Lecture 11

Regular Expressions and Non-regular Languages

07-30

Recap

Regular Languages

The following are equivalent

- A is regular
- There is a DFA M such that $L(M) = A$
- There is a NFA N such that $L(N) = A$

Closure

If A, B are regular, so are

- $\overline{A}$ ← flip the states
- $A \cup B$
- $A \cap B$

} product construction. "parallel".

- A/B
- A^n
- A^*

} using NFAs.

Regular Expressions

Let Σ be an alphabet. Define the set of regular expressions $\mathcal{R}_\Sigma$ recursively as follows.

$\mathcal{R}_\Sigma$ is the smallest set such that

- $\emptyset \in \mathcal{R}_\Sigma$
- $\epsilon \in \mathcal{R}_\Sigma$
- $a \in \mathcal{R}_\Sigma$ for each $a \in \Sigma$
- $R \in \mathcal{R}_\Sigma \implies (R)^* \in \mathcal{R}_\Sigma$
- $R_1, R_2 \in \mathcal{R}_\Sigma \implies (R_1 R_2) \in \mathcal{R}_\Sigma$
- $R_1, R_2 \in \mathcal{R}_\Sigma \implies (R_1 | R_2) \in \mathcal{R}_\Sigma$

Regular Expressions

The language if a regular expression R, denoted $L(R)$ is the set of strings that R matches. Formally,

- $L(\emptyset) = \emptyset$
- $L(\epsilon) = \{\epsilon\}$
- $L(a) = \{a\}$, for $a \in \Sigma$

Base case

- $L(R^*) = L(R)^*$ for $R \in \mathcal{R}^*_\Sigma$
- $L(R_1 R_2) = L(R_1)L(R_2)$ for $R_1, R_2 \in \mathcal{R}_\Sigma$
- $L(R_1|R_2) = L(R_1) \cup L(R_2)$ for $R_1, R_2 \in \mathcal{R}_\Sigma$

inductive steps.

if $A \subseteq \Sigma^*$ is a language:

$\rightarrow A^n$: $\underbrace{AA \ldots A}_{n} = \{ a_1 a_2 \ldots a_n : a_1, \ldots, a_n \in A \}$.

$$A^* = \bigcup_{i=0}^{\infty} A^i = \{ a_1 \ldots a_k : \begin{matrix} k \in \mathbb{N}, \\ a_i, \ldots, a_k \in A \end{matrix} \}.$$

Today

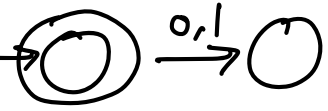
- NFA $\iff$ Regular Expressions (!!!!)
- Non-regular languages (!!!!)

Equivalence of NFAs and Regular Expressions

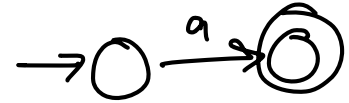
Regex -> NFA

$\forall R \in R_\Sigma$, $\exists$ a NFA, N s.t. $L(R) = L(N)$.

Base cases: Language.

ε : $\{\varepsilon\}$ 

$\emptyset$: $\{\}$ 

$a \in \Sigma$. : $\{a\}$ 

Inductive step. suppose R, S are regular expressions and suppose they are regular ($\exists$ a NFA, N_R $L(N_R)=L(R)$, N_S : $L(N_S)=L(S)$).

