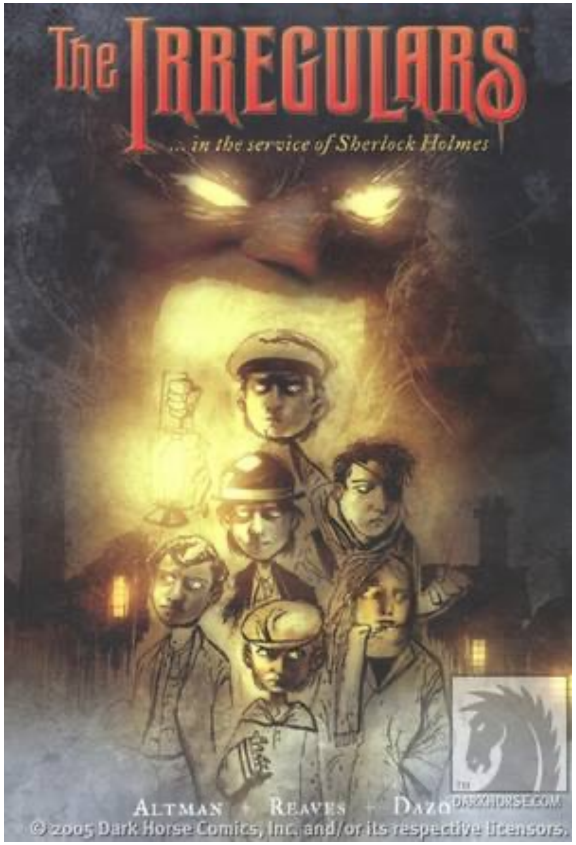
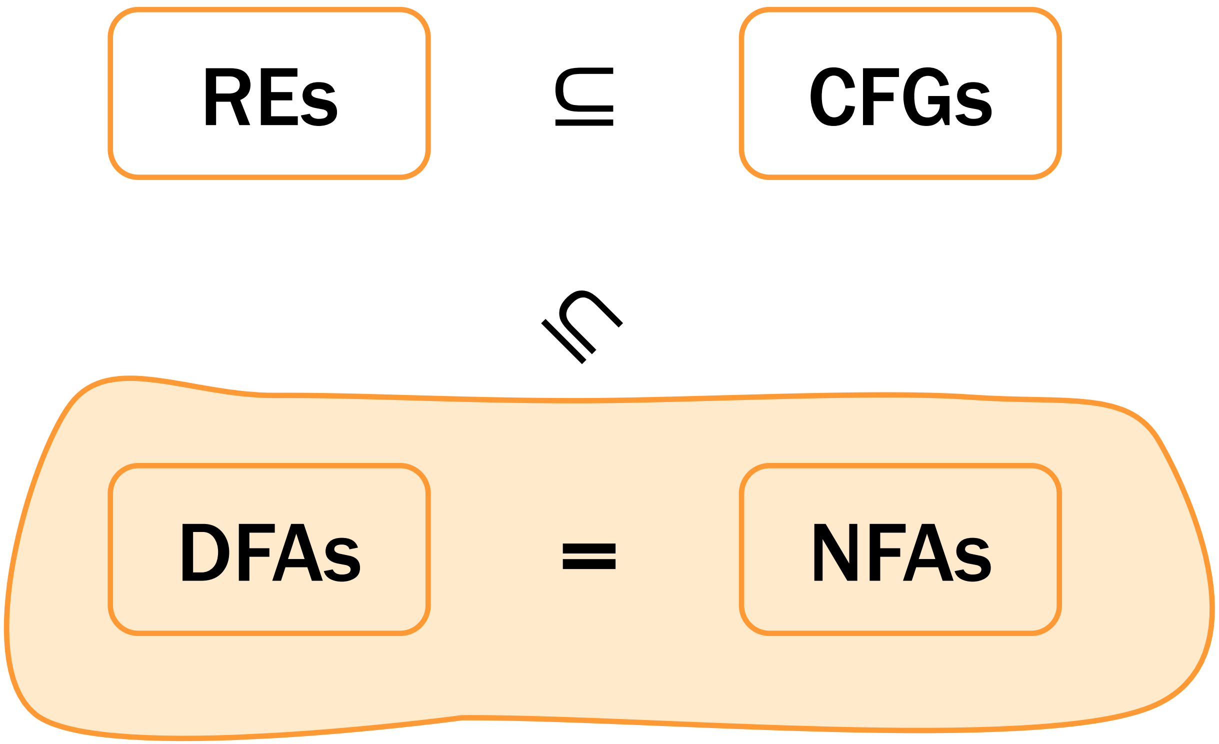
CSE 311: Foundations of Computing

Lecture 25: Languages vs Representations: Limitations of Finite Automata and Regular Expressions



The story so far...



