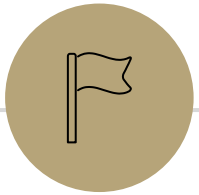
Irregular Languages

CSE 311 Summer 2025
Lecture 21

Announcements

- HW7 is out and due this Friday, 8/15
- HW6 Resubmissions will open today and are due this Friday



Review

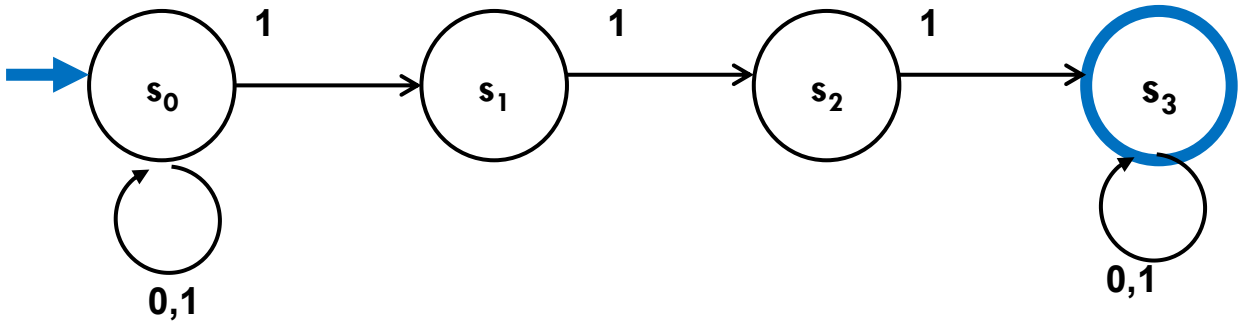
Nondeterministic Finite Automata

An NFA:

Still has exactly one start state and any number of final states.

The NFA accepts x if there is some path from a start state to a final state labeled with x.

From a state, you can have 0,1, or many outgoing arrows labeled with a single character. You can choose any of them to build the required path.



Three ways to think about NFAs

"Outside Observer": is there a path labeled by x from the start state, to the final state (if we know the input in advance can we tell the NFA which decisions to make)

"Perfect Guesser": The NFA has input x, and whenever there is a choice of what to do, it **magically** guesses a transition that will eventually lead to acceptance (if one exists)

"Parallel exploration": The NFA computation runs all possible computations on x in parallel (updating each possible one at every step)

Regularity

So NFAs/DFAs what can and can't they do?

Can NFAs do more than DFAs?

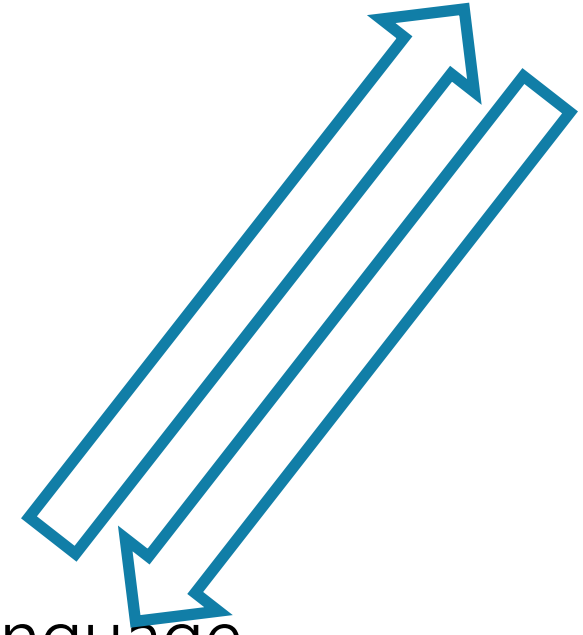
How do they relate to context-free-grammars? Regular expressions?

Kleene's Theorem

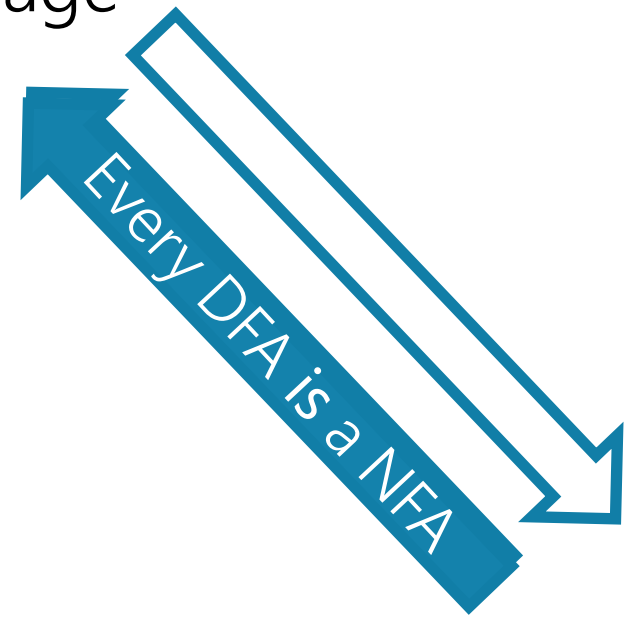
For every language L:
L is the language of a regular expression if and only if
L is the language of a DFA if and only if
L is the language of an NFA

Proof [sketch]

L is the language of an NFA.



