Given a regular expression $R$, over an alphabet $\Sigma$, define

$$L(R) = \{ x \in \Sigma^* : x \text{ matches } R \}.$$

**Intent:** how does a string $x$ match $R$?

**Intension:** concentration on the language as a whole.

**Extension:** concentration on the language as a whole.

**Formal Inductive Definition of Regular Expressions, Their Languages, With NFA “Pictures” Too.**

**Basis:** (Lecture: Temporarily use $\sim$ to say something)

- $\emptyset$ is a regexp, $L(\emptyset) = \emptyset$
- $\varepsilon$ is a regexp, $L(\varepsilon) = \{ \varepsilon \}$
- For any char $c \in \Sigma$, $c$ is a regexp, $L(c) = \{ c \}$

**Induction:** Let any two regexps $R_1$ and $R_2$ be given, along with:
- Their languages $L_1 = L(R_1)$ and $L_2 = L(R_2)$
- Their NFA “pictures” $N_1$ and $N_2$ such that $L(N_1) = L_1$ and $L(N_2) = L_2$

*Cannot process any nonempty string.*

*Can process “$c$” but no more.*

*Must process one $c$, so $c \in L(c)$.*
Then: \( R_3 = (R_1 \cup R_2) \) is a regexp [also write \( R_3 = R_1 \cup R_2 \)].

and denotes \( L(R_1) \cup L(R_2) = L(R_3). \)

From the given NFA's \( N_1 \) s.t. \( L(N_1) = L(R_1) \) and \( N_2 \) s.t. \( L(N_2) = L(R_2) \) build \( N_3 = \)

\[
\begin{array}{c}
\text{N_3} \\
\end{array}
\]

\[ s_1 \quad \epsilon \]
\[ \epsilon \quad s_2 \quad \epsilon \]
\[ \epsilon \quad \epsilon \quad \epsilon \quad f_3 \]

build \( N_3 \)

and complete the induction by showing
\[ L(N_3) = L_3 \]
\[ L(R_3) = L_1 \cup L_2. \]

\[ \text{N_3 can process a string x from s_2 to f_3} \Leftrightarrow \]
\[ N_1 \text{ can process x from } s_1 \text{ to } f_1 \text{ or } N_2 \text{ can process x from } s_2 \text{ to } f_2. \]

\[ L(N_3) = L(N_1) \cup L(N_2) = L(R_1) \cup L(R_2) = L_1 \cup L_2 = L_3 \]

by machine construction [by induction hyp. \( L(N_3) = L(R_3) \).]

Then \( R_4 = (R_1 \cdot R_2) \) is a regexp, [parentheses and dot optional]

\[ L(R_4) = \text{def } L(R_1) \cdot L(R_2). \]

Then the NFA's \( N_1 \) and \( N_2 \), build

\[ \begin{array}{c}
N_4:
\end{array} \]

\[ s_1 \quad \epsilon \quad s_2 \quad \epsilon \]
\[ \epsilon \quad s_1 \quad \epsilon \quad s_2 \]
\[ \epsilon \quad \epsilon \quad \epsilon \quad f_4 \]

Then \( N_4 \) can process a string \( x \) from \( s_4 \) to \( f_4 \) \( \Leftrightarrow \) \( x \) can be broken as \( x = yz \) such that \( N_1 \) can process \( y \) from \( s_1 \) to \( f_1 \) and \( N_2 \) can process \( z \) from \( s_2 \) to \( f_2. \)

\[ L(N_4) = L(N_1) \cdot L(N_2) \]

[End of text, Skip this part, which is just for neatness.]
Recall: \( A \cdot B = \{ x : x \text{ can be broken as } y \cdot z \text{ st. } y \in A \land z \in B \} \). So \( L(R_4) = \text{def } L(R_1) \cdot L(R_2) = \{ x : x \text{ can be broken as } y \in L(R_1) \land z \in L(R_2) \} \).

By induction hypothesis: \( L(R_1) = L(N_1) \)
\( L(R_2) = L(N_2) \).

\[ \therefore L(R_4) = \{ x : x \text{ can be broken as } y \in L(N_1) \land z \in L(N_2) \} \]
\[ = L(N_1) \cdot L(N_2) = L(N_4) \]
\[ \therefore L(N_4) = L(R_1) \cdot L(R_2) \]
\[ = L(R_4) \cdot \emptyset \]

\( \star \)

Finally define \( R_S = (R_1)^* \) [\( R_2 \) not involved]. And define \( L(R_S) = L(R_1)^* \)
\[ = \{ x \in \Sigma^* : x \text{ can be broken as } \text{ such that } \forall k \in L(R_1) \} \]

Build \( N_S \) given \( N_1 \) like so:

**Feedback Circuit** (with bypass)

Then \( L(N_S) = \exists x : x \text{ can be broken into zero or more } L(R_1) \) substrings each processed by \( N_1 \), \( L(R_S) \)

**Diagram Construction**

**Theorem**: For every regular expression \( R \), we can (recursively!) build an NFA \( N_R \) such that \( L(N_R) = L(R) \).
Example 1: \( R = 1 \cdot 0^* \cdot N \cdot 0^* \cdot N_1 \cdot 0^* \cdot N \),
\[ N = \{ s, 0 \} \]
Literal Recursion.

Much simpler (closer to text):
Formally an NFA but has no actual non-determinism. Hence can complete to a DFA.

Example 2: \((0+1)^* 010 (0+1)^* \cdot \)
"Aesthetic Point": The NFA \( N \) is the most immediate picture for \( 1^* \cdot 0^+ \cdot M \).

Has actual non-determinism, so DFA more complicated. "Nirvana" state

Extra - added after lecture:

Example 3: \( R = 1^* 0^* \cdot \) Wrong is \( q_0 \) - that is \( (1+0)^* \)
Correct is \( \cdot \)\( q_0 \)\( \epsilon \)\( q_0 \)
Thus \( \epsilon \)-arcs are sometimes helpful.

Also good is \( \cdot \)\( q_0 \)\( \epsilon \)\( q_0 \)\( q_0 \)
This needs the start state to be accepting too. Can be completed to a DFA as shown in blue.

Example 4:
\( R = 1^* 0^* \cdot \)
\( \cdot \)\( q_0 \)\( \epsilon \)\( q_0 \)\( q_0 \)
\( q_0 \)\( \epsilon \)\( q_0 \)\( q_0 \)\( q_0 \)
\( q_0 \)\( \epsilon \)\( q_0 \)\( q_0 \)\( q_0 \)
OFA is as shown in blue.