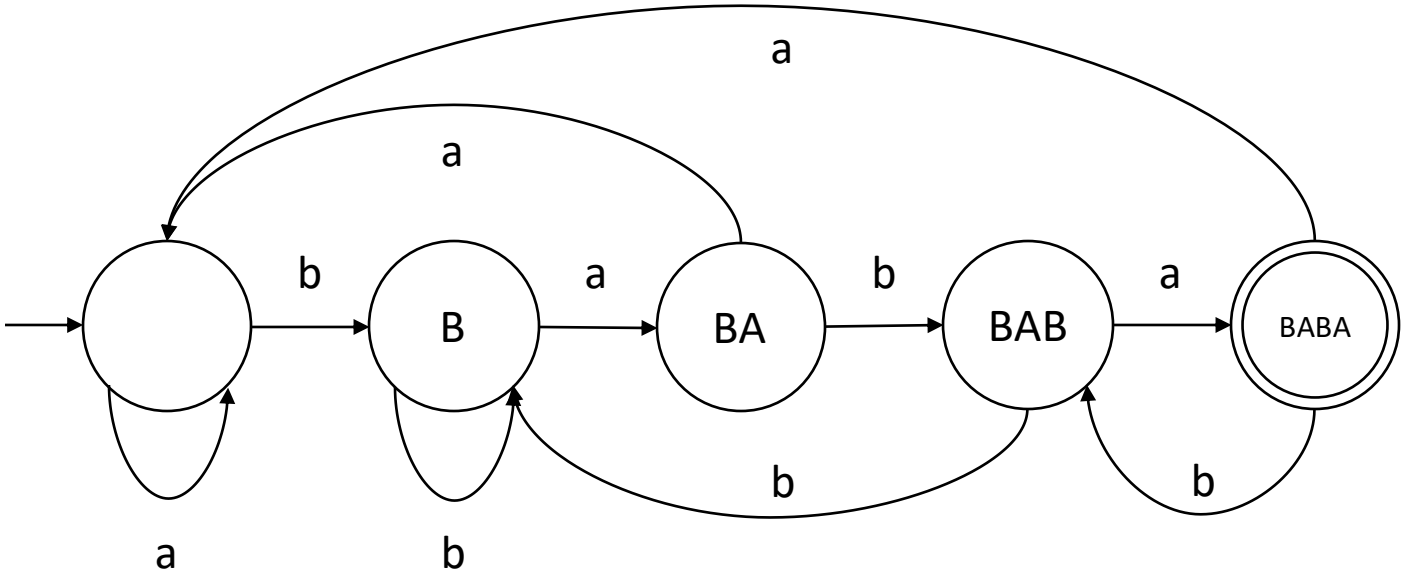
Nondeterministic Finite Automata

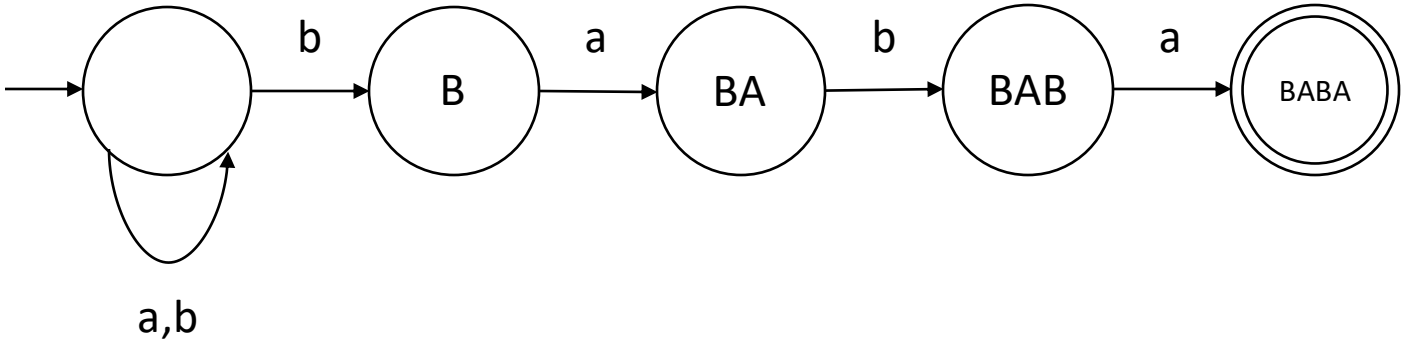
DFA: Strings Ending in "baba"

- In an earlier lecture, we constructed this DFA for the regular expression (a|b)*baba.
- The DFA is much more complicated than the regular expression.



