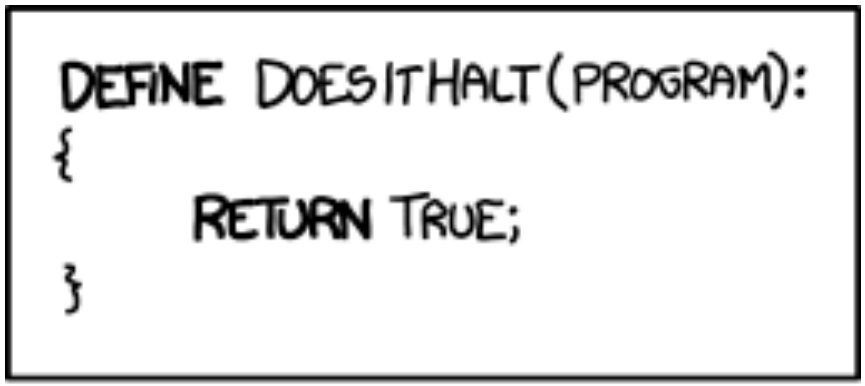
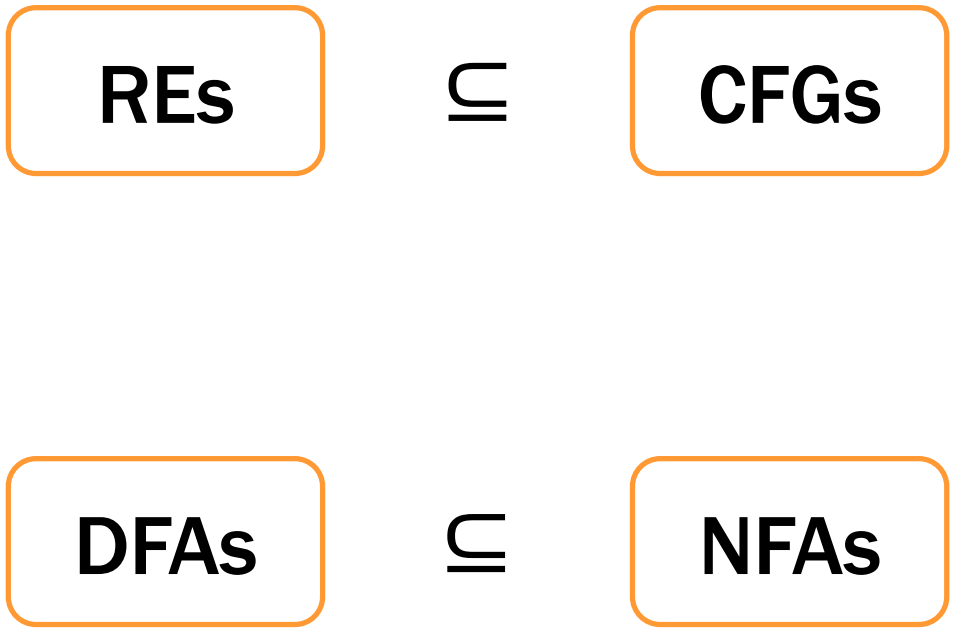
CSE 311: Foundations of Computing

Topic 6: Computability



THE BIG PICTURE SOLUTION
TO THE HALTING PROBLEM

The story so far...



NFAs and regular expressions

Theorem: For any regular expression, there is an NFA that recognizes the same language

Proof idea: Structural induction based on the recursive definition of regular expressions…

Regular Expressions over Σ

- **Basis:**
  - ε is a regular expression
  - ***a*** is a regular expression for any $a \in \Sigma$

- **Recursive step:**
  - If **A** and **B** are regular expressions, then so are:

    $A \cup B$

    AB

    A*

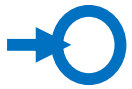
Base Case

- Case ε:

- Case *a*:

Base Case

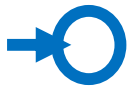
- Case ε:



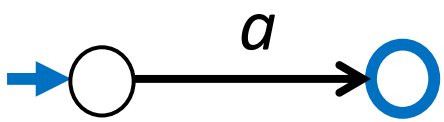
- Case *a*:

Base Case

- Case ε:



- Case ***a***:



Regular Expressions over Σ

- **Basis:**
  - ε is a regular expression
  - ***a*** is a regular expression for any $a \in \Sigma$

- **Recursive step:**
  - If **A** and **B** are regular expressions, then so are:

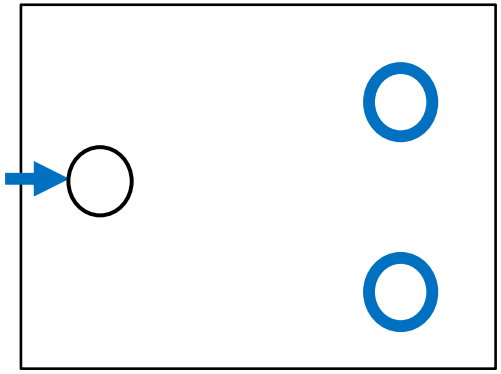
    $A \cup B$

    AB

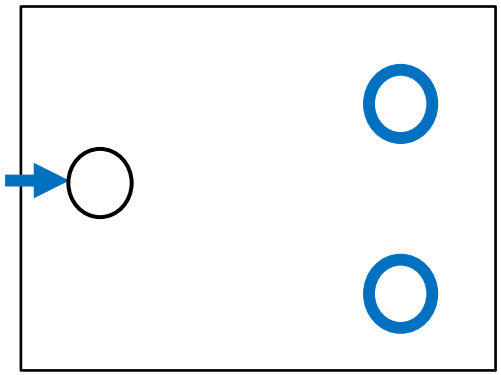
    A*

Inductive Hypothesis

- Suppose that for some regular expressions A and B there exist NFAs N_A and N_B such that N_A recognizes the language given by A and N_B recognizes the language given by B



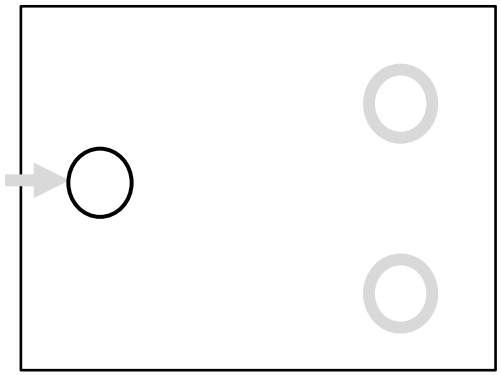
N_A



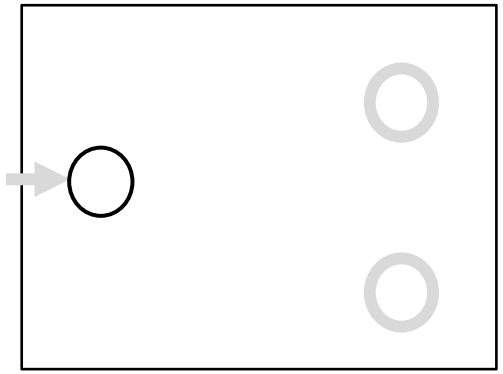
N_B

Inductive Step

Case $A \cup B$:



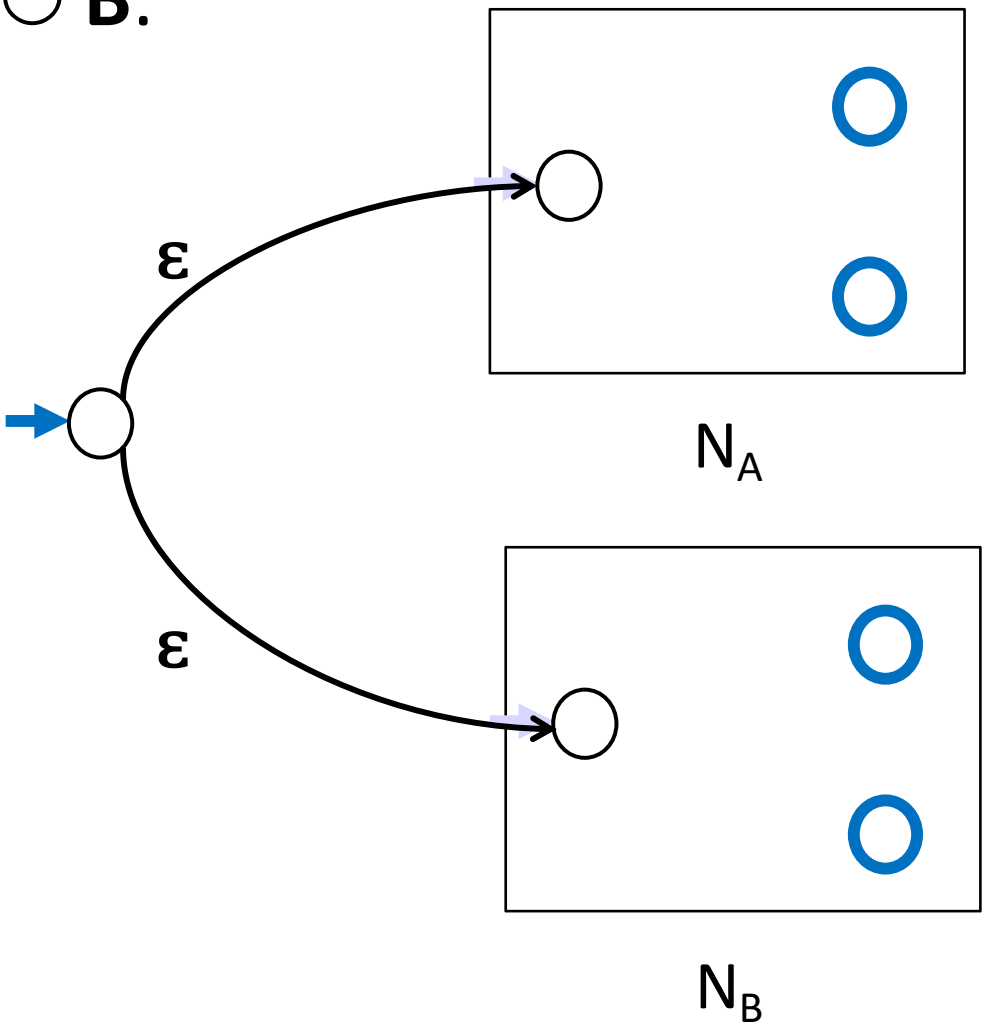
N_A



N_B

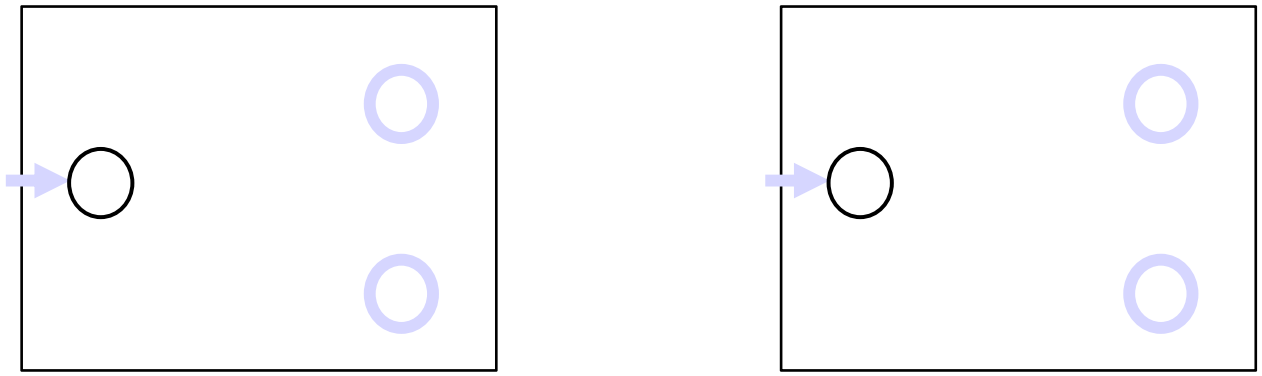
Inductive Step

Case $A \cup B$:



Inductive Step

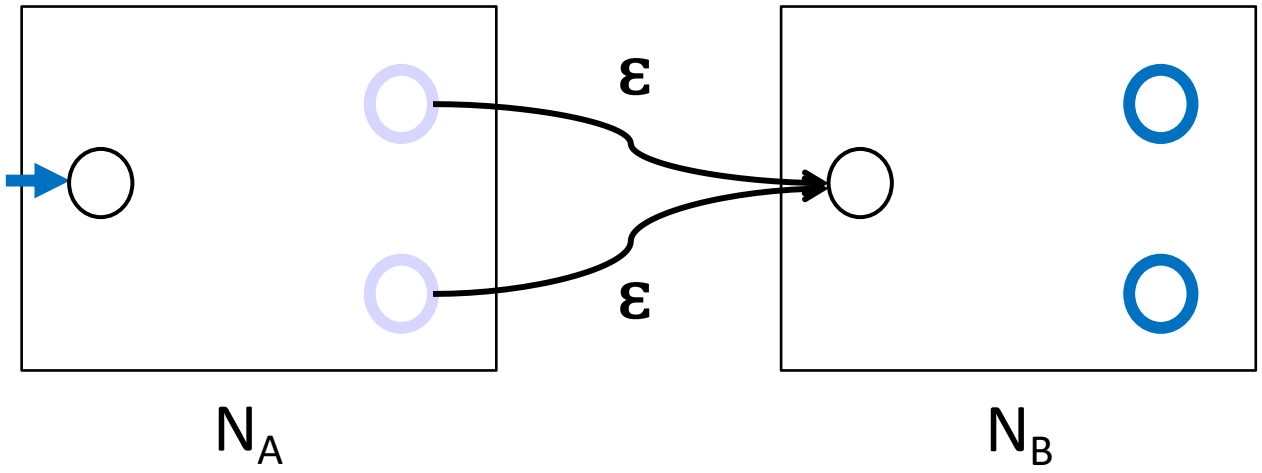
Case AB:



N_A N_B

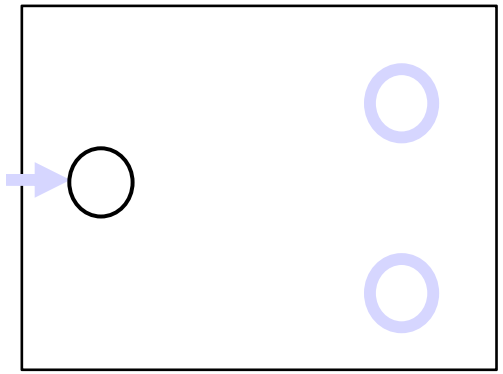
Inductive Step

Case AB:



Inductive Step

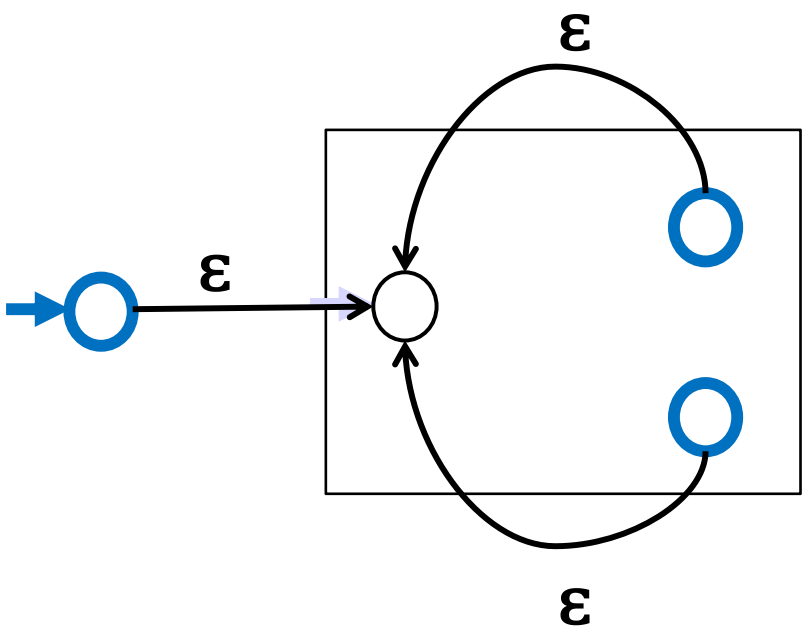
Case A*



N_A

Inductive Step

Case A*

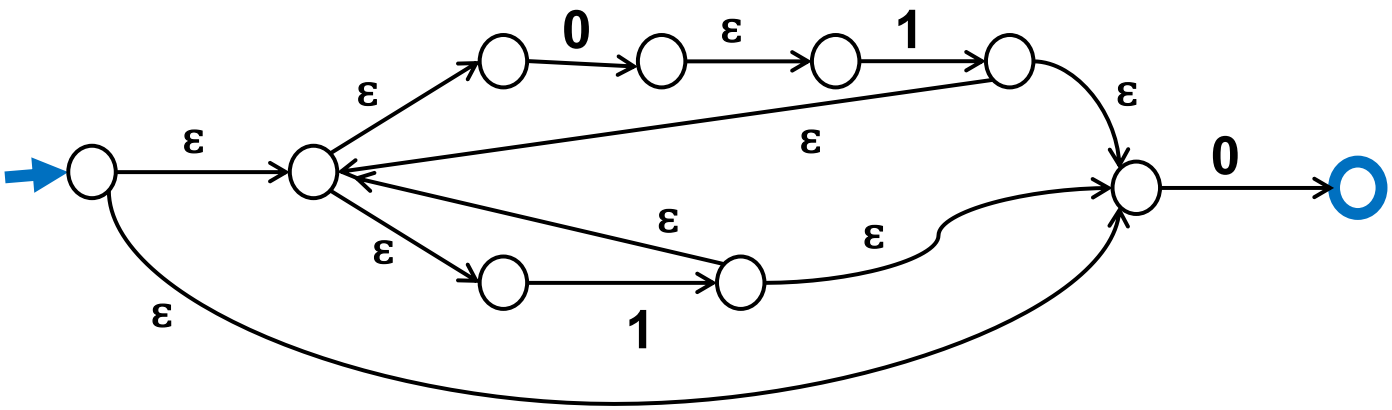


N_A

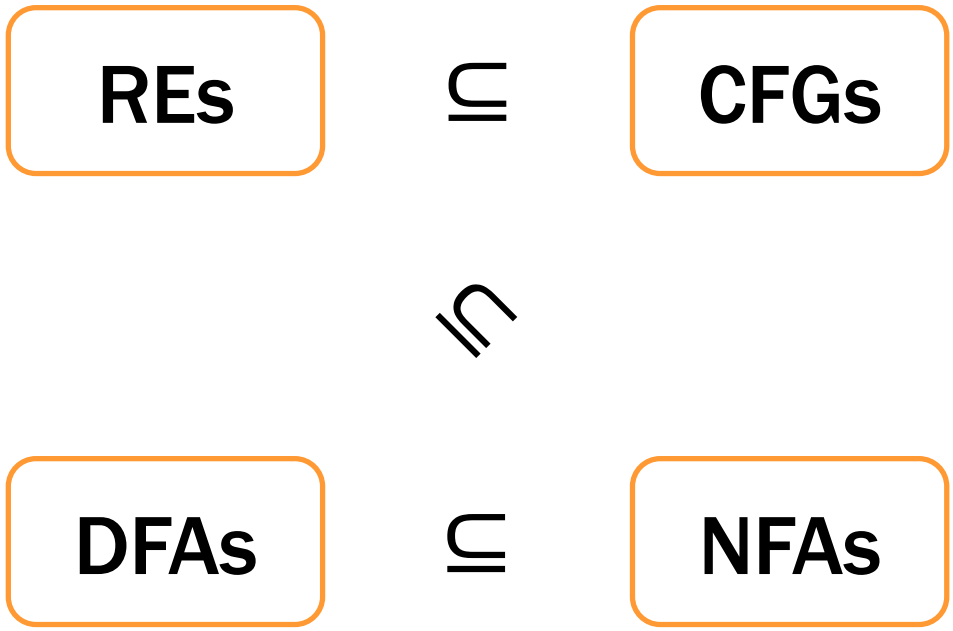
Build an NFA for $(01 \cup 1)^*0$

Solution

(01 ∪1)*0



The story so far...



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

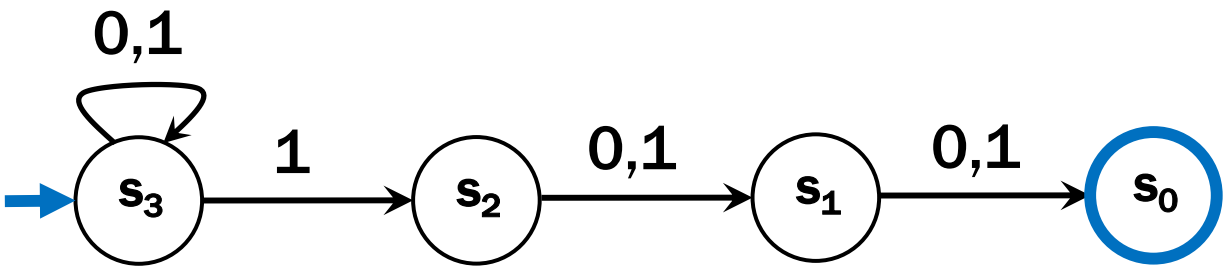
Three ways of thinking about NFAs

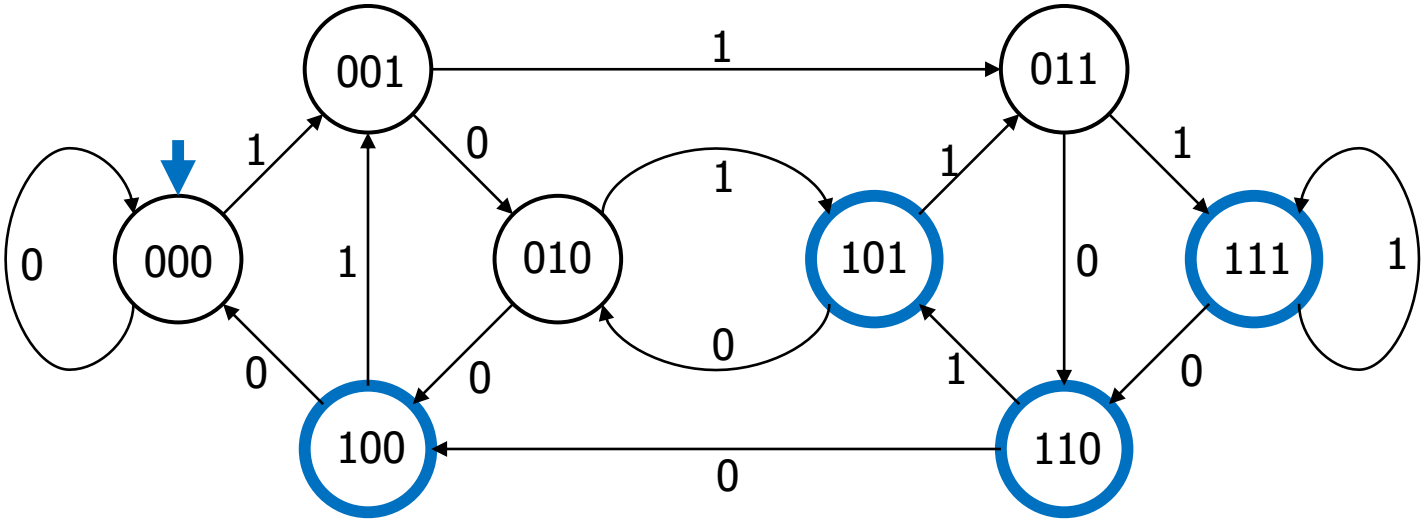
- Perfect guesser: The NFA has input x and whenever there is a choice of what to do it magically guesses a good one (if one exists)

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

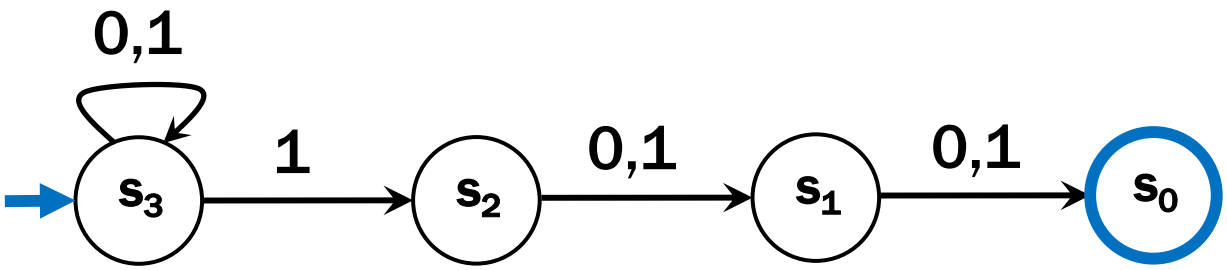
- Parallel exploration: The NFA computation runs all possible computations on x step-by-step at the same time in parallel

Recall: Compare with the smallest DFA

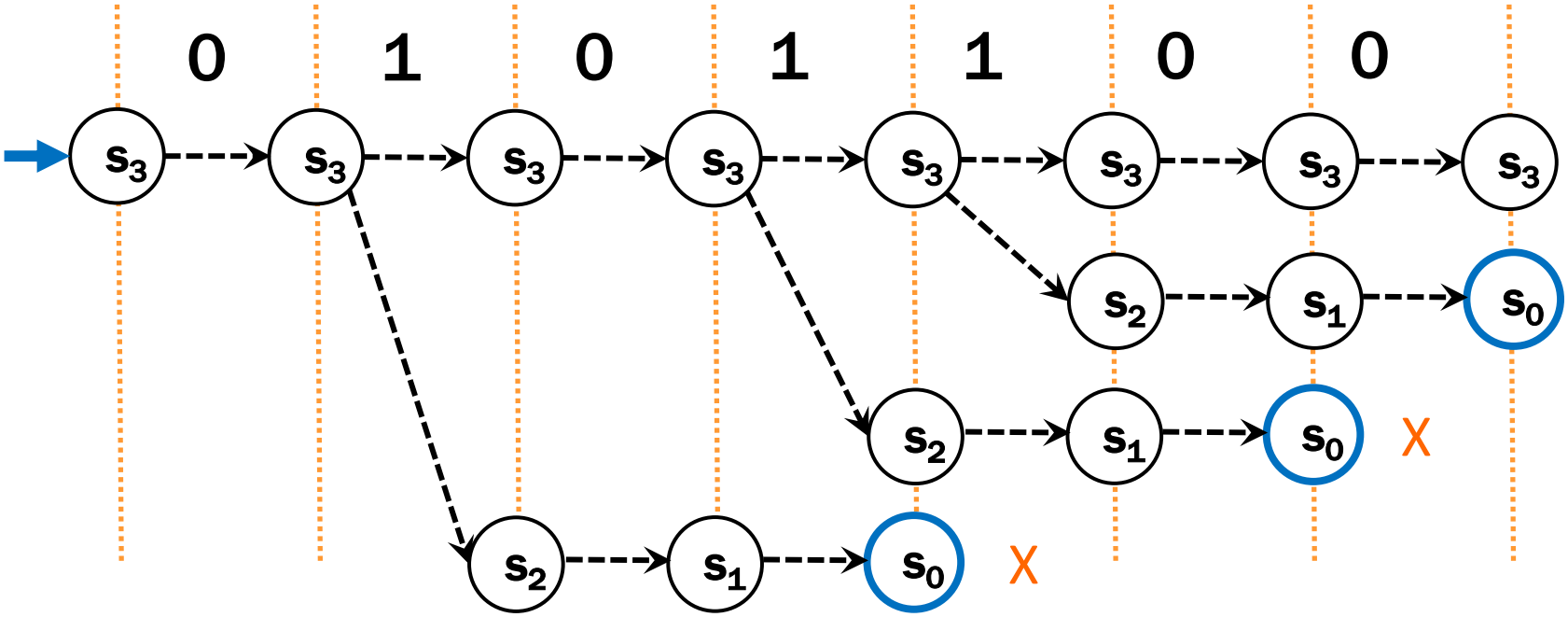




Parallel Exploration view of an NFA



Input string 0101100



Conversion of NFAs to a DFAs

- **Construction Idea:**
  - **The DFA keeps track of ALL states reachable in the NFA along a path labeled by the input so far**

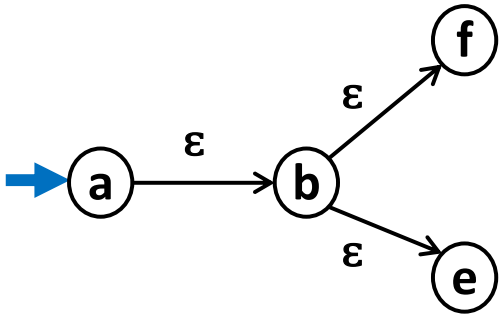
    (Note: not all *paths*; all *last states* on those paths.)

  - **There will be one state in the DFA for each *subset* of states of the NFA that can be reached by some string**

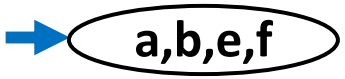
Conversion of NFAs to a DFAs

New start state for DFA

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



NFA

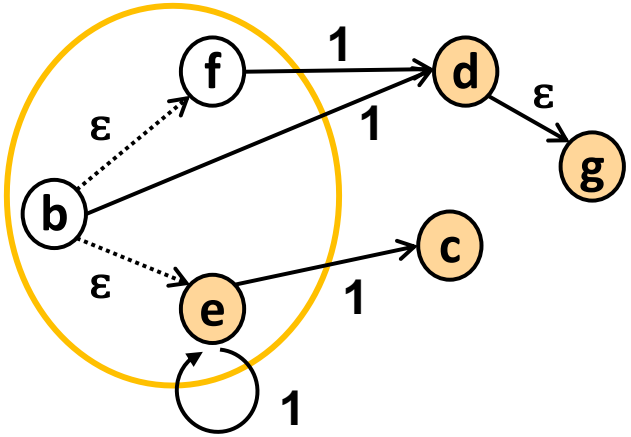


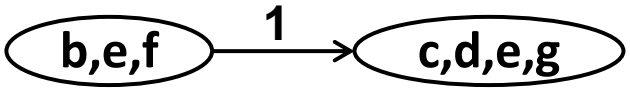
DFA

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**

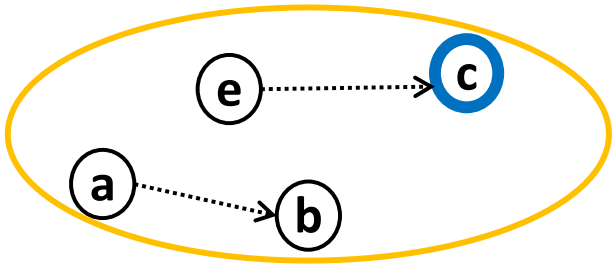




Conversion of NFAs to a DFAs

Final states for the DFA

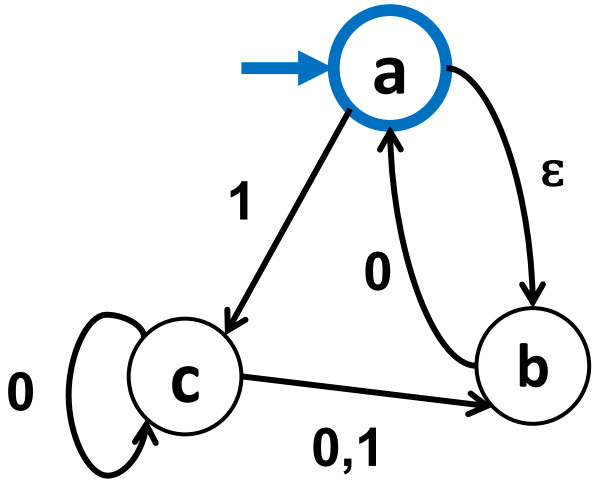
– All states whose set contain some final state of the NFA