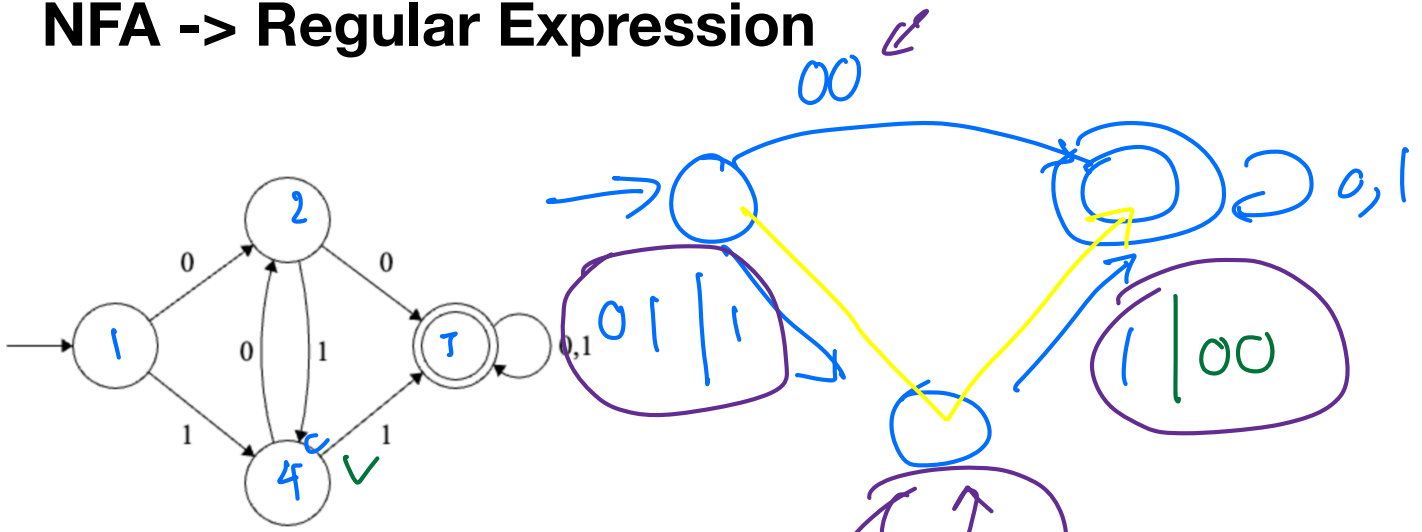
WTS: RS, R^*, R|S are regular.

$L(RS) = L(R)L(S)$, $L(R|S) = L(R) \cup L(S)$.