Multiple Choices

- Why not represent (a|b)*baba with this drawing?



- The diagram has the same structure as the regular expression.
- But if we read 'b' in the first state, it requires **making a choice** about which state to go (stay in the first state or go to B).
- Our automaton is no longer **deterministic**.

Nondeterministic Computation

- Is it a problem if our automaton's behaviour is nondeterministic?
- In a practical context, yes, but in a theoretical context, not really.
- There are two theoretical interpretations of what it means to solve a problem with **nondeterministic computation**.

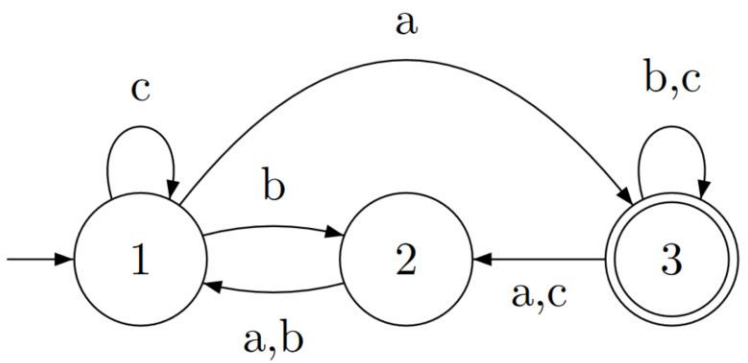
1. When there is a choice between multiple states, the computer *always picks the right one*. The problem is solved if there is *any sequence of choices* that leads to a solution.
2. When there is a choice between states, the computer takes *all choices simultaneously.* In other words, it occupies multiple states at once. The problem is solved if it is solved in *any occupied state.*

Nondeterministic Finite Automata

- A **nondeterministic finite automaton** (NFA) has the same components as a DFA:
  - A finite set of *states* (represented as circles).
  - An *initial* or *starting* state (marked with an arrow pointing inwards).
  - A set of *accepting* states (the double-circled states).
  - A collection of *arrows* between the states, where each arrow is labelled with a character.
- The difference is that there are no restrictions on the arrows.
  - In a DFA, a state cannot have two arrows leading out that are labelled with the same character. NFAs have no such restriction.

DFA or NFA?

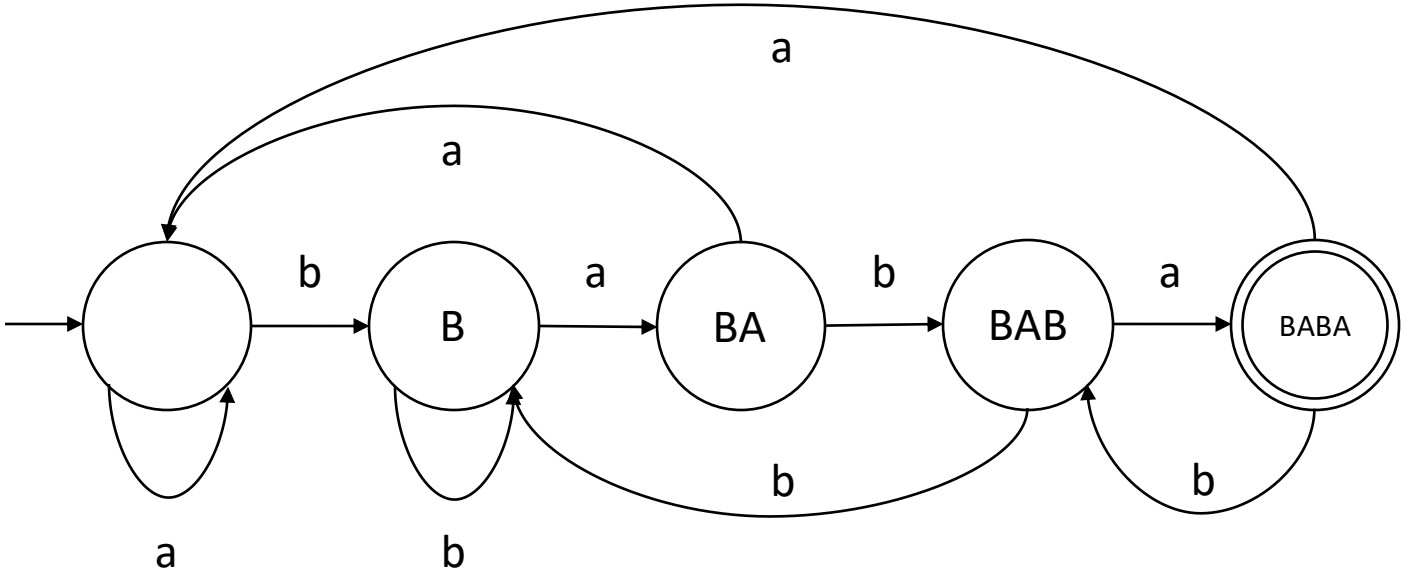
- Is this a DFA or an NFA?



