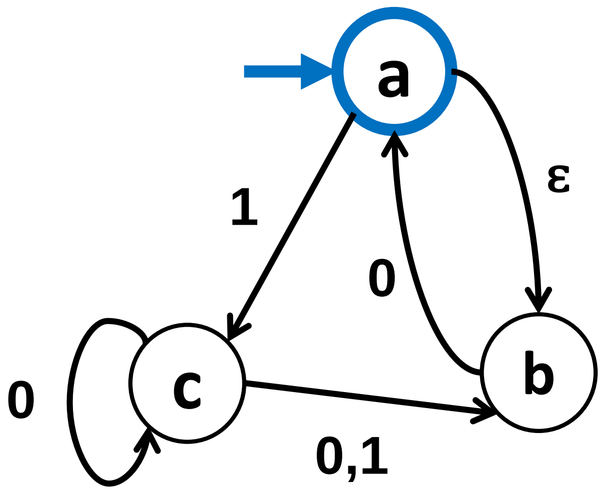
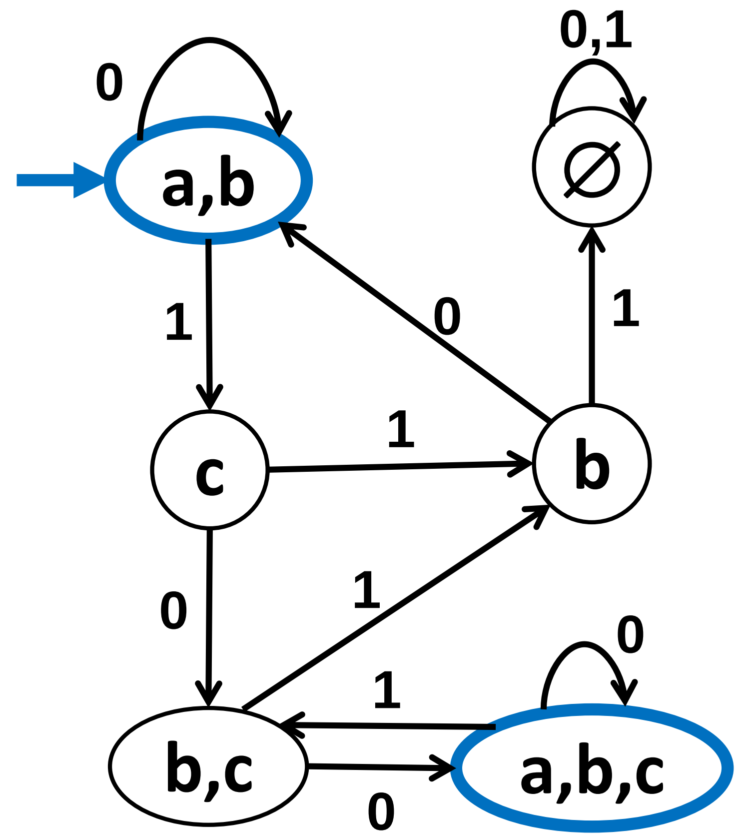
CSE 311: Foundations of Computing

Lecture 25: Pattern Matching, DFA≡NFA≡Regex
Languages vs Representations

Last time: NFA to DFA



NFA



DFA

Exponential Blow-up in Simulating Nondeterminism

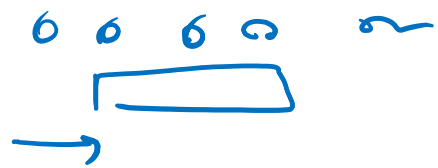
- In general the DFA might need a state for every subset of states of the NFA
  - Power set of the set of states of the NFA
  - n-state NFA yields DFA with at most 2^n states
  - We saw an example where roughly 2^n is necessary

    "Is the n^{th} char from the end a 1?"

The famous "P=NP?" question asks whether a similar blow-up is always necessary to get rid of nondeterminism for polynomial-time algorithms

Pattern matching

- **Given**
  - a string s of n characters
  - a pattern p of m characters
  - usually $m \ll n$



- **Find**
  - all occurrences of the pattern p in the string s

- **Obvious algorithm:**
  - try to see if p matches at each of the positions in s

    stop at a failed match and try matching at the next position

string **s** = x y x x y x y x y y x y x y x y y x y x y x x

pattern **p** = x y x y y x y x y x x

string **s** = x y x x y x y x y y x y x y x y y x y x y x x
x y x y y x y x y x x