L is the language of a regular expression.



L is the language of a DFA.

Suppose L is the language of some DFA M. M also satisfies the requirements for an NFA, so L is also the language of an NFA.

More formally (the "powerset construction")

The original NFA

States: Q

Start state: q_0

Transition function: $\delta(q, a)$

Outputs set of all states reachable from q using one a transition (and any number of ε-transitions)

Final States: F

The constructed DFA

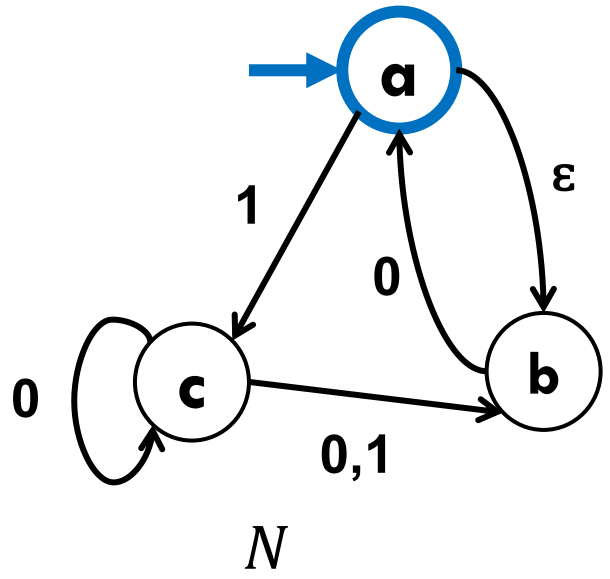
States: $\mathcal{P}(Q)$

Start state: $\{q' : q'$ reachable from q_0 with only ε-transitions $\}$

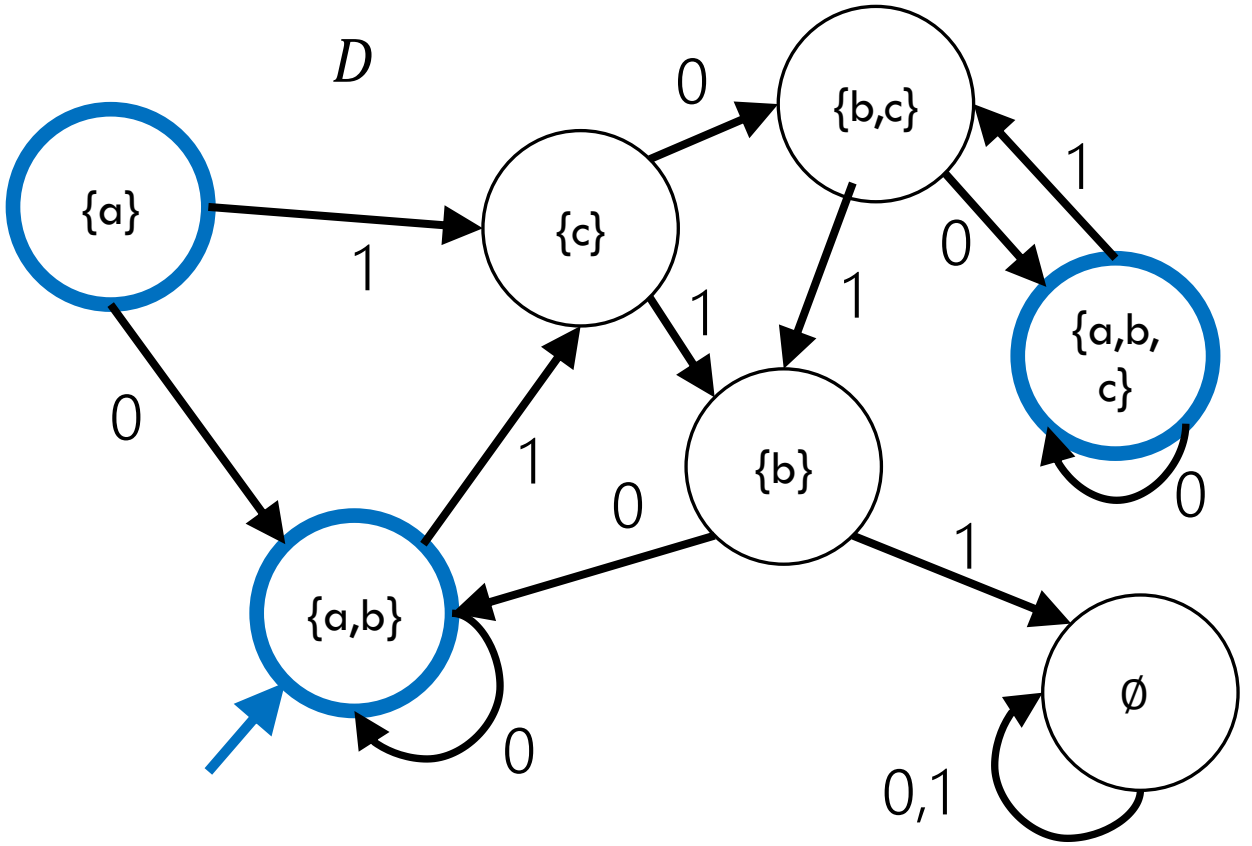
Transition function: $\delta_D(S, a) = \mathrm{U}_{q \in S}\, \delta(q, a)$.

