Lecture 7 — Regular Expressions and Finite Automata

CSE205: Theory of Computation

Minseok Jeon

2026 Fall

Equivalence of Regular Expressions and Finite Automata

Regular expressions and finite automata describe *exactly* the same class of languages — the *regular languages*.

Theorem (From RE to FA)

Every language defined by a regular expression is also defined by a finite automaton.

Theorem (From FA to RE)

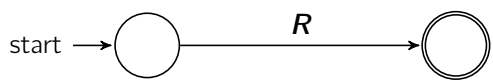
Every language defined by a finite automaton is also defined by a regular expression.

We prove both directions constructively.

From Regular Expression to Finite Automaton

Given a regular expression $\boldsymbol{R}$, we build an $\boldsymbol{\varepsilon}$-NFA accepting $\boldsymbol{L}(\boldsymbol{R})$ with a special *normal form*:

- it has exactly **one accepting state**,
- **no arcs into** the initial state, and
- **no arcs out of** the accepting state.

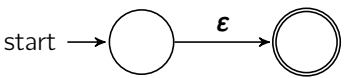


The construction is by *structural induction* on $\boldsymbol{R}$ (the *Thompson construction*). The normal form makes it easy to plug sub-automata together.

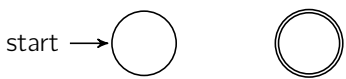
RE to FA: Base Cases

Base cases.

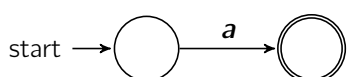
$R = \varepsilon$:



$R = \emptyset$:



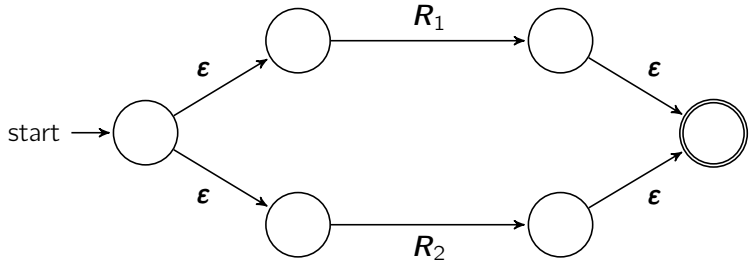
$R = a$ $(\in \Sigma)$:



For $R = \emptyset$ there is no transition, so no string is accepted.

RE to FA: Inductive Case — Union $R_1 + R_2$

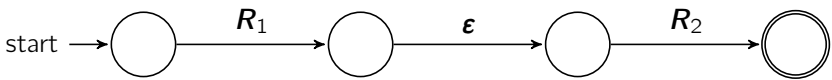
Let A_1, A_2 be the automata for R_1, R_2. Add a new start state and a new accept state, joined by ε-transitions:



A string is accepted iff it is accepted by A_1 *or* A_2, so the language is $L(R_1) \cup L(R_2)$.

RE to FA: Inductive Case — Concatenation $R_1 R_2$

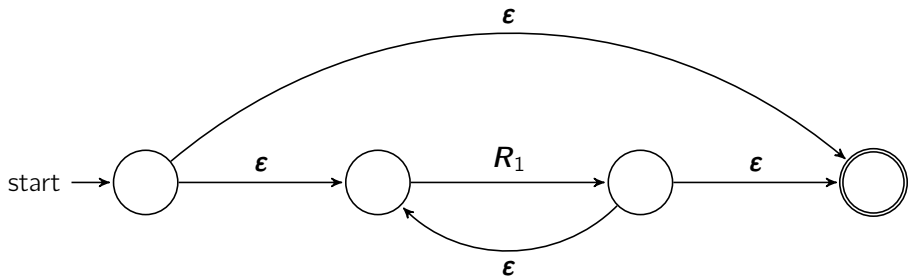
Place A_1 before A_2, linking the accept state of A_1 to the start state of A_2 by an ε-transition:



The new start state is that of A_1; the new accept state is that of A_2. The language is $L(R_1)\,L(R_2)$.