- It is an NFA. The state 3 has two arrows leading out of it that are both labelled with the letter c.

DFA or NFA?

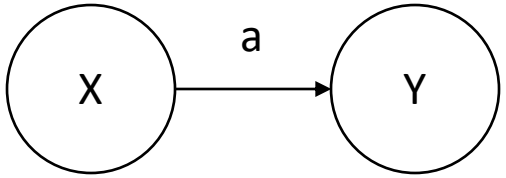
- Is this a DFA or an NFA?



- It's both! There are no nondeterministic arrows, but we still consider it an NFA because NFAs simply remove restrictions on arrows.

Terminology Notes

- An NFA is just a DFA with the restriction on arrows removed.
- Therefore, *every DFA is an NFA*.
- However, *not all NFAs are DFAs*.
- The "nondeterministic" in NFA should be thought of as saying "nondeterminism allowed" instead of "it must be nondeterministic".
- By the way, "arrows" in NFAs and DFAs are usually called **transitions**.
- *There is a **transition** from X to Y on a.*

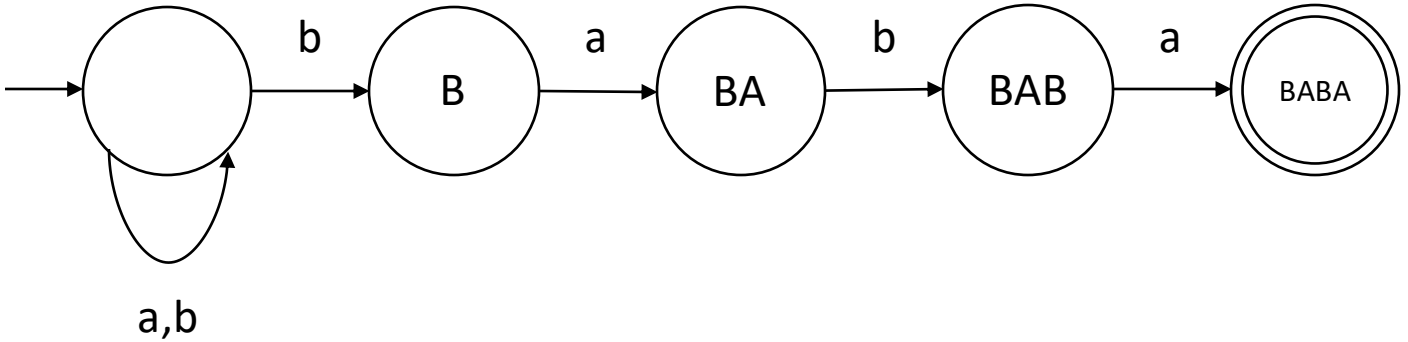


Language Recognition for NFAs

- We introduced DFAs to specify *language recognition programs*.
  - These programs take a string as input, and return "true" or "false" indicating whether the string is in a particular *language* (set of strings).
- As mentioned, we can think about nondeterminism in two ways, and this leads to two ways to think about language recognition.
  - An NFA will “always make the correct choice” if a correct choice exists, so if there exists a path to an accepting state for the current string, it will find such a path and accept the string. Otherwise it rejects the string.
  - An NFA will “make all possible choices”, potentially occupying multiple states at once. Once it finishes reading the string, it accepts if at least one of the states it occupies is accepting. If none are accepting, it rejects the string.