string **s** = x y x x y x y x y y x y x y x y y x y x y x x

x y x y

x y x y y x y x y x x

string **s** = x y x x y x y x y y x y x y x y y x y x y x x

x y x y

x

x y x y y x y x y x x

string **s** = x y x x y x y x y y x y x y x y y x y x y x x

x y x y

x

x y

x y x y y x y x y x x

string **s** = x y x x y x y x y y x y x y x y y x y x y x x

x y x y

x

x y

x y x y y

x y x y y x y x y x x

string **s** = x y x x y x y x y y x y x y x y y x y x y x x

x y x y

x

x y

x y x y y

x

x y x y y x y x y x x

string **s** = x y x x y x y x y y x y x y x y y x y x y x x

x y x y

x

x y

x y x y y

x

x y x y y x y x y x x

x y x y y x y x y x x

string **s** = x y x x y x y x y y x y x y x y y x y x y x x

x y x y

x

x y

x y x y y

x

x y x y y x y x y x x

x

x y x y y x y x y x x

string **s** = x y x x y x y x y y x y x y x y y x y x y x x

x y x y
x
x y
x y x y y
x
x y x y y x y x y x x
x
x y x
x y x y y x y x y x x

string **s** = x y x x y x y x y y x y x y x y y x y x y x x

x y x y

x

x y

x y x y y

x

x y x y y x y x y x x

x

x y x

x

x y x y y x y x y x x

string **s** = x y x x y x y x y y x y x y x y y x y x y x x

x y x y

x

x y

x y x y y

x

x y x y y x y x y x x

x

x y x

x

x

x y x y y x y x y x x

string **s** = x y x x y x y x y y x y x y x y y x y x y x x

x y x y

x

x y

x y x y y

x

x y x y y x y x y x x

x

x y x

x

x

x y x y y

x y x y y x y x y x x

string **s** = x y x x y x y x y y x y x y x y y x y x y x x

x y x y

x

x y

x y x y y

x

x y x y y x y x y x x

x

x y x

x

x

x y x y y

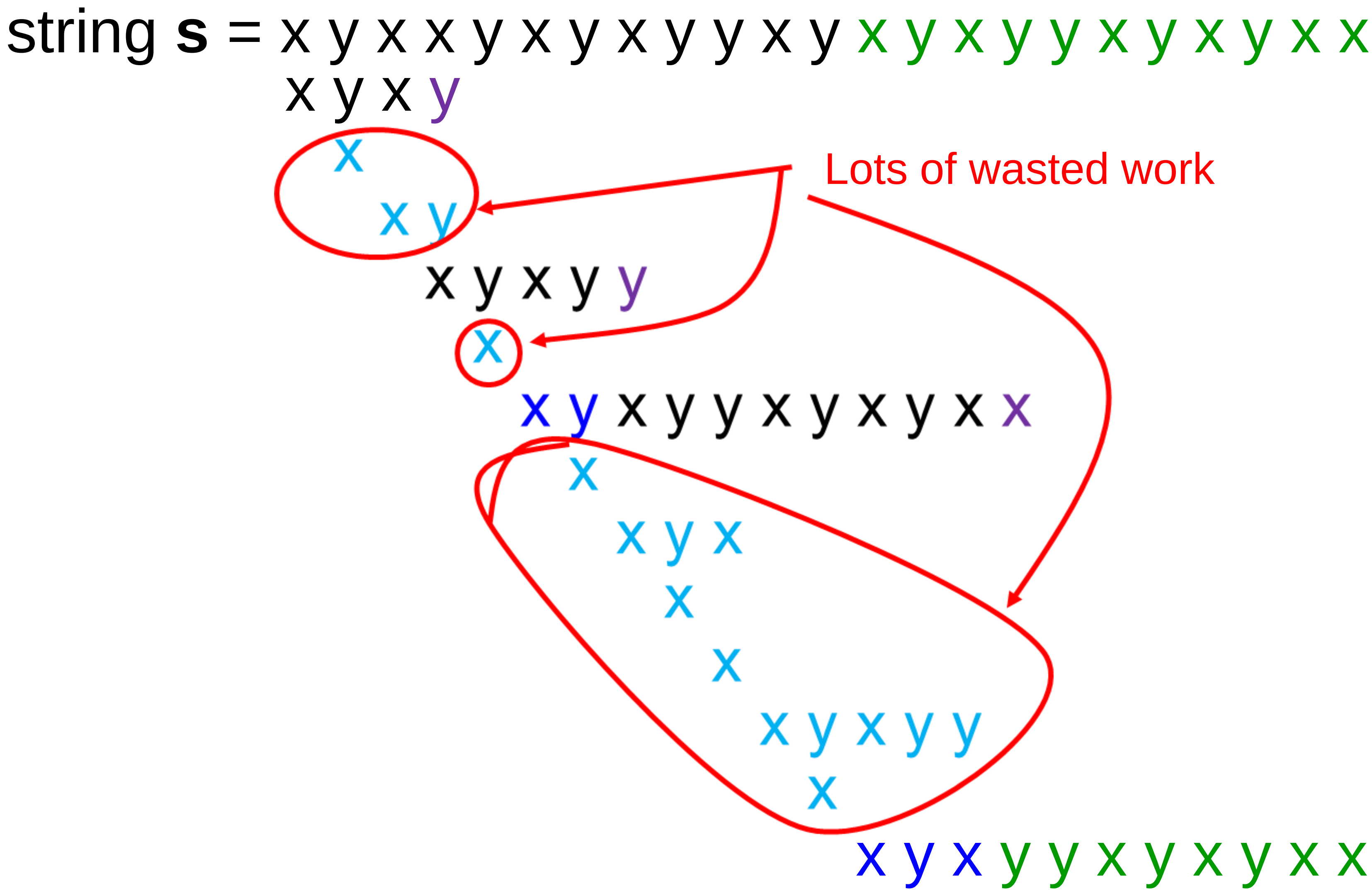
x

x y x y y x y x y x x

Worst-case time

$\boldsymbol{O(mn)}$

$(n - m)\, m$

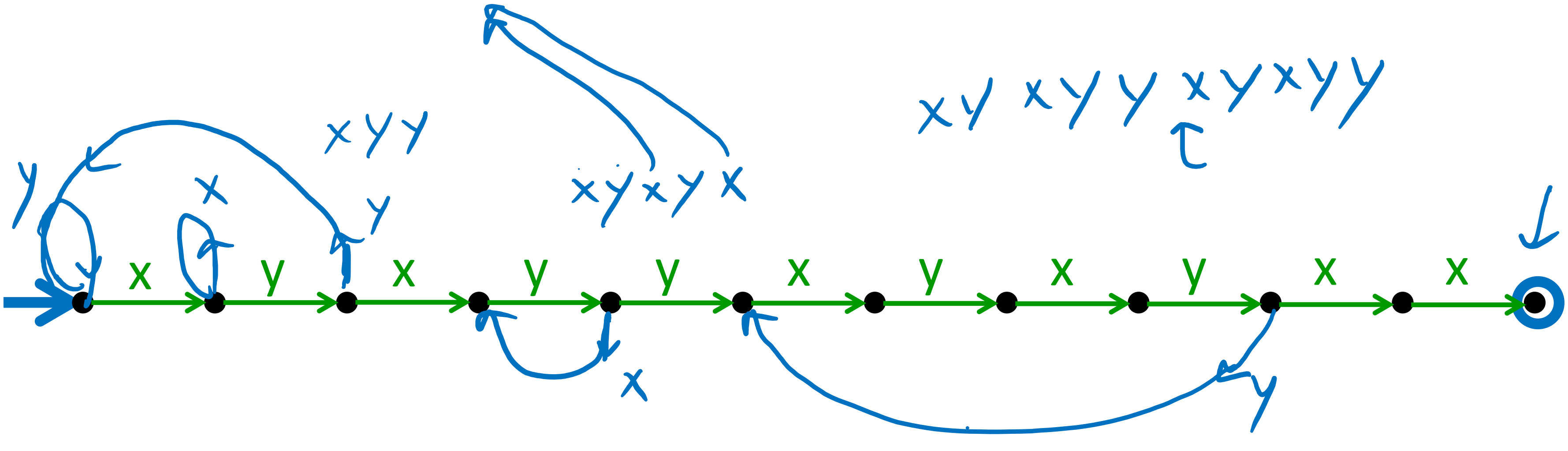

string **s** = x y x x y x y x y y x y x y x y y x y x y x x
x y x y
x
x y
x y x y y
x
x y x y y x y x y x x
x
x y x
x
x
x y x y y
x
x y x y y x y x y x x
Lots of wasted work

Better pattern matching via finite automata

- Build a DFA for the pattern (preprocessing) of size $O(m)$
  - Keep track of the ‘longest match currently active’
  - The DFA will have only $m + 1$ states
- Run the DFA on the string n steps
- Obvious construction method for DFA will be $O(m^2)$ but can be done in $O(m)$ time.
- Total $O(m + n)$ time

