$$

$$N_{A}$$

$$N_{B}$$

NFA for $A^*$

$$N$$

Note that $A^*$ consists of all strings $x_1x_2\cdots x_k$ s.t. $x_i \in A$ for every $1 \leq i \leq k$. We can start with $N_A$ and add $\varepsilon$-moves from
its accept states to its start state $q_1$. This ensures that

$x_1$ takes $N_A$ from $q$ to an accept state then using $\varepsilon$-move, back to $q$, then

$x_2$ takes $N_A$ from $q$ to an accept state then using $\varepsilon$-move, back to $q$,

... etc... etc and finally

$x_k$ takes $N_A$ from $q$ to an accept state.

Thus $N$ takes $x_1x_2...x_k$ as input and takes $q$ to an accept state. Therefore $N$ accepts $A^*$. (Almost!)

One needs to handle the string $\varepsilon$ separately. Note that $\varepsilon \in A^*$ (corresponding to $k=0$) even if $\varepsilon \notin A$. To fix this, we add the new start state $\text{Start}$ and add the $\varepsilon$-move

$$\text{Start} \xrightarrow[\varepsilon]{\text{Start}} q_0.$$
Exercise - Give a formal construction of N in terms of $N_A$, $N_B$.
- Write the full argument (in English) that N accepts precisely the language $A \cup B$, $A \cap B$, or $A^*$, as the case may be.

Regular Expressions

DFA/NFAs give a characterization of regular languages in terms of a computational model. We'll see now an equivalent characterization that is syntactic, in terms of regular exprs.

Examples $\Sigma = \{0, 1\}$.

1. The regular expression $\{0 \cup 1\}^* 001$ describes the language $L = \{w \mid w \text{ ends with suffix } 001 \}$.

2. $0 \{0 \cup 1\}^* 0 \cup 1 \{0 \cup 1\}^* 1$ describes
\[ L = \{ w \mid \text{the first and the last symbol of } w \text{ is the same} \} \]

(01 U 10 U 00 U 11)* describes \[ L = \{ w \mid |w| \text{ is even} \}. \]

Formal Inductive Definition

Def. \( R \) is a regular expression (over alphabet \( \Sigma \)) if

- \( R = a \) for some \( a \in \Sigma \)
- \( R = \varepsilon \)
- \( R = \emptyset \)
- \( R = (R_1 U R_2) \) \( \quad \) where \( R_1, R_2 \) are regular expressions defined already.
- \( R = (R_1 \circ R_2) \)
- \( R = (R_1^*) \)

Note While writing, we often omit the parentheses ( ) or the \( \circ \) sign.

The language defined by an expression \( R \) is described inductively in a natural manner.
**Definition:** The language $L(R)$ defined by regular expression $R$ is

- $L(R) = \emptyset$ if $R = \emptyset$
- $L(R) = \{\varepsilon\}$ if $R = \varepsilon$
- $L(R) = \{a\}$ if $R = a, a \in \Sigma$
- $L(R) = L(R_1) \cup L(R_2)$ if $R = R_1 \cup R_2$
- $L(R) = L(R_1) \circ L(R_2)$ if $R = R_1 \circ R_2$
- $L(R) = L(R_1)^*$ if $R = R_1^*$

**Theorem:** A language $L$ is regular iff $L = L(R)$ for some regular expression $R$.

**Proof of $\leftarrow$:**

It is easily observed that if $R$ is a regular expression then $L(R)$ is regular. This follows simply from the above inductive definition of $L(R)$ and that the class of regular languages is closed under $\cup, \circ, \ast$. 
If one wishes, one can build an NFA accepting $L(R)$ by "parsing" the expression "bottom-up". E.g. for the expression $(ab \cup a)^*$, we can first build NFAs for $ab$ and $a$ as

Then we build NFA for $ab \cup a$ using the NFA construction for $\cup$:

Finally NFA for $(ab \cup a)^*$, using the construction for $\ast$.
Proof of \( \Rightarrow \)
We now show that given an NFA, we can construct an equivalent regular expr.

**Example** \( \Sigma = \{a, b\} \)

\[
\begin{array}{c}
\quad a \\
\downarrow \\
q_1 \\
\downarrow b \\
q_2 \\
\downarrow a, b \\
t
\end{array}
\]

Introduce new dummy

Start, accept states

\[
\begin{array}{c}
S \quad \rightarrow \quad q_1 \\
\uparrow a \\
\downarrow b \\
q_1 \\
\downarrow a, b \\
q_2 \\
\uparrow \epsilon \\
t
\end{array}
\]

Eliminate \( q_1 \)

The equivalent regular expression is \( a^* b (a \cup b)^* \).
The general construction can be sketched as:

- **k-State NFA**
- Add new dummy state
- Start, accept states
- **(k+2)-State NFA**
- Eliminate k original states one by one.

**Note** - The final regular expression may depend on order of elimination.

- During this process, one has NFAs whose transition arrows are labeled by regular expressions. Such NFAs may be referred to as **Generalized NFAs**, GNFA.

**Eliminating State q**

This involves the following operation for every $q', q'' \neq q$:

- $R_4 q' \rightarrow q$
- $R_2 q \rightarrow q''$
- $R_3 q'' \rightarrow q'$

After these operations, $q$ is deleted.
It may be the case that $q', q''$ are the same. After all elimination steps, one is left with the GNFA

\[ \sim \rightarrow S \]
\[ \rightarrow R \]
\[ \rightarrow t \]

where $s, t$ are the dummy start, accept states.

$R$ is the final regular expression equivalent to original NFA.

Example

\[ \sim \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \]

\[ \sim \rightarrow q'_1 \rightarrow q'_{2} \rightarrow q'_{3} \]

Add dummy states

```
Eliminate $q_1$
```
$$S \xrightarrow{0^*1} q_2 \xrightarrow{1} q_3 \xrightarrow{1} t$$

Eliminate $q_2$

$$S \xrightarrow{0^*11^*0} q_3$$

Eliminate $q_3$

$$S \xrightarrow{0^*11^*} q_3 \xrightarrow{0^*11^*0 (1 \cup 01^*0)^* 01^*} t$$

$R$ is a regular expression that is equivalent to the NFA $N$. 