Language Recognition Example

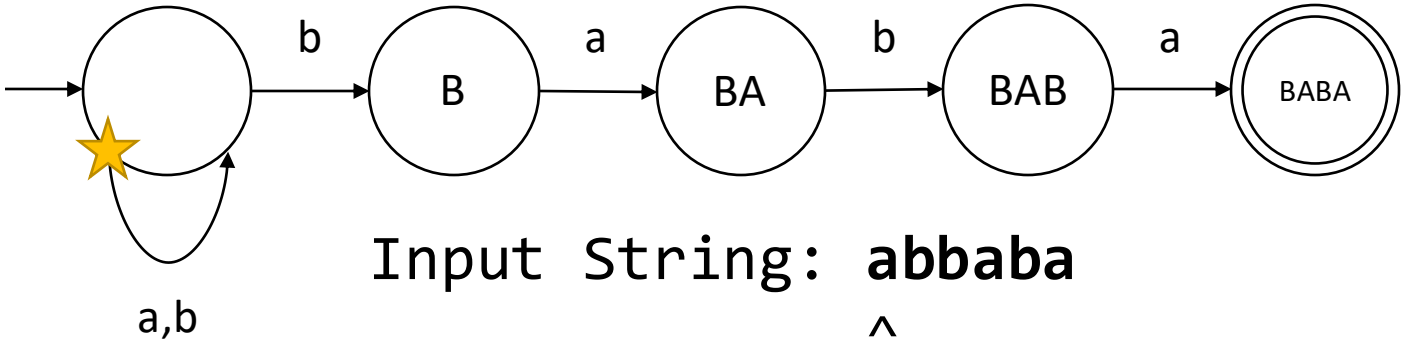
- Let's consider this NFA.



- How do we tell whether it recognizes "abbaba"?
- From the "always makes the correct choice" perspective, we just need to find *some* path that leads to the NFA accepting the string.

Language Recognition Example

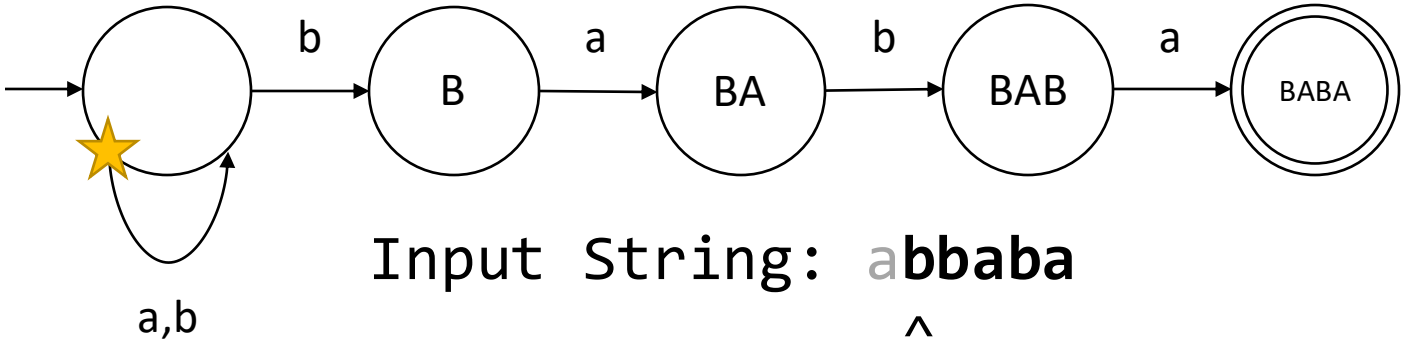
- Let's consider this NFA.



- How do we tell whether it recognizes "abbaba"?
- From the "always makes the correct choice" perspective, we just need to find *some* path that leads to the NFA accepting the string.