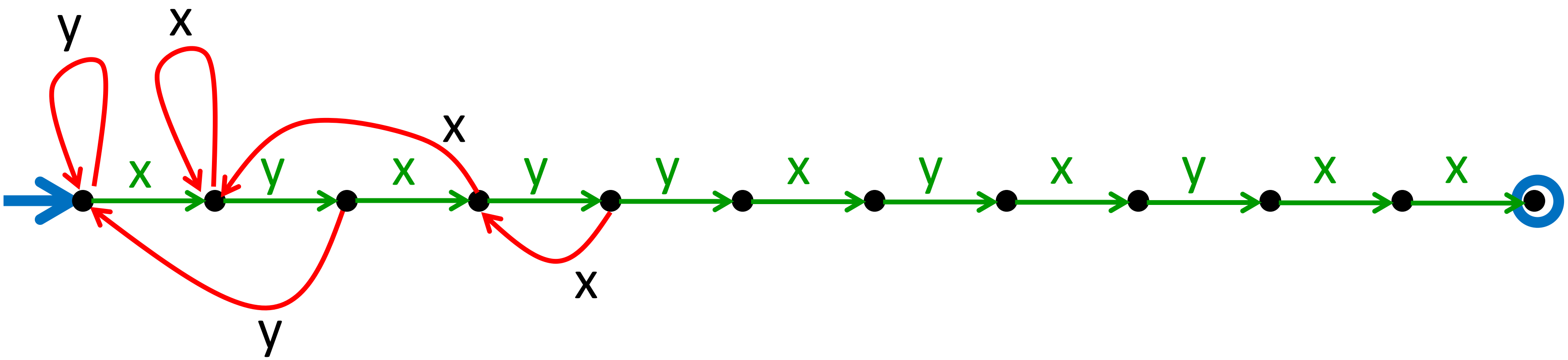
Building a DFA for the pattern

pattern **p**=x y x y y x y x y x x



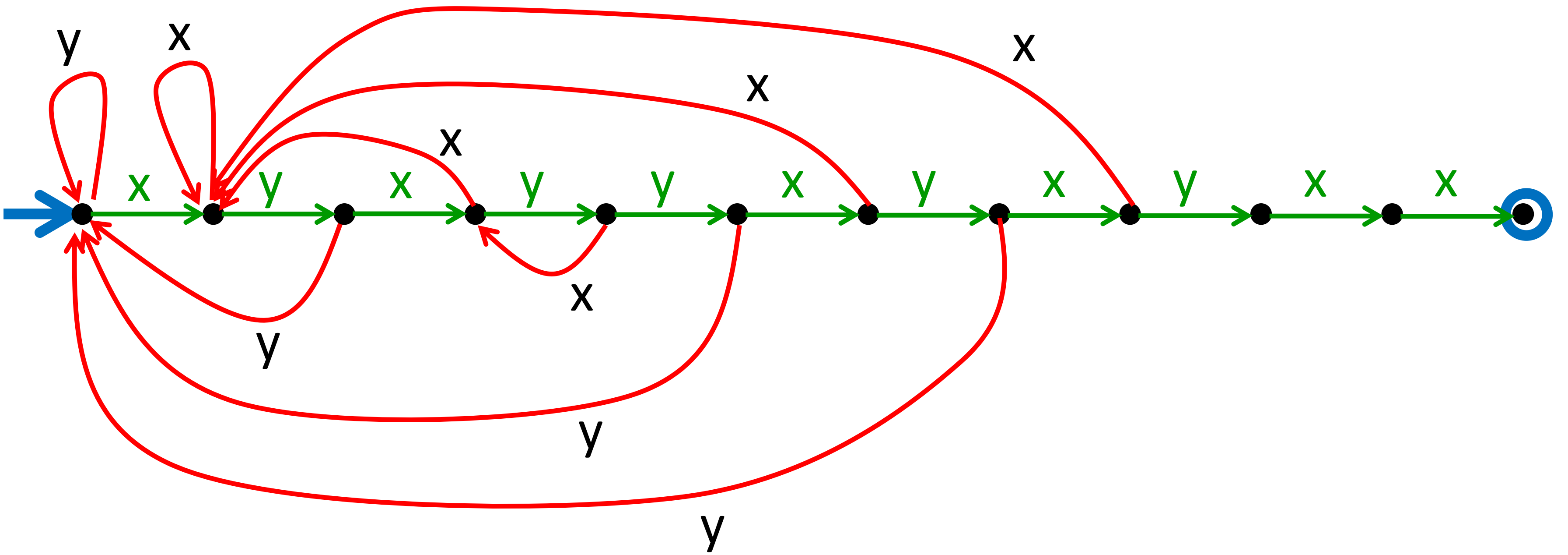
Building a DFA for the pattern

pattern **p**=x y x y y x y x y x x



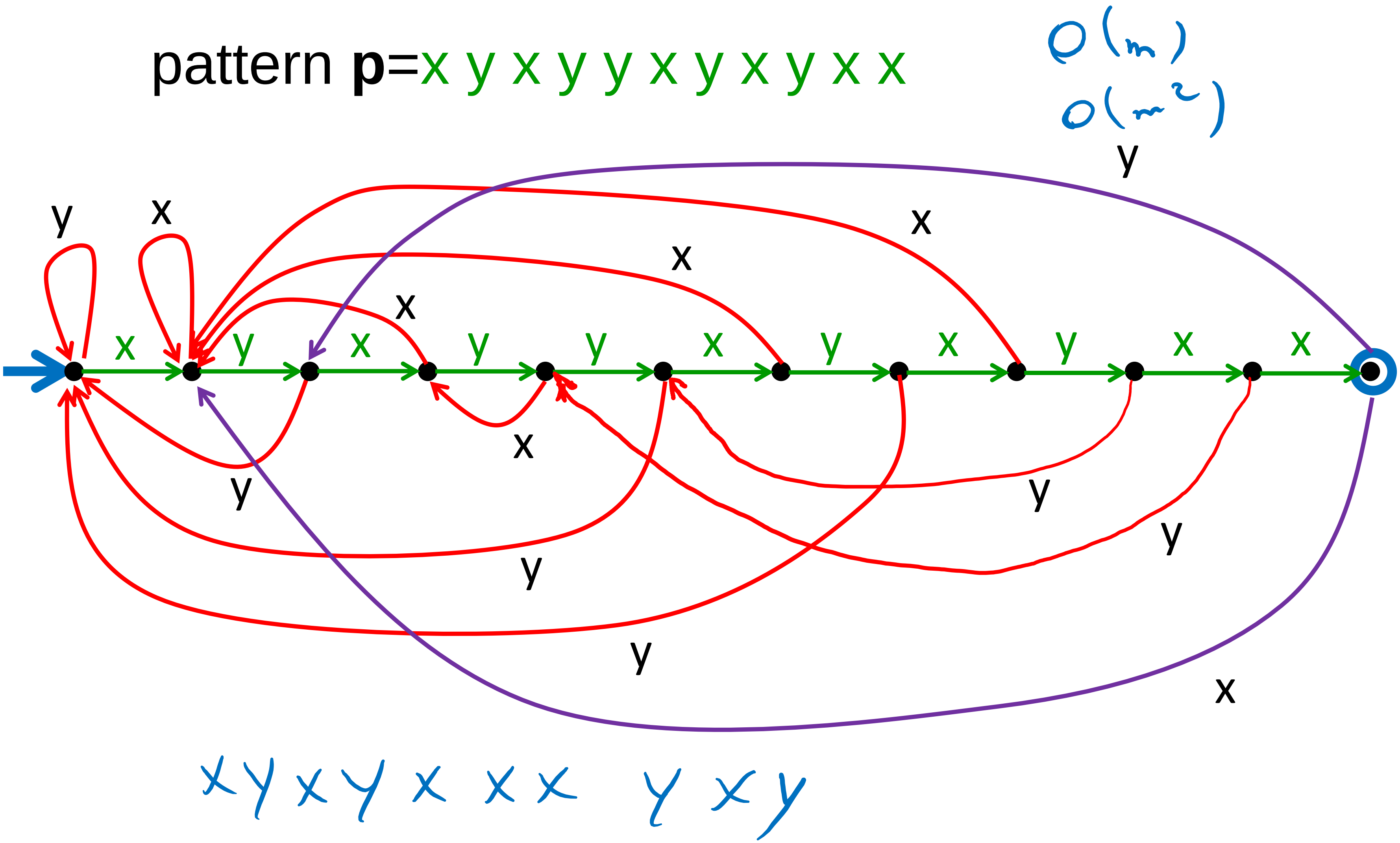
Building a DFA for the pattern

pattern **p**=x y x y y x y x y x x



Building a DFA for the pattern

pattern **p**=x y x y y x y x y x x

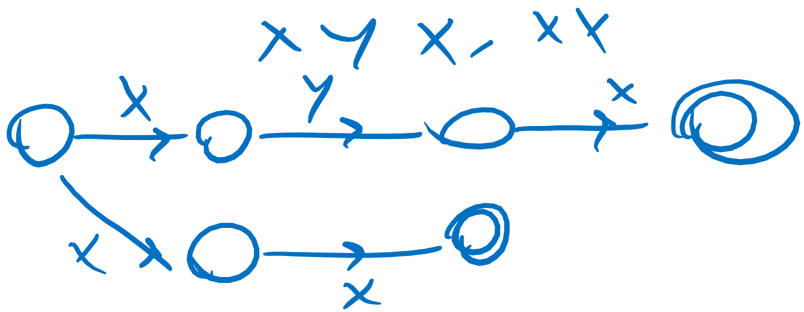


Generalizing

- **Can search for arbitrary combinations of patterns**
  - **Not just a single pattern**
  - **Build NFA for pattern then convert to DFA 'on the fly*'.**

    * Only add states when the input string actually needs to use them

    (Compare DFA constructed above with subset construction for the obvious NFA.)



DFAs ≡ NFAs ≡ Regular expressions

We have shown how to build an optimal DFA for every regular expression

- Build NFA
- Convert NFA to DFA using subset construction
- Minimize resulting DFA

Theorem: A language is recognized by a DFA (or NFA) if and only if it has a regular expression

You need to know this fact but we won't ask you anything about the "only if" direction from DFA/NFA to regular expression. For fun, we sketch the idea.