RE to FA: Inductive Case — Star R_1^*

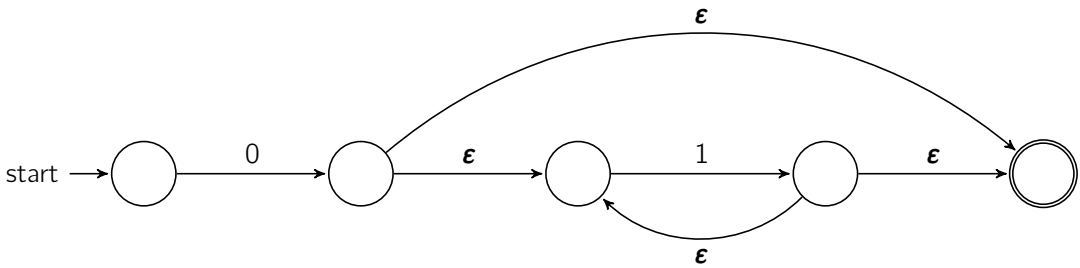
Add a new start state and a new accept state. An $\boldsymbol{\varepsilon}$-transition lets us skip $\boldsymbol{A}_1$ (for $\boldsymbol{\varepsilon}$); a back $\boldsymbol{\varepsilon}$-transition lets us repeat it:



The top $\boldsymbol{\varepsilon}$-arc from start to accept admits the empty string; the lower back-arc admits one or more repetitions of $\boldsymbol{R}_1$. The language is $(\boldsymbol{L}(\boldsymbol{R}_1))^*$.

Example: $0 \cdot 1^*$

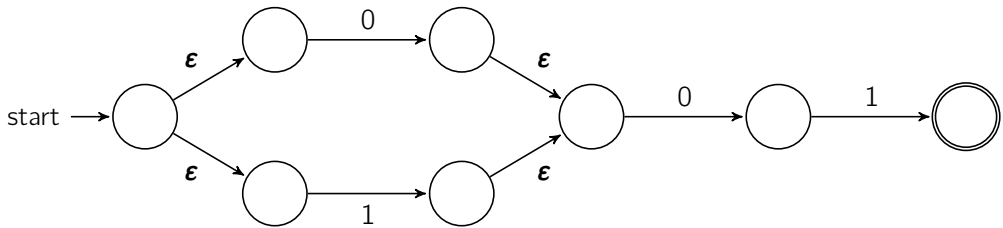
Build 1^* first, then prepend 0.



This $\boldsymbol{\varepsilon}$-NFA accepts $\{0, 01, 011, 0111, \dots\} = \boldsymbol{L}(0\,1^*)$.

Example: $(0 + 1) \cdot 0 \cdot 1$

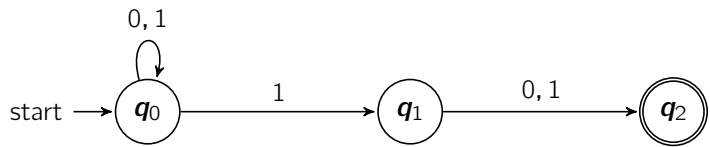
First $(0 + 1)$, then concatenate 0 and 1.



Accepts $\{001, 101\} = \boldsymbol{L}((0 + 1)\,0\,1)$.

Example: $(0+1)^* \cdot 1 \cdot (0+1)$

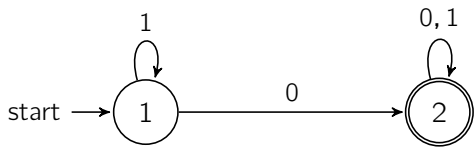
A simplified (NFA) view: guess where the marked 1 occurs — it accepts exactly the strings whose *second-to-last* symbol is 1.



The full Thompson $\boldsymbol{\varepsilon}$-NFA can be built from the union, concatenation, and star constructions; the NFA above is an equivalent, smaller machine accepting $\boldsymbol{L}\big((0+1)^*\,1\,(0+1)\big)$.

From Automaton to Regular Expression

Consider a DFA $\boldsymbol{D}$ with states $\{1, 2, \ldots, \boldsymbol{n}\}$, for example:



Idea. Progressively allow more paths through the graph. Let $\boldsymbol{R}_{\boldsymbol{ij}}^{(\boldsymbol{k})}$ name a regular expression for the set of labels of paths from state $\boldsymbol{i}$ to state $\boldsymbol{j}$ such that no *intermediate* node has a number greater than $\boldsymbol{k}$.

From Automaton to RE: Base Case $k = 0$

With $k = 0$, paths may use no intermediate states — only direct arcs.

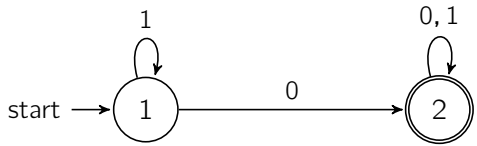
When $i \neq j$: consider every arc $i \xrightarrow{a} j$.

