Last class:

Introduced regular expressions.
Began proving the equivalence of regular expressions and finite automata.

Today:

Construct a regular expression for any finite automaton.

Assignments:

Assignment 1 due today, start of class.
Assignment 2 out today, due in a week.

Exam I:

Tentatively scheduled for Oct. 2 (two weeks from today)
Survey: In-class vs take-home? Trade offs...
**Regular expressions**

**DEFINITION.** $R$ is a *regular expression* if $R$ is

1. $a$ for some $a \in \Sigma$
2. $\varepsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1$ and $R_2$ are regular expressions
5. $(R_1 \cdot R_2)$, where $R_1$ and $R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression

Every regular expression arises by a finite number of applications of these six rules.

---

**Equivalence of FA languages and RE languages**

Regular expressions and finite automata have equivalent descriptive power.

We can convert any regular expression into a finite automaton that recognizes the language it describes — and vice versa.

---

**Constructing an NFA-$\varepsilon$ from a regular expression**

Cover the six cases in the definition of REs:

1. $R = a$ for some $a \in \Sigma$. Then $L(R) = \{a\}$ and the following NFA recognizes $L(R)$

2. $R = \varepsilon$

3. $R = \emptyset$

The languages accepted by DFA, NFA, NFA-$\varepsilon$, and described by RE are called the *regular languages*
4. $R = (R_1 \cup R_2)$

For two languages $R_1$ and $R_2$, take two NFAs $N_1$ and $N_2$ and combine them into one new NFA $N$. $N$ must accept input if either $N_1$ or $N_2$ accepts input.

5. $R = (R_1 \cdot R_2)$

For two languages $R_1$ and $R_2$, take two NFAs $N_1$ and $N_2$ and combine them sequentially into one new NFA $N$.

6. $R = (R_1)^*$

For a language $R_1$, modify $N_1$ to accept $(R_1)^*$.

Example

$ab(a \cup a)^*$
Proof

We will prove this set of equivalences by

✓ Showing how to construct an NFA-$\varepsilon$ from a regular expression

Showing how to construct a regular expression from a finite automaton

Constructing a regular expression from a finite automaton

Two algorithms:

State elimination: gives smaller expression, in general, and easier to apply.

Inductive construction: covered in the appendix slides.

DFA-to-RE by state elimination

Basic idea:

Eliminate a state $s$ (i.e., remove all arcs into and out of $s$)

Label arcs from $q$ to $p$ that went through $s$ with a regular expression representing the sequence of symbols on that path.

Generalized nondeterministic finite automaton (GNFA)

An NFA where the transition arrows can have a regular expression as the label.

It can read blocks of symbols from the input rather than just one symbol at a time.

Used as an intermediate state in this proof by construction.
General process

- Remove state \( q \)
- Label new paths from
  - \( q \) to \( q \)
  - \( q \) to \( q \)
  - \( q \) to \( q \)
  - \( q \) to \( q \)

Alternative method

We can simplify things if we ensure the following before applying the procedure for state elimination:

- There is a single final state
- There are no transitions into the initial state
- There are no transitions out of the final state

Since the procedure works on NFA-\( \epsilon \)s, this is easy to do:

Procedure

Before

After

Example

Add new start and end state

Remove state 2

Remove state 1

Remove state 1
Example

STEP 1: Modify to create a unique start and end state:

STEP 2: Eliminate state 1
path from s to 2 is $a^*b$
path from 3 to 2 is $aa^*b$.