$L(R^*) = L(R)^*$ ... by closure properties

Equivalence of NFAs and Regular Expressions

NFA -> Regular Expression



00

0,1

01 | 1

1 | 00

01

(01|1)(01)*(1|00)|00

0

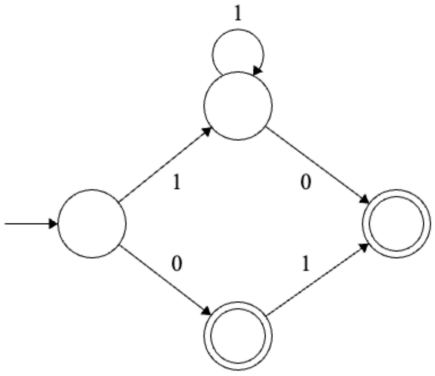
0

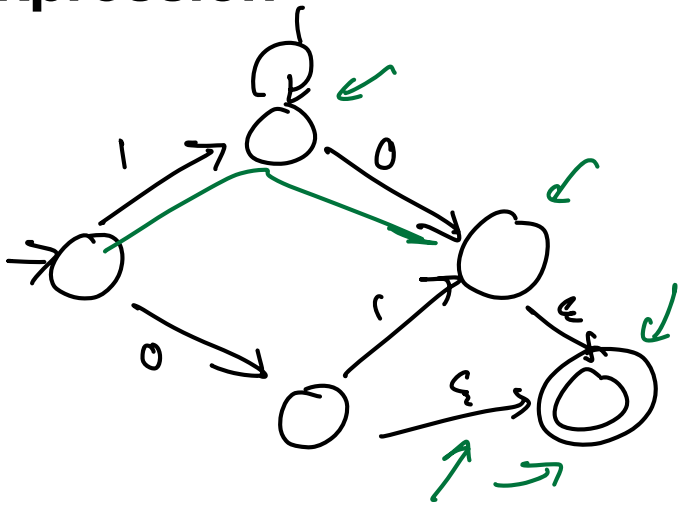
0,1

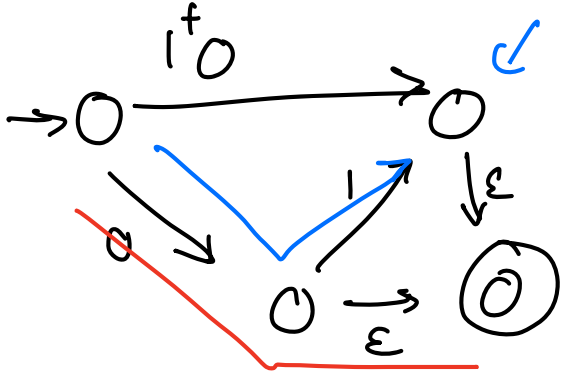
((01|1)(01)*(1|00)|00){0,1}*

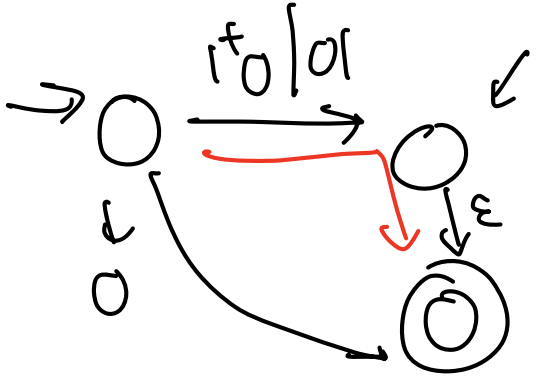
Equivalence of NFAs and Regular Expressions

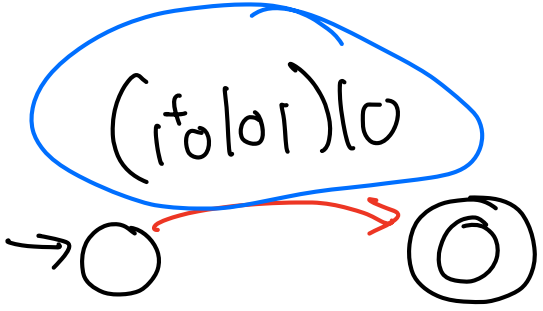
NFA -> Regular Expression





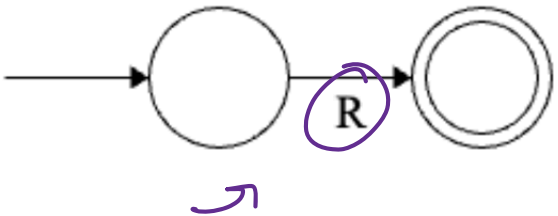






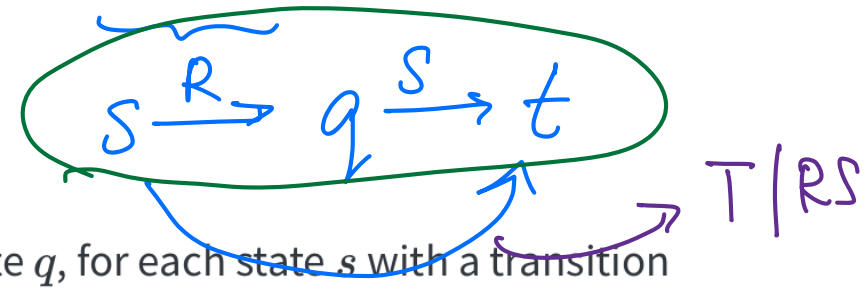
Sketch of Formal Proof

- Alter the NFA so there's just one accepting state (using ϵ transitions).
- Iteratively rip out states, replacing transitions with regular expressions until you have something that looks like



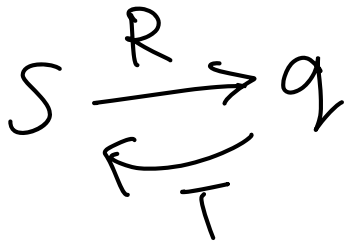
R is the equivalent regular expression.