- no such arc: $R_{ij}^{(0)} = \emptyset$;
- exactly one arc labelled a: $R_{ij}^{(0)} = a$;
- arcs $a_1, \ldots, a_m$: $R_{ij}^{(0)} = a_1 + a_2 + \cdots + a_m$.

When $i = j$: also include ε (the empty path).

- no self-loop: $R_{ij}^{(0)} = \varepsilon$;
- one self-loop a: $R_{ij}^{(0)} = \varepsilon + a$;
- self-loops $a_1, \ldots, a_m$: $R_{ij}^{(0)} = \varepsilon + a_1 + a_2 + \cdots + a_m$.

Example: Base Case



$$R_{11}^{(0)} = \varepsilon + 1$$
$$R_{12}^{(0)} = 0$$
$$R_{21}^{(0)} = \emptyset$$
$$R_{22}^{(0)} = \varepsilon + 0 + 1$$

From Automaton to RE: Inductive Step $k > 0$

A path from i to j using no intermediate node greater than k falls into two cases:

1. The path does **not** use state k at all.
   Its label is in $R_{ij}^{(k-1)}$.
2. The path goes **through** state k one or more times:
   go from i to the first visit of k, loop on k zero or more times, then go from k to j:

$$R_{ik}^{(k-1)} \left(R_{kk}^{(k-1)}\right)^* R_{kj}^{(k-1)}.$$

Combining the two cases:

$$\boxed{R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} \left(R_{kk}^{(k-1)}\right)^* R_{kj}^{(k-1)}}$$

The final answer is the union of $R_{ij}^{(n)}$ over the start state i and all accepting states j.

Example: Inductive Step ($k = 1$)

Using $R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)}(R_{kk}^{(k-1)})^* R_{kj}^{(k-1)}$ with $k = 1$:

$$
\begin{aligned}
R_{11}^{(1)} &= (\varepsilon + 1) + (\varepsilon + 1)(\varepsilon + 1)^*(\varepsilon + 1) = 1^* \\
R_{12}^{(1)} &= 0 + (\varepsilon + 1)(\varepsilon + 1)^* 0 = 1^* 0 \\
R_{21}^{(1)} &= \emptyset + \emptyset(\varepsilon + 1)^*(\varepsilon + 1) = \emptyset \\
R_{22}^{(1)} &= (\varepsilon + 0 + 1) + \emptyset(\varepsilon + 1)^* 0 = \varepsilon + 0 + 1
\end{aligned}
$$

Example: Inductive Step ($k = 2$)

Now $k = 2$:

$$
\begin{aligned}
R_{11}^{(2)} &= 1^* + 1^*0\,(\varepsilon + 0 + 1)^*\emptyset = 1^* \\
R_{12}^{(2)} &= 1^*0 + 1^*0\,(\varepsilon + 0 + 1)^*(\varepsilon + 0 + 1) = 1^*0\,(0 + 1)^* \\
R_{21}^{(2)} &= \emptyset + (\varepsilon + 0 + 1)(\varepsilon + 0 + 1)^*\emptyset = \emptyset \\
R_{22}^{(2)} &= (\varepsilon + 0 + 1) + (\varepsilon + 0 + 1)(\varepsilon + 0 + 1)^*(\varepsilon + 0 + 1) = (0 + 1)^*
\end{aligned}
$$

State 1 is the start state and state 2 is accepting, so

$$L(D) = R_{12}^{(2)} = 1^*0\,(0 + 1)^*.$$

State Elimination (Generalized NFA)

An equivalent, often more practical method is *state elimination*.

- Allow arcs to be labelled by *regular expressions* — a **generalized NFA** (GNFA).
- Add a new start state (with an $\boldsymbol{\varepsilon}$-arc to the old start) and a new single accept state (with $\boldsymbol{\varepsilon}$-arcs from the old accepting states).
- Repeatedly **eliminate** an intermediate state $\boldsymbol{q}$: for every pair of remaining states $\boldsymbol{p} \to \boldsymbol{q} \to \boldsymbol{r}$, replace the path by a direct arc

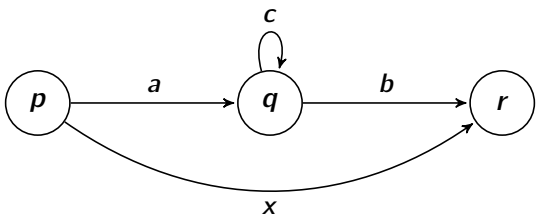
$$\boldsymbol{p} \xrightarrow{R_{pq}\,(R_{qq})^*\,R_{qr}} \boldsymbol{r},$$

  combining with any existing $\boldsymbol{p} \to \boldsymbol{r}$ arc by union.
- When only the start and accept states remain, the label on the single arc between them is the resulting regular expression.

