CSE 311: Foundations of Computing

Lecture 26: From NFAs to DFAs and from NFAs to REs

Recap: Concepts to describe languages

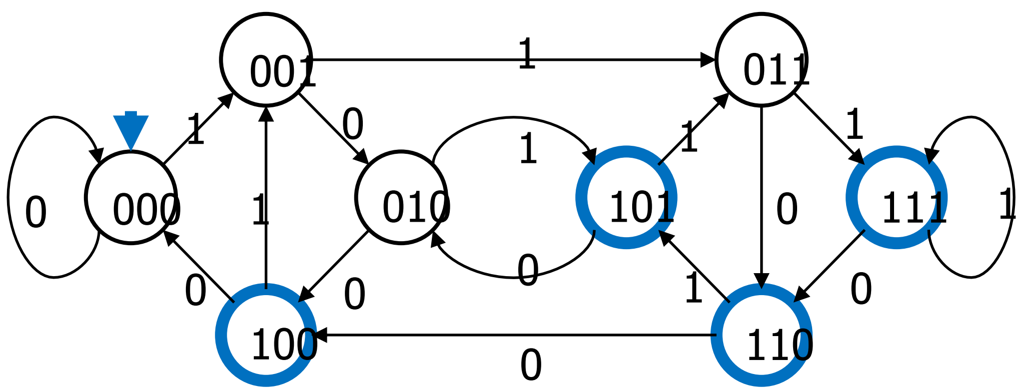
REs $\subseteq$ **CFGs**

$\subseteq$

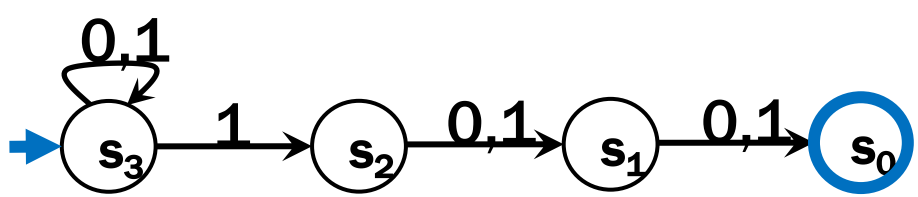
DFAs $\subseteq$ **NFAs**

Regular expression: $(0 \cup 1)^* 1 (0 \cup 1)(0 \cup 1)$

DFA:



NFA:



NFAs and DFAs

Every DFA **is** an NFA

– DFAs have requirements that NFAs don't have

Can NFAs recognize more languages?

NFAs and DFAs

Every DFA **is** an NFA

- DFAs have requirements that NFAs don't have

Can NFAs recognize more languages? No!

Theorem: For every NFA there is a DFA that recognizes exactly the same language

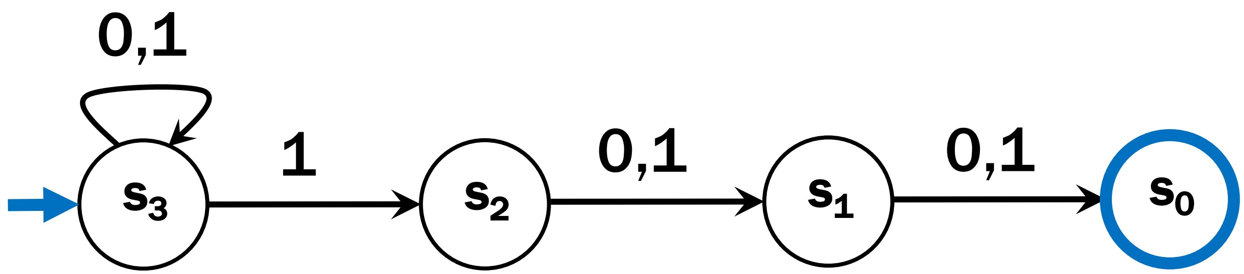
Three ways of thinking about NFAs

- Outside observer: Is there a path labeled by x from the start state to some final state?

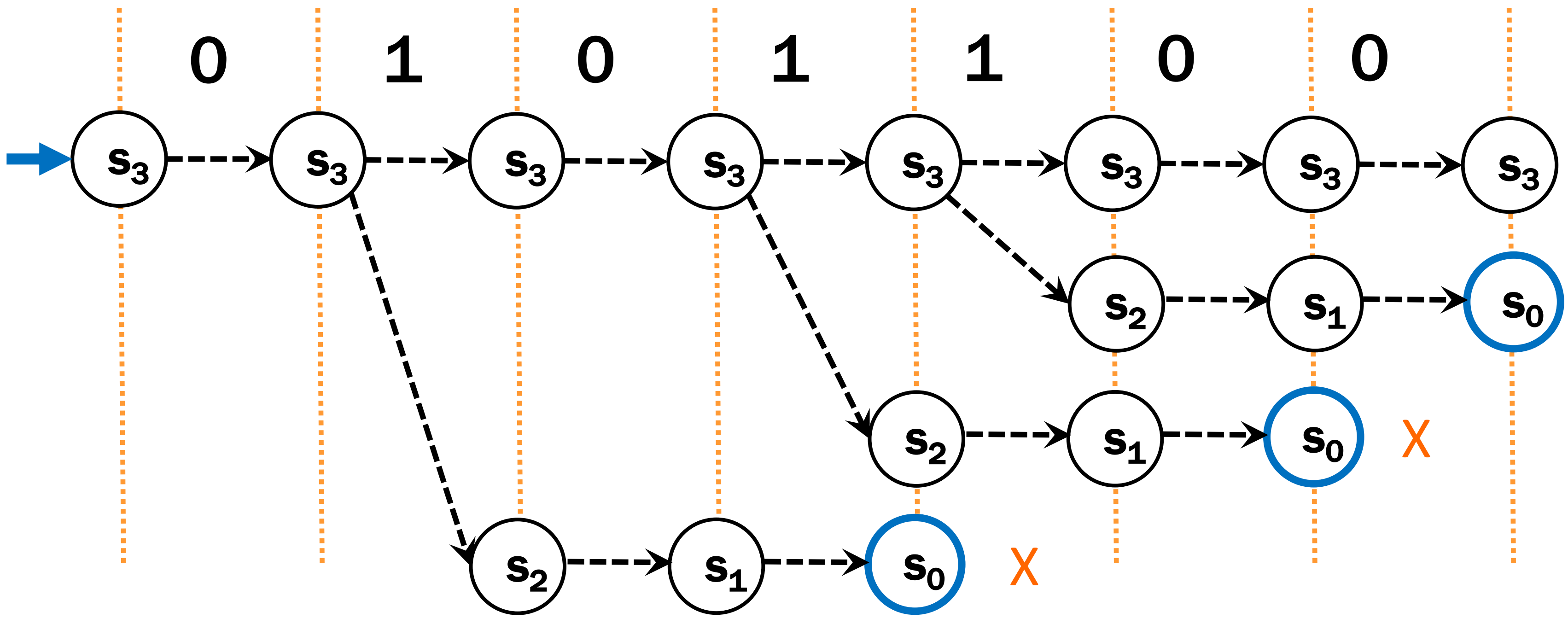
- Perfect guesser: The NFA has input x and whenever there is a choice of what to do it magically guesses a good one (if one exists)

- Parallel exploration: The NFA computation runs all possible computations on x step-by-step at the same time in parallel

