

# CS421 Topic 12: Converting Regular Expressions to DFAs<sup>1</sup>

Sameer Sundresh  
sundresh@uiuc.edu

University of Illinois at Urbana-Champaign

June 14, 2007

---

<sup>1</sup>Based on slides by Mattox Beckman, as updated by Vikram Adve, Gul Agha, Elsa Gunter, and Mark Hills

## Using Regular Expressions

NFA to DFA

RE to NFA

Activity

## Using Regular Expressions

We've seen that regular expressions have nice descriptive properties, providing a convenient way to specify syntax. We've also seen that finite automata provide a nice model for matching strings, providing small “machines” tuned to match strings. How can we leverage the nice characteristics of both for recognizing language syntax?

- ▶ First, convert regular expressions to NFAs, since there is a nice, direct conversion from REs to NFAs
- ▶ Second, convert NFAs to DFAs, since they provide equivalent computational power and since DFAs have a nicer computational model (no backtracking needed)

## Subset Construction

To convert from an NFA to a DFA, we somehow need to account for multiple paths with the same label.

- ▶ **Key Concept:** All states reachable in NFA along a given path should be same state in DFA
- ▶ DFA states will thus be *subsets* of set of NFA states

## $\epsilon$ Closure

Also, we don't have  $\epsilon$  in DFAs – how do we eliminate this?

- ▶ The  $\epsilon$  closure  $S(\epsilon)$  of  $S$ , a set of states, is the smallest set
  - ▶ containing  $S$
  - ▶ for all  $s \in S$  and all states  $t$ , if  $(s, \epsilon, t)$  is an edge then  $t \in S$

# Reachable States

For any letter  $\alpha \in \Sigma$ , with  $\Sigma$  the alphabet, the  $\alpha$  reachable states from  $S$ ,  $S(\alpha)$ , are defined as:

$$S(\alpha) = \{t \mid (s, \alpha, t) \text{ an edge for some } s \in S\}$$

- ▶ **Key Intuition:** To figure out where we can get to from a set of states when given a character from the alphabet, find all states with an incoming edge labeled with the character and coming from some state in the set

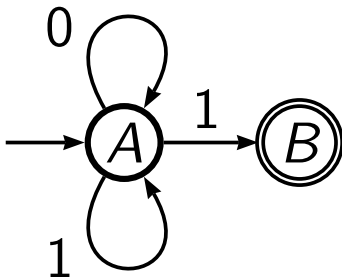
## Performing the Subset Construction

- ▶ Begin with  $S = \{s_0\}(\epsilon)$ , the  $\epsilon$ -closure of the set containing just the NFA start state – this is our DFA start state
- ▶ While our states keep changing
  - ▶ for each new state  $S$  and letter  $\alpha$ , add state  $S(\alpha)(\epsilon)$ , the  $\epsilon$  closure of all states  $\alpha$ -reachable from  $S$
  - ▶ add edge  $(S, \alpha, S(\alpha)(\epsilon))$
- ▶ Mark as final states any set of states in the DFA containing a final state from the NFA

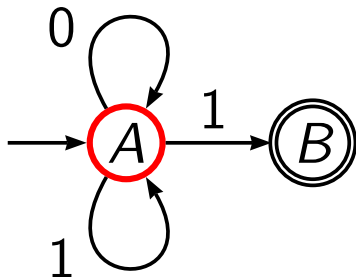
Any problems?

- ▶ Potentially exponential in the number of NFA states

## Example 1



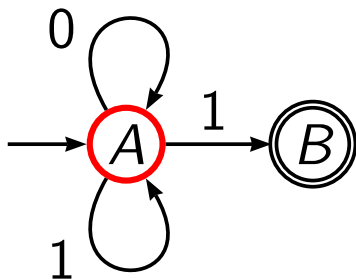
## Example 1



New states:  $\{A\}$

New edges:

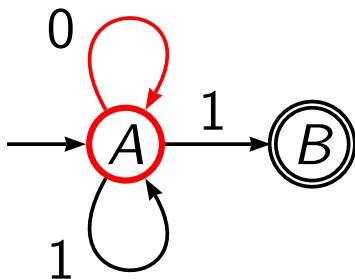
## Example 1



New states:  $\{A\}$

New edges:  $(\{A\}, 0, )$ ,  $(\{A\}, 1, )$

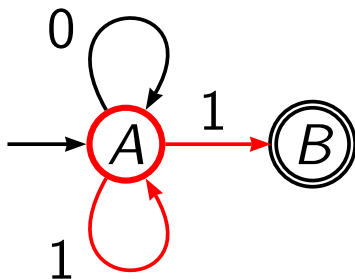
## Example 1



New states:  $\{A\}$

New edges:  $(\{A\}, 0, \{A\}), (\{A\}, 1, )$

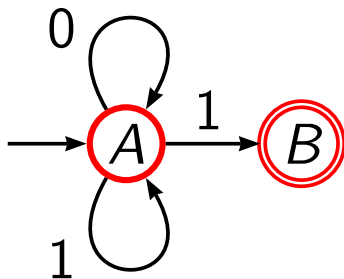
## Example 1



New states:  $\{A\}$

New edges:  $(\{A\}, 0, \{A\}), (\{A\}, 1, \{A, B\})$

## Example 1

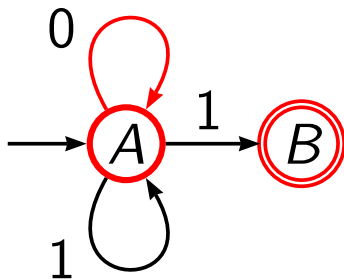


New states:  $\{A\}, \{A, B\}$

New edges:

$(\{A\}, 0, \{A\}), (\{A\}, 1, \{A, B\}), (\{A, B\}, 0, \quad), (\{A, B\}, 1, \quad)$

## Example 1

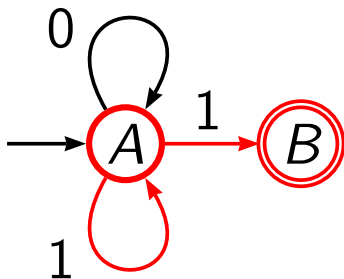


New states:  $\{A\}, \{A, B\}$

New edges:

$(\{A\}, 0, \{A\}), (\{A\}, 1, \{A, B\}), (\{A, B\}, 0, \{A\}), (\{A, B\}, 1, )$

## Example 1

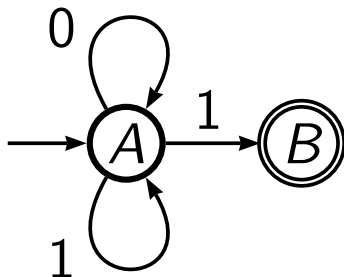


New states:  $\{A\}, \{A, B\}$

New edges:

$(\{A\}, 0, \{A\}), (\{A\}, 1, \{A, B\}), (\{A, B\}, 0, \{A\}), (\{A, B\}, 1, \{A, B\})$

## Example 1



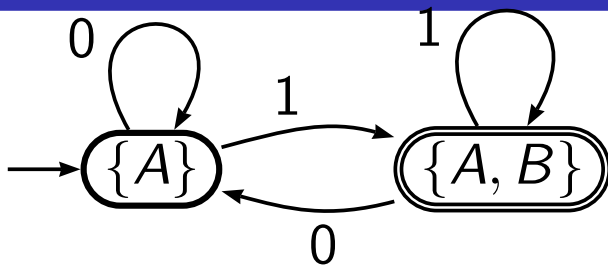
New states:  $\{A\}$ ,  $\{A, B\}$

New edges:

$(\{A\}, 0, \{A\})$ ,  $(\{A\}, 1, \{A, B\})$ ,  $(\{A, B\}, 0, \{A\})$ ,  $(\{A, B\}, 1, \{A, B\})$

Final states:  $\{A, B\}$

## Example 1



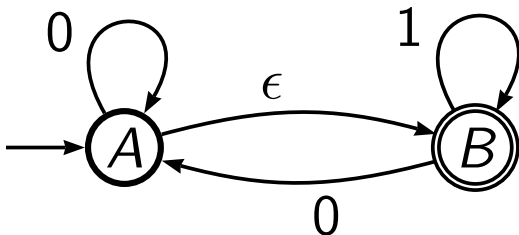
New states:  $\{A\}, \{A, B\}$

New edges:

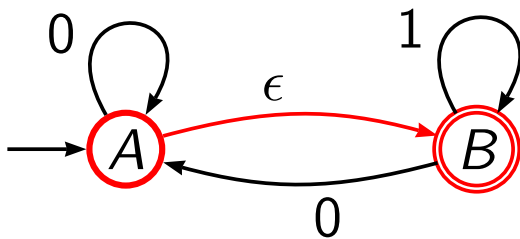
$(\{A\}, 0, \{A\}), (\{A\}, 1, \{A, B\}), (\{A, B\}, 0, \{A\}), (\{A, B\}, 1, \{A, B\})$

Final states:  $\{A, B\}$

## Example 2



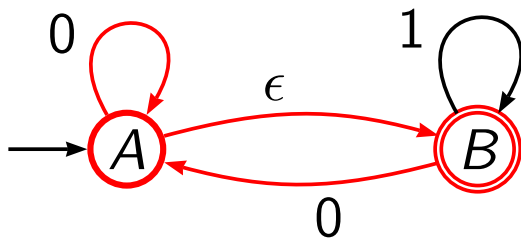
## Example 2



New states:  $\{A\}(\epsilon) = \{A, B\}$

New edges:  $(\{A, B\}, 0, \quad), (\{A, B\}, 1, \quad)$

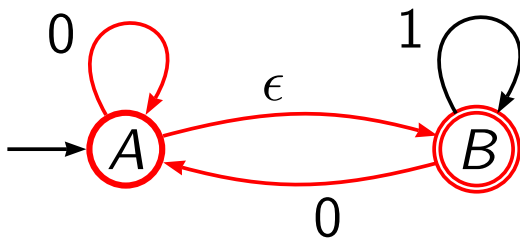
## Example 2



New states:  $\{A, B\}$

New edges:  $(\{A, B\}, 0, \{A\}(\epsilon)), (\{A, B\}, 1, )$

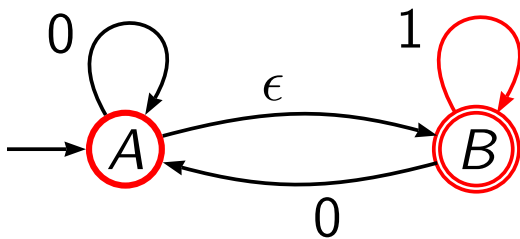
## Example 2



New states:  $\{A, B\}$

New edges:  $(\{A, B\}, 0, \{A, B\}), (\{A, B\}, 1, )$

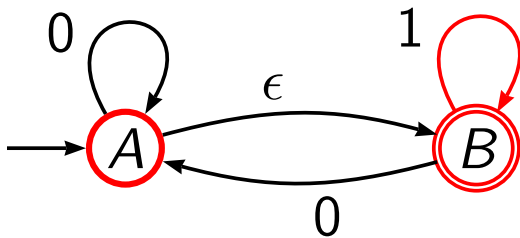
## Example 2



New states:  $\{A, B\}, \{B\}(\epsilon)$

New edges:  $(\{A, B\}, 0, \{A, B\}), (\{A, B\}, 1, \{B\}(\epsilon))$

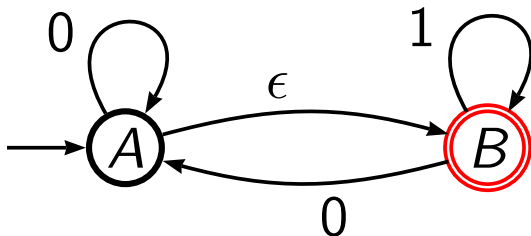
## Example 2



New states:  $\{A, B\}, \{B\}(\epsilon) = \{B\}$

New edges:  $(\{A, B\}, 0, \{A, B\}), (\{A, B\}, 1, \{B\})$

## Example 2

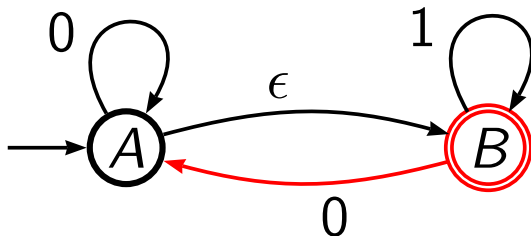


New states:  $\{A, B\}, \{B\}$

New edges:

$(\{A, B\}, 0, \{A, B\}), (\{A, B\}, 1, \{B\}), (\{B\}, 0, \{A, B\}), (\{B\}, 1, \{B\})$

## Example 2

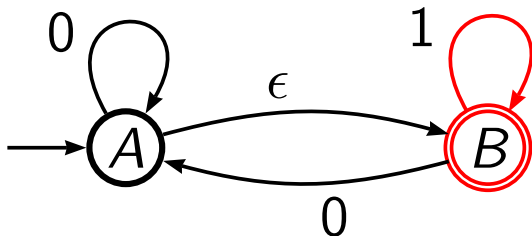


New states:  $\{A, B\}, \{B\}$

New edges:

$(\{A, B\}, 0, \{A, B\}), (\{A, B\}, 1, \{B\}), (\{B\}, 0, \{A, B\}), (\{B\}, 1, )$

## Example 2

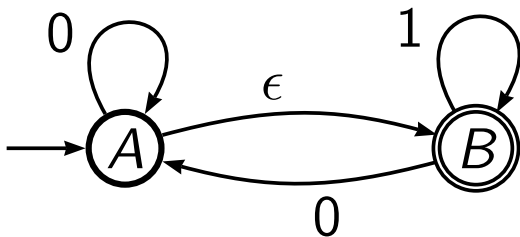


New states:  $\{A, B\}, \{B\}$

New edges:

$(\{A, B\}, 0, \{A, B\}), (\{A, B\}, 1, \{B\}), (\{B\}, 0, \{A, B\}), (\{B\}, 1, \{B\})$

## Example 2



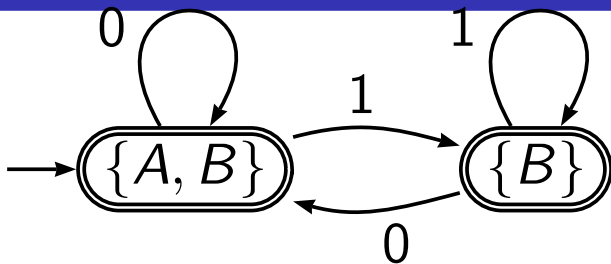
New states:  $\{A, B\}, \{B\}$

New edges:

$(\{A, B\}, 0, \{A, B\}), (\{A, B\}, 1, \{B\}), (\{B\}, 0, \{A, B\}), (\{B\}, 1, \{B\})$

Final states:  $\{\{A, B\}, \{B\}\}$

## Example 2



New states:  $\{A, B\}, \{B\}$

New edges:

$(\{A, B\}, 0, \{A, B\}), (\{A, B\}, 1, \{B\}), (\{B\}, 0, \{A, B\}), (\{B\}, 1, \{B\})$

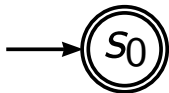
Final states:  $\{\{A, B\}, \{B\}\}$

## Converting Regular Expressions to NFAs

Now that we've seen how to tackle the second part, converting an NFA to a DFA, we can tackle the first, how to generate an NFA that will recognize the same language as a regular expression. We will take a “tinker toy” approach, assembling “widgets” representing each of the parts of the regular expression.

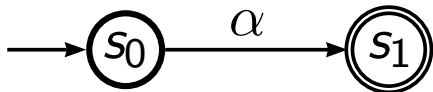
## Converting $\epsilon$

A widget for  $\epsilon$  essentially does nothing – we start in an accepting state.



## Converting Individual Characters

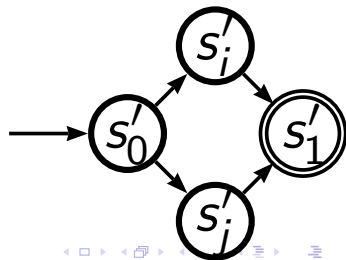
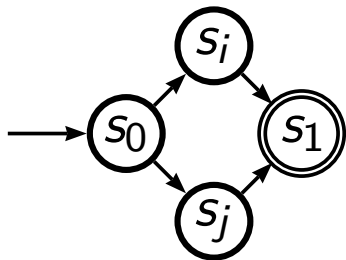
A widget for an individual character just needs to take that character as input and accept it. For any character  $\alpha$ :



## Concatenation

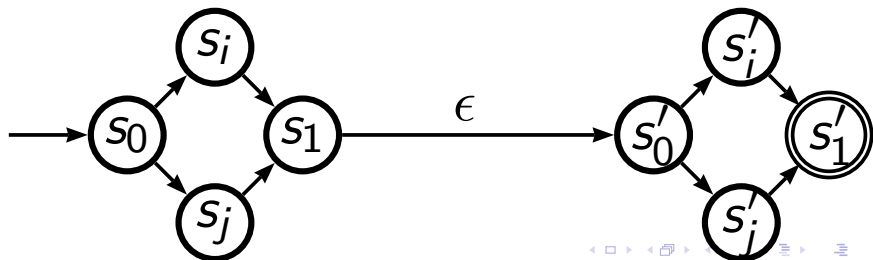
To handle concatenation, we will put in  $\epsilon$  transitions from the final state in the first to the start state of the second, and then only keep the final state in the second and the start state in the first.