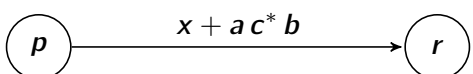
Elimination Step, Concretely: General Case

Eliminating q replaces the two-hop path (and any pre-existing direct arc) by a single arc:

Before:

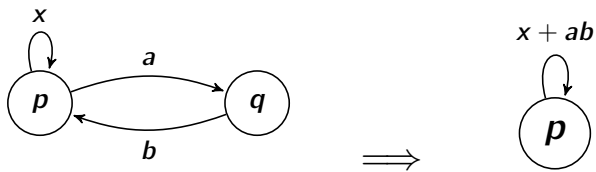


After (eliminate q; x is any pre-existing $p \to r$ label, or $\emptyset$ if there was none):



Elimination Step, Concretely: A Detour Becomes a Self-Loop

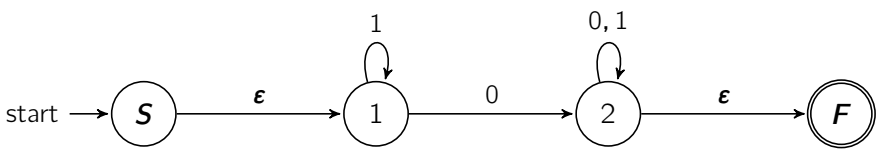
If $p = r$, eliminating q *adds to* p*'s own self-loop* instead of creating a new arc between two states.



Before (left): self-loop x at p, plus $p \to q \to p$. **After** (right): eliminate q, giving self-loop $x + ab$ at p.

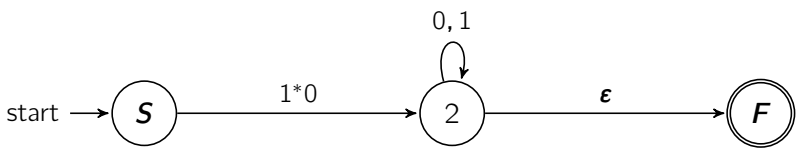
Example: State Elimination — Setup

The same DFA from earlier (recognizing $1^*0(0+1)^*$). Add a new start S (with ε to state 1) and new accept F (with ε from state 2) to form a GNFA:



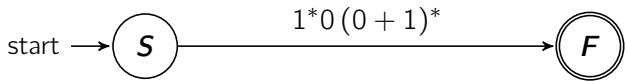
Example: State Elimination — Eliminate State 1

State 1 has incoming arc $S \xrightarrow{\varepsilon} 1$, self-loop $R_{11} = 1$, and outgoing arc $1 \xrightarrow{0} 2$ (no other outgoing arc). Eliminating 1 adds $S \xrightarrow{\varepsilon \cdot 1^* \cdot 0} 2$, i.e. $S \xrightarrow{1^*0} 2$:



Example: State Elimination — Final Result

State 2 has incoming arc $S \xrightarrow{1^*0} 2$, self-loop $R_{22} = 0 + 1$, and outgoing arc $2 \xrightarrow{\varepsilon} F$. Eliminating 2 leaves a single arc from S to F:



$$\boxed{L(D) = 1^*0\,(0+1)^*}$$

This matches $R_{12}^{(2)} = 1^*0\,(0+1)^*$ computed earlier with the $R_{ij}^{(k)}$ recurrence — two different methods, same answer.

Summary

- Regular expressions and finite automata are **equivalent**: they define exactly the regular languages.
- **RE** $\rightarrow$ **FA**: Thompson's construction builds an $\boldsymbol{\varepsilon}$-NFA by structural induction (base cases, union, concatenation, star).
- **FA** $\rightarrow$ **RE**: the $R_{ij}^{(k)}$ recurrence (or equivalently state elimination on a GNFA) yields a regular expression.
- Therefore regular expressions, DFAs, NFAs, and $\boldsymbol{\varepsilon}$-NFAs all have the same expressive power.