STEP 3: Eliminate state 2:
path from s to 3 is $a^*bb^*a$
path from s to f is $a^*bb^*$
path from 3 to f is $(b \cup aa^*b)b^*$
path from 3 to 3 is $(b \cup aa^*b)b^*a$

STEP 4: Eliminate state 3:
label on the path from s to f yields the final regular expression:

$$a^*bb^* \cup (a^*bb^*a((b \cup aa^*b)b^*a)((b \cup aa^*b)b^*a)^* \cup \varepsilon))$$
Another example

Remove state 1

Simplify

Remove state 2

Remove state 3

Remove state 4

Done!
Proof

We will prove this set of equivalences by

✓ Showing how to construct an NFA-ε from a regular expression
✓ Showing how to construct a regular expression from a finite automaton

The regular languages are recognized by NFA-εs, NFAs, DFAs, and REs (...and GNFAs)

What about languages that aren’t regular languages?
Next time!

Appendix

Inductive construction

Let A be a FA with states 1, 2, … n.

Let \( R_{ij}^{(k)} \) be a RE whose language is the set of labels of paths that go from state i to state j without passing through any state numbered above k.

Construction, and the proof that the expressions for these REs are correct, are inductions on \( k \).
**Basis:** $k = 0$. Path can’t go through any states.

Thus, path is either an arc or the null path (a single node).

If $i \neq j$, then $R_{ij}^{(0)}$ is the sum of all symbols $a$ such that $A$ has a transition from $i$ to $j$ on symbol $a$ ($\emptyset$ if none).

If $i = j$, then add $\varepsilon$ to above.

**Induction:** Assume we have correctly developed expressions for the $R_{ij}^{(k-1)}$s. Then for the $R_{ij}^{(k)}$s:

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

**Proof it works:** A path from $i$ to $j$ that goes through no state higher than $k$ either:

Never goes through $k$, in which case the path’s label is (by the IH) in the language of $R_{ij}^{(k-1)}$; or

Goes through $k$ one or more times. In this case:
- $R_{ik}^{(k-1)}$ contains the portion of the path that goes from $i$ to $k$ for the first time.
- $(R_{kk}^{(k-1)})^*$ contains the portion of the path (possibly empty) from the first $k$ visit to the last.
- $R_{kj}^{(k-1)}$ contains the portion of the path from the last $k$ visit to $j$.

**Final step:** The RE for the entire FA is the sum (union) of the RE’s $R_{ij}^{(n)}$, where $i$ is the start state and $j$ is one of the accepting states.

Note that superscript $(n)$ represents no restriction on the path at all, since $n$ is the highest-numbered state.
Example
The “clamping” automaton, with states named by integers:

![Automaton Diagram]

Some basis expressions:

\[
\begin{aligned}
R_{11}^{(0)} &= \varepsilon \\
R_{12}^{(0)} &= 1 \\
R_{22}^{(0)} &= \varepsilon + 0 + 1 \\
R_{21}^{(0)} &= 1 \\
R_{32}^{(0)} &= R_{21}^{(0)} = \emptyset
\end{aligned}
\]

Two inductive examples:

\[
\begin{aligned}
R_{32}^{(0)} &= R_{32}^{(0)} \cup R_{31}^{(0)}(R_{11}^{(0)} \ast R_{12}^{(0)}) = \emptyset \cup 1 \varepsilon \ast 1 = 11
\end{aligned}
\]

*Uses algebraic laws:

- \( \varepsilon \) is the identity for concatenation:
  \[
  \begin{aligned}
  \varepsilon \ast \varepsilon &= \varepsilon \\
  R\varepsilon &= \varepsilon R = R
  \end{aligned}
  \]

- \( \emptyset \) is the identity for union:
  \[
  \emptyset \cup R = R \cup \emptyset = R
  \]

\[
\begin{aligned}
R_{22}^{(1)} &= R_{22}^{(0)} \cup R_{21}^{(0)}(R_{11}^{(0)} \ast R_{12}^{(0)}) = \varepsilon \cup 0 \cup 1 \cup \emptyset \varepsilon \ast 1 = \varepsilon \cup 0 \cup 1
\end{aligned}
\]

*Additional algebraic law used:

- \( \emptyset \) is the annihilator for concatenation:
  \[
  \emptyset R = R \emptyset = \emptyset
  \]