This allows us to recognize the first string and then automatically “flow” into the second NFA.



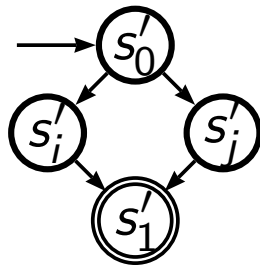
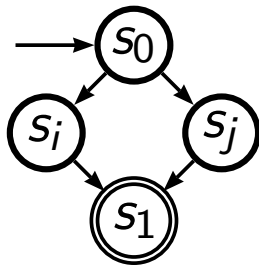
## Concatenation

What if we have multiple final states? We can either add edges from all of them to the start state of the second NFA or we can “fake” having just one by adding a new final state, putting in  $\epsilon$  transitions from all the old final states in the NFA, and then changing them to normal states.

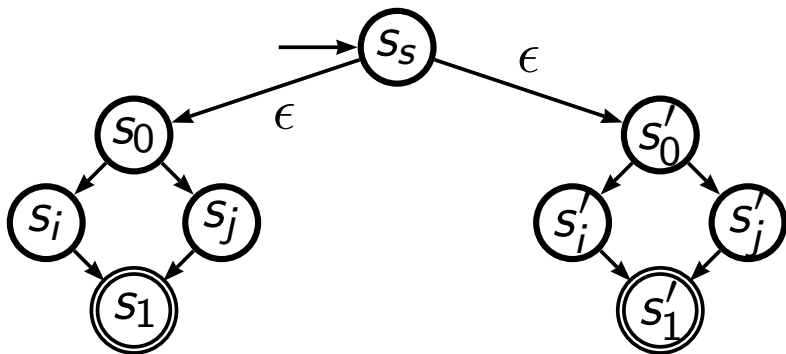


# Choice

To handle choice, we can add a new start state that transitions to the old start states, taking advantage of non-determinism.

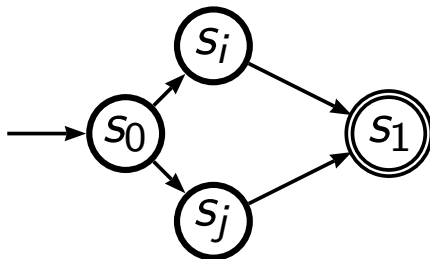


# Choice



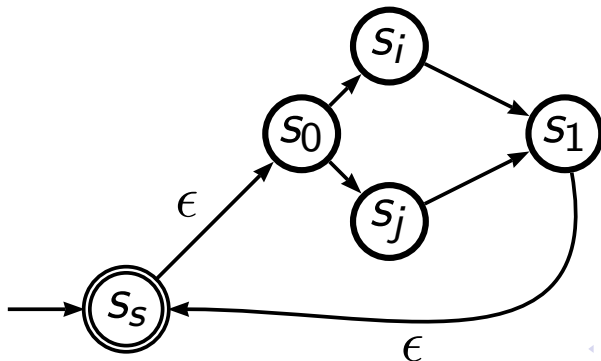
# Kleene \*

To handle the  $*$  operation, we need to be able to successfully accept no input; we also need to be able to accept input and then “loop back”, to accept more than 1.



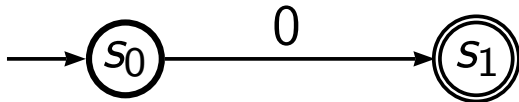
# Kleene \*

We will add a new start state, and also make it the final state, and add in edges from the old final states to the new final state.