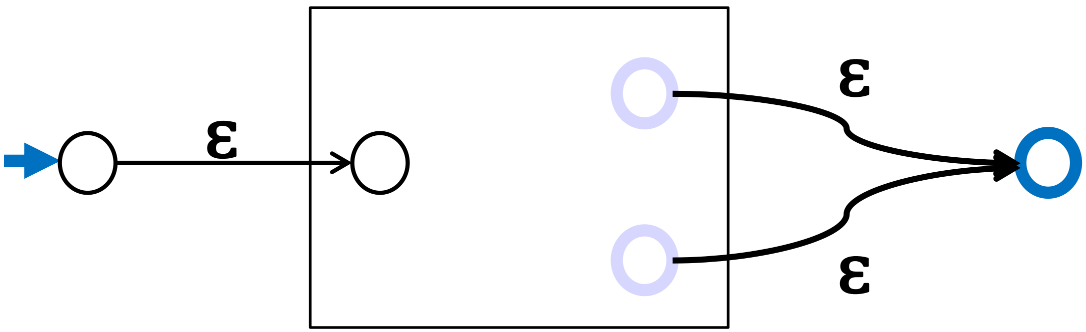
Generalized NFAs

- Like NFAs but allow
  - Parallel edges
  - Regular Expressions as edge labels

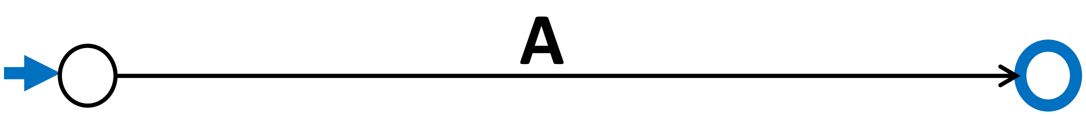
    NFAs already have edges labeled **ε** or ***a***
- An edge labeled by **A** can be followed by reading a string of input chars that is in the language represented by **A**
- Defn: A string x is accepted iff there is a *path* from start to final state *labeled by a regular expression* whose language contains x

Starting from an NFA

Add new start state and final state



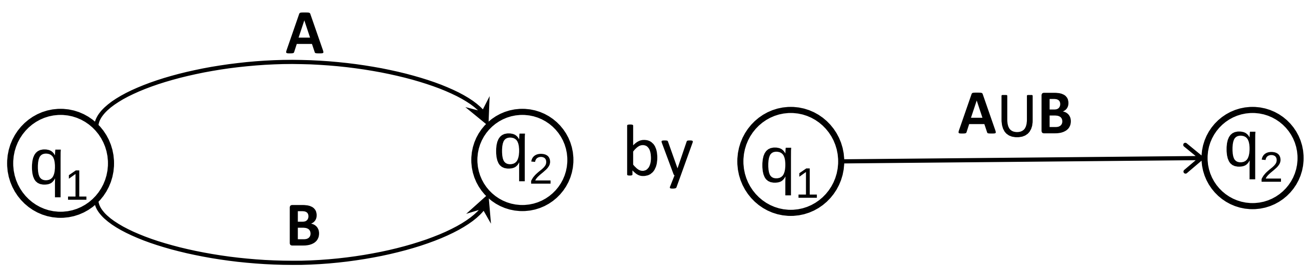
Then eliminate original states one by one, keeping the same language, until it looks like:



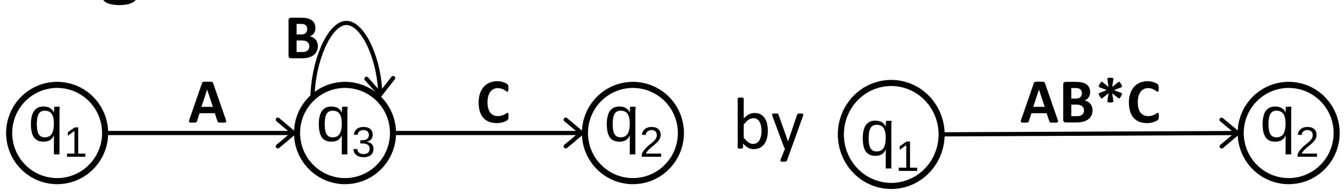
Final regular expression will be **A**

Only two simplification rules

- **Rule 1**: For any two states q_1 and q_2 with parallel edges (possibly q_1=q_2), replace



- **Rule 2**: Eliminate non-start/final state q_3 by replacing all

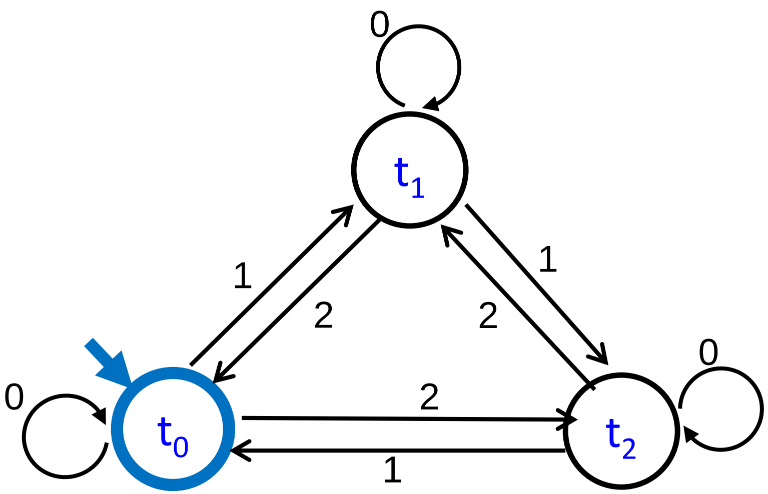


for *every* pair of states q_1, q_2 (even if q_1=q_2)

Converting an NFA to a regular expression

Consider the DFA for the mod 3 sum

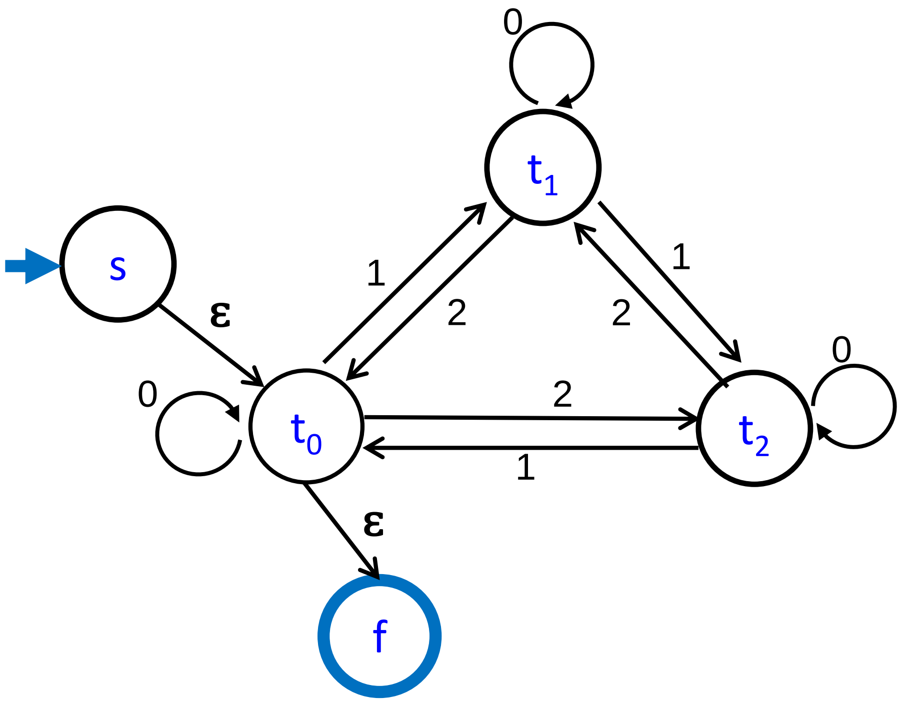
- Accept strings from {0,1,2}* where the digits mod 3 sum of the digits is 0