Regular expressions ⊆ NFAs ≡ DFAs

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

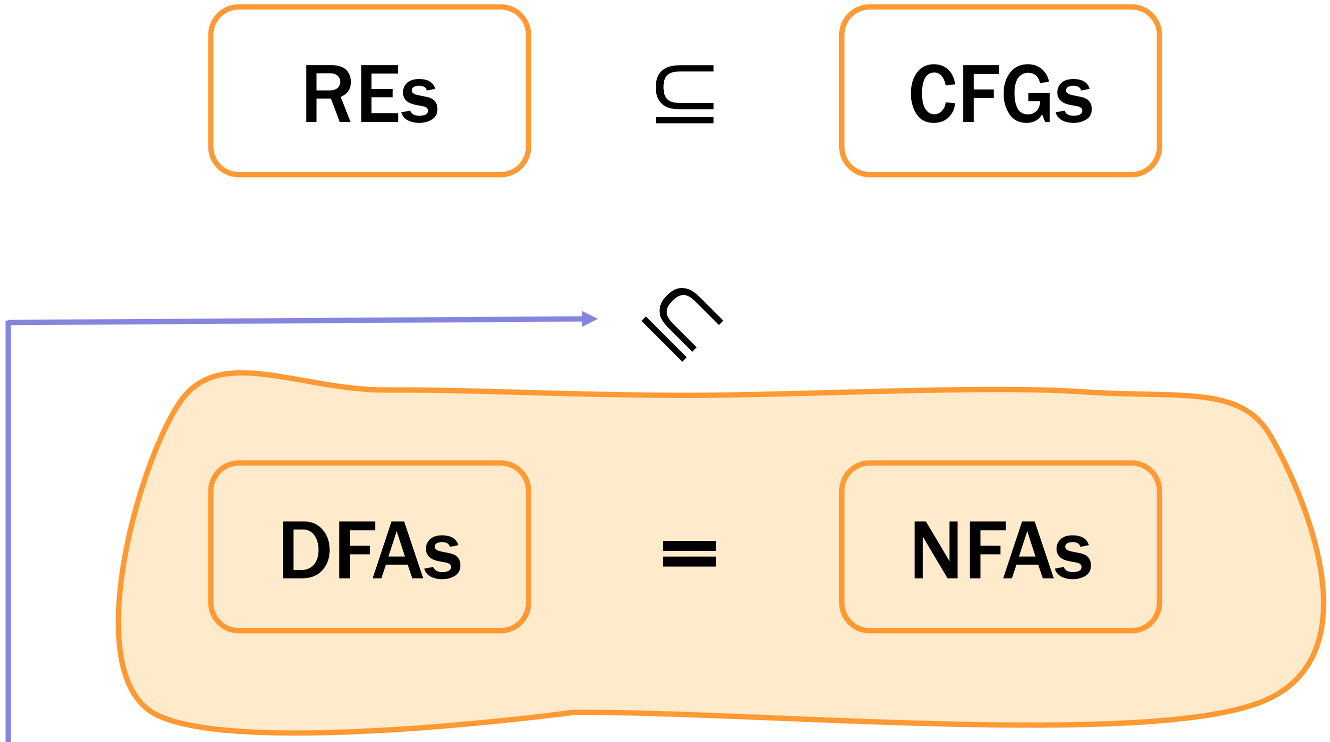
Thus, we could now implement a RegExp library

- most RegExp libraries actually simulate the NFA

  by constructing just the parts that are needed during the execution

- (even better: one can combine the two approaches: apply DFA minimization lazily while simulating the NFA)

The story so far...



Is this ⊆ really "=" or "⊊"?

Regular expressions ≡ NFAs ≡ DFAs

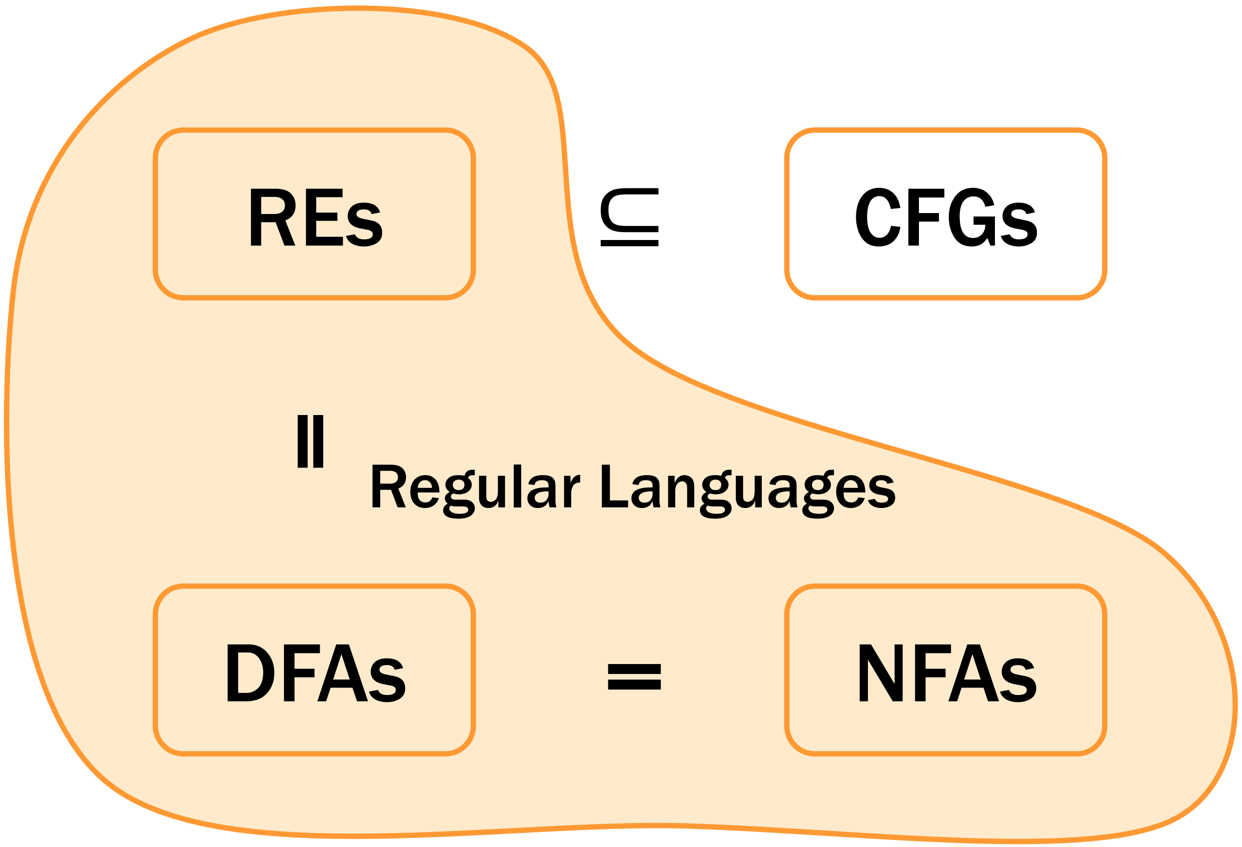
Theorem: For any NFA, there is a regular expression that defines the same language