Language Recognition Example

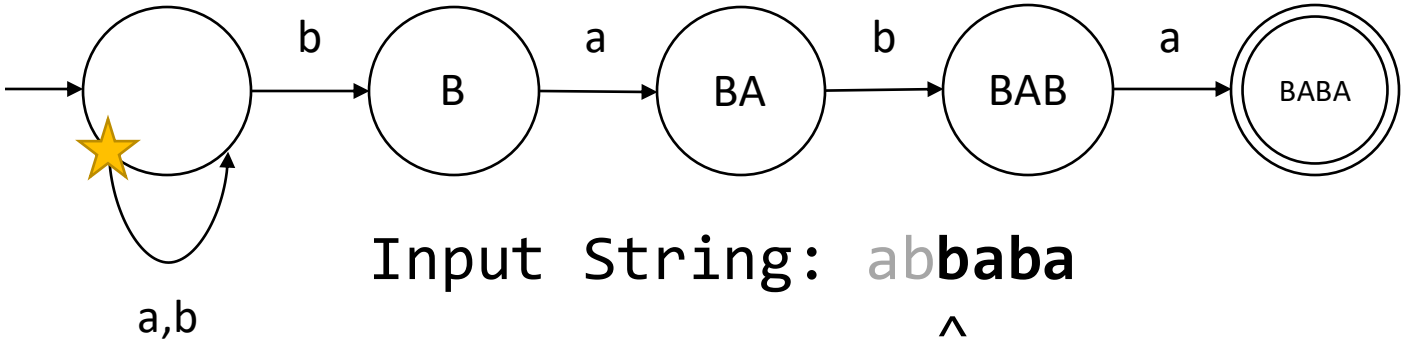
- Let's consider this NFA.



- How do we tell whether it recognizes "abbaba"?
- From the "always makes the correct choice" perspective, we just need to find *some* path that leads to the NFA accepting the string.

Language Recognition Example

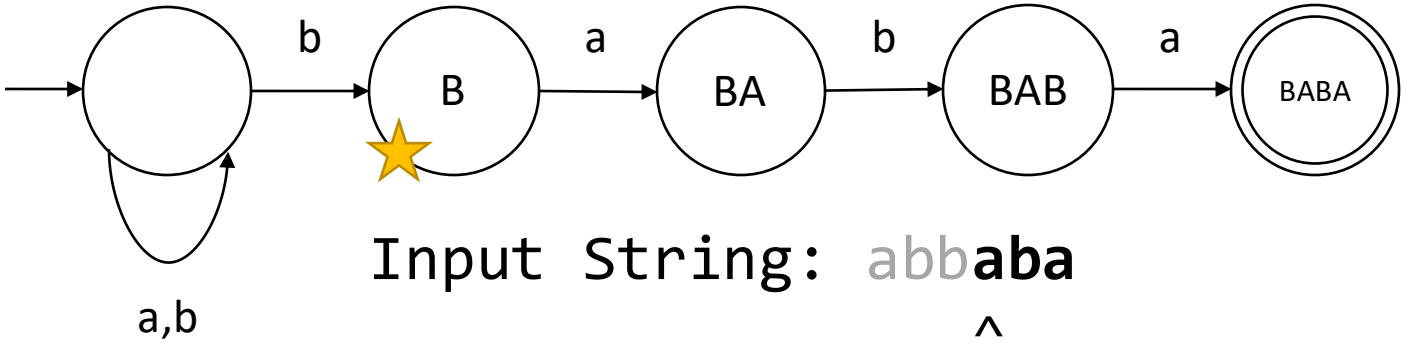
- Let's consider this NFA.



- How do we tell whether it recognizes "abbaba"?
- From the "always makes the correct choice" perspective, we just need to find *some* path that leads to the NFA accepting the string.