Splicing out a state t_1

Regular expressions to add to edges

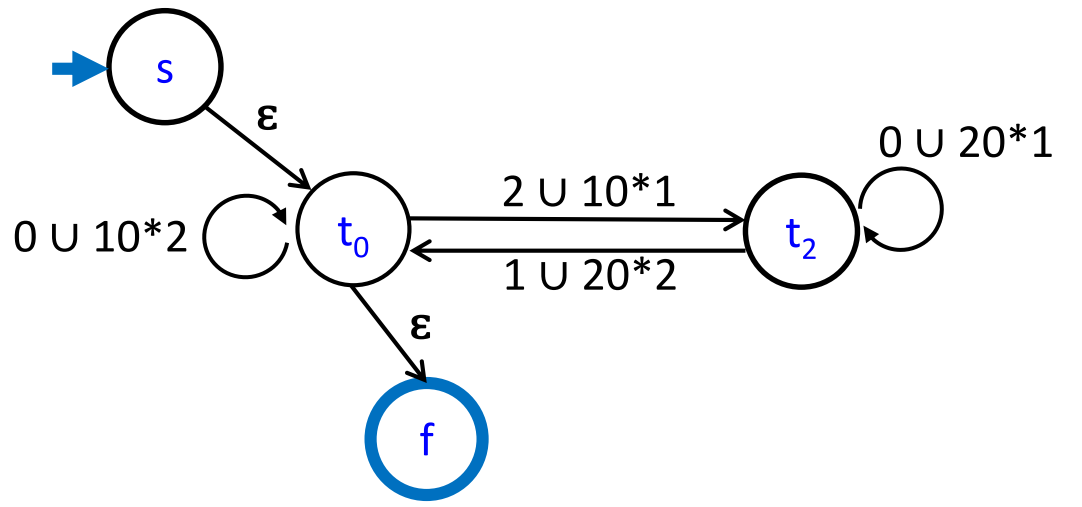
$t_0 \to t_1 \to t_0$: 10*2
$t_0 \to t_1 \to t_2$: 10*1
$t_2 \to t_1 \to t_0$: 20*2
$t_2 \to t_1 \to t_2$: 20*1



Splicing out a state t_1

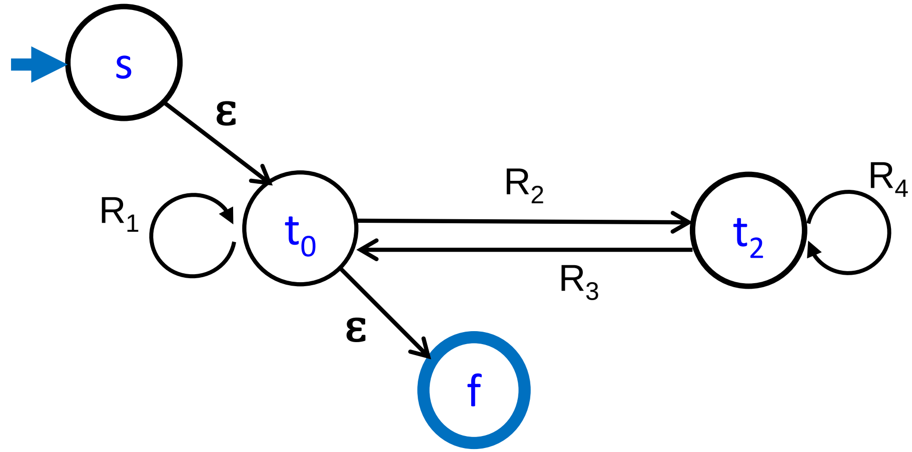
Regular expressions to add to edges

$t_0 \rightarrow t_1 \rightarrow t_0$: 10*2
$t_0 \rightarrow t_1 \rightarrow t_2$: 10*1
$t_2 \rightarrow t_1 \rightarrow t_0$: 20*2
$t_2 \rightarrow t_1 \rightarrow t_2$: 20*1

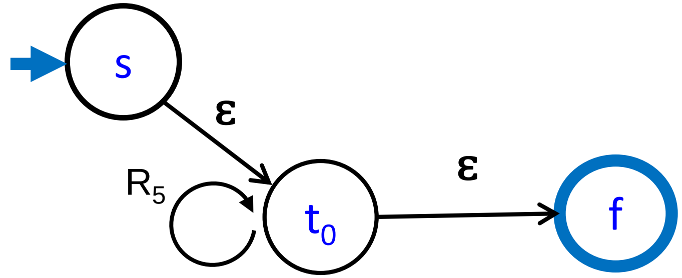


Splicing out state t_2 (and then t_0)

R_1: $0 \cup 10^*2$
R_2: $2 \cup 10^*1$
R_3: $1 \cup 20^*2$
R_4: $0 \cup 20^*1$



R_5: $R_1 \cup R_2R_4{}^*R_3$

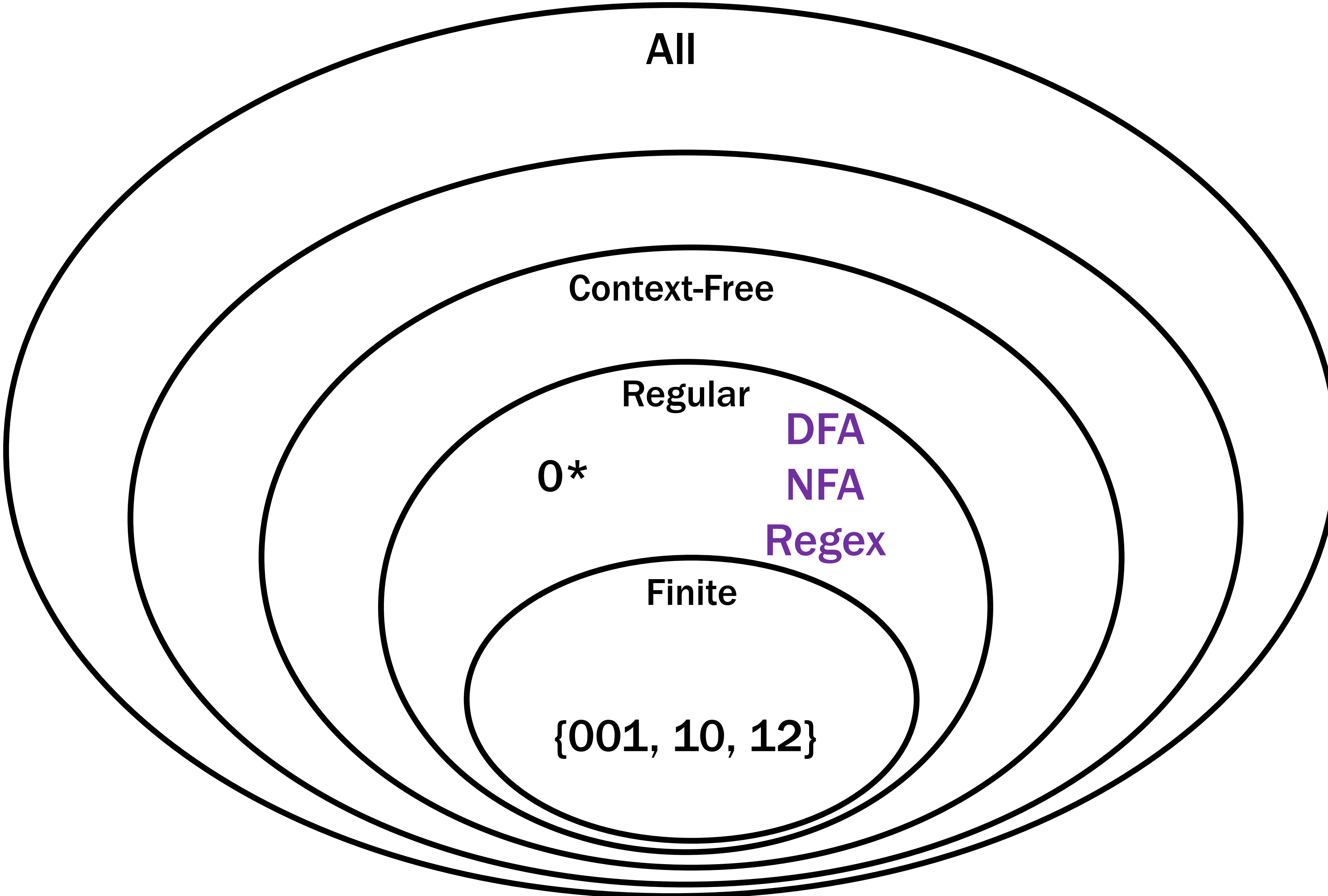


Final regular expression: $R_5{}^*$=
$(0 \cup 10^*2 \cup (2 \cup 10^*1)(0 \cup 20^*1)^*(1 \cup 20^*2))^*$