NFA

DFA

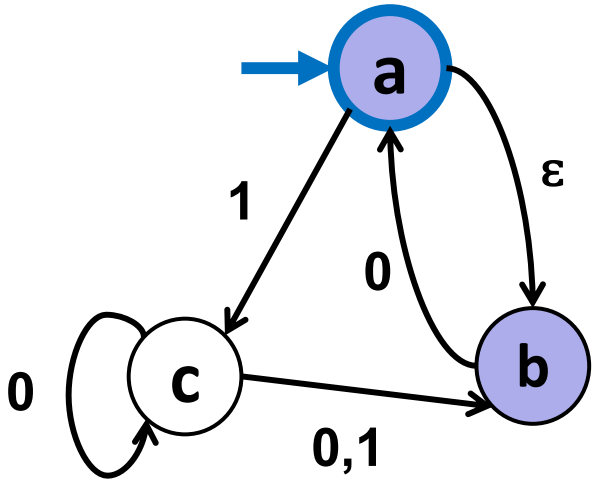
Example: NFA to DFA



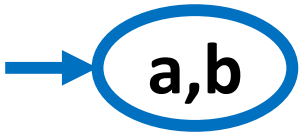
NFA

DFA

Example: NFA to DFA

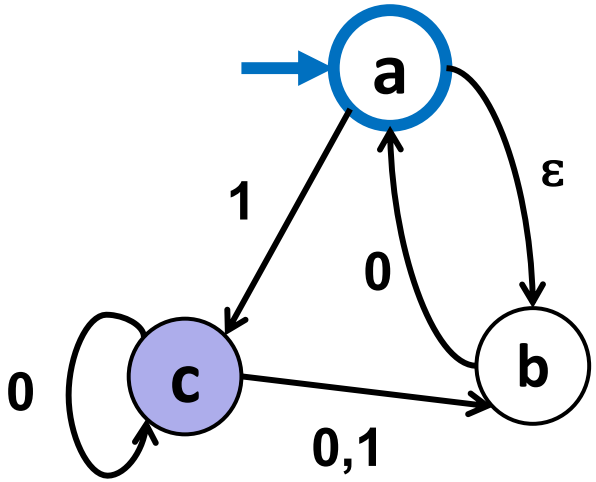


NFA

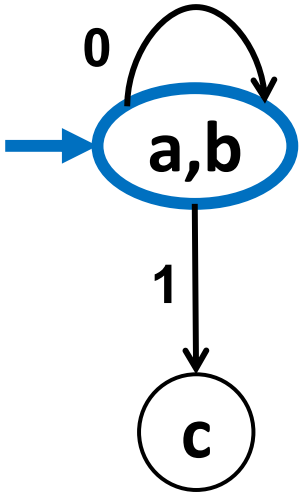


DFA

Example: NFA to DFA

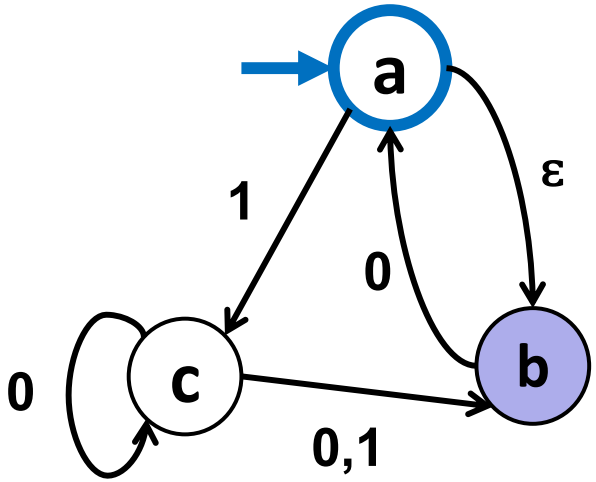


NFA

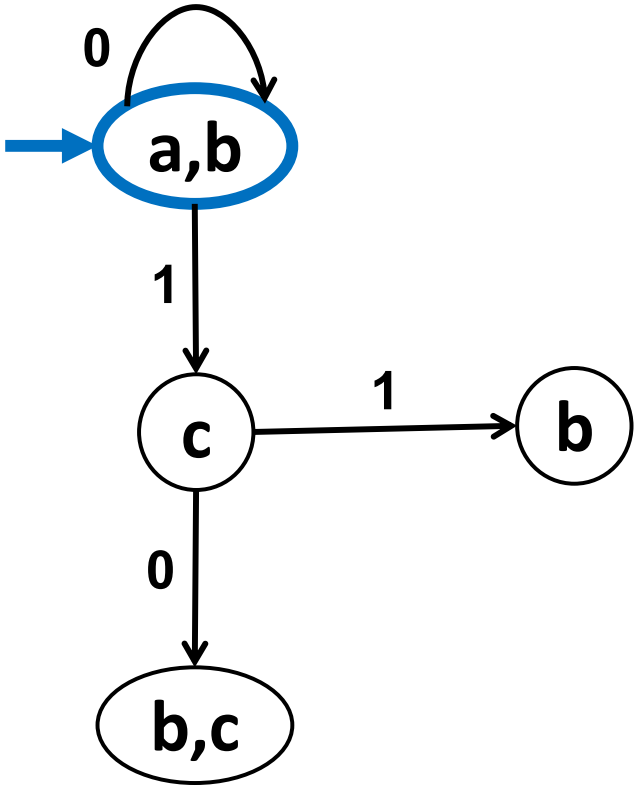


DFA

Example: NFA to DFA

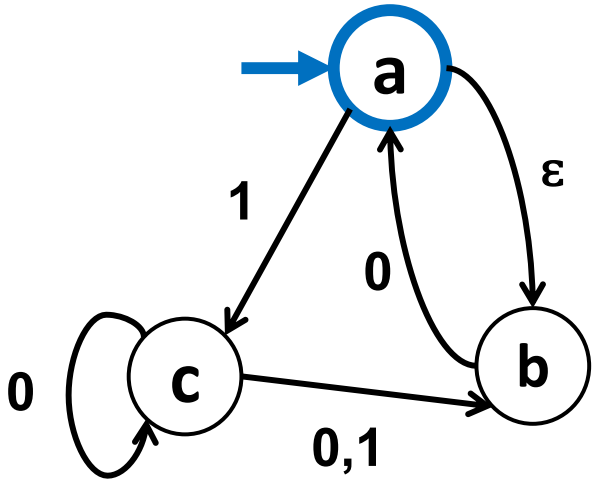


NFA

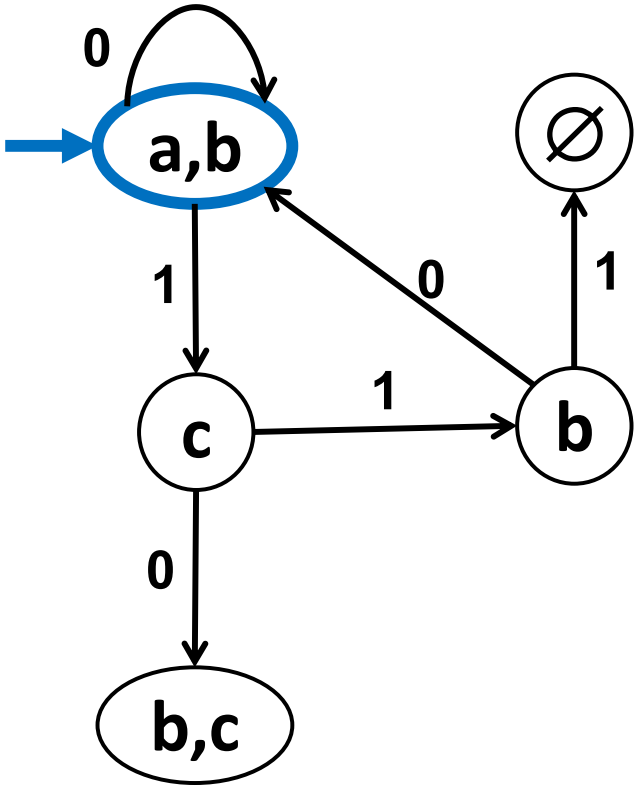


DFA

Example: NFA to DFA

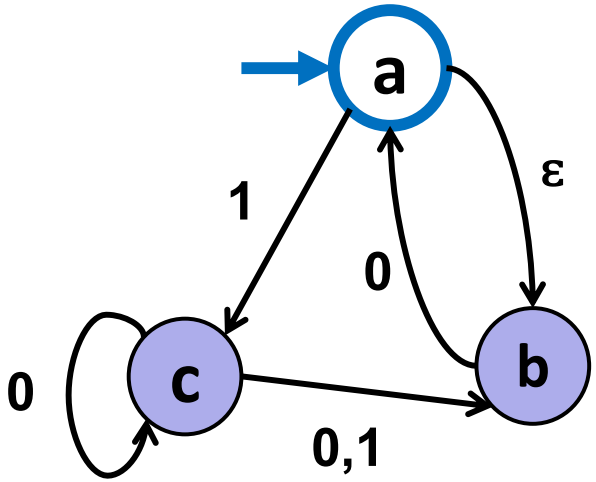


NFA

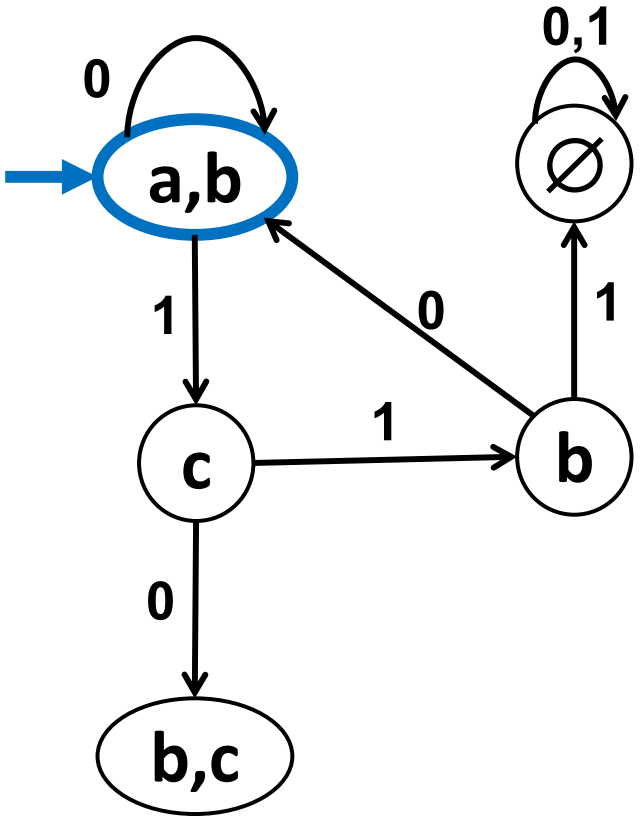


DFA

Example: NFA to DFA

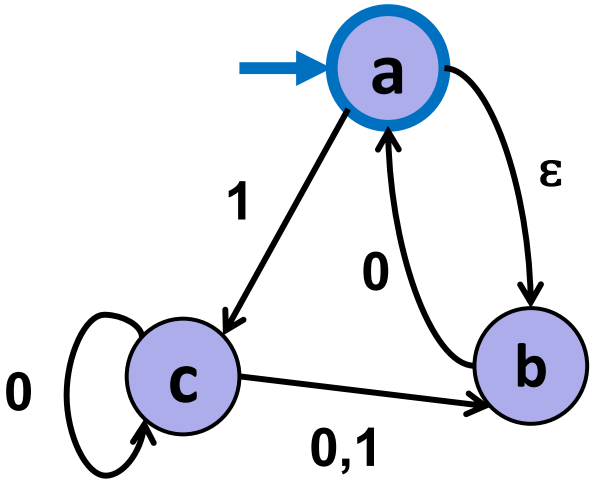


NFA

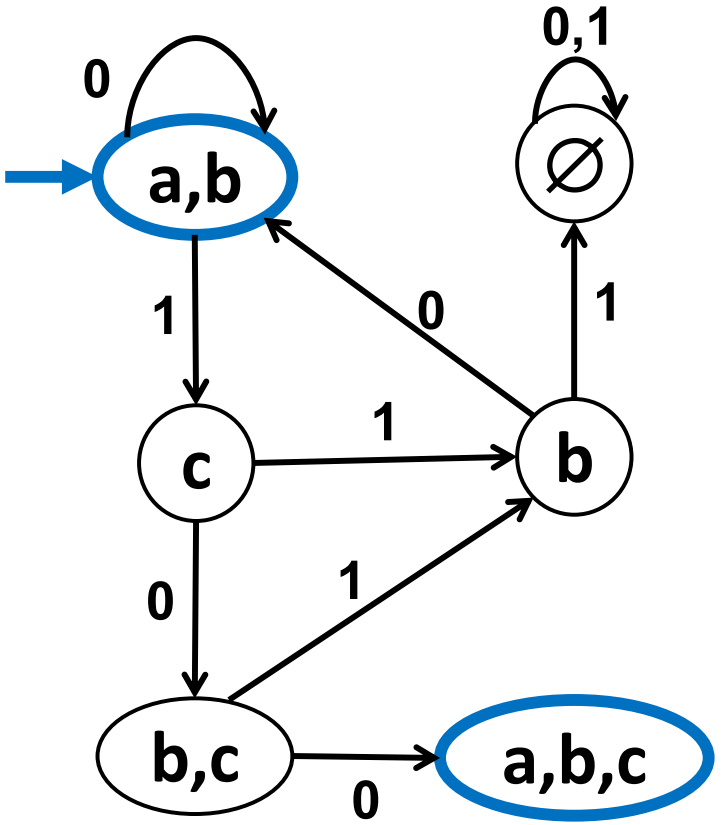


DFA

Example: NFA to DFA

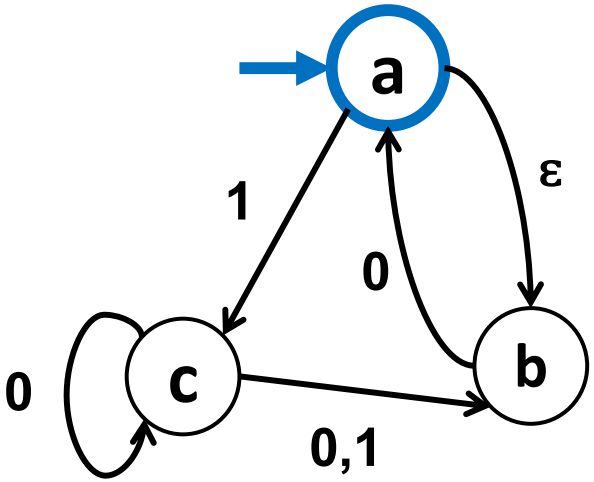


NFA

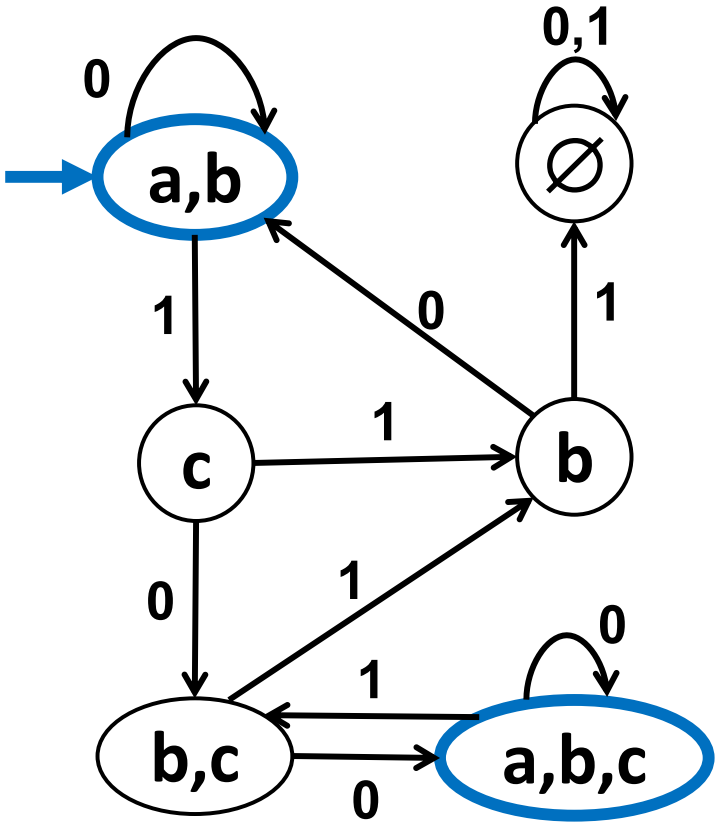


DFA

Example: NFA to DFA



NFA



DFA

Regular expressions, NFAs, & DFAs

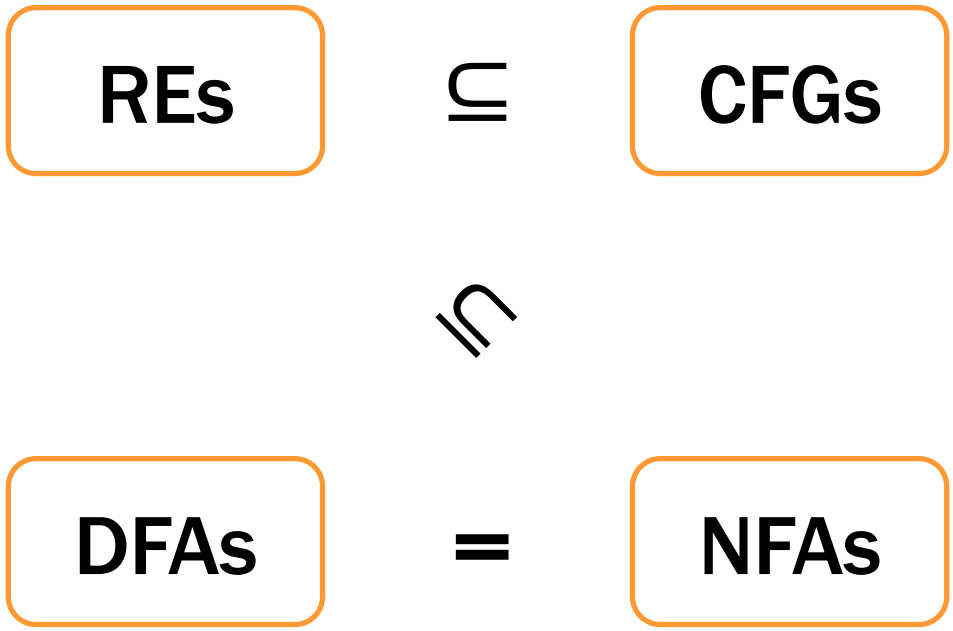
We have shown how to build a DFA for every RE