Final States: $\{S : S \cap F \neq \emptyset\}$

An example

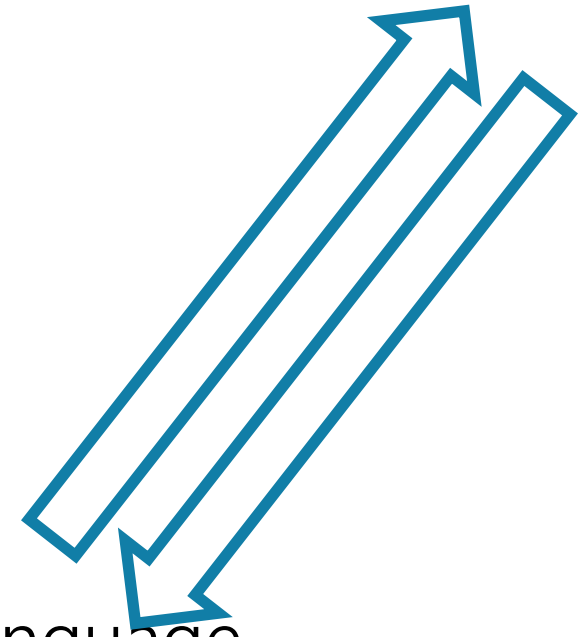


N



Proof [sketch]

L is the language of an NFA.



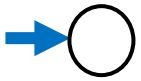
L is the language of a regular expression.

L is the language of a DFA.

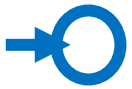
Let $P(A)$ be "There is an NFA whose language is the same as the language for A."

Base Cases:

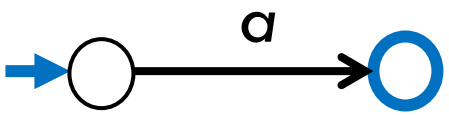
$\emptyset$



ε



$a\ (a \in \Sigma)$

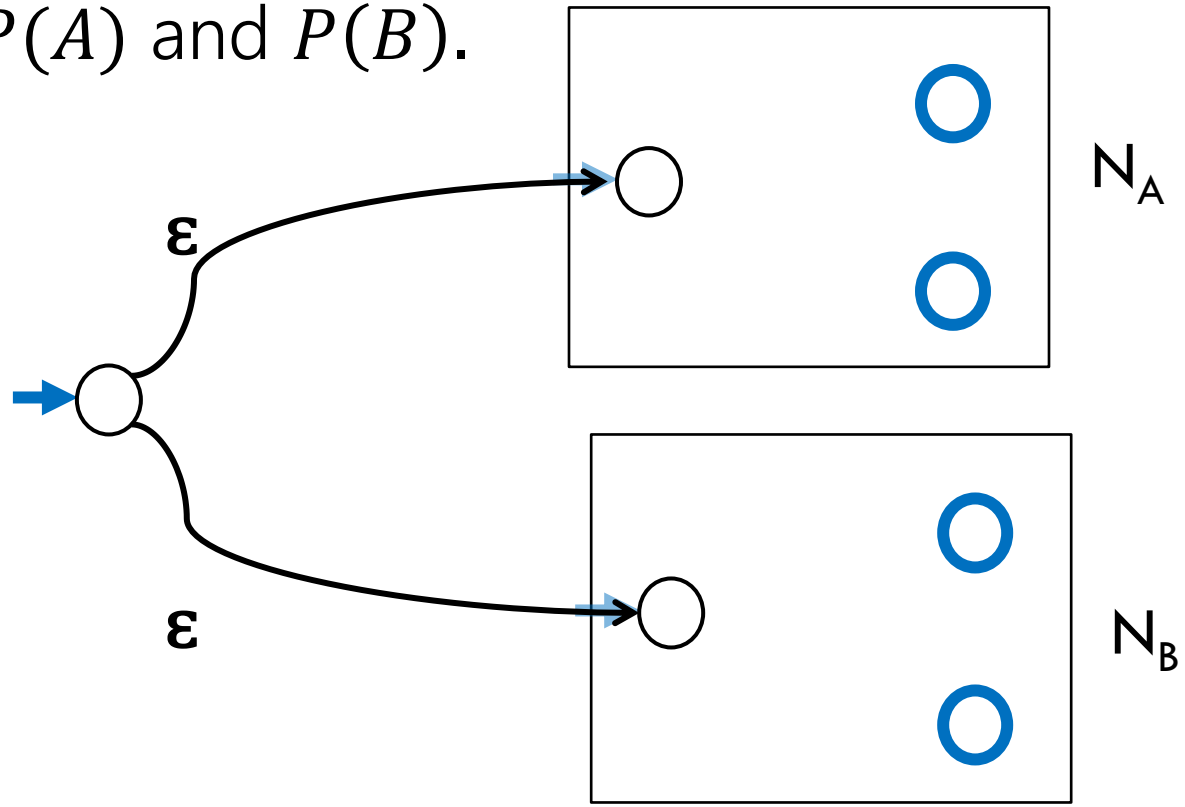


Let $P(A)$ be "There is an NFA whose language is the same as the language for A."

Let R be a regex not covered by the base cases. By the exclusion rule, $R = A \cup B$ or AB or A^* from some regexes A, B
Inductive Hypothesis: Suppose $P(A)$ and $P(B)$.

Inductive Step: **Case 1: $A \cup B$**

Match $A \cup B$? Then you match one of the two regexes. New machine transitions into start state of appropriate old machine. Will be accepted.
Accepted by the machine? First step **has** to be an ε-transition into one of the machines, so would have been accepted by the smaller machine, so must have matched A or B.

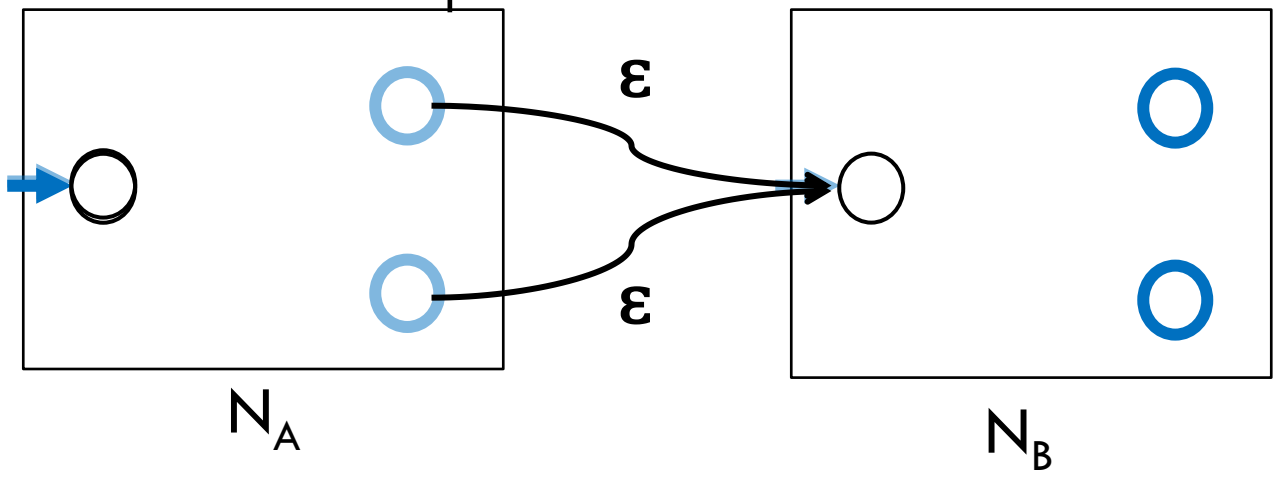


Want a machine that accepts exactly strings matched by A or B.

Let $P(A)$ be "There is an NFA whose language is the same as the language for A."

Let R be a regex not covered by the base cases. By the exclusion rule, $R = A \cup B$ or AB or A^* from some regexes A, B
Inductive Hypothesis: Suppose $P(A)$ and $P(B)$.