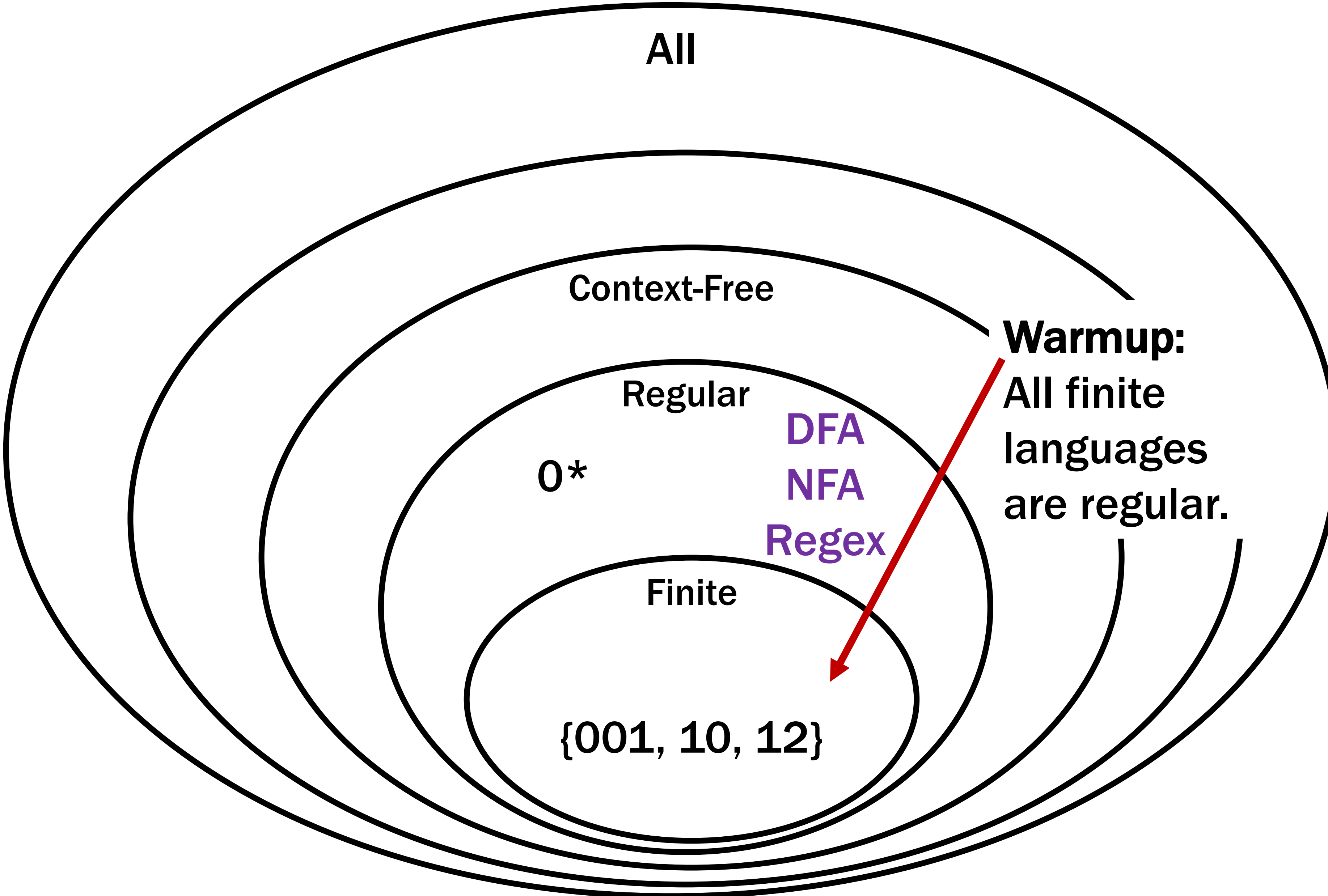
What languages have DFAs? CFGs?

All of them?

Languages and Representations!



Languages and Representations!



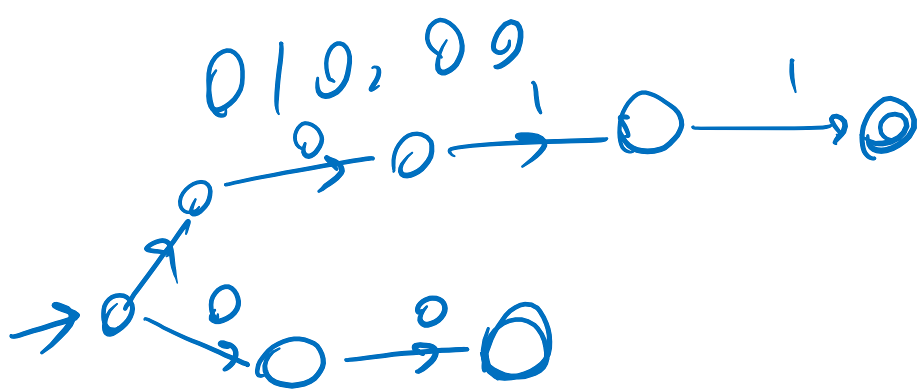
DFAs Recognize Any Finite Language

010

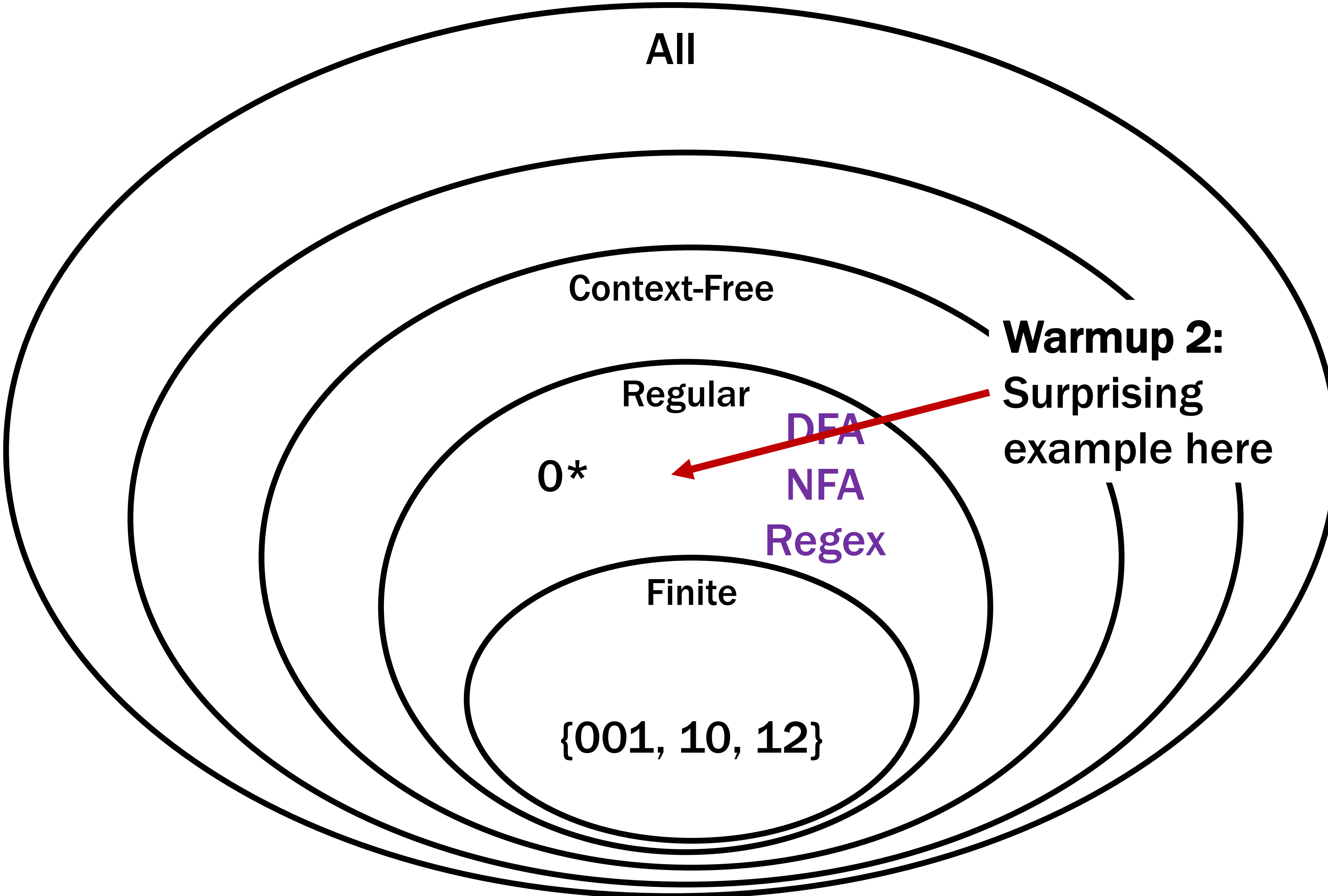
DFAs Recognize Any Finite Language

Construct a DFA for each string in the language.