Parallel Exploration view of an NFA



Input string 0101100



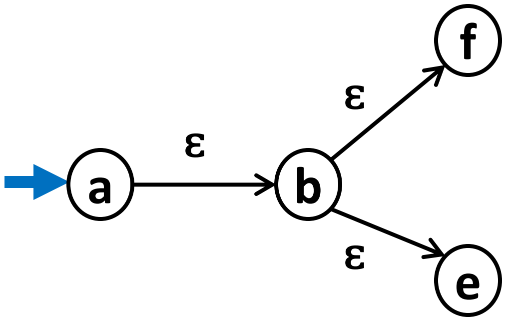
Conversion of NFAs to a DFAs

- Proof Idea:
  - The DFA keeps track of ALL the states that the part of the input string read so far can reach in the NFA
  - There will be one state in the DFA for each *subset* of states of the NFA that can be reached by some string

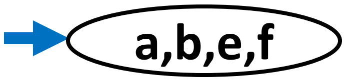
Conversion of NFAs to a DFAs

New start state for DFA

– The set of all states reachable from the start state of the NFA using only edges labeled ε



NFA



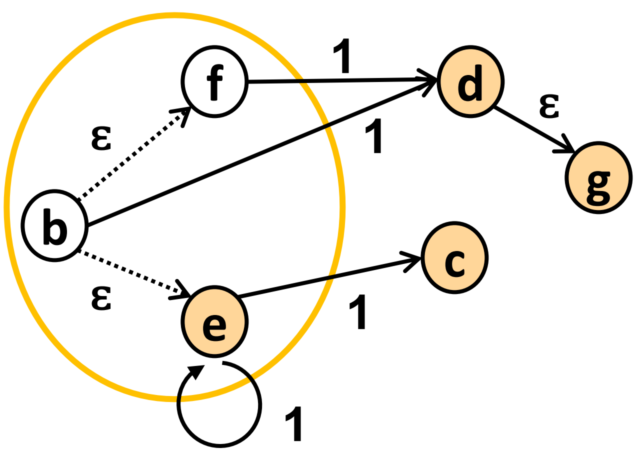
DFA

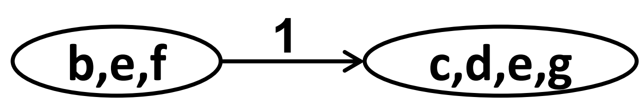
Conversion of NFAs to a DFAs

For each state of the DFA corresponding to a set S of states of the NFA and each symbol s

- **Add an edge labeled** s **to state corresponding to** T**, the set of states of the NFA reached by**
  - starting from some state in S, then
  - following one edge labeled by s, and

    then following some number of edges labeled by **ε**
- T **will be ∅ if no edges from** S **labeled** s **exist**