Inductive Step: **Case 2:** AB



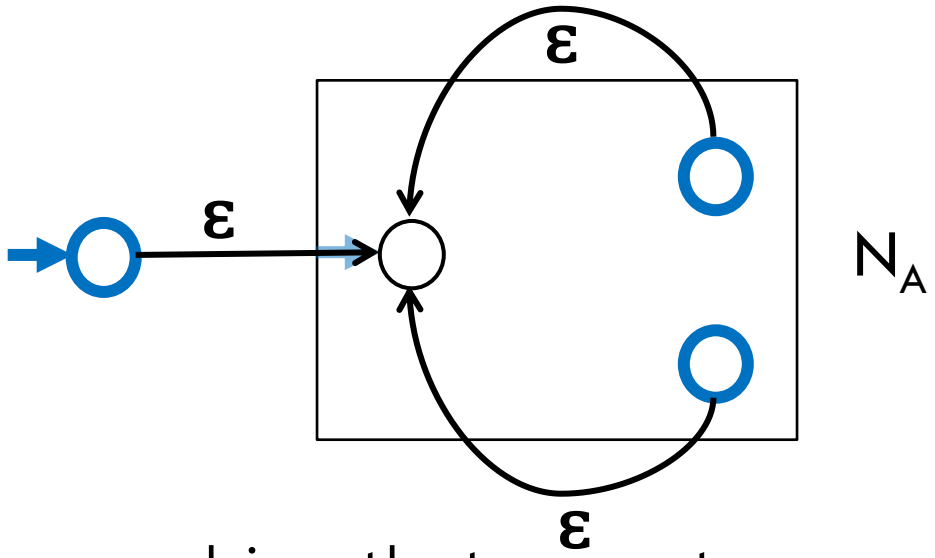
String x that matches AB can divide into yz where y matches A, z matches B.
NFA can run as N_A would on y take ε-transition, then run as N_B would on z so accepted by N
String x that is accepted?
N must run in N_A take ε-transition, then run in N_B until acceptance. Substring read in N_A must match A. Substring read in N_B must match B (by IH) so string matches AB.

Want a machine that accepts exactly strings matched by AB.

Let $P(A)$ be "There is an NFA whose language is the same as the language for A."

Let R be a regex not covered by the base cases. By the exclusion rule, $R = A \cup B$ or AB or A^* from some regexes A, B
Inductive Hypothesis: Suppose $P(A)$ and $P(B)$.

Inductive Step: **Case 3: A^***



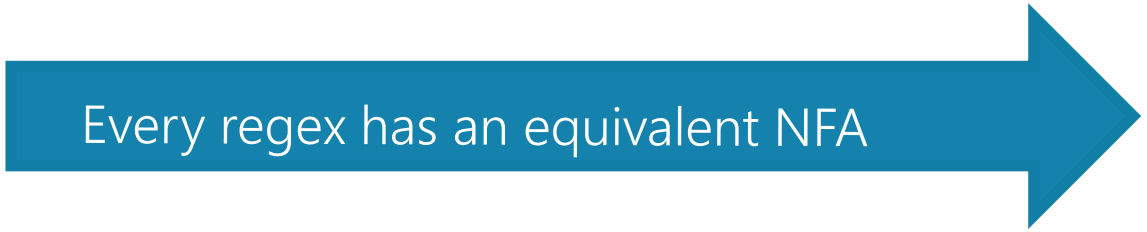
Want a machine that accepts exactly strings matched by A^*.

If x matches A^*, then by def of $*$ $x = \varepsilon$ or $x = x_1 \ldots x_k$ with each x_i matching A. If $x = \varepsilon$, machine accepts by not transitioning. Otherwise run accepting computation in N_A for each x_i return to start until x_k then end in accept state (all possible by IH)
If accepted by N,
Either ε or go from start state of N_A to final state and ε-transition back to start some number of times. So we can break string into parts accepted by N_A by IH we can break string into substrings all matched by A, i.e. we match A^*.

Let $P(A)$ be "There is an NFA whose language is the same as the language for A."

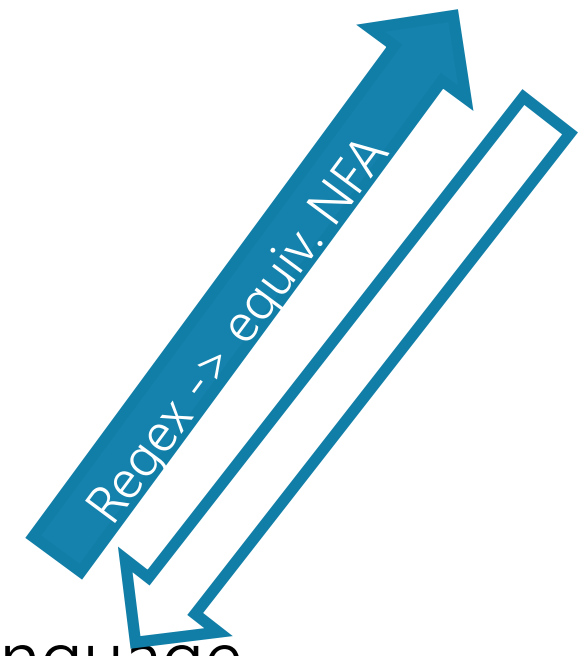
By principle of structural induction, $P(A)$ holds for all regular expressions A.