- Build NFA
- Convert NFA to DFA using subset construction
- (Later: minimize resulting DFA)

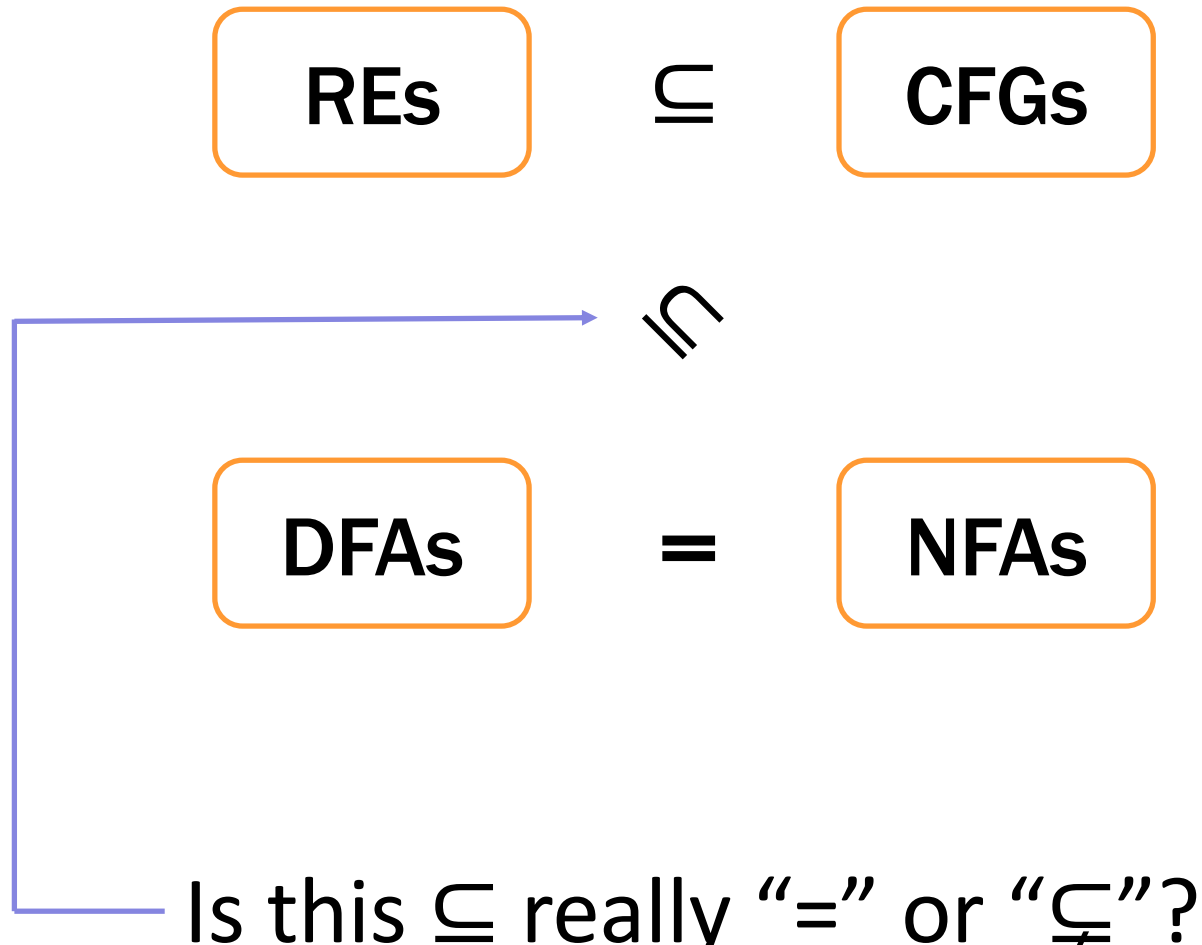
Thus, we could now implement a RegExp library

- most RegExp libraries actually simulate the NFA
- (even better: one can combine the two approaches: apply DFA minimization lazily while simulating the NFA)

The story so far...



The story so far...



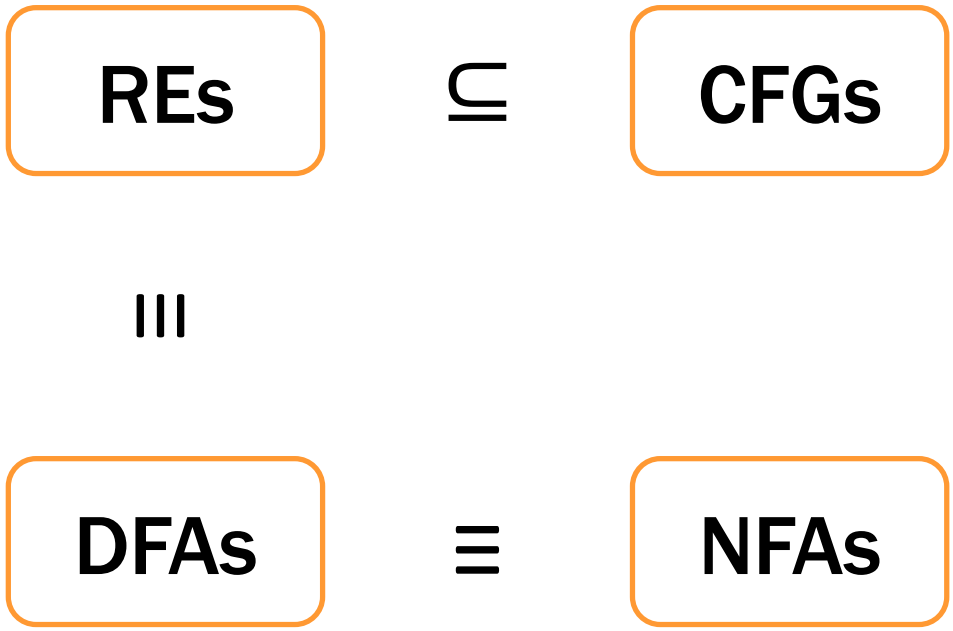
Regular expressions ≡ NFAs ≡ DFAs

Theorem: For any NFA, there is a regular expression that accepts 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 story so far...



Languages represented by DFA, NFAs, or regular expressions are called **Regular Languages**

Example Corollary of These Results

Corollary: If $\mathbf{A}$ is the language of a regular expression, then $\overline{\mathbf{A}}$ is the language of a regular expression*.

(This is the complement with respect to the universe of all strings over the alphabet, i.e., $\overline{\mathbf{A}} = \boldsymbol{\Sigma}^* \setminus \mathbf{A}$.)

Recall: Algorithms for Regular Languages

We have algorithms for

- RE to NFA
- NFA to DFA
- DFA/NFA to RE (not shown)
- DFA minimization (next...)

State Minimization

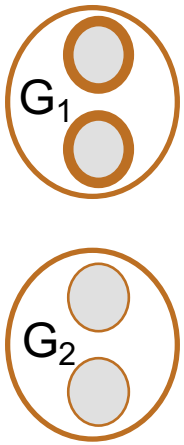
- Many FSMs (DFAs) for the same problem
- Take a given FSM and try to reduce its state set by combining states
  - Algorithm will always produce the unique minimal equivalent machine (up to renaming of states) but we won't prove this

State Minimization Algorithm

- Put states into groups
- Try to find groups that can be collapsed into one state
  - states can keep track of information that isn't necessary to determine whether to accept or reject
- Group states together until we can *prove* that collapsing them can change the accept/reject result

State Minimization Algorithm

1. Put states into groups based on their outputs (whether they accept or reject)

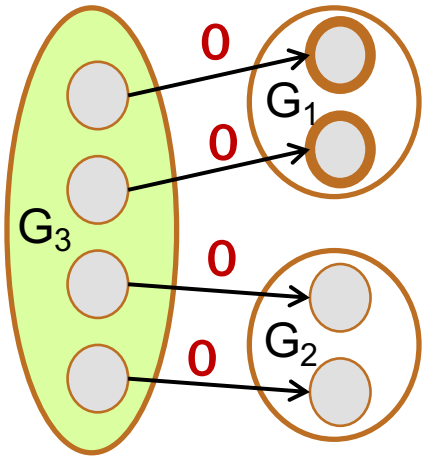


If these were one state,
it would not behave the same
at the end of the input

Must separate G_1 from G_2 because
G_1 is accepting and G_2 is rejecting

State Minimization Algorithm

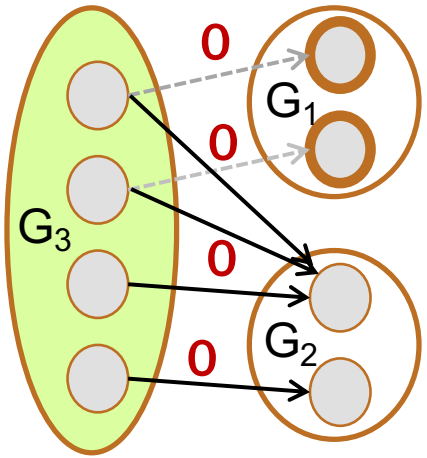
1. Put states into groups based on their outputs (whether they accept or reject)



Must separate top of G_3 from bottom of G_3 because they behave differently on 0

State Minimization Algorithm

1. Put states into groups based on their outputs (whether they accept or reject)

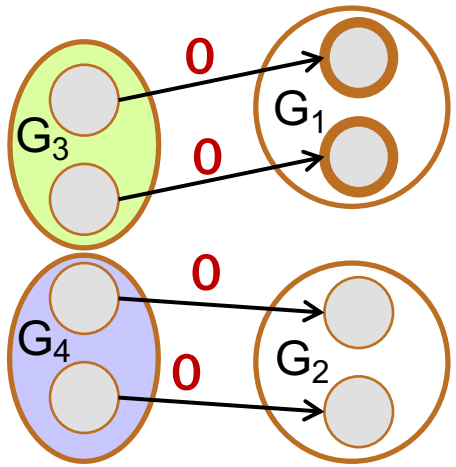


The top two states behave differently, so this is not the same machine

Same problem if we send all transitions to G_1

State Minimization Algorithm

1. Put states into groups based on their outputs (whether they accept or reject)

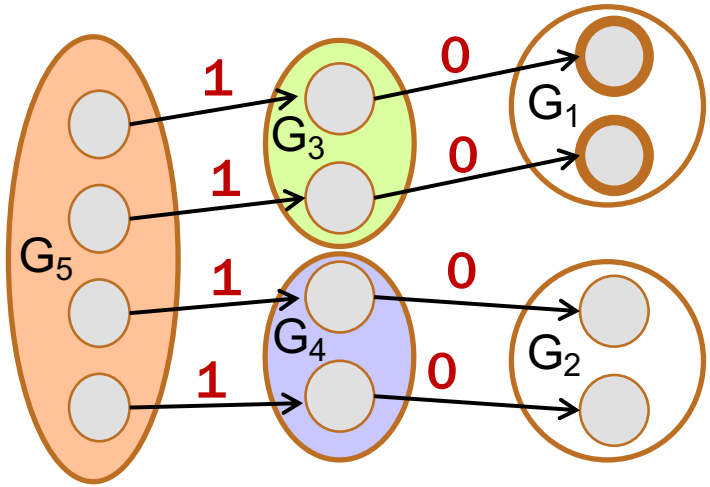


Can only collapse a group into a single state if all states in it go to the same place (group) on every character of the alphabet

State Minimization Algorithm

1. Put states into groups based on their outputs (whether they accept or reject)

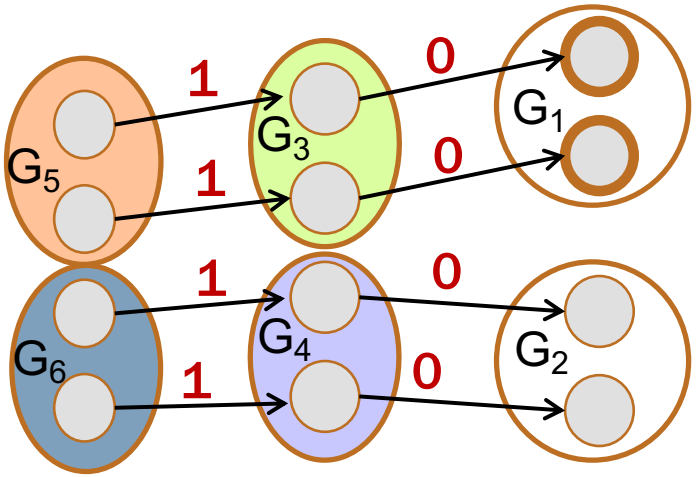
For later: need to know G_5 vs G_6 if next two characters are "10"



Splitting one group can cause another to need splitting

State Minimization Algorithm

1. Put states into groups based on their outputs (whether they accept or reject)

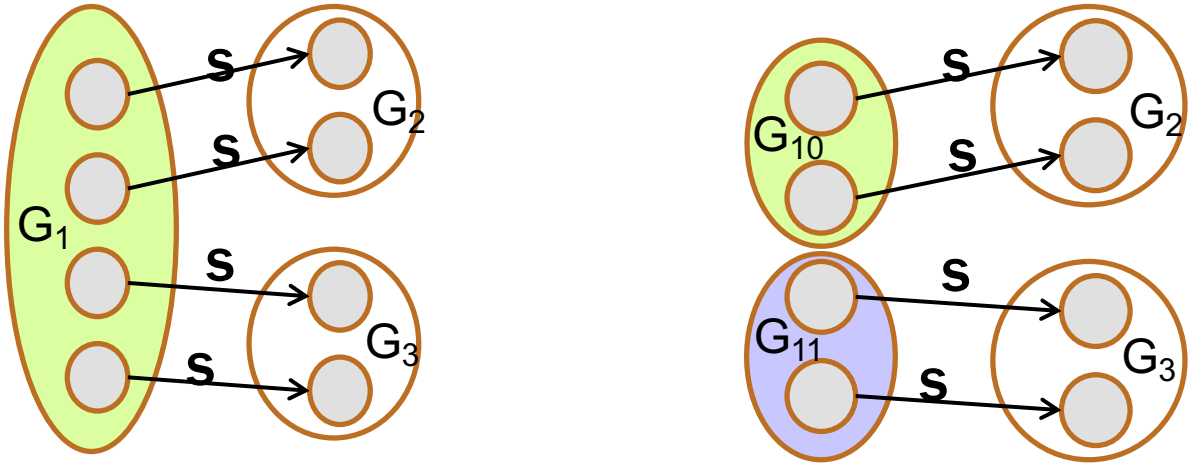


For later: need to know G_5 vs G_6 if next two characters are "10"

Must separate G_4 from G_6 because they behave differently on 1

State Minimization Algorithm

1. Put states into groups based on their outputs (whether they accept or reject)
2. Repeat the following until no change happens
   a. If there is a letter s so that not all states in a group G agree on which group s leads to, split G into smaller groups based on which group the states go to on s