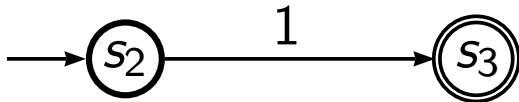
## An Example: $(0 + 1)^*1$

- ▶ NFA for 0



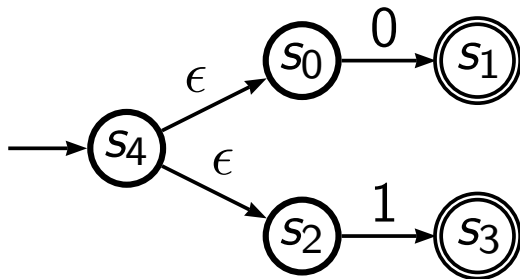
# An Example: $(0 + 1)^*1$

- NFA for 1



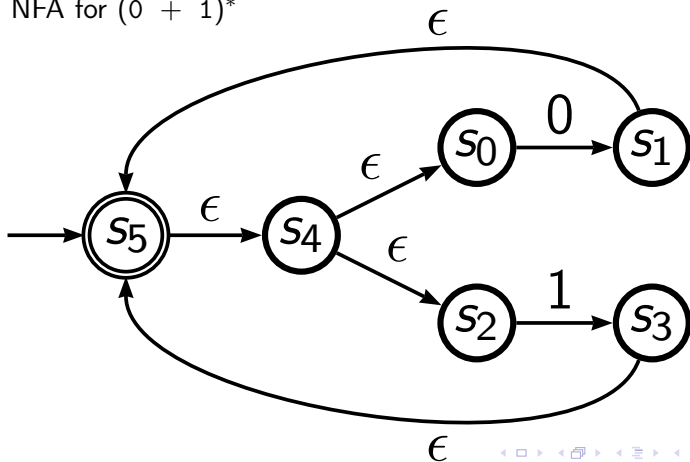
# An Example: $(0 + 1)^*1$

- NFA for  $0 + 1$



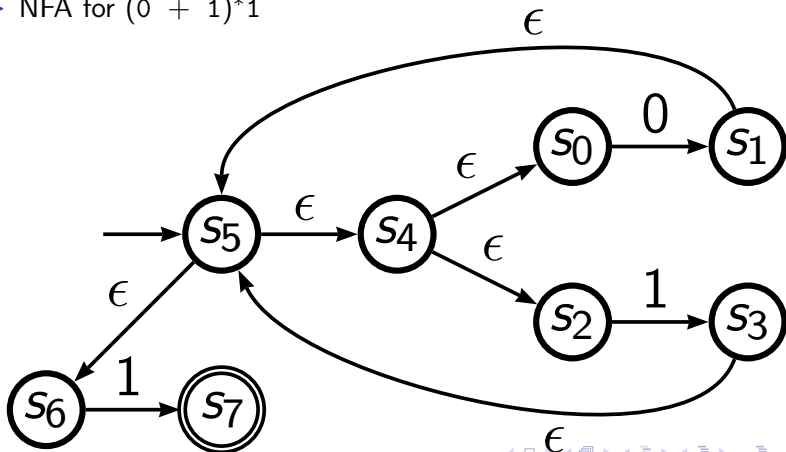
# An Example: $(0 + 1)^*1$

- NFA for  $(0 + 1)^*$



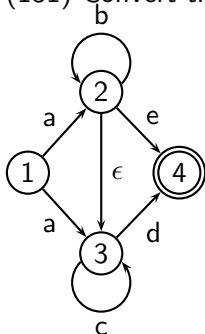
# An Example: $(0 + 1)^*1$

- NFA for  $(0 + 1)^*1$

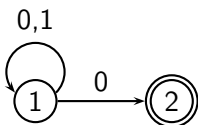


## Activity!

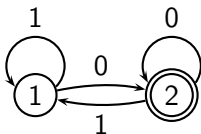
1. (129) Draw an NFA that accepts even binary numbers.
2. (130) Draw a DFA that accepts even binary numbers.
3. (131) Convert this example into a DFA.



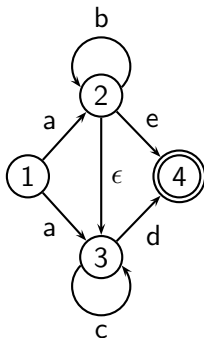
Draw an NFA that accepts even binary numbers.



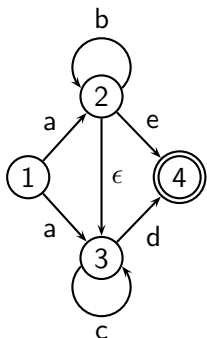
Draw a DFA that accepts even binary numbers.



# Convert this example into a DFA



# The DFA



becomes