For two states q_1, q_2 with a transition between them, let $f(q_1, q_2)$ be the regular expression labelling the transition.



Here are the steps to rip out a state q.

1. **Remove the loop**: If there is a self loop on state q, for each state s with a transition into q, update the transition $f(s, q) = f(s, q)f(q, q)^*$. For each state s' with a transition out of q, update the transition $f(q, s') = f(q, q)^* f(q, s')$
2. **Bypass** q: for each path (s, q, t) of length 2 through q, update $f(s, t) = f(s, t)|f(s, q)f(q, t)$. Note that it is possible that $s = t$, in which case this step adds a loop.
3. Remove q.



Regular Languages

The following are equivalent

- A is regular
- There is a DFA M such that $L(M) = A$
- There is a NFA N such that $L(N) = A$
- **There is a regular expression R such that $L(R) = A$**

Showing a language is regular

- Find either a DFA, NFA, or Regular Expression for the language!
-

How to choose

- I typically use regular expressions for languages that seem to require some form of 'matching'. For example *contains 121* as a substring, or *ends with 11*. Regular expressions are typically faster to find and write out in an exam setting.

- I'll use NFAs when I can't easily figure out a regular expression for something. These are usually languages for which memory seems to be useful like the Dogwalk example from hw.

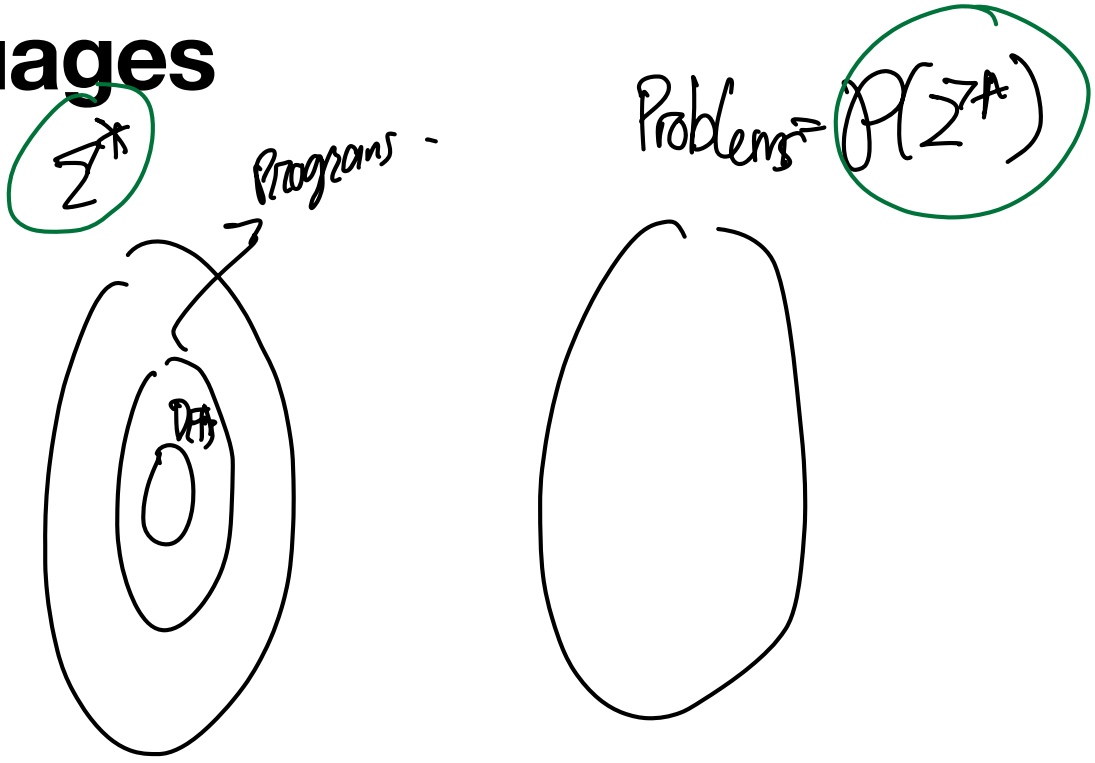
- Stuff involving negations also seems easier to do with NFAs than with regular expressions. For example, *contains the substring* 011 is easy with regular expression, but *doesn't contain the substring* 011 is a bit more complicated.