3. Finally, convert groups to states

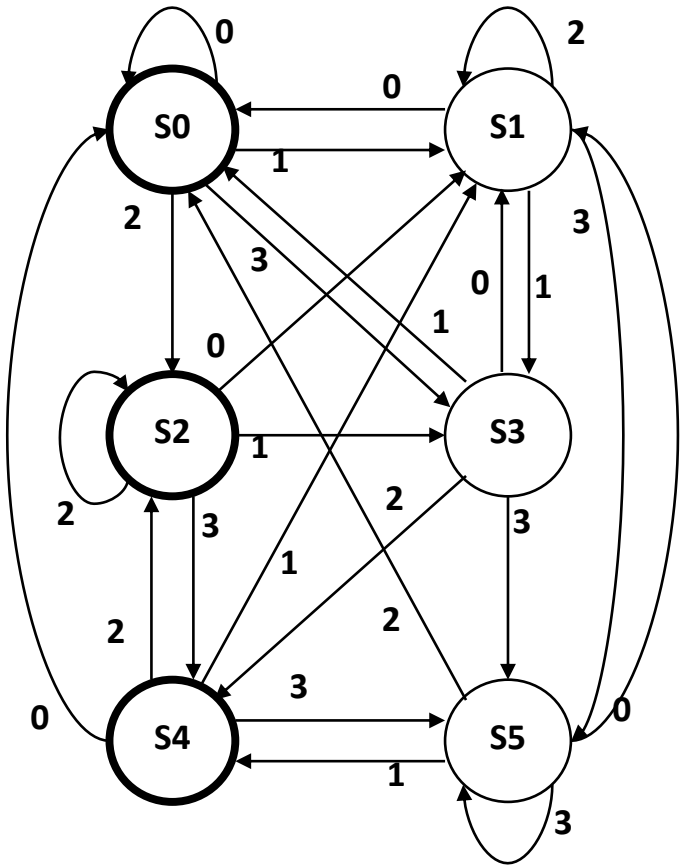
State Minimization Algorithm

- Put states into groups
- Try to find groups that can be collapsed into one state
  - states can keep track of information that isn't necessary to determine whether to accept or reject
- Group states together until we can *prove* that collapsing them can change the accept/reject result
  - find a specific string x such that:

    starting from state A, following edges according to x ends in **accept**

    starting from state B, following edges according to x ends in **reject**
  - algorithm could be modified to calculate these strings

State Minimization Example

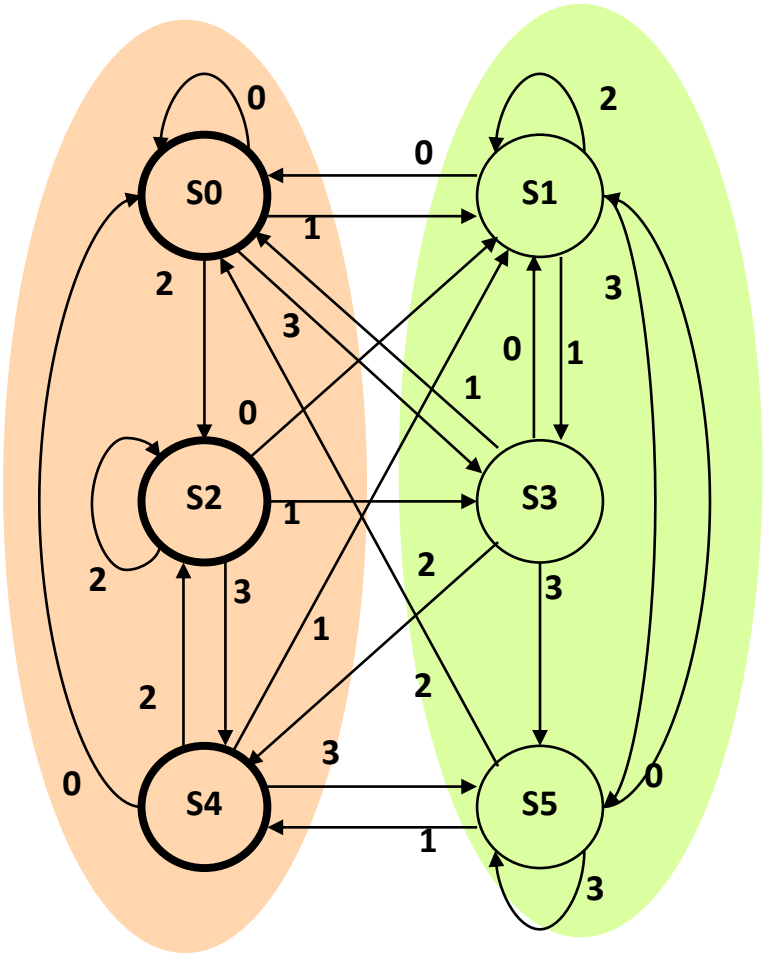


present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

state transition table

Put states into groups based on their outputs (or whether they accept or reject)

State Minimization Example

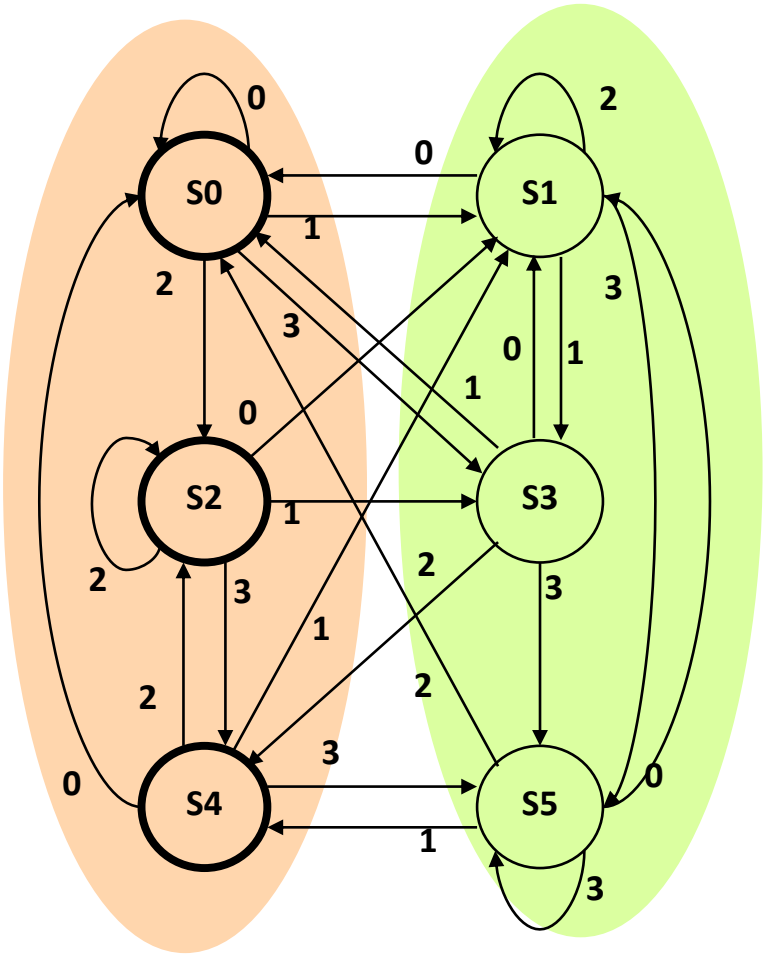


present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

state transition table

Put states into groups based on their outputs (or whether they accept or reject)

State Minimization Example



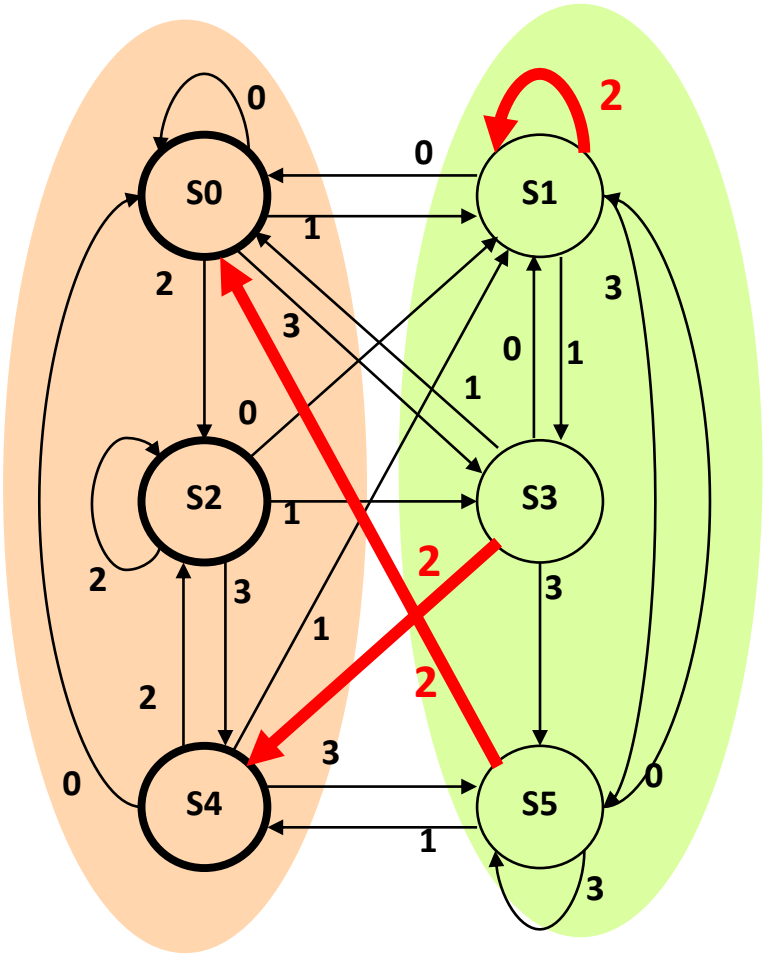
present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

state transition table

Put states into groups based on their outputs (or whether they accept or reject)

If there is a symbol **s** so that not all states in a group **G** agree on which group **s** leads to, split **G** based on which group the states go to on **s**

State Minimization Example



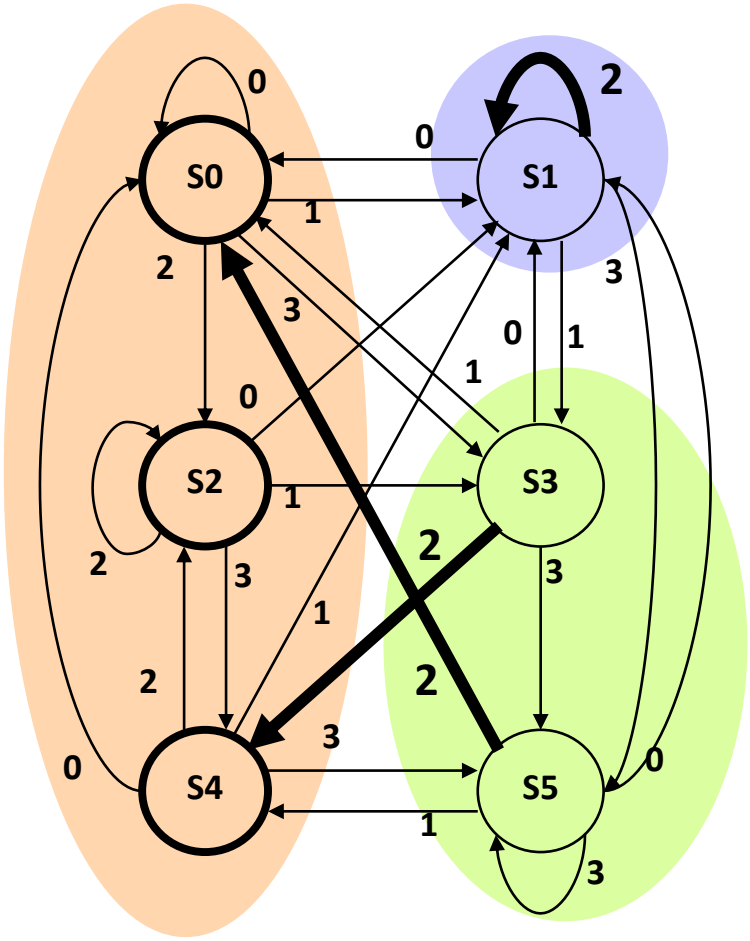
present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

state transition table

Put states into groups based on their outputs (or whether they accept or reject)

If there is a symbol **s** so that not all states in a group **G** agree on which group **s** leads to, split **G** based on which group the states go to on **s**

State Minimization Example



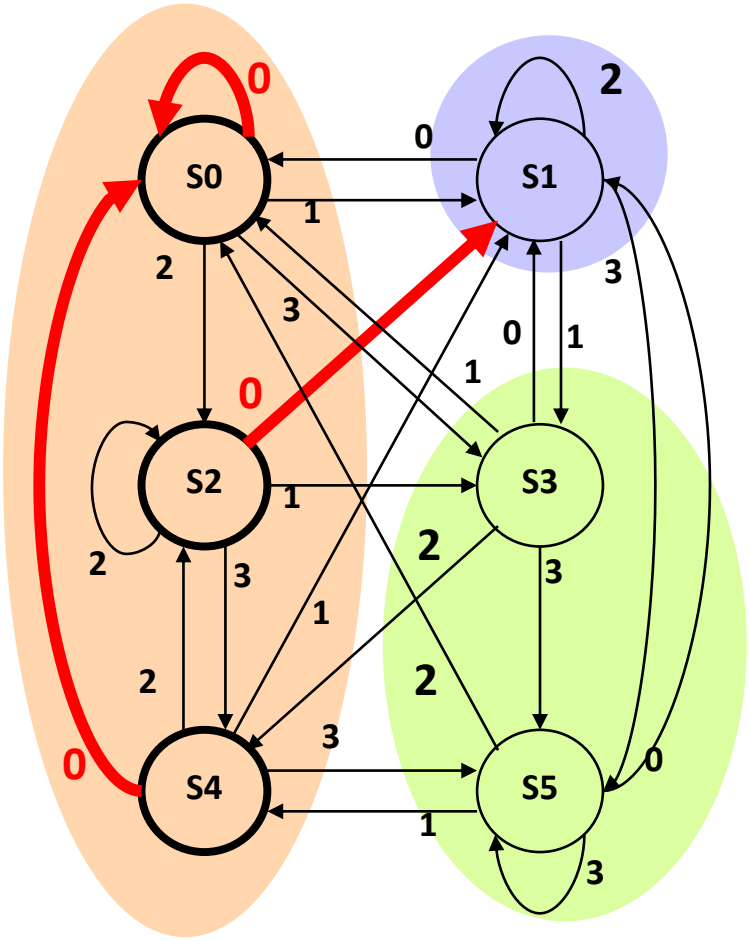
present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

state transition table

Put states into groups based on their outputs (or whether they accept or reject)

If there is a symbol **s** so that not all states in a group **G** agree on which group **s** leads to, split **G** based on which group the states go to on **s**

State Minimization Example



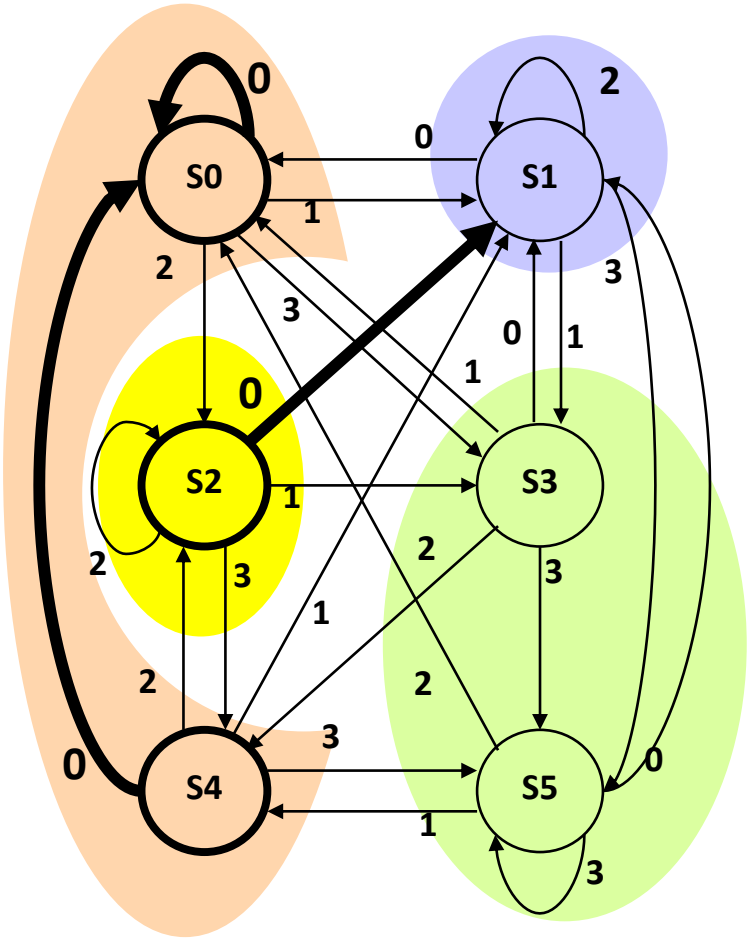
present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

state transition table

Put states into groups based on their outputs (or whether they accept or reject)

If there is a symbol **s** so that not all states in a group **G** agree on which group **s** leads to, split **G** based on which group the states go to on **s**