Then, put them together using the union construction.



Languages and Machines!



An Interesting Infinite Regular Language

L = {x∈ {0, 1}*: x has an equal number of substrings 01 and 10}.

L is infinite.

0, 00, 000, …

L is regular. How could this be?

(It seems to be comparing counts and counting seems hard for DFAs.)

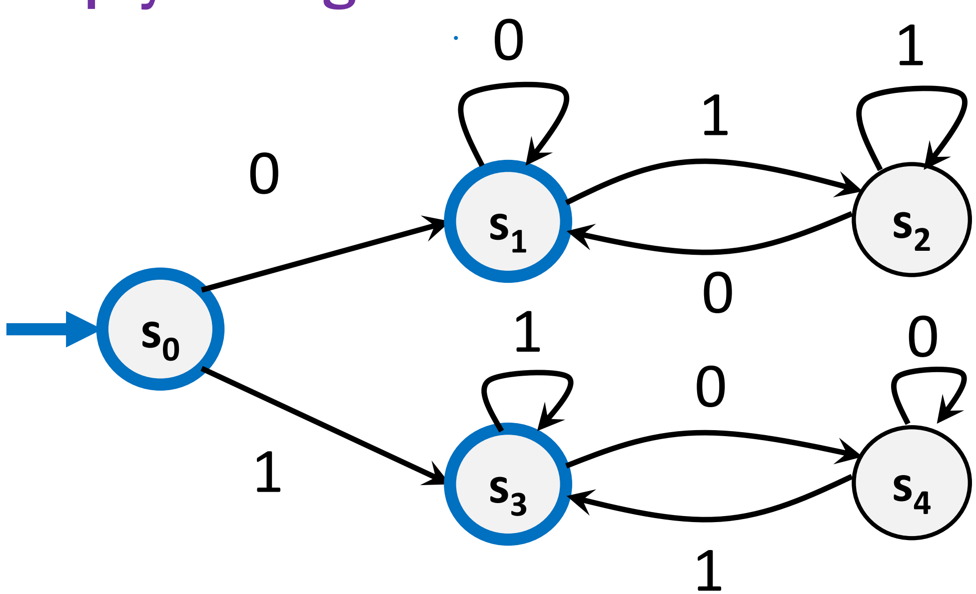
An Interesting Infinite Regular Language

L = {x∈ {0, 1}*: x has an equal number of substrings 01 and 10}.

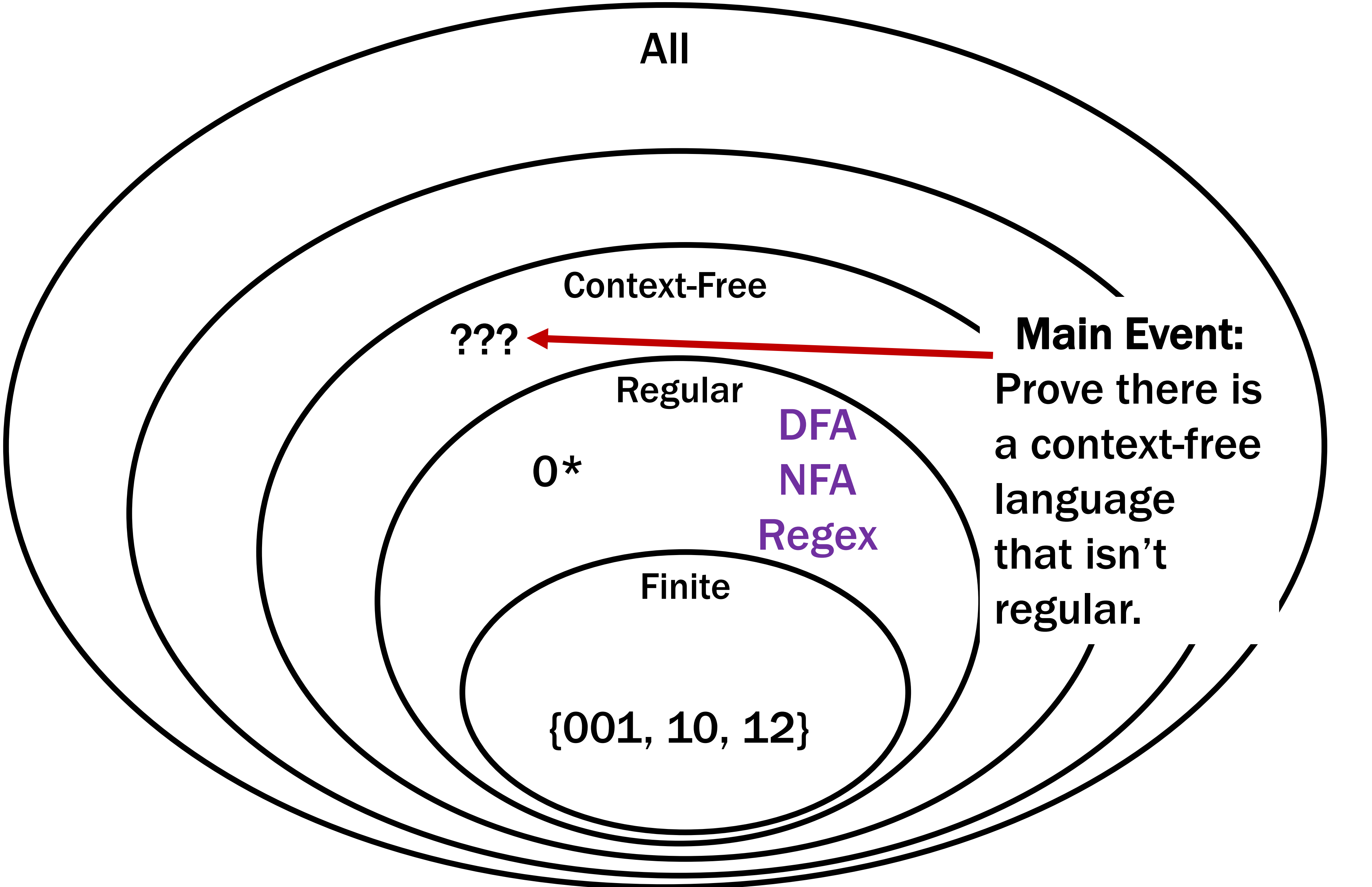
L is infinite.

0, 00, 000, …

L is regular. How could this be? It is just the set of binary strings that are empty or begin and end with the same character!



Languages and Representations!



The language of “Binary Palindromes” is Context-Free

$$S \rightarrow \varepsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1$$

Is the language of “Binary Palindromes” Regular ?

Is the language of “Binary Palindromes” Regular ?

Intuition (NOT A PROOF!):

Q: What would a DFA need to keep track of to decide the language?

A: It would need to keep track of the “first part” of the input in order to check the second part against it

…but there are an infinite # of possible first parts and we only have finitely many states.

B = {binary palindromes} can't be recognized by any DFA

The general proof strategy is:

- Assume (for contradiction) that it's possible.
- Therefore, some DFA (call it M) exists that recognizes B

B = {binary palindromes} can't be recognized by any DFA

The general proof strategy is:

– Assume (for contradiction) that it's possible.
– Therefore, some DFA (call it M) exists that recognizes B
– Our goal is to show that M must be "confused"… we want to show it "does the wrong thing".

How can a DFA be "wrong"?

– when it accepts or rejects a string it shouldn't.