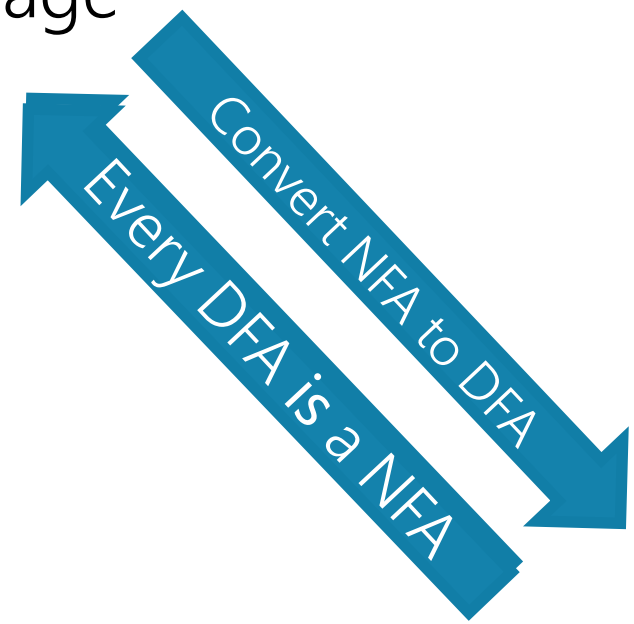
Thus every regular expression has an equivalent NFA.



Proof [sketch]

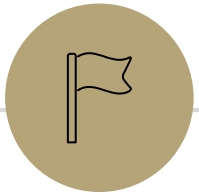
L is the language of an NFA.





L is the language of a regular expression.

L is the language of a DFA.



Irregular Languages

What can't NFAs/DFAs/Regexes do?

There are languages that can't be recognized by a DFA.

$\{0^k 1^k | k \geq 0\}$ is the classic example.

Let's prove it.

$\{0^k 1^k | k \geq 0\}$ is not regular

Not a proof:

$\{0^k 1^k | k \geq 0\}$ is regular if and only if there's a DFA the recognizes it.

To know if the number of 0's is equal to the number of 1's that DFA is going to have to count that number k.

But that number could be arbitrarily big.

And we have some finite number of states less than that.

So it's not regular!

"the DFA must work like this" isn't a rigorous argument.

“The DFA must do this”

Consider “The set of all binary strings, where the number of “01” substrings is equal to the number of “10” substrings.”

The DFA must count the number of “01” and “10” substrings, right?

NO!

Between every 01 substring there’s a 10 substring.

00011110000111101011

You don’t have to count the total, just the difference between them. And that difference never exceeds 1.

In general…

Try to avoid saying "in order to solve problem X, a machine must do Y"

It's almost impossible to rigorously justify.

And it's easy to trick yourself – you'll have to argue no one is more clever than you are.

The right way to argue around this claim is to argue by contradiction. Show that every conceivable DFA does the wrong thing (rather than indirectly that they aren't doing what you think they should).

A Proof Outline

Claim: $\{0^k 1^k : k \geq 0\}$ is an irregular language.

Proof:

Suppose, for the sake of contradiction, that $\{0^k 1^k : k \geq 0\}$ is regular.

Then there is a DFA M such that M accepts exactly $\{0^k 1^k : k \geq 0\}$.

We want to show M ***doesn't*** *actually accept $\{0^k 1^k : k \geq 0\}$*

How do we do that?

Forcing a Mistake

We don’t know anything about the DFA.

Well except that it **is** a DFA.

So it is deterministic. I.e. if you’re in a particular state and you read a certain character there’s only one place to go.

It’s also finite. I.e. there are not an infinite number of states.

Forcing a Mistake

What if…

We could find a set of strings that all **must** be in different states.

Too many of them. Enough that two must be in the same state.

If we had a 4 state machine, how many strings would guarantee a “collision”? 5 strings. If there are more strings than states, they can’t all get their own state.

How many strings is enough for us?

infinity

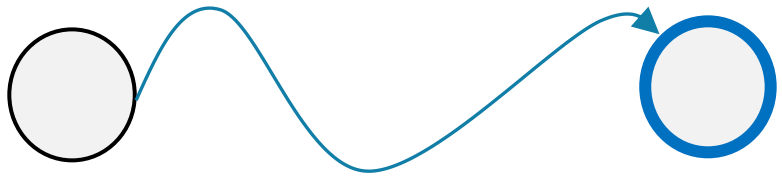
Forcing a Mistake

How do we know x, y must be in different states?