Conversion of NFAs to a DFAs

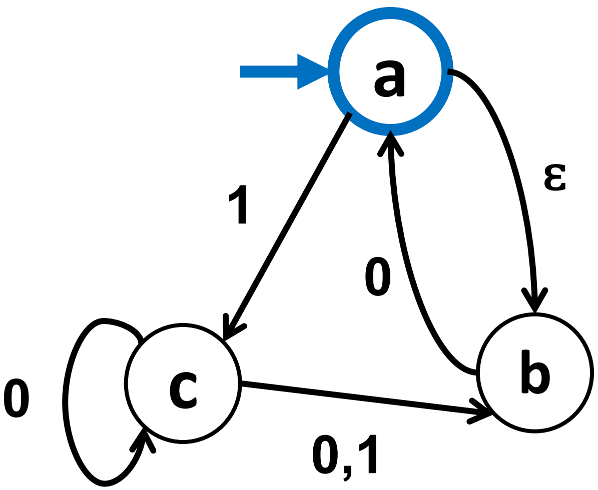
Final states for the DFA

- All states whose set contain some final state of the NFA

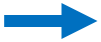
NFA

DFA

Example: NFA to DFA

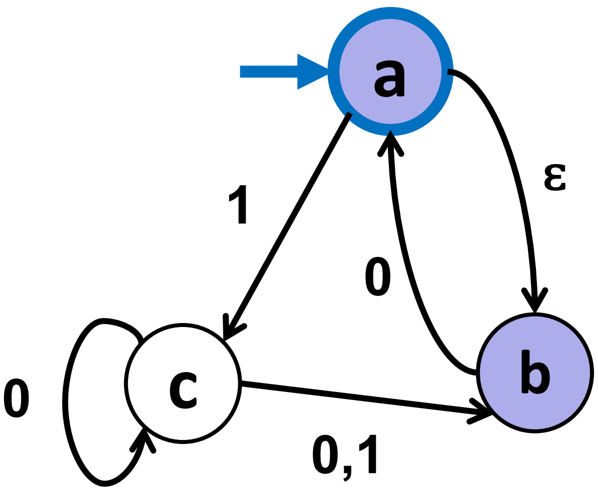


NFA

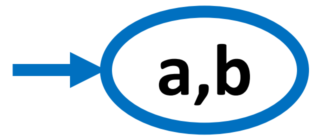


DFA

Example: NFA to DFA

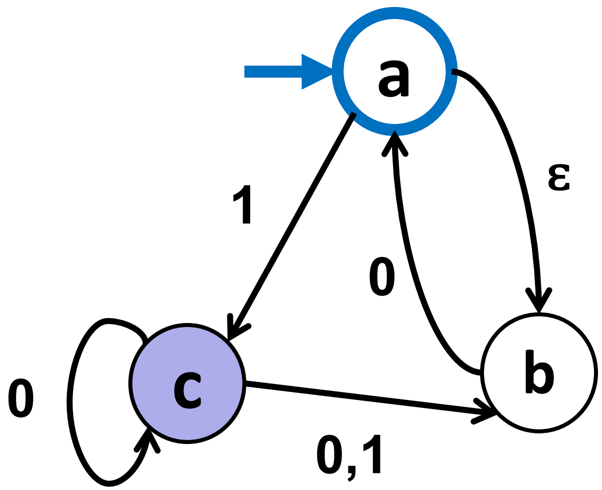


NFA

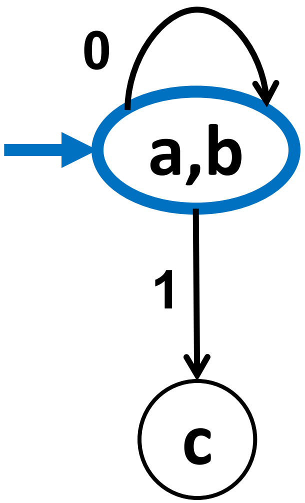


DFA

Example: NFA to DFA

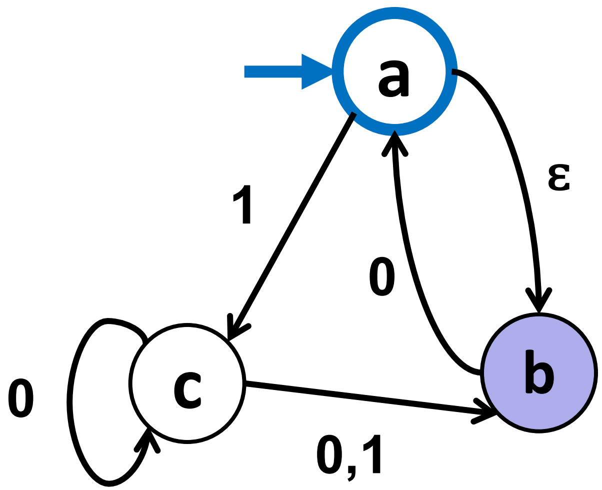


NFA

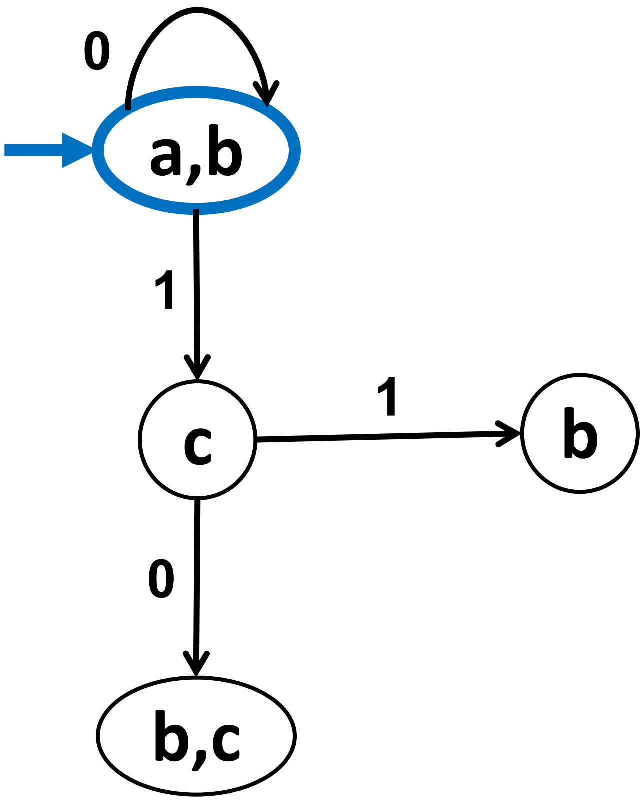


DFA

Example: NFA to DFA

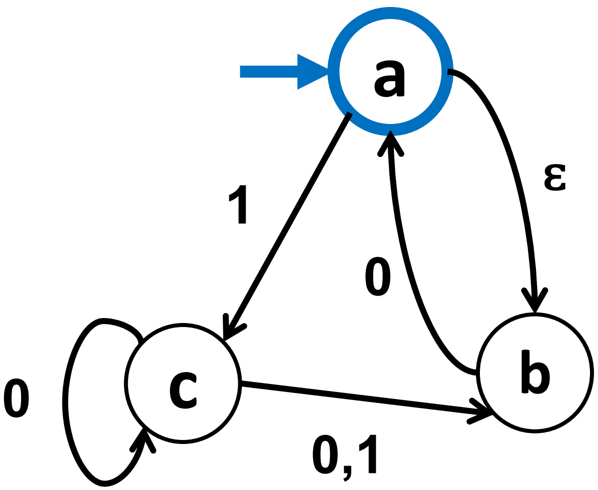


NFA

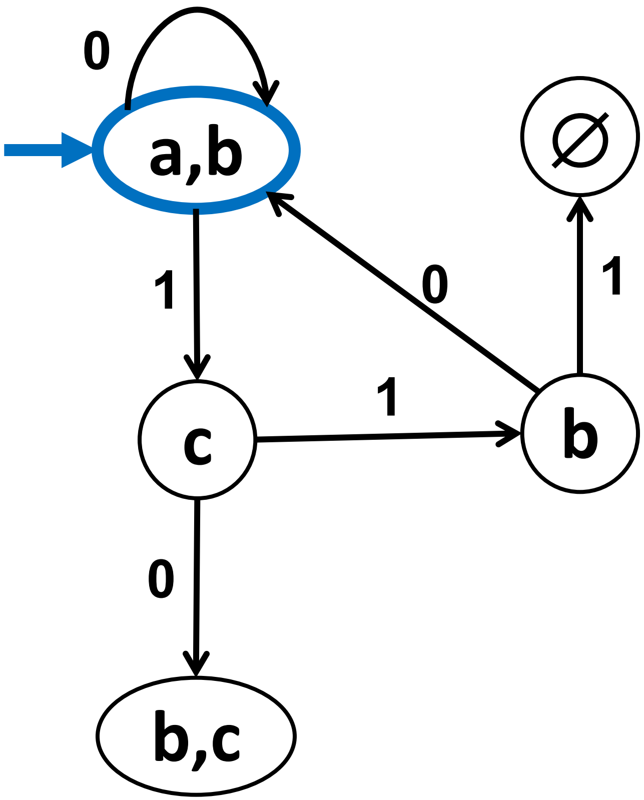