- Alternatively, decompose the language into simpler languages and use closure properties.

- Finally, if the language is finite it's automatically regular (in HW).

Non-Regular Languages

- Are all languages regular?



Key Intuition

↳ for regular languages.

- Regular ⟺ Computable with "finite memory" → DFA

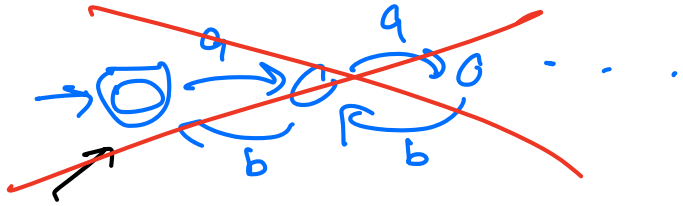
finite.

finite # of states.

fixed.

Example

$\Sigma = \{a, b\}$.



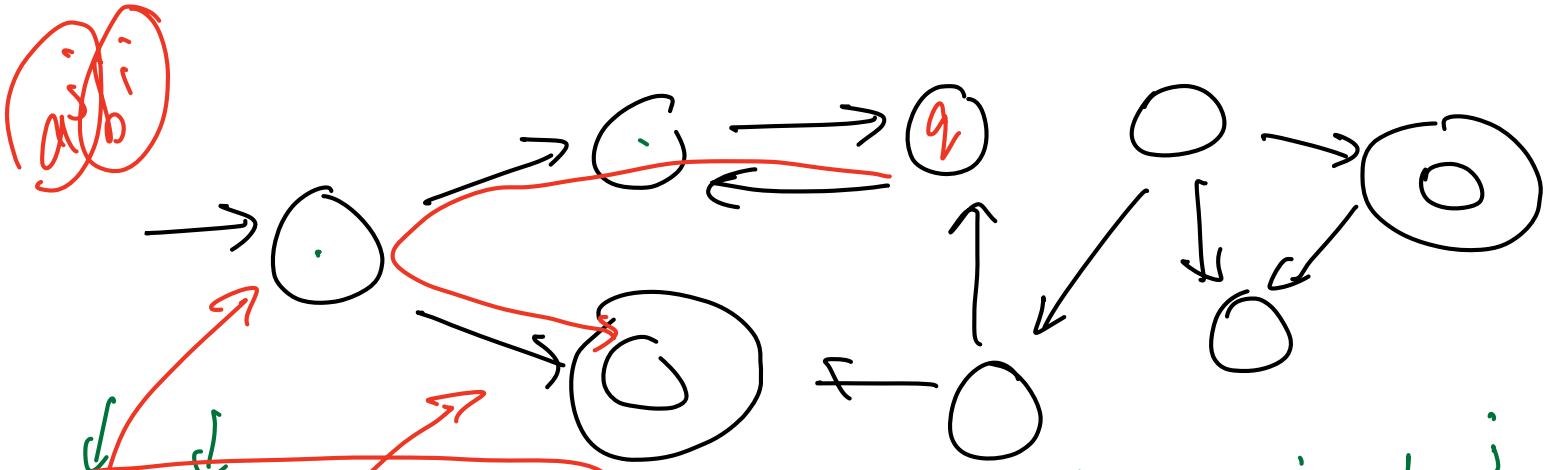
- Let $X = \{a^n b^n : n \in \mathbb{N}\}$. Claim: X is not regular
- Why is this the case, using the intuition from the previous slide?