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

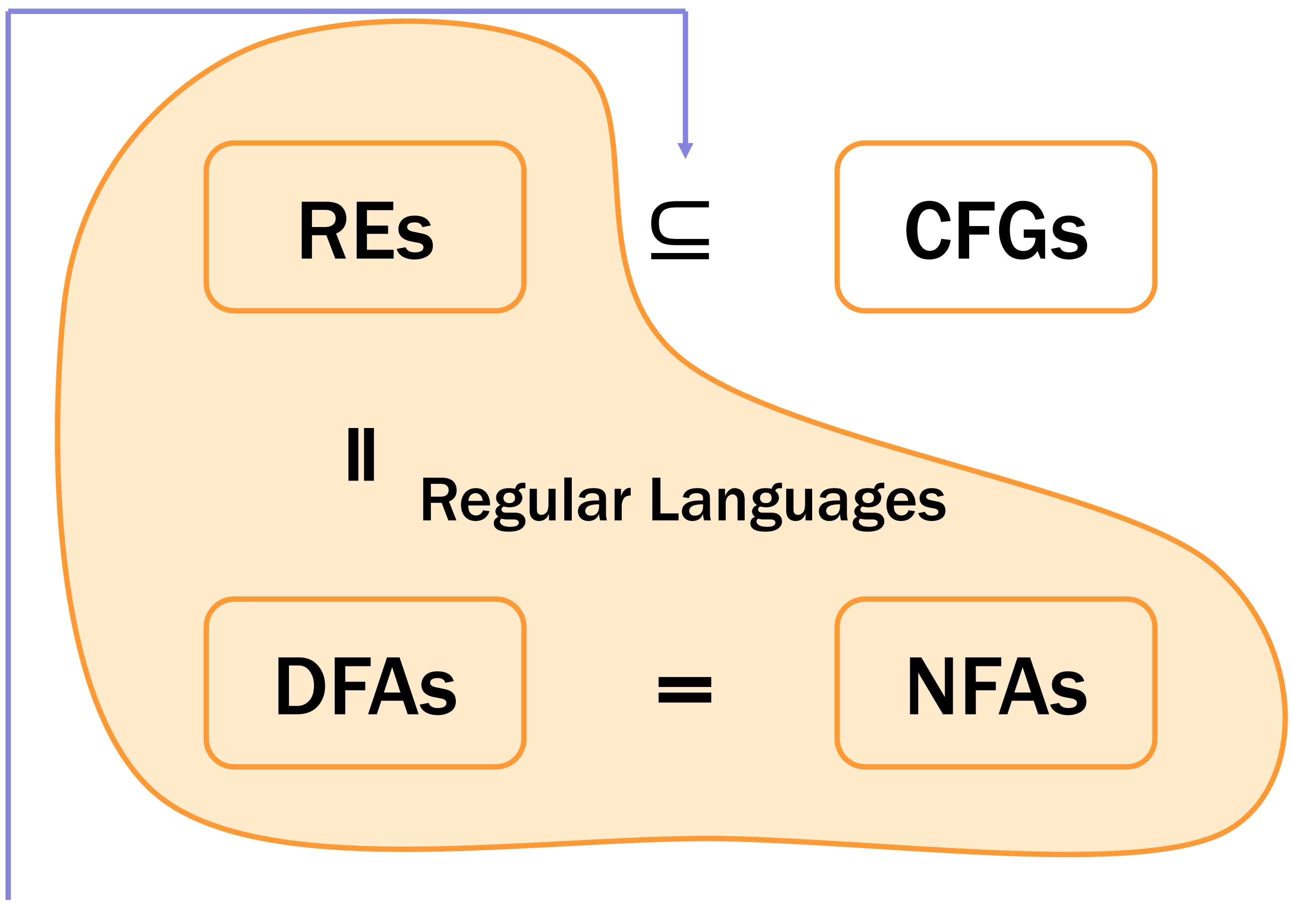
You need to know these facts

- the construction for the Theorem is included in the slides after this, but you will not be tested on it

The story so far...



The story so far...



Next time: Is this ⊆ really "=" or "⊊"?

Last time: Algorithms for Regular Languages

We have seen algorithms for

- RE to NFA
- NFA to DFA
- DFA/NFA to RE (not tested)
- DFA minimization

Practice three of these in HW.

(May also be on the final.)

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

Applications of FSMs

- Implementation of regular expression matching in programs like `grep`
- Control structures for sequential logic in digital circuits
- Algorithms for communication and cache-coherence protocols
  - Each agent runs its own FSM
- Design specifications for reactive systems
  - Components are communicating FSMs

Applications of FSMs

- Formal verification of systems
  - Is an unsafe state reachable?
- Computer games
  - FSMs provide worlds to explore
- Minimization algorithms for FSMs can be extended to more general models used in
  - Text prediction
  - Speech recognition