State Minimization Example



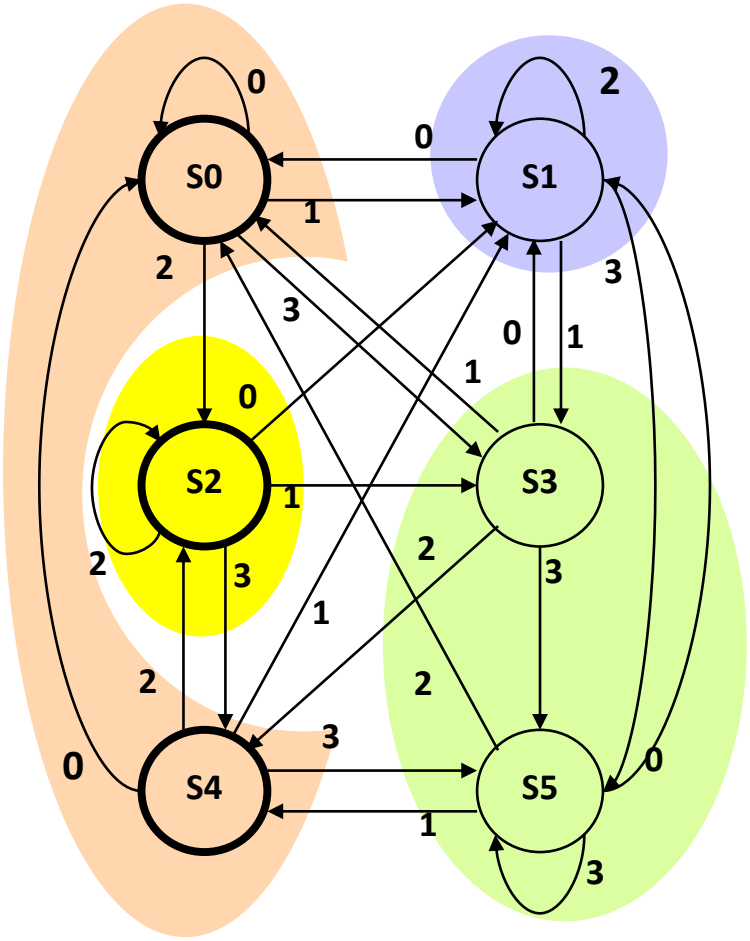
present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

state transition table

Put states into groups based on their outputs (or whether they accept or reject)

If there is a symbol **s** so that not all states in a group **G** agree on which group **s** leads to, split **G** based on which group the states go to on **s**

State Minimization Example



present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

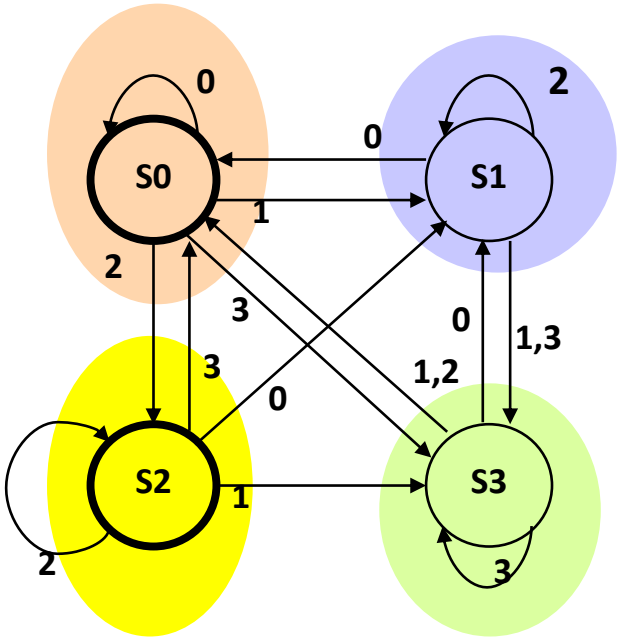
state transition table

Finally convert groups to states:

Can combine states S0-S4 and S3-S5.

In table replace all S4 with S0 and all S5 with S3

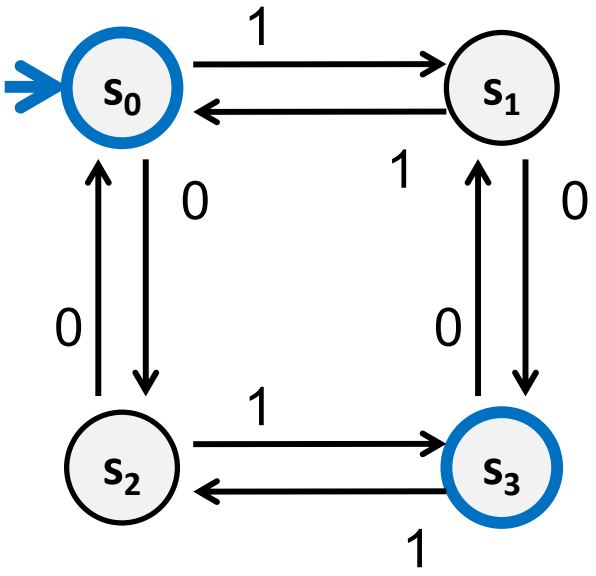
Minimized Machine



present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S3	0
S2	S1	S3	S2	S0	1
S3	S1	S0	S0	S3	0

state transition table

A Simpler Minimization Example



#0s is even

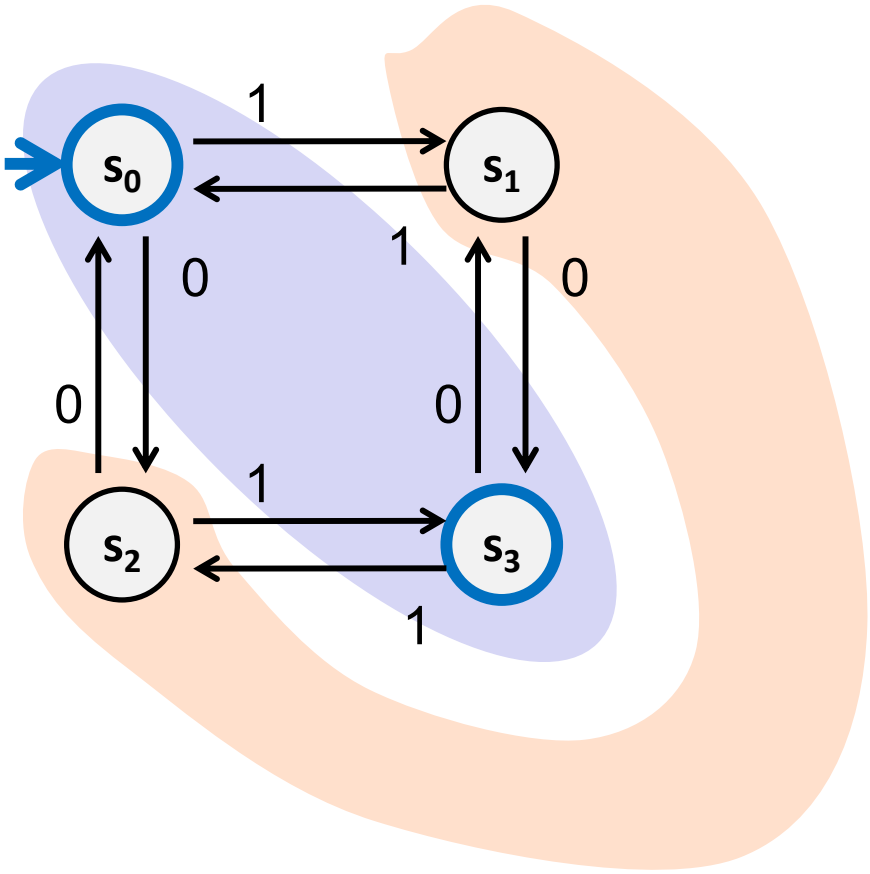
#0s is odd

#1s is even

#1s is odd

The set of all binary strings with # of 1's ≡ # of 0's (mod 2).

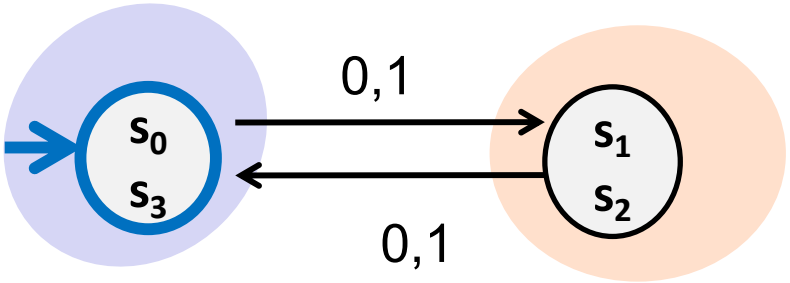
A Simpler Minimization Example



Split states into accept/reject groups

Every symbol causes the DFA to go from one group to the other so neither group needs to be split

Minimized DFA

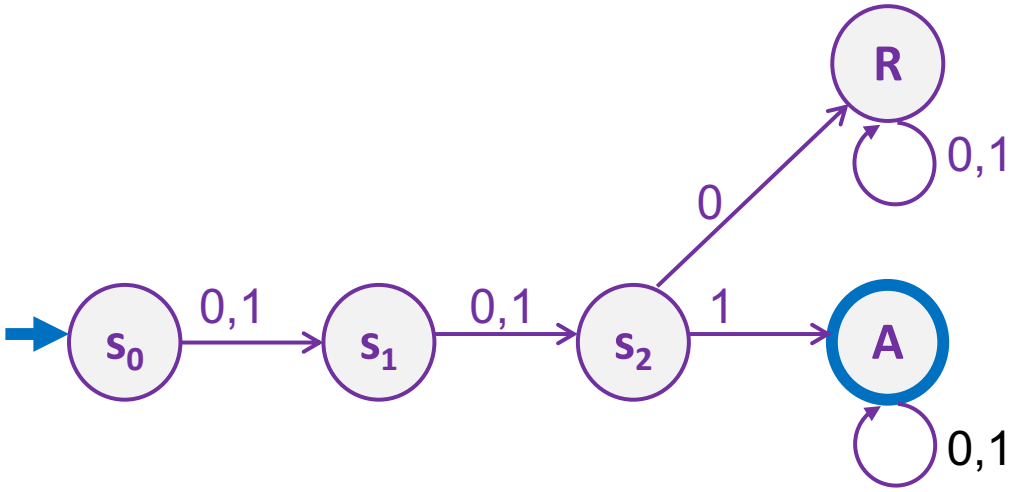


length is even length is odd

The set of all binary strings with # of 1's ≡ # of 0's (mod 2).

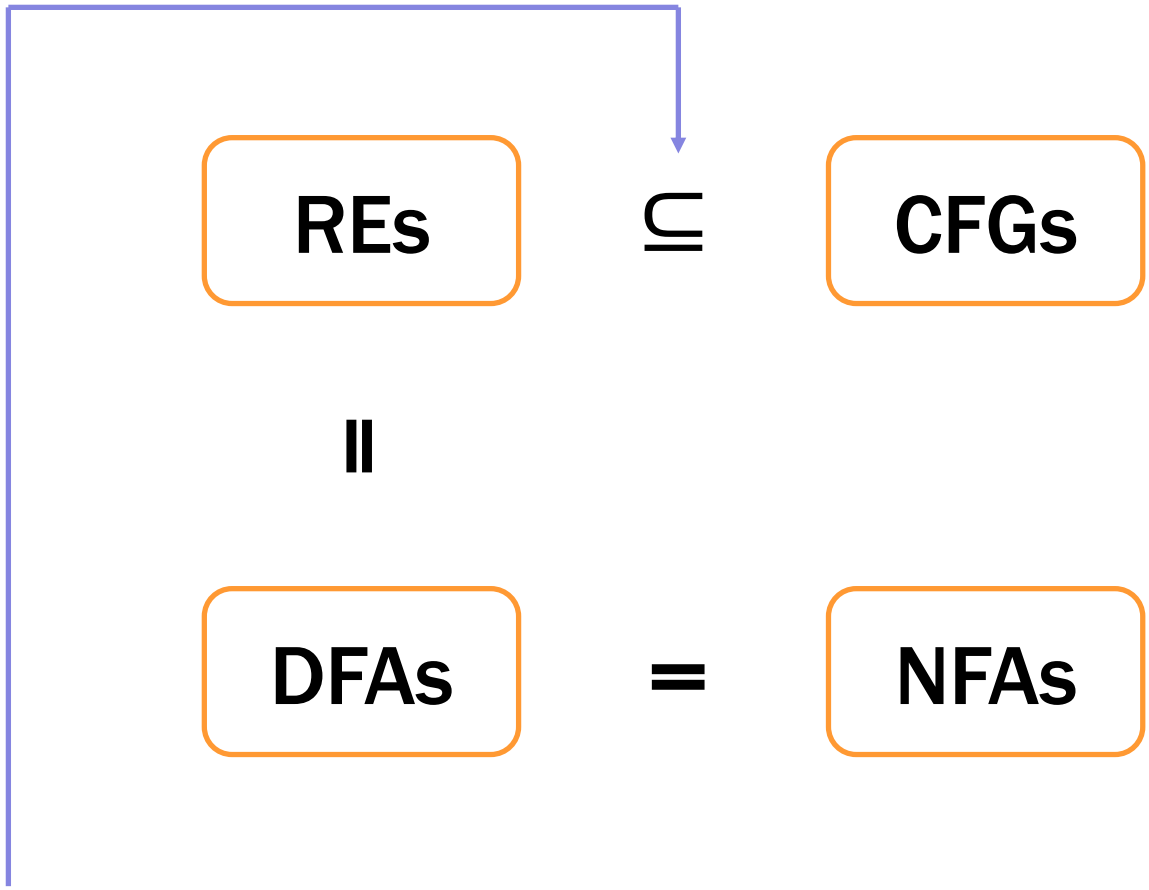
= The set of all binary strings with even length.

Recall: Binary strings with a 1 in the 3rd position from the start



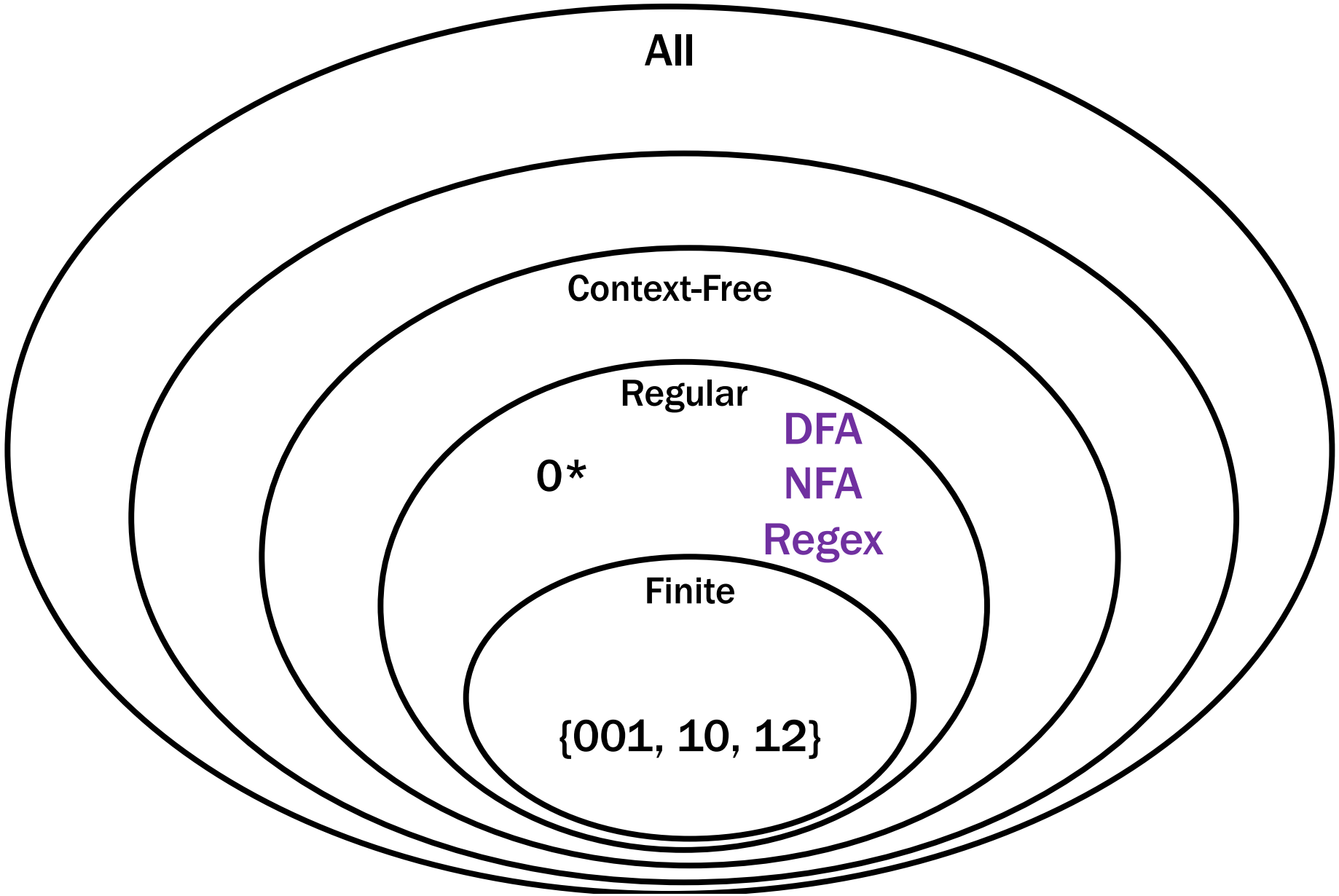
None of these states can be grouped!