DFA

Example: NFA to DFA

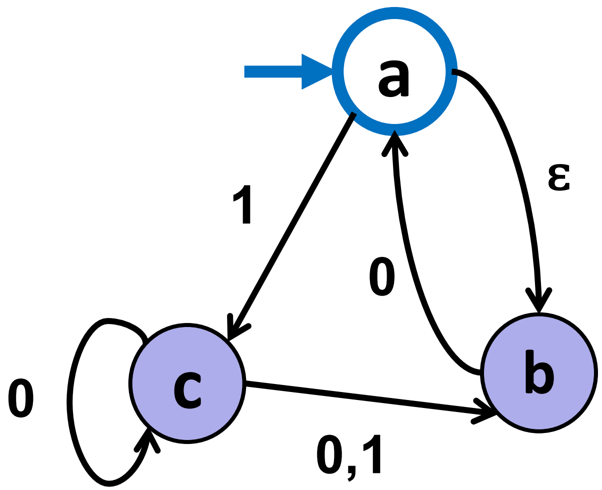


NFA

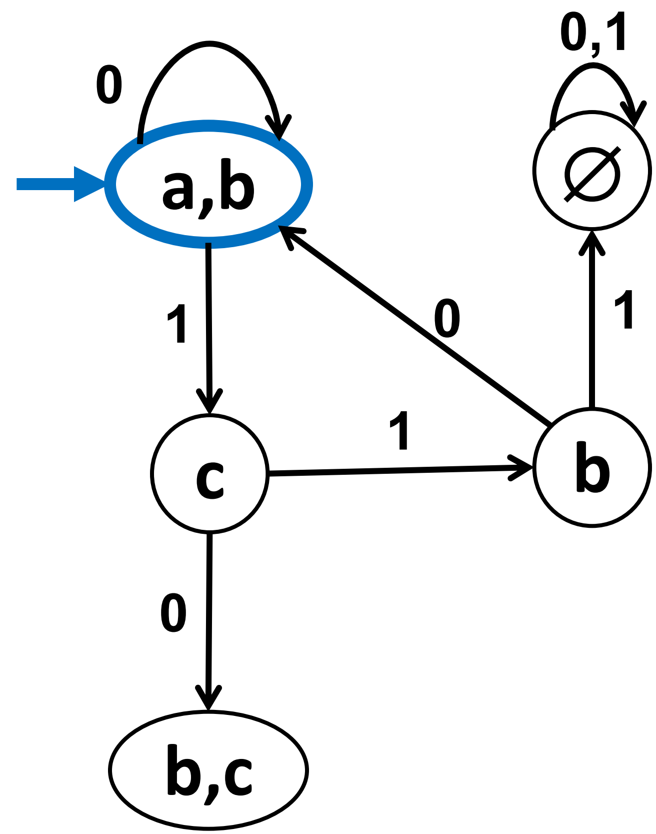


DFA

Example: NFA to DFA

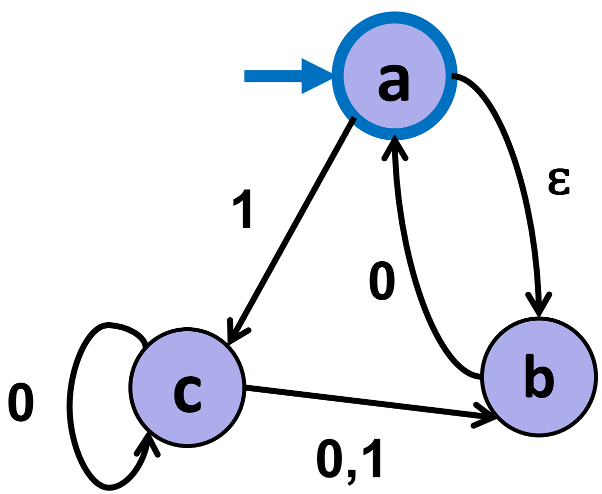


NFA

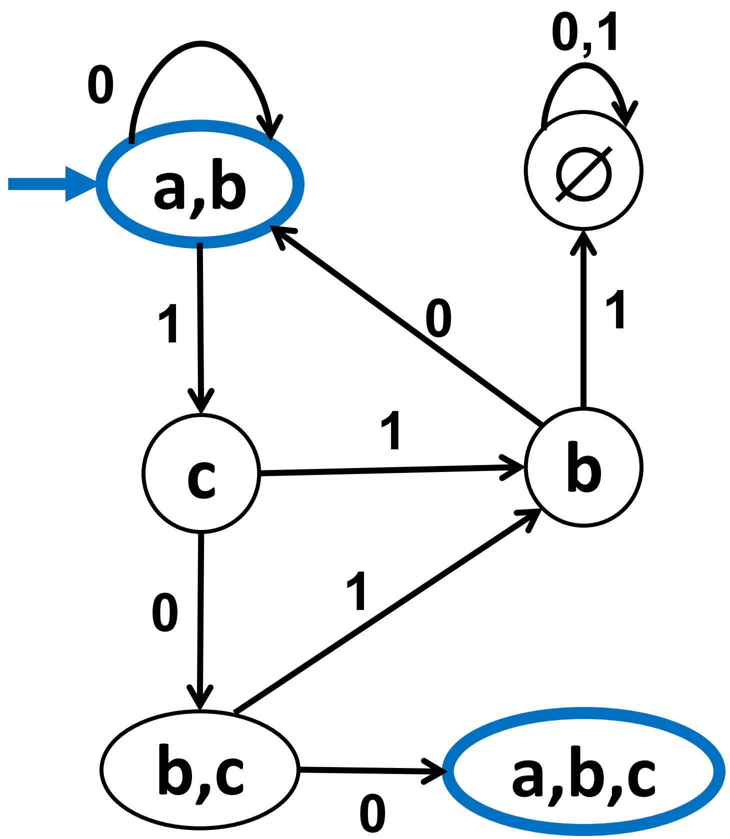


DFA

Example: NFA to DFA

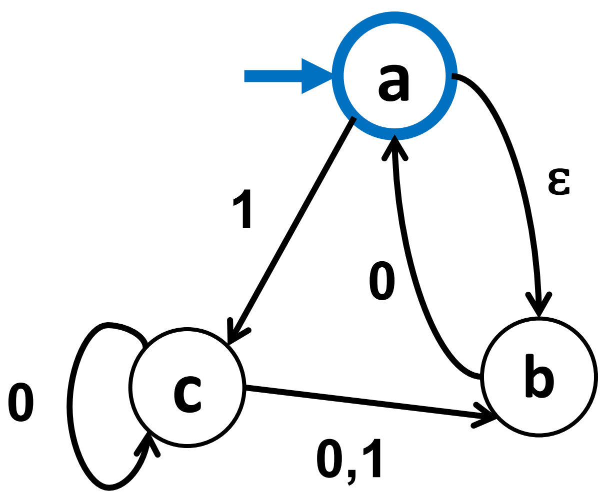


NFA

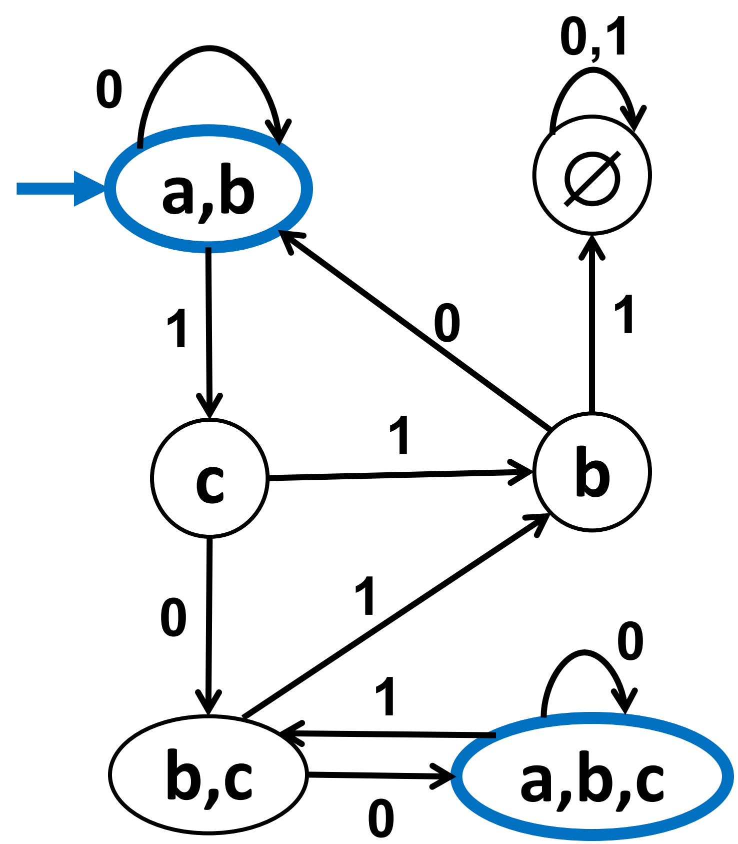


DFA

Example: 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

The story so far...

REs $\subseteq$ **CFGs**

$\subseteq$

DFAs $\equiv$ **NFAs**

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

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

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.