Language Recognition Example

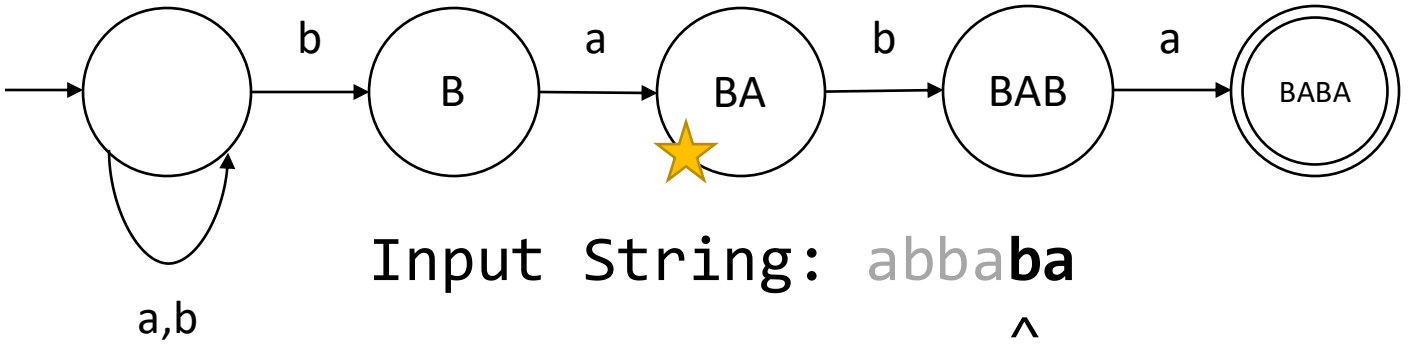
- Let's consider this NFA.



- How do we tell whether it recognizes "abbaba"?
- From the "always makes the correct choice" perspective, we just need to find *some* path that leads to the NFA accepting the string.

Language Recognition Example

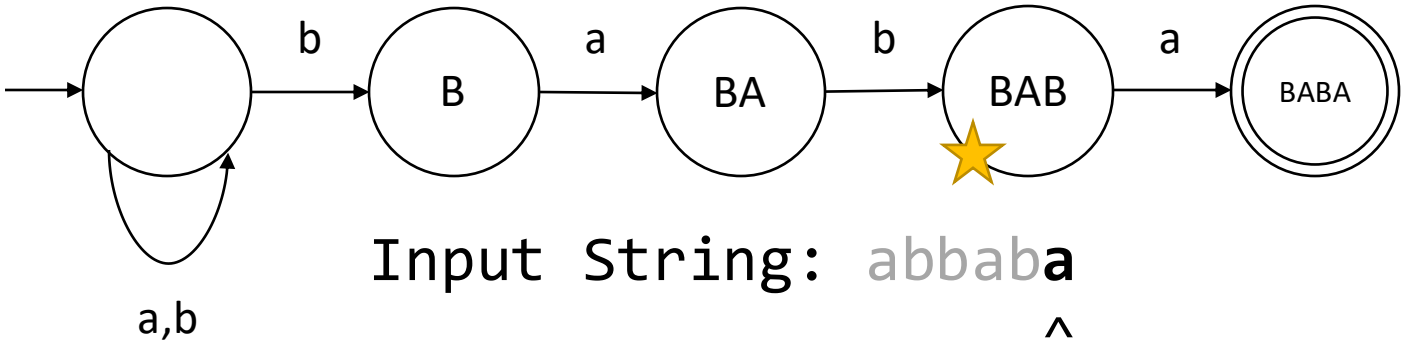
- Let's consider this NFA.



- How do we tell whether it recognizes "abbaba"?
- From the "always makes the correct choice" perspective, we just need to find *some* path that leads to the NFA accepting the string.

Language Recognition Example

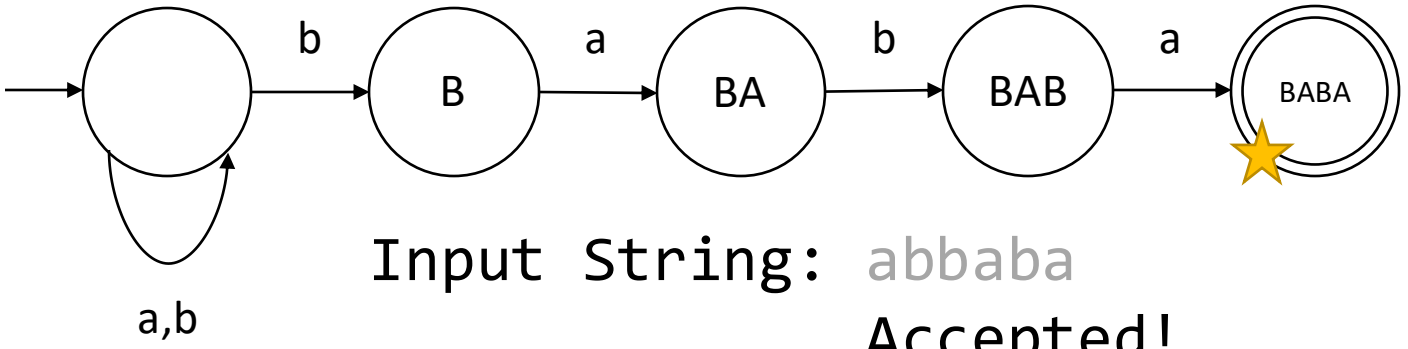
- Let's consider this NFA.