The story so far...

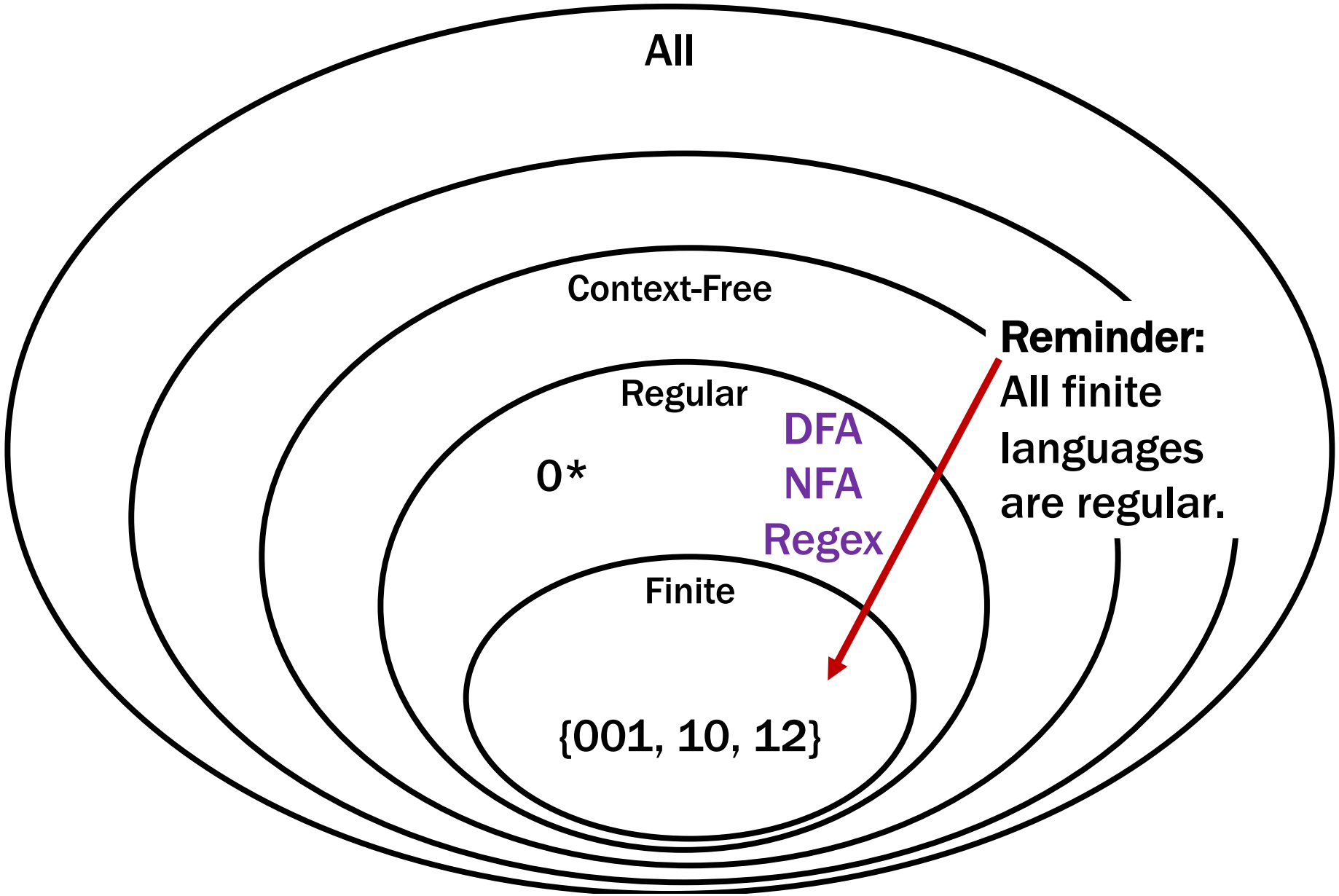


Now: Is this ⊆ really "=" or "⊊"?

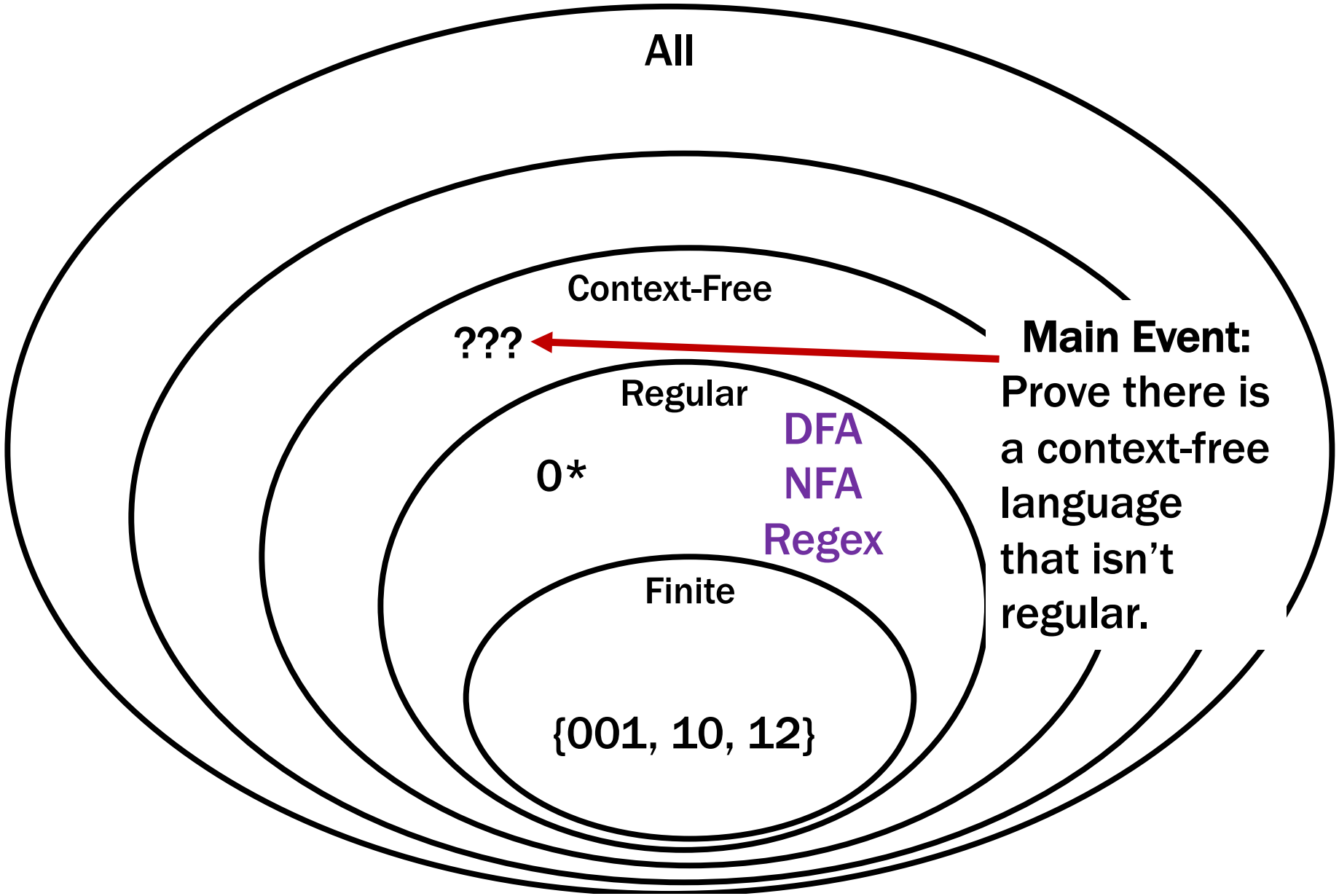
Languages and Representations!



Languages and Representations!

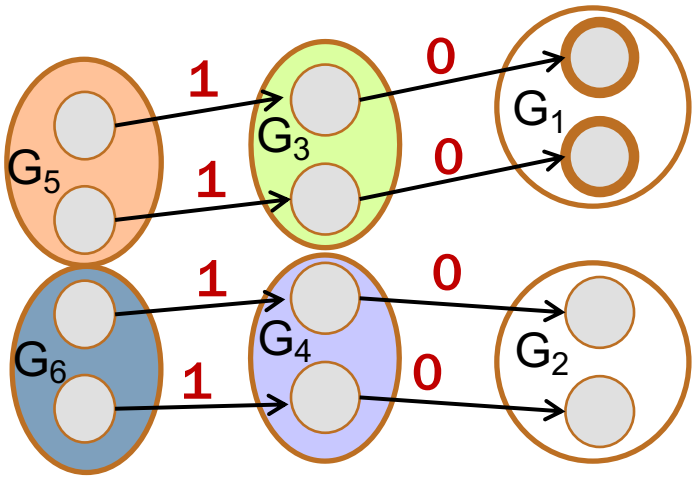


Languages and Representations!



Recall: Minimization Algorithm

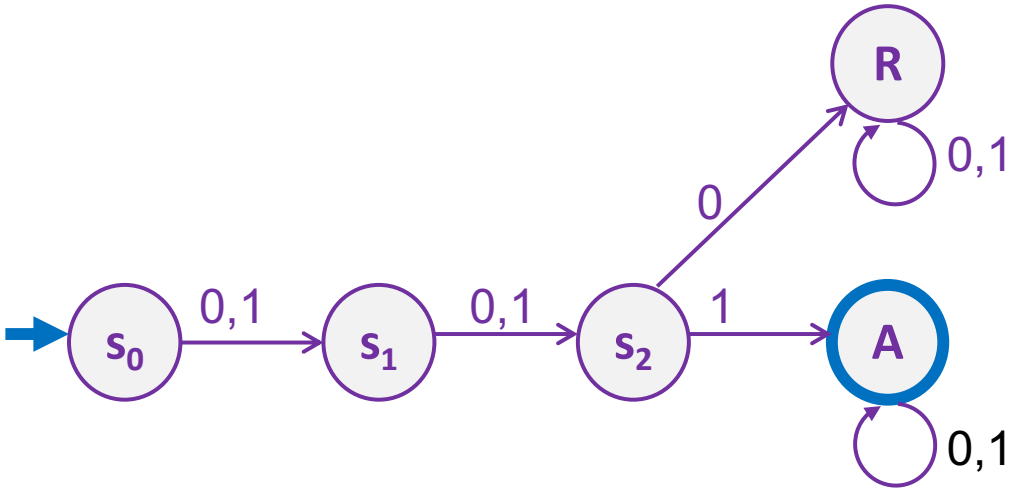
- Split states if they go different places on some input:
  - goal was to match the behavior of the *original* machine
  - but not splitting would actually change output...



Must split G_5 from G_6 because they lead to different outcomes (accept vs reject) if next two characters are "10"

A machine that did not split G_5 from G_6 would be wrong on some strings

Recall: Binary strings with a 1 in the 3rd position from the start



Split {A} from {s_0, s_1, s_2, R} different on ε

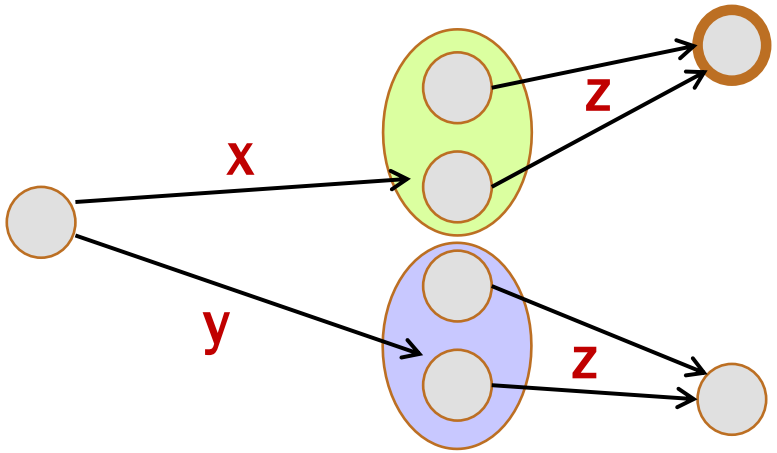
Split {s_2} from {s_0, s_1, R} different on 1

Split {s_1} from {s_0, R} different on 01

Split {s_0} from {R} different on 001

How to prove a DFA minimal?

- Find strings that must be "distinguished"
  - rewrite the argument to talk about strings, not states



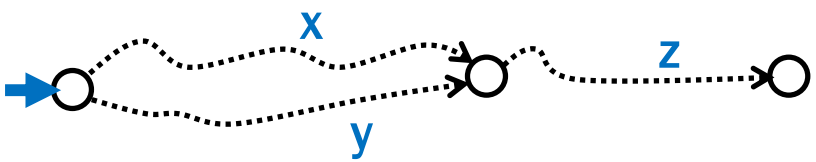
$x \bullet z \in L$

$y \bullet z \notin L$

Useful Lemmas about DFAs

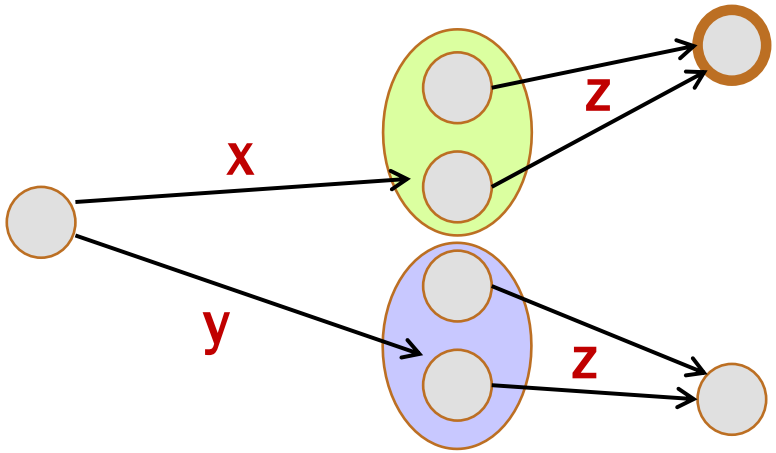
Lemma 2: If DFA **M** takes $\mathbf{x}, \mathbf{y} \in \Sigma^*$ to the same state, then for every $\mathbf{z} \in \Sigma^*$, M accepts $\mathbf{x} \bullet \mathbf{z}$ iff it accepts $\mathbf{y} \bullet \mathbf{z}$.

M can't remember if the input was $\mathbf{x}$ or $\mathbf{y}$.



How to prove a DFA minimal?

- Find strings that must be "distinguished"
  - two strings must be *distinguished* if some completion puts one the language and the other not

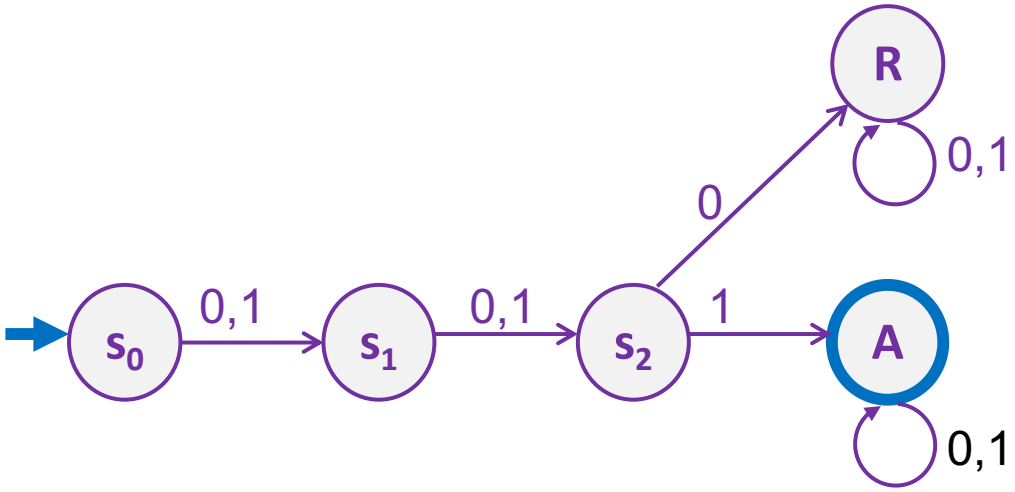


$x \bullet z \in L$

$y \bullet z \notin L$

Any correct machine must take x and y to different states.
The completion proves they must be "distinguished".