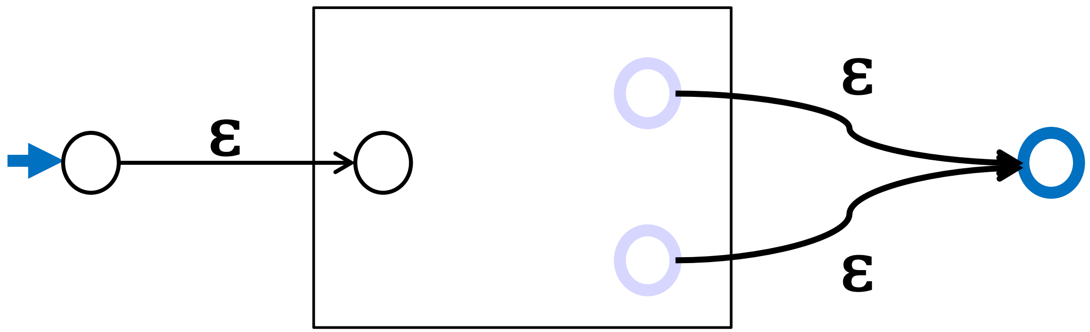
Generalized NFAs

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

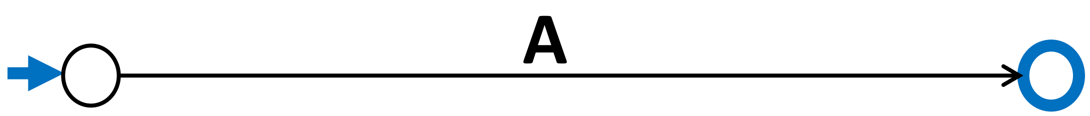
    NFAs already have edges labeled **ε** or ***a***
- **An edge labeled by A can be followed by reading a string of input chars that is in the language represented by A**
- **Defn: A string** x **is accepted iff there is a *path* from start to final state *labeled by a regular expression* whose language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

Starting from an NFA

Add new start state and final state



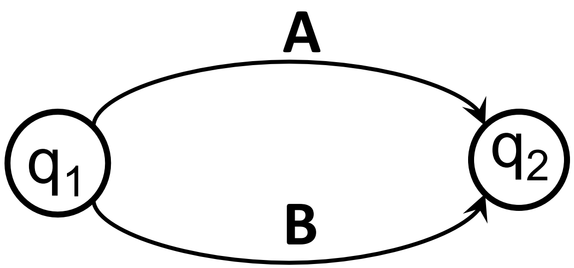
Then eliminate original states one by one, keeping the same language, until it looks like:



Final regular expression will be **A**

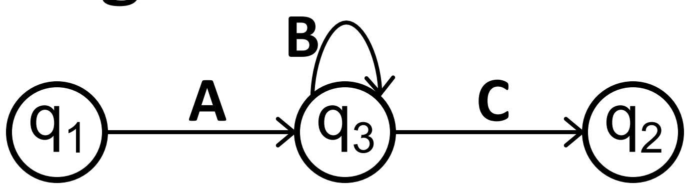
Only two simplification rules

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

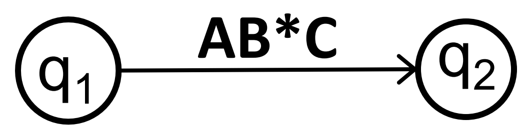


by **?**

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



by

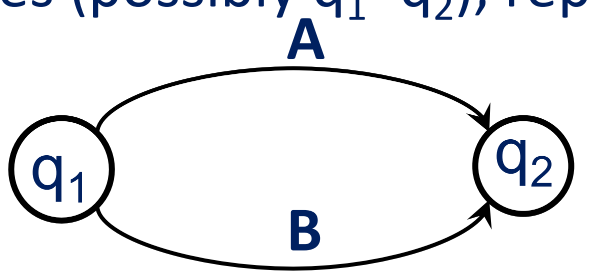


for *every* pair of states q_1, q_2 (even if q_1=q_2)

Lecture 26 Activity

- You will be assigned to **breakout rooms**. Please:
- Introduce yourself
- Choose someone to share screen, showing this PDF
- We are considering Generalized NFAs where we allow parallel edges and edges may be labelled with regular expressions.
- Our overall goal is to transform an arbitrary such generalized NFA into one that only has a single edge.
- Complete the following rule! Why does it work?

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



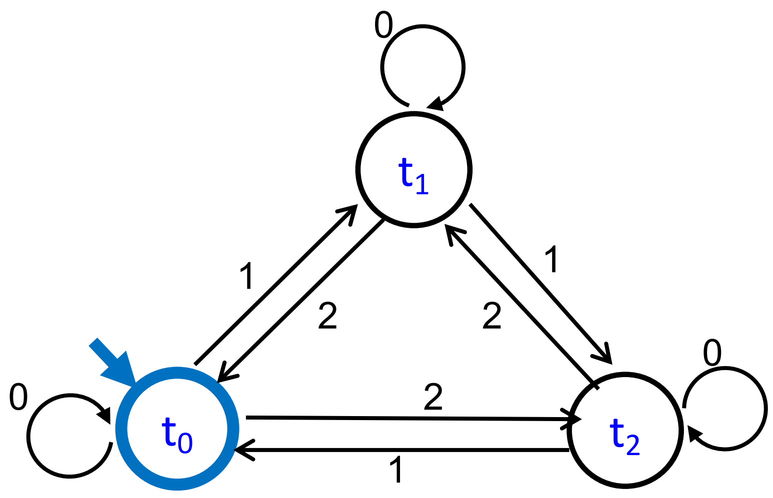
by ?

Fill out a poll everywhere for Activity Credit!
Go to pollev.com/thomas311 and login with your UW identity

Converting an NFA to a regular expression

Consider the DFA for the mod 3 sum

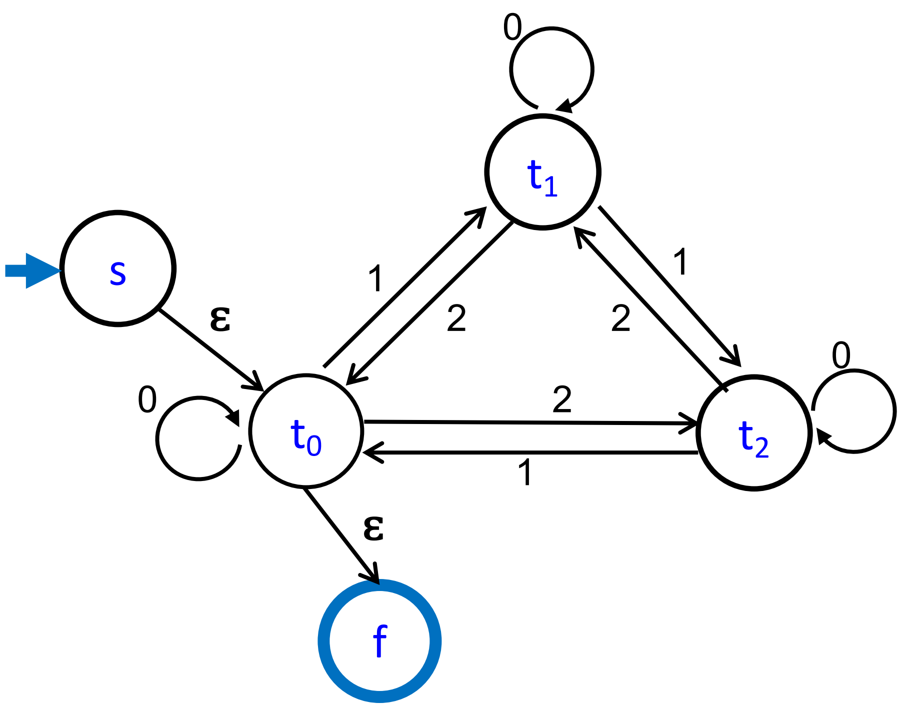
- Accept strings from {0,1,2}* where the digits mod 3 sum of the digits is 0