↳ need to keep track of how many as I've read.

a very large number.

$X = \{a^n b^n : n \in \mathbb{N}\}$.

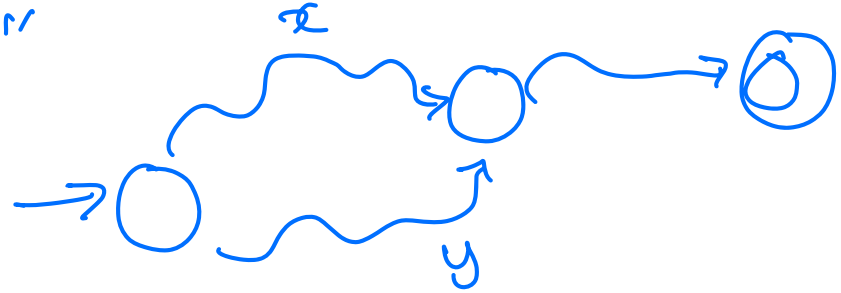
Proof By contradiction, suppose M is a DFA for X. Suppose M has k states.



By P.P. $\exists\, a^i, a^j$ s.t. a^i and a^j end up @ the same state q, and $i \neq j$.

suppose I now read b^i from q. Then $a^i b^i \in X$, so I should reach an accept state. but this means I also accept $a^j b^i$ contradiction $\notin X$.

Same State, Same Fate



- If two strings x and y reached the same state, then no matter what string w comes after, xw and yw will end up in the same state and hence will both be accepted or both be rejected
- Equivalently, different fates → different states
- X has infinitely many strings that have different fates, hence, there must be infinitely many states!

Distinguishable

two strings $x, y \in \Sigma^*$ are distinguishable relative to a language A iff $\exists\ w \in \Sigma^*$ s.t.

- $xw \in A$, $yw \notin A$

OR

- $xw \notin A$, $yw \in A$.

in a DFA for A.

Same state same fate: if x and y reach the same state then x and y are not distinguishable relative to A.

Myhill-Nerode Theorem

S can be any subset of strings. Common misconception is that $S \subseteq A$.

Let A be a language over Σ. Suppose there exists a set $S \subseteq \Sigma^*$ with the following properties

- (Infinite). S is infinite
- (Pairwise distinguishable). $\forall x, y \in S$, with $x \neq y$. x, and y are distinguishable relative to A.

Then A is not regular.

Proof: By contradiction, suppose M is a DFA for A.
Let Q be the set of states of M.
Let $f : S \to Q$ be a mapping of strings in S to the states they end up in. S is infinite, Q is finite. By P.P. $\exists\, x, y \in S$ $x \neq y$ s.t. $f(x) = f(y)$
Let's call $f(x) = f(y) = q$. Then x, and y are indistinguishable relative to A, which is a contradiction, b/c

Using the Myhill-Nerode Theorem

- By the Myhill-Nerode Theorem, to show a language A is not regular, it suffices to find a set $S \subset \Sigma^*$ such that S is infinite, and pairwise distinguishable relative to A.

Example

Showing X is not regular using the Myhill-Nerode Theorem.

$X = \{a^n b^n : n \in \mathbb{N}\}$.

By Myhill-Nerode Theorem, it suffices to find $S \subseteq \Sigma^*$ s.t. S is infinite & pairwise distinguishable. Claim: $S = \{a^i : i \in \mathbb{N}\}$.

① S is infinite, since S contains one string for each natural number.

② S is pairwise distinguishable. Let $x, y \in S$ s.t. $x \neq y$. WTS x, y are distinguishable. Suppose $x = a^i$, $y = a^j$ s.t. $i \neq j$. The we claim that b^i distinguishes them. Indeed $xw = a^i b^i \in X$, whereas $yw = a^j b^i \notin X$.