- How do we tell whether it recognizes "abbaba"?
- From the "always makes the correct choice" perspective, we just need to find *some* path that leads to the NFA accepting the string.

Language Recognition Example

- Let's consider this NFA.

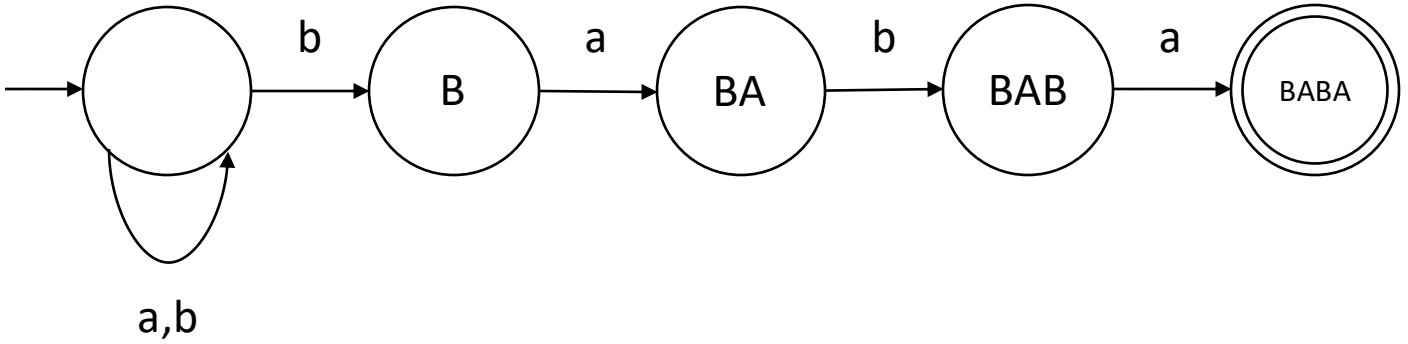



```
Input String: abbaba
              Accepted!
```

- How do we tell whether it recognizes "abbaba"?
- From the "always makes the correct choice" perspective, we just need to find *some* path that leads to the NFA accepting the string.

Language Recognition Example

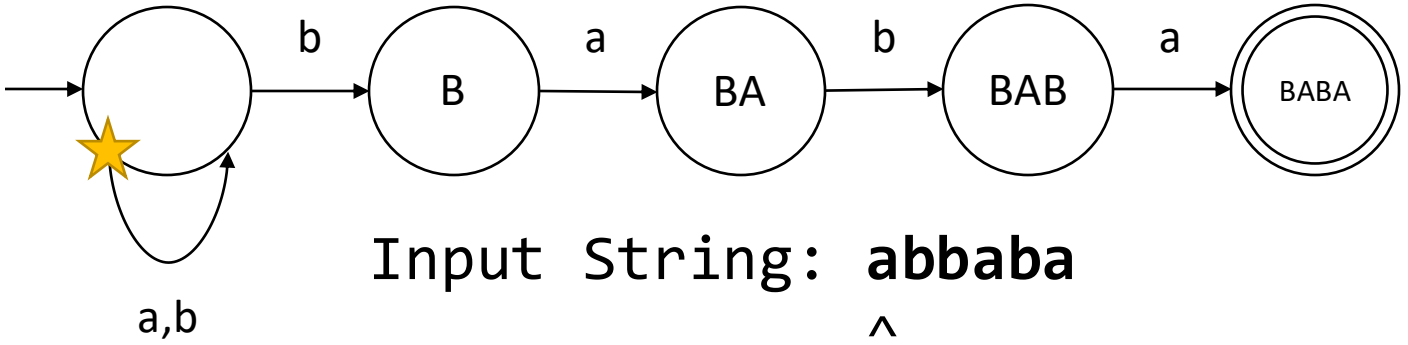
- Let's consider this NFA.