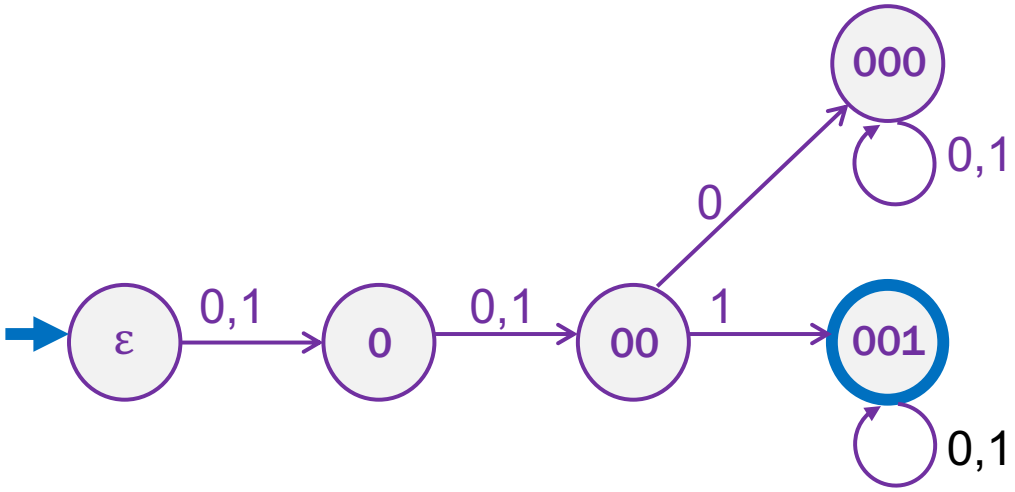
Recall: Binary strings with a 1 in the 3rd position from the start



Can turn this into an argument about strings…

- ε goes to s_0
- 0 goes to s_1
- 00 goes to s_2
- 000 goes to R
- 001 goes to A

Recall: Binary strings with a 1 in the 3rd position from the start

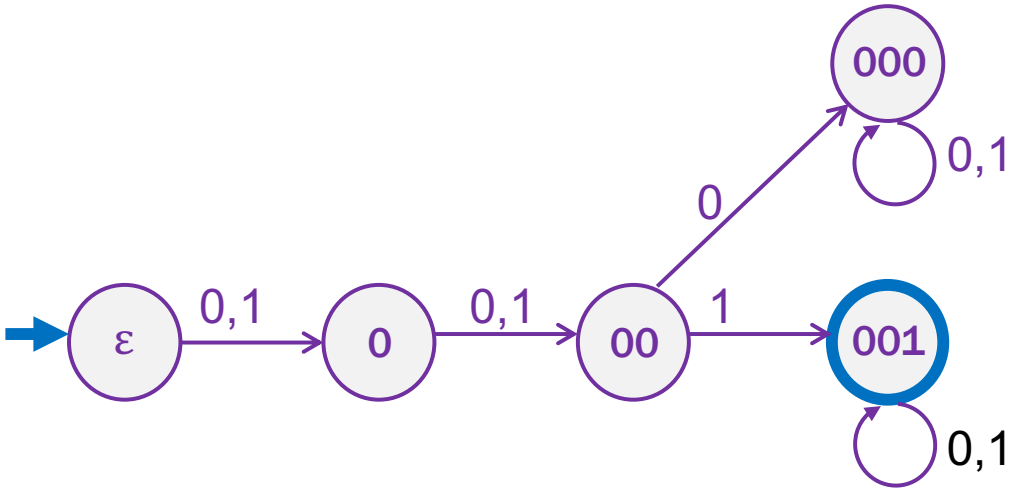


Minimization algorithm shows that these strings must be distinguished

– for every pair, x and y, there is a completion putting one in the language and one not

$$x \bullet z \in L$$
$$y \bullet z \notin L$$

Recall: Binary strings with a 1 in the 3rd position from the start



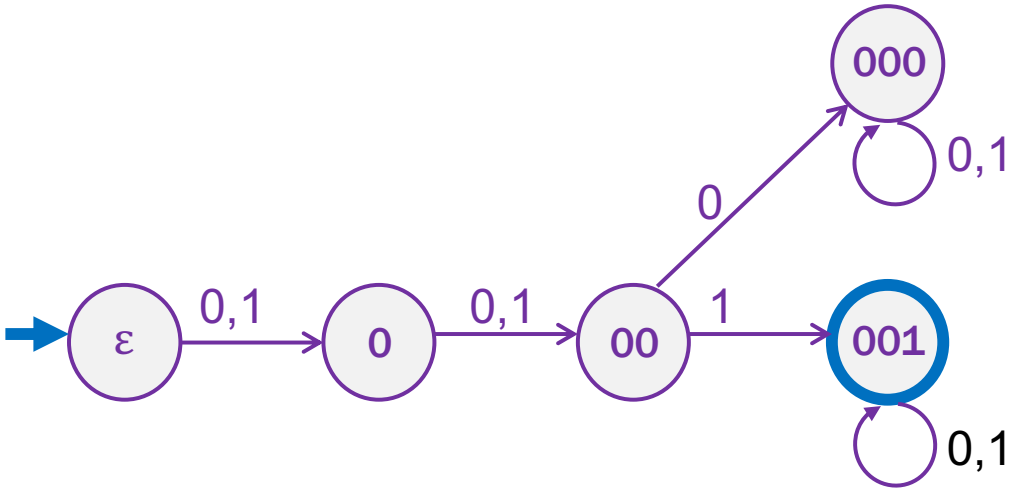
Each pair has a completion showing they must be distinguished:

	ε	0	00	000	001
ε					
0					
00					
000					
001					

$x \bullet z \in L$

$y \bullet z \notin L$

Recall: Binary strings with a 1 in the 3rd position from the start



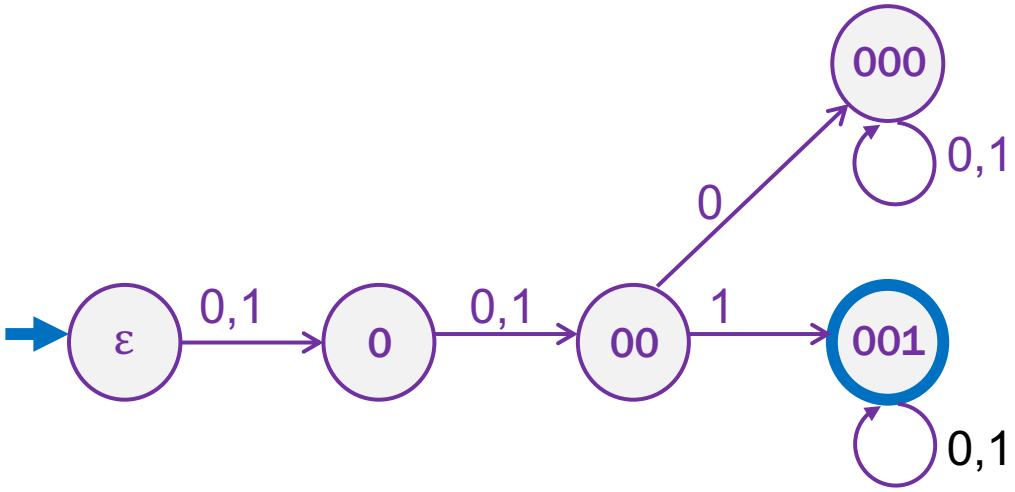
Each pair has a completion showing they must be distinguished:

	ε	0	00	000	001
ε					ε
0					ε
00					ε
000					ε
001					

$001 \bullet \varepsilon \in L$

$0 \bullet \varepsilon \notin L$

Recall: Binary strings with a 1 in the 3rd position from the start



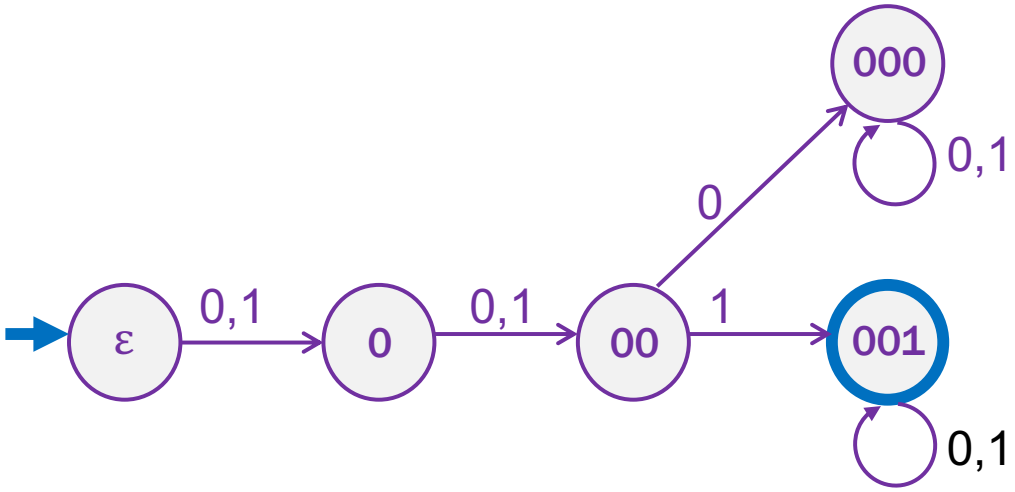
Each pair has a completion showing they must be distinguished:

	ε	0	00	000	001
ε			1		ε
0			1		ε
00				1	ε
000					ε
001					

$00 \bullet 1 \in L$

$0 \bullet 1 \notin L$

Recall: Binary strings with a 1 in the 3rd position from the start



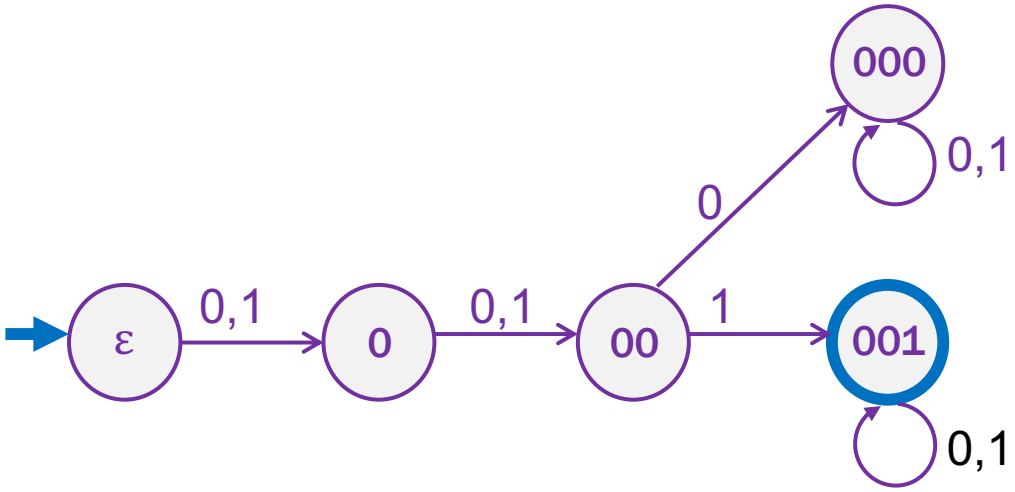
Each pair has a completion showing they must be distinguished:

	ε	0	00	000	001
ε		01	1		ε
0			1	01	ε
00				1	ε
000					ε
001					

$0 \bullet 01 \in L$

$000 \bullet 01 \notin L$

Recall: Binary strings with a 1 in the 3rd position from the start



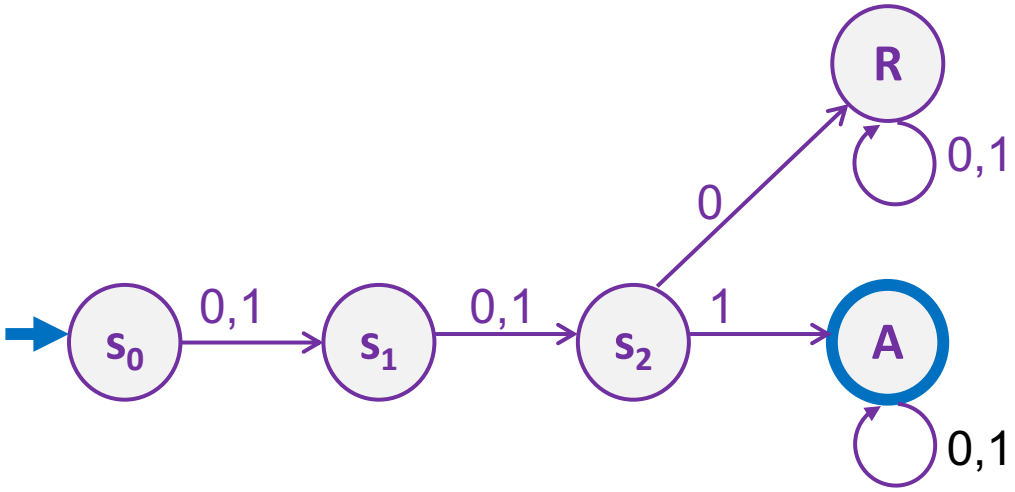
Each pair has a completion showing they must be distinguished:

	ε	0	00	000	001
ε		01	1	001	ε
0			1	01	ε
00				1	ε
000					ε
001					

ε • 001 ∈ L

000 • 001 ∉ L

Recall: Binary strings with a 1 in the 3rd position from the start



{ε, 0, 00, 000, 001} is a distinguishing set for this language, so any correct machine has at least 5 states

Hence, this machine is minimal!

The language of "Binary Palindromes" is Context-Free

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

Can prove this is not regular (irregular)
by finding an *infinite* distinguishing set!

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

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

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

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

Useful Lemmas about DFAs

Lemma 1: If DFA **M** has **n** states and a set **S** contains *more* than **n** strings, then **M** takes at least two strings from **S** to the same state.

M can't take n+1 or more strings to different states because it doesn't have n+1 different states.

So, some pair of strings must go to the same state.

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

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

We will 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, by Lemma 1, 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

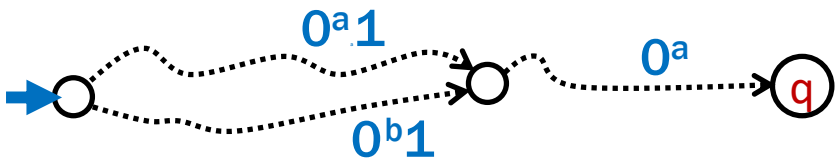
Suppose for contradiction that some DFA, M, accepts B.

We will 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, by Lemma 2, 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.



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

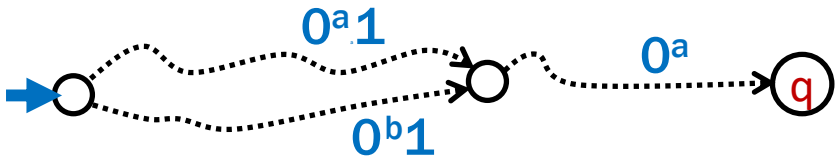
Suppose for contradiction that some DFA, M, accepts B.

We will 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, by Lemma 2, 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.



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, accepts B.

We will 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, by Lemma 2, 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.

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.

This proves that M does not recognize B, contradicting our assumption that it does. 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 prefixes (which we intend to complete later).
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_at and s_bt end at the same state q of M. Since $s_at \in L$ and $s_bt \notin L$, M does not recognize L."
6. "Thus, no DFA recognizes L."

Showing that a Language L is not regular

The choice of S is the creative part of the proof

You must find an infinite set S with the property that *no two* strings can be taken to the same state

- i.e., for ***every pair*** of strings S there is a "rest of the string" that makes one accepting and one rejecting

Prove **A** = $\{0^n1^n : n \geq 0\}$ 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**.

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 $A = \{0^n 1^n : n \geq 0\}$ 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 "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_at and s_bt end at the same state q of M. Since $s_at \in L$ and $s_bt \notin L$, M does not recognize L."
6. "Thus, no DFA recognizes L."