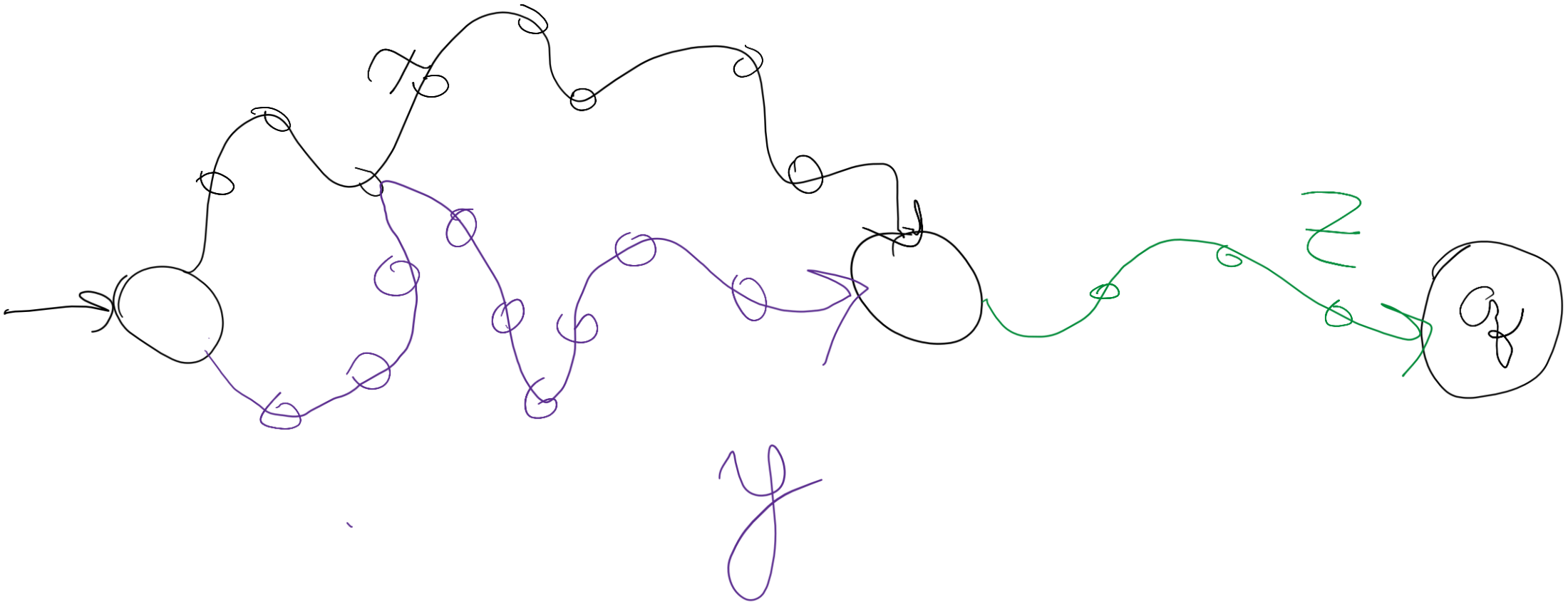
Well, if one would be accepted and the other rejected, that would be a clear sign.

Or if there's some string z where xz is accepted but yz is rejected (or vice versa).

The machine is deterministic! If x and y take you to the same state, then xz and yz are also in the same state!



Intuition



A Proof Outline

Claim: $\{0^k 1^k : k \geq 0\}$ is an irregular language.

Proof:

Suppose, for the sake of contradiction, that $\{0^k 1^k : k \geq 0\}$ is regular.

Then there is a DFA M such that M accepts exactly $\{0^k 1^k : k \geq 0\}$.

We want to show M ***doesn't*** *actually accept $\{0^k 1^k : k \geq 0\}$*

How do we do that?

A Proof Outline

Claim: $\{0^k 1^k : k \geq 0\}$ is an irregular language.

Proof:

Suppose, for the sake of contradiction, that $\{0^k 1^k : k \geq 0\}$ is regular.

Then there is a DFA M such that M accepts exactly $\{0^k 1^k : k \geq 0\}$.

Let $S =$ [TODO]. *S is an infinite set of strings.*

Because the DFA is finite, there are two (different) strings x, y in S such that x and y go to the same state. *We don't get to choose x, y*

Consider the string $z =$[TODO] *We do get to choose z* depending on x, y

A Proof Outline

Claim: $\{0^k 1^k : k \geq 0\}$ is an irregular language.

…

Let S = [TODO]. *S is an infinite set of strings.*

Because the DFA is finite, there are two (different) strings x, y in S such that x and y go to the same state. *We don't get to choose x, y*

Consider the string z =[TODO] *We do get to choose z* depending on x, y

Since x, y led to the same state and M is deterministic, xz and yz will also lead to the same state q in M. Observe that $xz \in \{0^k 1^k : k \geq 0\}$ but $yz \notin \{0^k 1^k : k \geq 0\}$. Since q is can be only one of an accept or reject state, M does not actually recognize $\{0^k 1^k : k \geq 0\}$. That's a contradiction!

Therefore, $\{0^k 1^k : k \geq 0\}$ is an irregular language.

Filling in the blanks

Need S

Such that

1. S is infinite

2. For every pair of strings x, y in S, there is a suffix z such that one of xz, yz is in the language and the other is not in the language.