Application of FSMs: Pattern matching

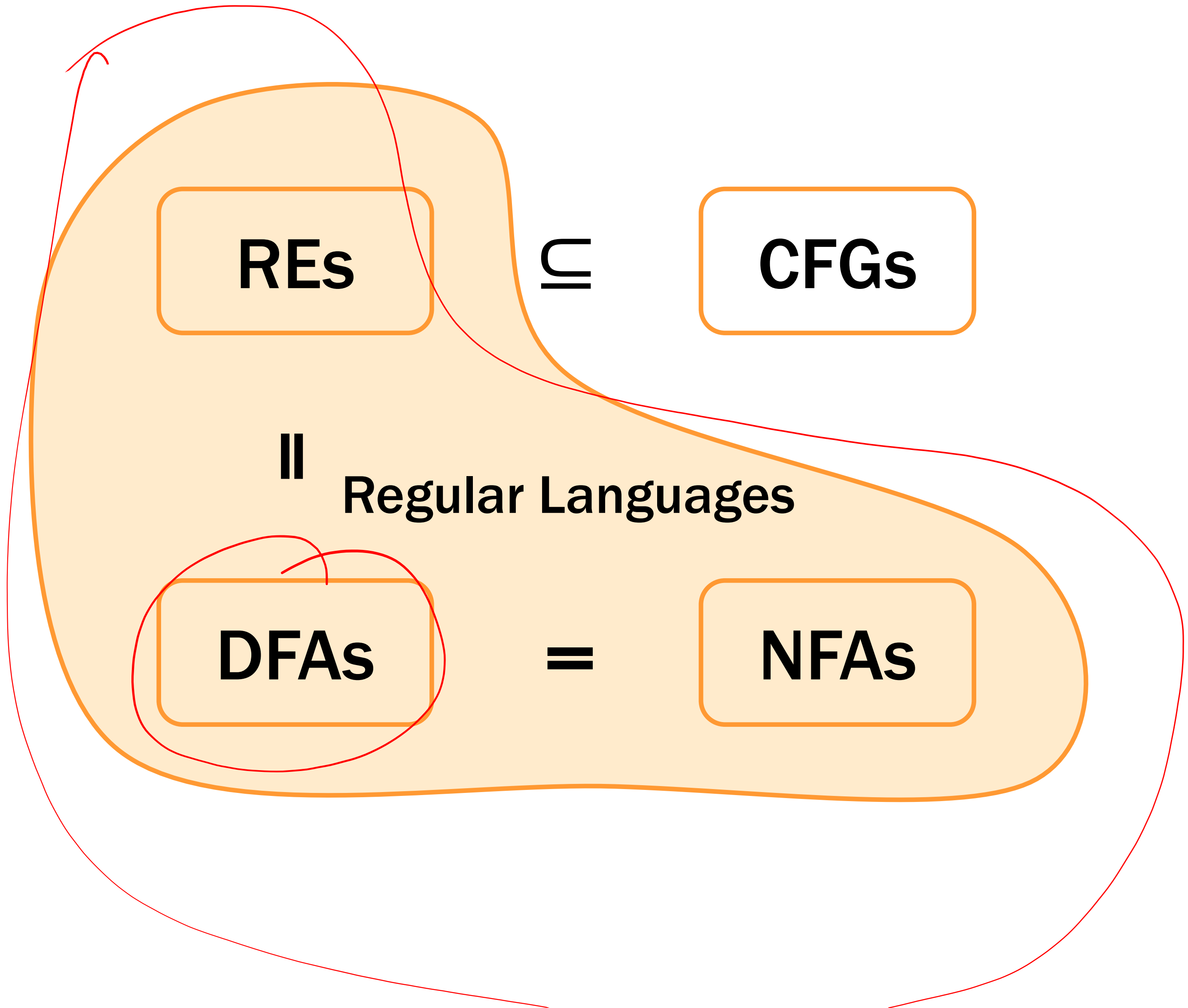
- Given
  - a string s of n characters
  - a pattern p of m characters
  - usually $m \ll n$
- Find
  - all occurrences of the pattern p in the string s
- Obvious algorithm:
  - try to see if p matches at each of the positions in s
    stop at a failed match and try matching at the next position: $O(mn)$ running time.

Application of FSMs: Pattern Matching

- With DFAs can do this in $O(m+n)$ time.

- See Extra Credit problem on HW8 for some ideas of how to get to $O(m^2 + n)$ time.

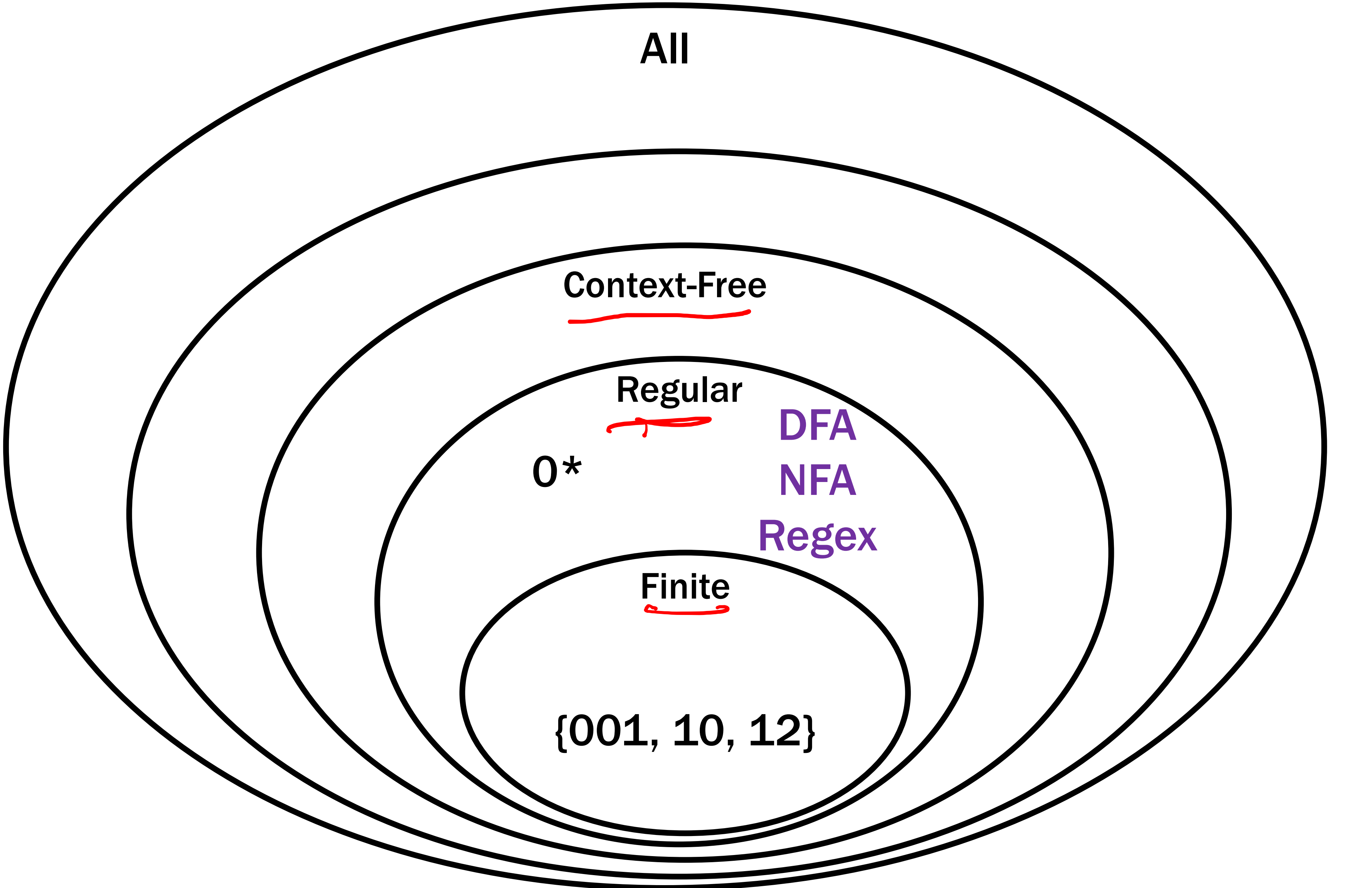
The story so far...