Splicing out a state t_1

Regular expressions to add to edges

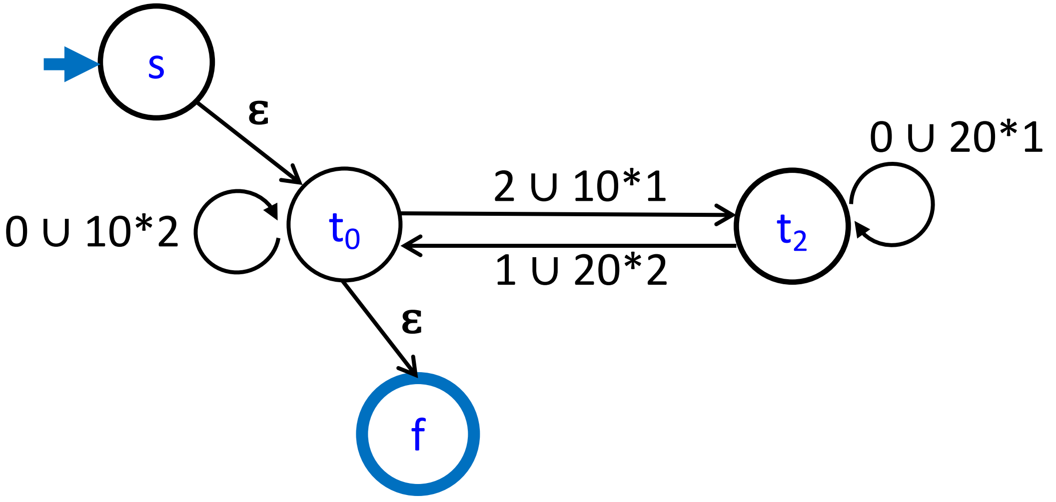
$t_0 \rightarrow t_1 \rightarrow t_0$: 10*2
$t_0 \rightarrow t_1 \rightarrow t_2$: 10*1
$t_2 \rightarrow t_1 \rightarrow t_0$: 20*2
$t_2 \rightarrow t_1 \rightarrow t_2$: 20*1



Splicing out a state t_1

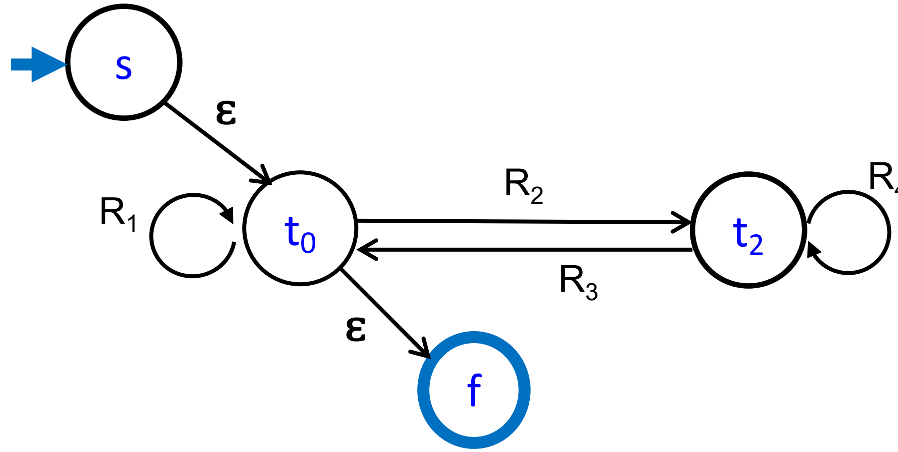
Regular expressions to add to edges

$t_0 \to t_1 \to t_0$: 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

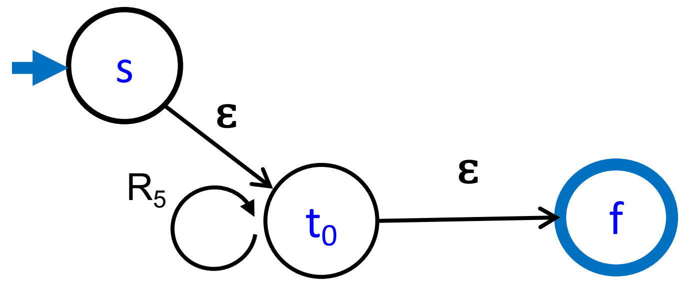


Splicing out state t_2 (and then t_0)

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

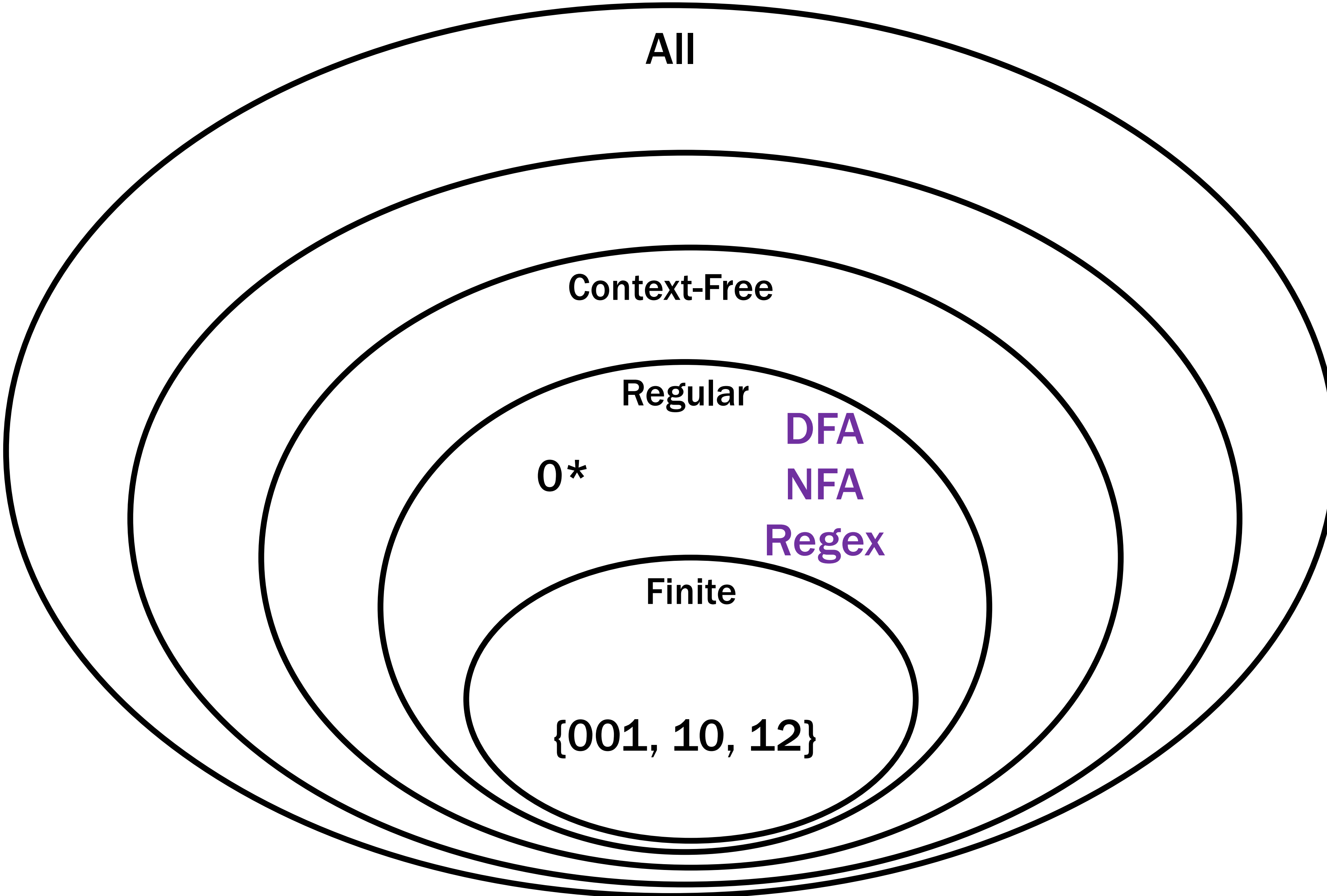
The story so far...