Let $S = \{0^k : k \geq 0\}$

$x = 0^a$, $y = 0^b$ with $a \neq b$. Take $z = 1^a$.

Claim: $\{0^k 1^k : k \geq 0\}$ is an irregular language.

Proof:

Suppose, for the sake of contradiction, that $\{0^k 1^k : k \geq 0\}$ is regular.

Then there is a DFA M such that M accepts exactly $\{0^k 1^k : k \geq 0\}$.

Let $S = \{0^k : k \geq 0\}$.

Because the DFA is finite and S is infinite, there are two (different) strings x, y in S such that x and y go to the same state when read by M. Since both are in S, $x = 0^a$ for some integer a, and $y = 0^b$ for some integer b, with $a \neq b$.

Consider the string $z = 1^a$. $xz = 0^a 1^a \in \{0^k 1^k : k \geq 0\}$ but $yz = 0^b 1^a \notin \{0^k 1^k : k \geq 0\}$.

Since x, y both end up in the same state, and we appended the same z, both xz and yz end up in the same state of M.

Since $xz \in \{0^k 1^k : k \geq 0\}$ and $yz \notin \{0^k 1^k : k \geq 0\}$, M does not recognize$\{0^k 1^k : k \geq 0\}$. But that's a contradiction!

So$\{0^k 1^k : k \geq 0\}$ must be an irregular language.

Full outline

1. Suppose for the sake of contradiction that L is regular. Then there is some DFA M that recognizes L.

2. Let S be [fill in with an infinite set of prefixes].

3. Because the DFA is finite and S is infinite, there are two (different) strings x, y in S such that x and y go to the same state when read by M *[you don't get to control x, y other than having them not equal and in S]*

4. Consider the string z [argue exactly one of $\mathbf{xz}, \mathbf{yz}$ will be in $\mathbf{L}$]

5. Since x, y both end up in the same state, and we appended the same z, both xz and yz end up in the same state of M. Since $xz \in L$and $yz \notin L$, M does not recognize L. But that's a contradiction!

6. So L must be an irregular language.

Practical Tips

When you're choosing the set S, think about what the DFA would "have to count"

That is fundamentally why a language is irregular. The set S is the way we prove it! Whatever we "need to remember" it's different for every element of S.

If your strings have an "obvious middle" (like between the 0's and 1's) that's a good place to start.

Let's Try another

The set of strings with balanced parentheses is not regular.

What do you want S to be? What would you have to count?

The number of unclosed parentheses.

Let $S =$...

Let's Try another

The set of strings with balanced parentheses is not regular.

What do you want S to be? What would you have to count?

The number of unclosed parentheses.

Want S to be a set with infinitely many strings with different numbers of unclosed parentheses.

Let $S = (^*$

Outline for $(^*$

1. Suppose for the sake of contradiction that L is regular. Then there is some DFA M that recognizes L.

2. Let S be $(^*$

3. Because the DFA is finite and S is infinite, there are two (different) strings x, y in S such that x and y go to the same state when read by M Observe that $x = (^a$ for some integer a, $y = (^b$ for some integer b with $a \neq b$.

4. Consider the string z [argue exactly one of xz, yz will be in L]

5. Since x, y both end up in the same state, and we appended the same z, both xz and yz end up in the same state of M. Since $xz \in L$and $yz \notin L$, M does not recognize L. But that's a contradiction!

6. So L must be an irregular language.

Full outline

1. Suppose for the sake of contradiction that L is regular. Then there is some DFA M that recognizes L.

2. Let S be $(^*$

3. Because the DFA is finite and S is infinite, there are two (different) strings x, y in S such that x and y go to the same state when read by M Observe that $x = (^a$ for some integer a, $y = (^b$ for some integer b with $a \neq b$.

4. Consider the string $z=)^a$ xz is a balanced set of parentheses (since there are the same number of each and all the open-parentheses come before the close parentheses). But yz is not balanced because $a \neq b$.

5. Since x, y both end up in the same state, and we appended the same z, both xz and yz end up in the same state of M. Since $xz \in L$and $yz \notin L$, M does not recognize L. But that's a contradiction!

6. So L must be an irregular language.

One more, just the key steps

What about $\{a^k b^k c^k : k \geq 0\}$?

One more, just the key steps

What about $\{a^k b^k c^k : k \geq 0\}$?

$S = \{a^k : k \geq 0\}$ or $S = \{a^k b^k : k \geq 0\}$ are equally good choices.

Your suffix is $b^j c^j$ for the first and c^j for the second.

$S = \{a^k b : k \geq 0\}$ also works. $z = b^{j-1} c^j$ is your suffix. The proof is a little more tedious, but you can make it through.

Todo

Tonight:

- CC 21 is out and due Friday at noon
- Start today on HW7 if you haven’t already