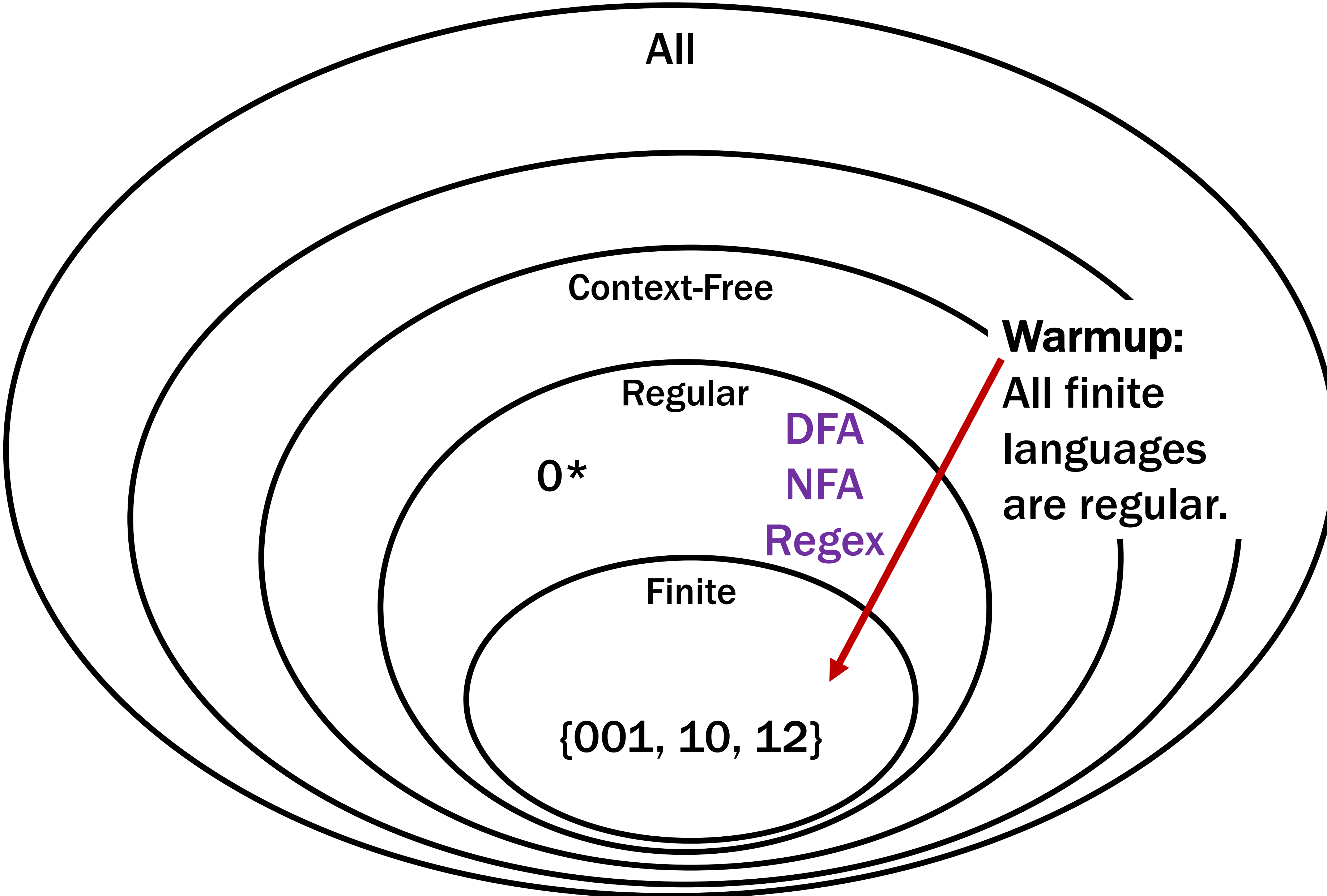
What languages have DFAs? CFGs?

All of them?

Languages and Representations!



Languages and Representations!