REs	$\subseteq$	CFGs
$\equiv$		
DFAs	$\equiv$	NFAs

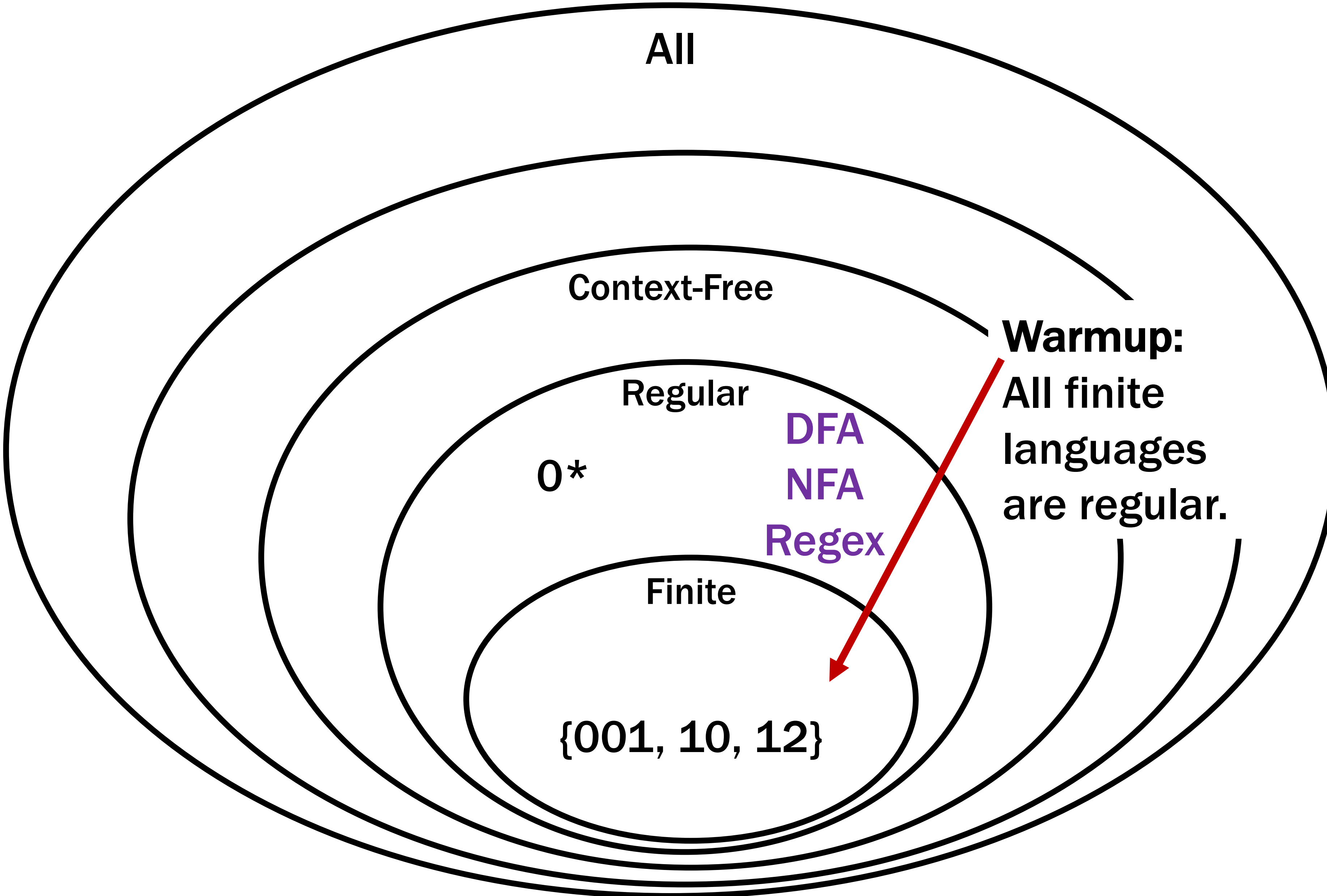
What languages have DFAs? CFGs?

All of them?

Languages and Representations!



Languages and Representations!



Warmup:
All finite languages are regular.