B = {binary palindromes} can't be recognized by any DFA

The general proof strategy is:

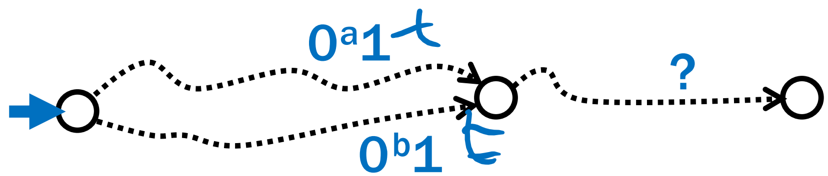
- Assume (for contradiction) that it's possible.
- Therefore, some DFA (call it M) exists that recognizes B
- Our goal is to show that M must be "confused"… we want to show it ~~"does the wrong thing"~~ accepts or rejects a string it shouldn't.

B = {binary palindromes} can't be recognized by any DFA

The general proof strategy is:

– Assume (for contradiction) that it's possible.
– Therefore, some DFA (call it M) exists that recognizes B
– We want to show: M accepts or rejects a string it shouldn't.

Key Idea 1: If two strings "collide" at any point, a DFA can no longer distinguish between them!

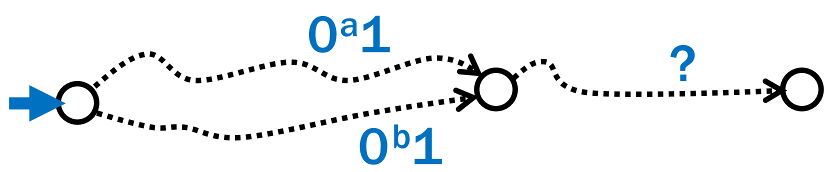


B = {binary palindromes} can't be recognized by any DFA

The general proof strategy is:

- Assume (for contradiction) that it's possible.
- Therefore, some DFA (call it M) exists that recognizes B
- We want to show: M accepts or rejects a string it shouldn't.

Key Idea 1: If two strings "collide" at any point, a DFA can no longer distinguish between them!



Key Idea 2: Our machine M has a finite number of states which means if we have infinitely many strings, two of them must collide!

B = {binary palindromes} can't be recognized by any DFA

The general proof strategy is:

- Assume (for contradiction) that it's possible.
- Therefore, some DFA (call it M) exists that recognizes B
- We want to show M accepts or rejects a string it shouldn't.

We choose an **INFINITE** set S of "half strings" (which we intend to complete later). It is imperative that for ***every pair*** of strings in our set there is an <u>"accept" completion</u> that the two strings DO NOT SHARE.

1______
01______
001______
0001______
00001______
....

B = {binary palindromes} can't be recognized by any DFA

Suppose for contradiction that some DFA, M, recognizes B.

We show M accepts or rejects a string it shouldn't.

Consider S={1, 01, 001, 0001, 00001, ...} = $\{0^n1 : n \geq 0\}$.

Key Idea 2: Our machine has a finite number of states which means if we have infinitely many strings, two of them must collide!

B = {binary palindromes} can't be recognized by any DFA

Suppose for contradiction that some DFA, M, recognizes B.

We show M accepts or rejects a string it shouldn't.

Consider S={1, 01, 001, 0001, 00001, ...} = $\{0^n1 : n \geq 0\}$.

Since there are finitely many states in M and infinitely many strings in S***, there exist strings*** $0^a1 \in S$ ***and*** $0^b1 \in S$ ***with*** $a \neq b$ ***that end in the same state of M.***

SUPER IMPORTANT POINT: You do not get to choose what a and b are. Remember, we've proven they exist…we have to take the ones we're given!

B = {binary palindromes} can't be recognized by any DFA

Suppose for contradiction that some DFA, M, recognizes B.

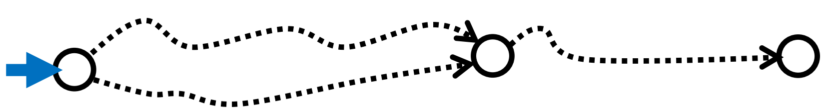
We show M accepts or rejects a string it shouldn't.

Consider $S = \{0^n1 : n \geq 0\}$.

Since there are finitely many states in M and infinitely many strings in S**, there exist strings** $0^a1 \in S$ **and** $0^b1 \in S$ **with** $a \neq b$ **that end in the same state of M.**

Now, consider appending 0^a *to both strings.*

Key Idea 1: If two strings "collide" at any point, a DFA can no longer distinguish between them!



B = {binary palindromes} can't be recognized by any DFA

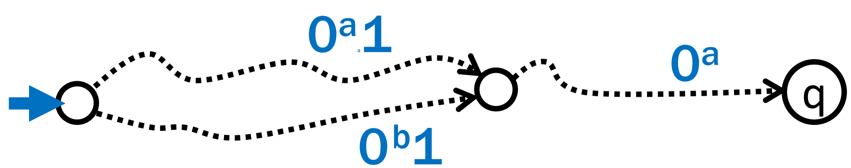
Suppose for contradiction that some DFA, M, recognizes B.

We show M accepts or rejects a string it shouldn't.

Consider $S = \{0^n1 : n \geq 0\}$.

Since there are finitely many states in M and infinitely many strings in S**, there exist strings** $0^a1 \in S$ **and** $0^b1 \in S$ **with** $a \neq b$ **that end in the same state of M.**

Now, consider appending 0^a **to both strings.**



Then, since 0^a1 ***and*** 0^b1 ***end in the same state,*** 0^a10^a ***and*** 0^b10^a ***also end in the same state, call it*** q. ***But then M must make a mistake:*** q ***needs to be an accept state since*** $0^a10^a \in$ ***B, but then M would accept*** $0^b10^a \notin$ ***B which is an error.***

B = {binary palindromes} can't be recognized by any DFA

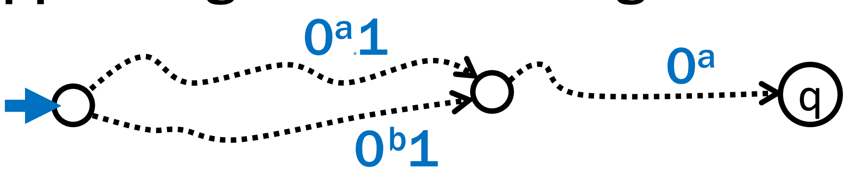
Suppose for contradiction that some DFA, M, recognizes B.

We show M accepts or rejects a string it shouldn't.

Consider $S = \{0^n1 : n \geq 0\}$.

Since there are finitely many states in M and infinitely many strings in S**, there exist strings** $0^a1 \in S$ **and** $0^b1 \in S$ **with** $a \neq b$ **that end in the same state of M.**

Now, consider appending 0^a **to both strings.**



Then, since 0^a1 **and** 0^b1 **end in the same state,** 0^a10^a **and** 0^b10^a **also end in the same state, call it** q**. But then M must make a mistake:** q **needs to be an accept state since** $0^a10^a \in B$**, but then M would accept** $0^b10^a \notin B$ **which is an error.**

This is a contradiction, since we assumed that M recognizes B. Since M was arbitrary, there is no DFA that** recognizes **B.

Showing that a Language L is not regular

1. "Suppose for contradiction that some DFA M recognizes L."
2. Consider an **INFINITE** set S of "half strings" (which we intend to complete later). It is imperative that for ***every pair*** of strings in our set there is an "accept" completion that the two strings DO NOT SHARE.
3. "Since **S** is infinite and **M** has finitely many states, there must be two strings $\mathbf{s_a}$ and $\mathbf{s_b}$ in **S** for some $\mathbf{s_a} \neq \mathbf{s_b}$ that end up at the same state of **M**."
4. Consider appending the (correct) completion to each of the two strings.
5. "Since $\mathbf{s_a}$ and $\mathbf{s_b}$ both end up at the same state of **M**, and we appended the same string **t**, both $\mathbf{s_a t}$ and $\mathbf{s_b t}$ end at the same state q of **M**. Since $\mathbf{s_a t} \in \mathbf{L}$ and $\mathbf{s_b t} \notin \mathbf{L}$, **M** does not recognize **L**."
6. "Since **M** was arbitrary, no DFA recognizes **L**."