DFAs Recognize Any Finite Language

000

110

0

000000

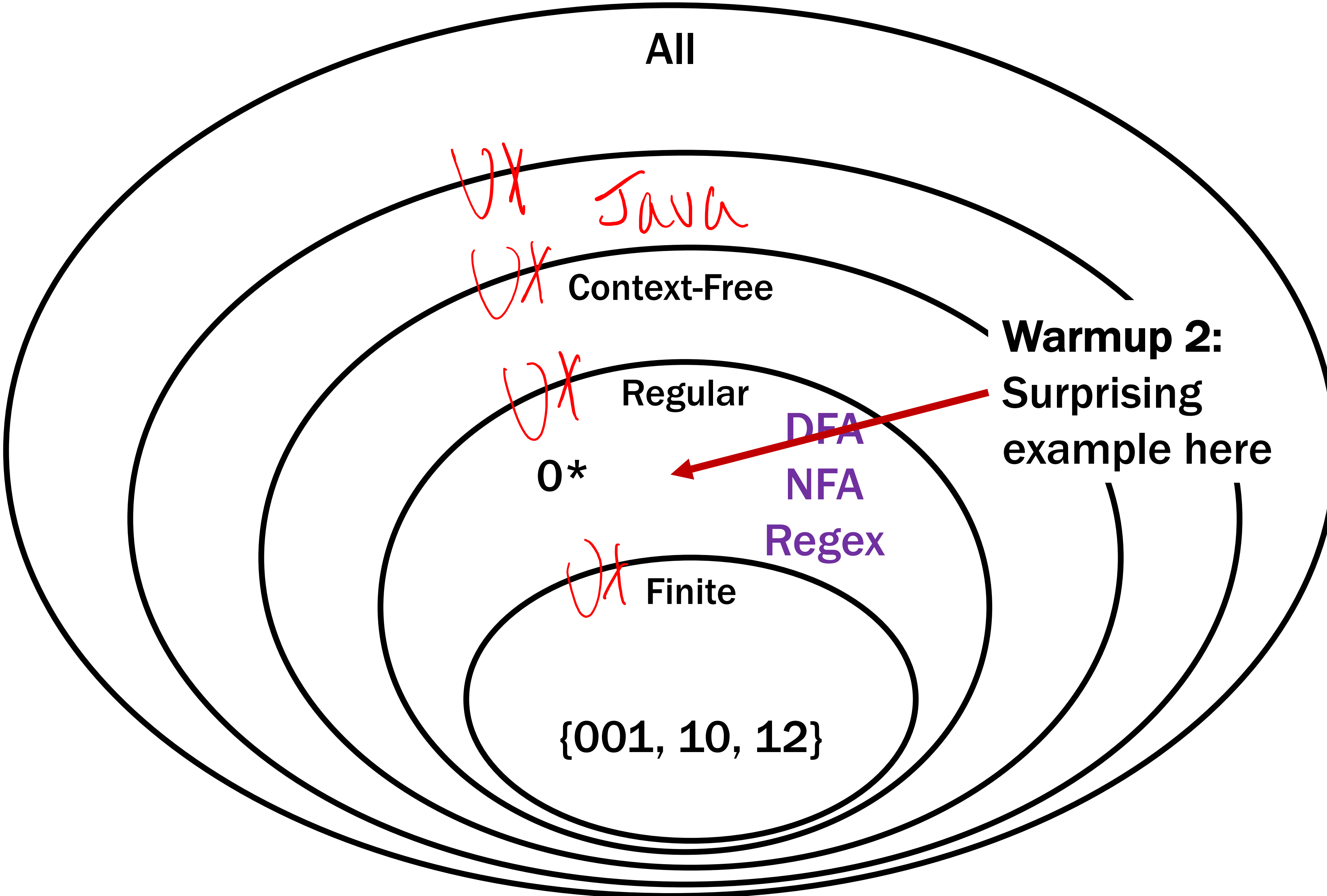
RE

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?

That seems to require comparing counts...

- easy for a CFG
- but seems hard for DFAs!

$0^n 1^n$

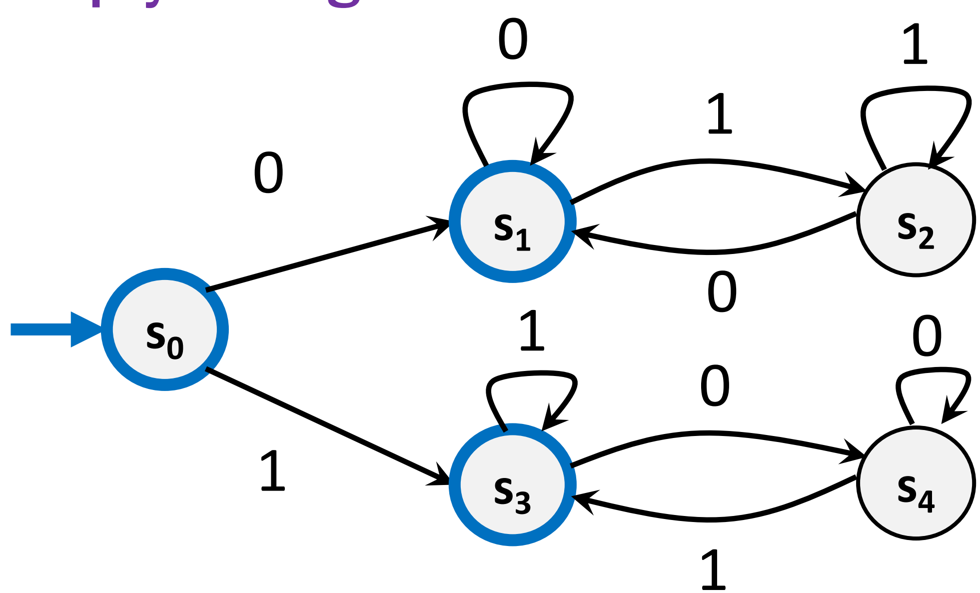
An Interesting Infinite Regular Language

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

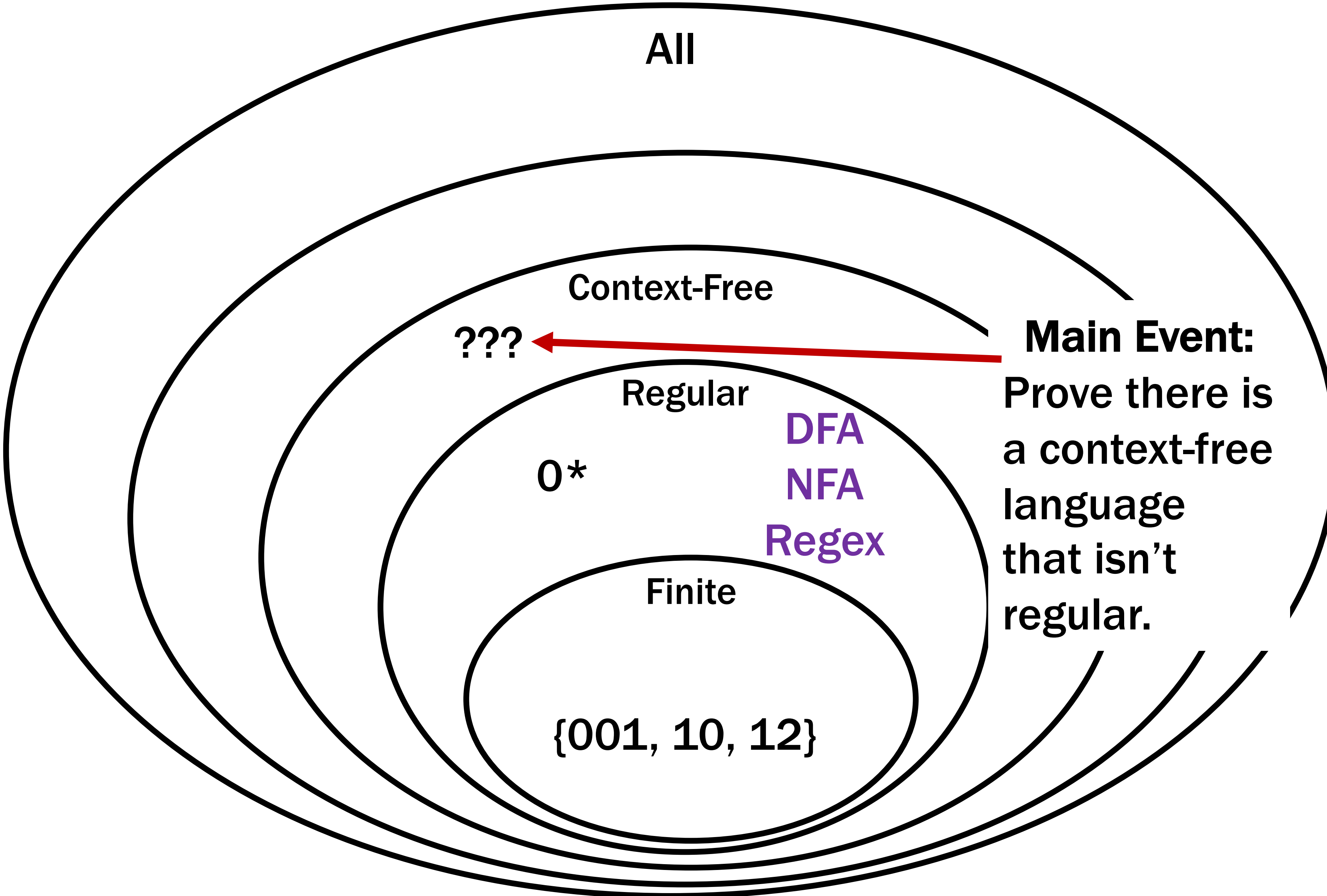
L is infinite.