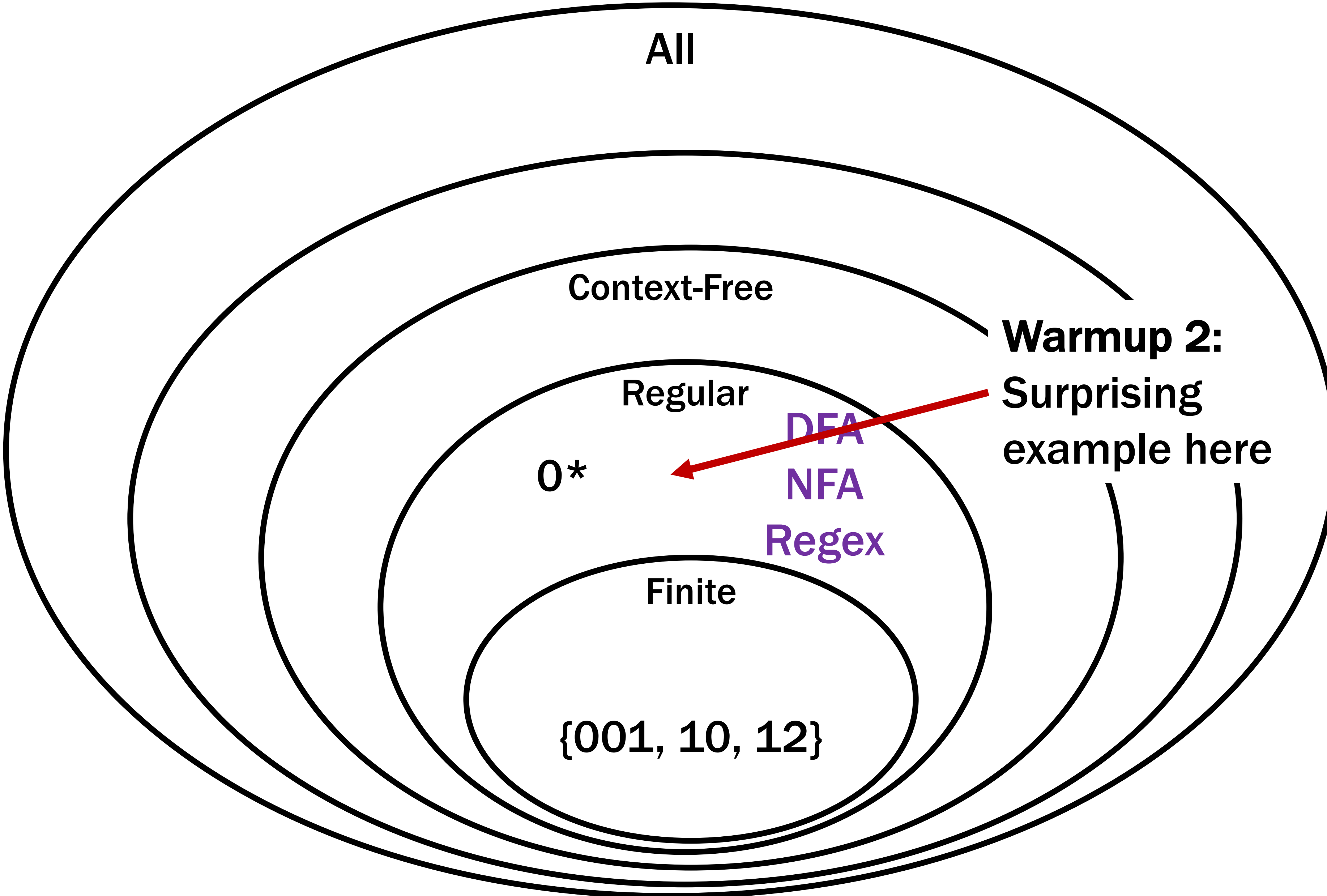
DFAs Recognize Any Finite Language

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!



Warmup 2: Surprising example here

An Interesting Infinite Regular Language

L = {x$\in$ {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 (see **section**: strings with equal # of 0s and 1s)
- but seems hard for DFAs!

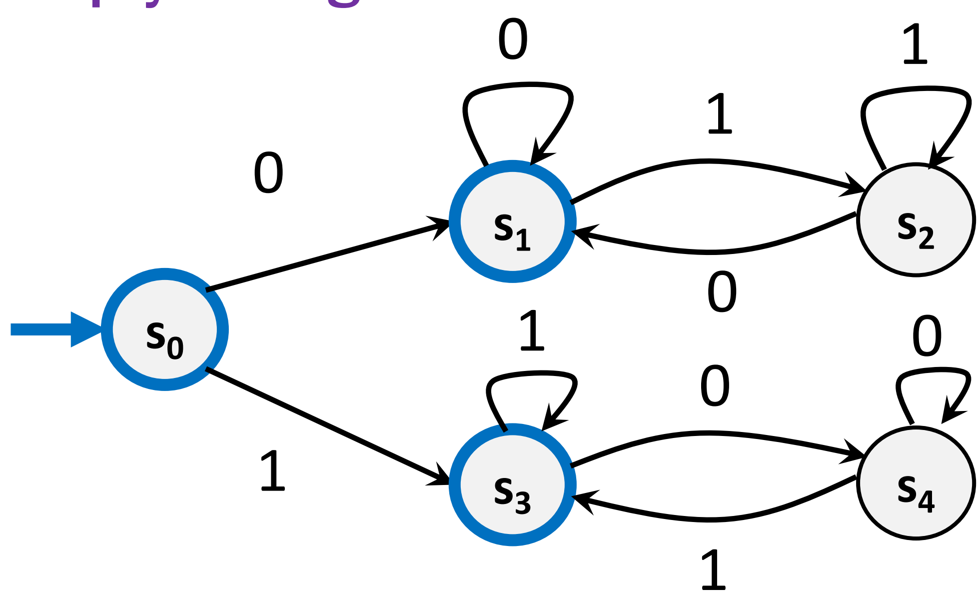
An Interesting Infinite Regular Language

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

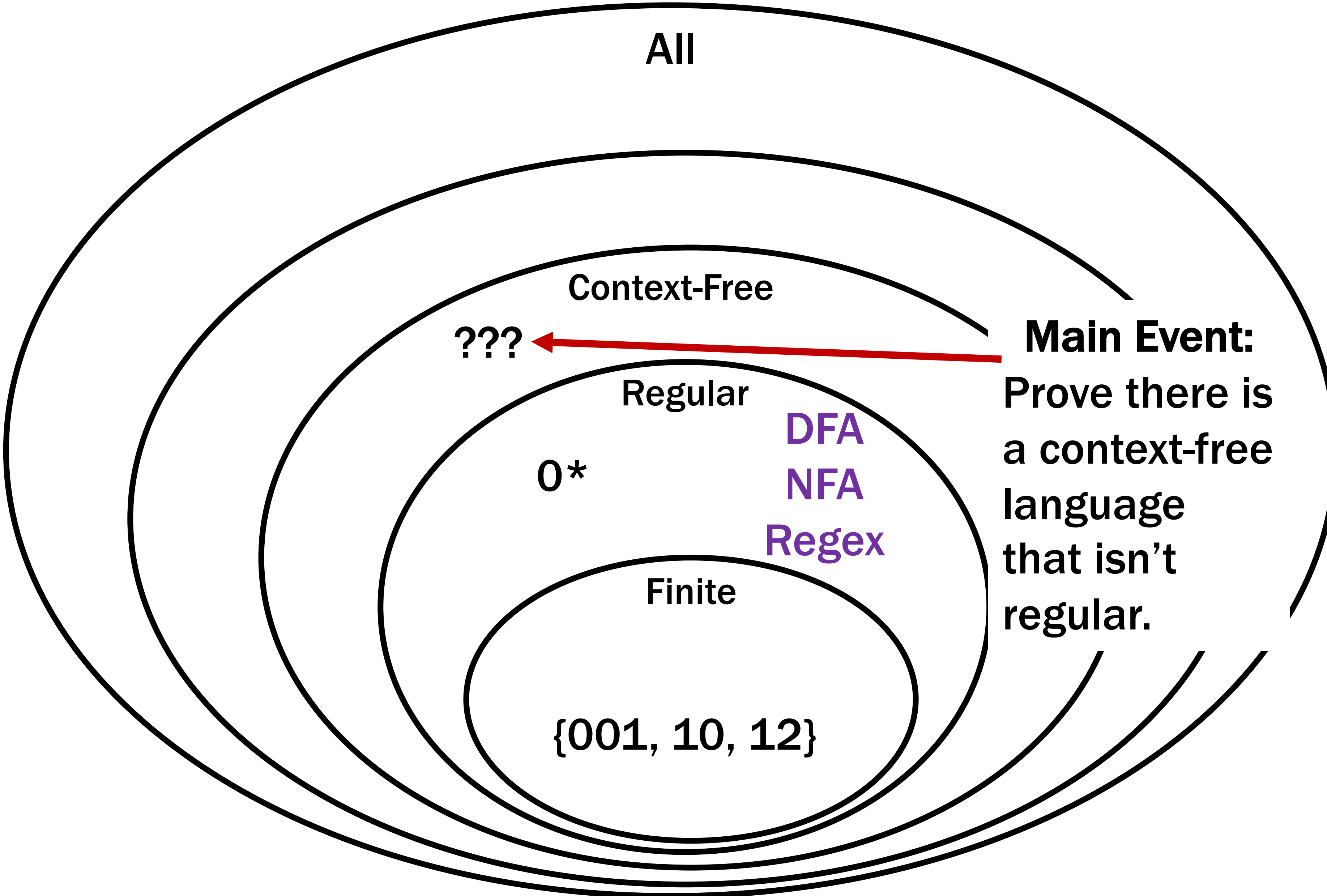
L is infinite.

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!



Main Event: Prove there is a context-free language that isn’t regular.