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 ?

Intuition (NOT A PROOF!):

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

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.

Proof idea: any machine that does not remember the entire first half will be wrong for some inputs

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

The general proof strategy is:

– Assume (for contradiction) that some DFA (call it M) exists that recognizes B

0^*

$S = \{0^n \mid n \geq 0\}$

$0^a z$

$0^b z$

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

The general proof strategy is:

- Assume (for contradiction) that some DFA (call it M) exists that recognizes B
- Our goal is to show that M actually does not recognize B

How can a DFA fail to recognize B?

- 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 some DFA (call it M) exists that recognizes B
– Our goal is to show that M actually does not recognize B, i.e., it accepts or rejects a string that it shouldn't

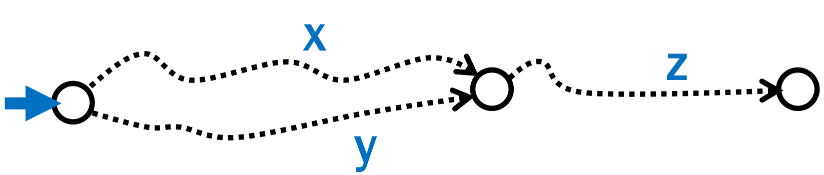
"M recognizes B" AND "M doesn't recognize B", which is a contradiction

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

The general proof strategy is:

- Assume (for contradiction) that
  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!



$x \bullet z = x_1\, x_2 \ldots x_n\, z_1\, z_2 \ldots z_k$

$y \bullet z = y_1\, y_2 \ldots y_m\, z_1\, z_2 \ldots z_k$

M is correct iff $\forall z \in \Sigma^* \, (x \bullet z \in B \leftrightarrow y \bullet z \in B)$

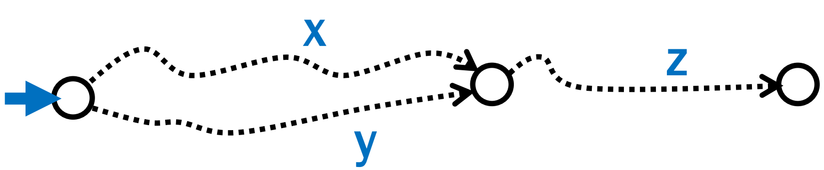
M is incorrect iff $\exists\, z \in \Sigma^* \, (x \bullet z \in B \not\leftrightarrow y \bullet z \in B)$

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

The general proof strategy is:

- Assume (for contradiction) that 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
  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 prefixes
(which we intend to complete later).

x,y z

1______

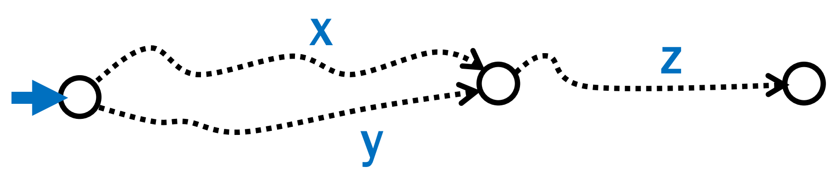
01______

001______

0001______

00001______

....



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

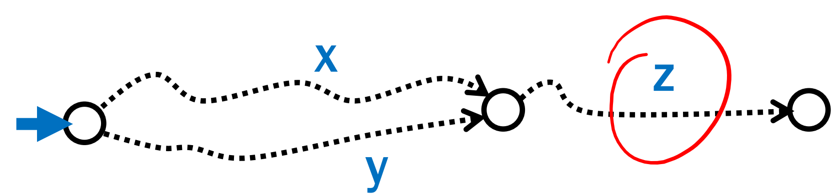
The general proof strategy is:

– Assume (for contradiction) that
  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 prefixes (which we intend to complete later). It is critical that for ***every pair*** of strings in our set there is an "accept" completion that the two strings DO NOT SHARE.

x,y   z
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 just proven they exist…we must take the ones we're given!

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

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

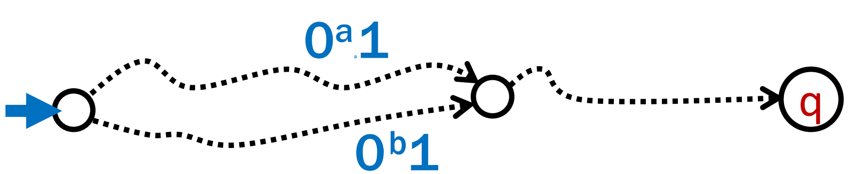
We show M accepts or rejects a string it shouldn't.

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

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

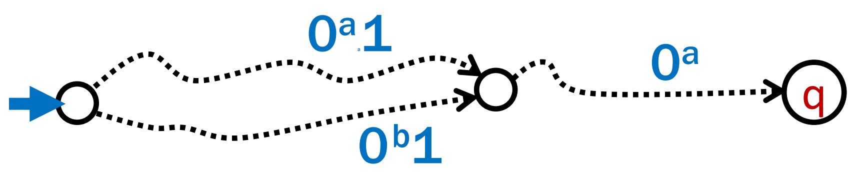
Suppose for contradiction that some DFA, M, recognizes B.

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

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

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 makes a mistake: q ***needs to be an accept state since*** $0^a10^a \in B$***, but M would accept*** $0^b10^a \notin B$ ***which is an error.***

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

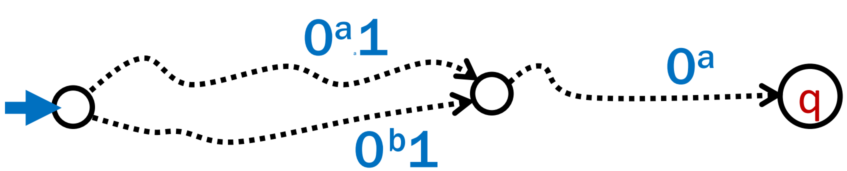
Suppose for contradiction that some DFA, M, recognizes B.

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

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

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. Thus, no DFA 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 prefxes (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 s_a and s_b in S for $s_a \neq s_b$ that end up at the same state of M."
4. Consider appending the (correct) completion t to each of the two strings.
5. "Since s_a and s_b both end up at the same state of M, and we appended the same string t, both $s_a t$ and $s_b t$ end at the same state q of M. Since $s_a t \in L$ and $s_b t \notin L$, M does not recognize L."
6. "Thus, no DFA recognizes L."

Prove $A = \{0^n 1^n : n \geq 0\}$ is not regular

Suppose for contradiction that some DFA, **M**, recognizes **A**.

Let **S =**

Prove A = $\{0^n1^n : n \geq 0\}$ is not regular

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

Let S = $\{0^n : n \geq 0\}$. Since S is infinite and M has finitely many states, there must be two strings, 0^a and 0^b for some $a \neq b$ that end in the same state in M.

$0^a z \in A$

$0^b z \notin A$

$z = 1^a$

$0^a 1^a \in A$

$0^b 1^a \notin A$

Prove A = $\{0^n1^n : n \geq 0\}$ is not regular

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

Let S = $\{0^n : n \geq 0\}$. Since S is infinite and M has finitely many states, there must be two strings, 0^a and 0^b for some $a \neq b$ that end in the same state in M.

Consider appending 1^a to both strings.

Prove A = $\{0^n1^n : n \geq 0\}$ is not regular

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

Let S = $\{0^n : n \geq 0\}$. Since S is infinite and M has finitely many states, there must be two strings, 0^a and 0^b for some $a \neq b$ that end in the same state in M.

Consider appending 1^a to both strings.

Note that $0^a1^a \in$ A, but $0^b1^a \notin$ A since $a \neq b$. But they both end up in the same state of M, call it q. Since $0^a1^a \in$ A, state q must be an accept state but then M would incorrectly accept $0^b1^a \notin$ A so M does not recognize A.

Thus, no DFA recognizes A.

Prove P = {balanced parentheses} is not regular

Suppose for contradiction that some DFA, M, accepts P.

Let S =

Prove P = {balanced parentheses} is not regular

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

Let S = $\{ (^n : n \geq 0\}$**. Since S is infinite and M has finitely many states, there must be two strings,** $(^a$ **and** $(^b$ **for some** $a \neq b$ **that end in the same state in M.**

Prove P = {balanced parentheses} **is not regular**

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

Let S = $\{ (^n : n \geq 0\}$. Since S is infinite and M has finitely many states, there must be two strings, $(^a$ and $(^b$ for some $a \neq b$ that end in the same state in M.

Consider appending $)^a$ to both strings.

()()

Prove P = {balanced parentheses} is not regular

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

Let $S = \{ (^n : n \geq 0\}$. Since S is infinite and M has finitely many states, there must be two strings, $(^a$ and $(^b$ for some $a \neq b$ that end in the same state in M.

Consider appending $)^a$ to both strings.

Note that $(^a)^a \in P$, but $(^b)^a \notin P$ since $a \neq b$. But they both end up in the same state of M, call it q. Since $(^a)^a \in P$, state q must be an accept state but then M would incorrectly accept $(^b)^a \notin P$ so M does not recognize P.

Thus, no DFA recognizes P.

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 prefixes (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. (You need to come up with S.)
3. "Since S is infinite and M has finitely many states, there must be two strings s_a and s_b in S for $s_a \neq s_b$ that end up at the same state of M."
4. Consider appending the (hard) completion t to each of the two strings. (You need to come up with a hard t for s_a, s_b)
5. "Since s_a and s_b both end up at the same state of M, and we appended the same string t, both $s_a t$ and $s_b t$ end at the same state q of M. Since $s_a t \in L$ and $s_b t \notin L$, M does not recognize L."
6. "Thus, no DFA recognizes L."

Fact: This method is optimal

- Suppose that for a language L, the set S is a *largest* set of prefixes with the property that, for every pair $s_a \neq s_b \in S$, there is some string t such that one of $s_a t$, $s_b t$ is in L but the other isn't.
- If S is infinite, then L is not regular
- If S is finite, then the minimal DFA for L has precisely $|S|$ states, one reached by each member of S.

Fact: This method is optimal

- Suppose that for a language L, the set S is a *largest* set of prefixes with the property that, for every pair $s_a \neq s_b \in S$, there is some string t such that one of $s_a t$, $s_b t$ is in L but the other isn't.
- If S is infinite, then L is not regular
- If S is finite, then the minimal DFA for L has precisely $|S|$ states, one reached by each member of S.

Corollary: Our minimization algorithm was correct.

– we separated *exactly* those states for which some t would make one accept and another not accept

Important Notes

- It is not necessary for our strings **xz** with **$x \in L$** to allow any string in the language
  - we only need to find a small "core" set of strings that must be distinguished by the machine
- It is **not true** that, if **L** is irregular and **$L \subseteq U$**, then **U** is irregular!
  - we always have **$L \subseteq \Sigma^*$** and **Σ^*** is regular!
  - our argument needs different answers: $xz \in L \not\leftrightarrow yz \in L$

    for **Σ^***, both strings are always in the language

Do not claim in your proof that, because $L \subseteq U$, U is also irregular