- How do we tell whether it recognizes "abbaba"?
- From the "makes all possible choices" perspective, we view the NFA as occupying multiple states at once. If any state is accepting at the end, we accept.

Language Recognition Example

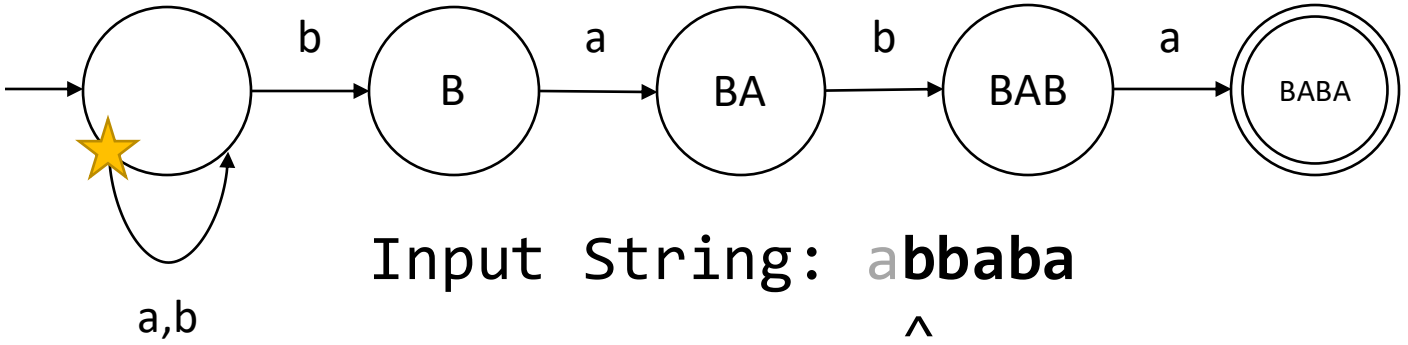
- Let's consider this NFA.



- How do we tell whether it recognizes "abbaba"?
- From the "makes all possible choices" perspective, we view the NFA as occupying multiple states at once. If any state is accepting at the end, we accept.

Language Recognition Example

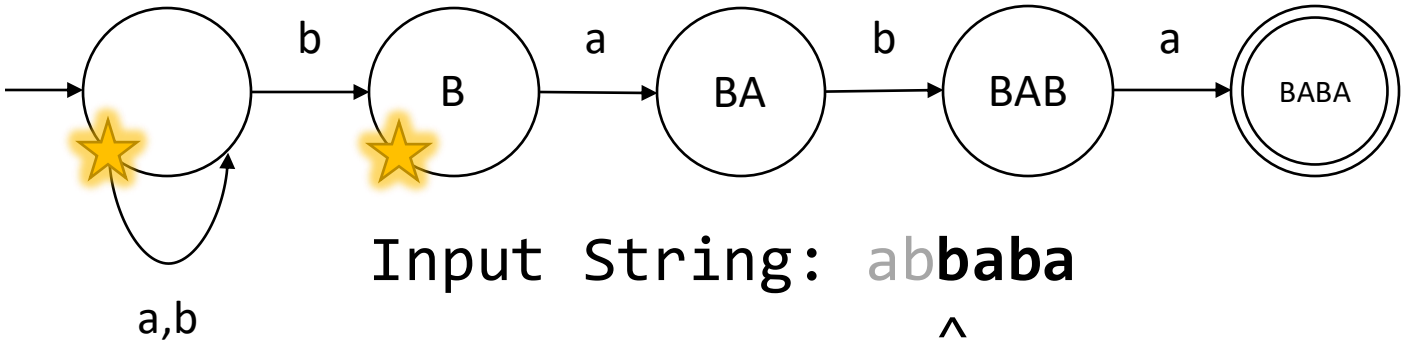
- Let's consider this NFA.



- How do we tell whether it recognizes "abbaba"?
- From the "makes all possible choices" perspective, we view the NFA as occupying multiple states at once. If any state is accepting at the end, we accept.

Language Recognition Example

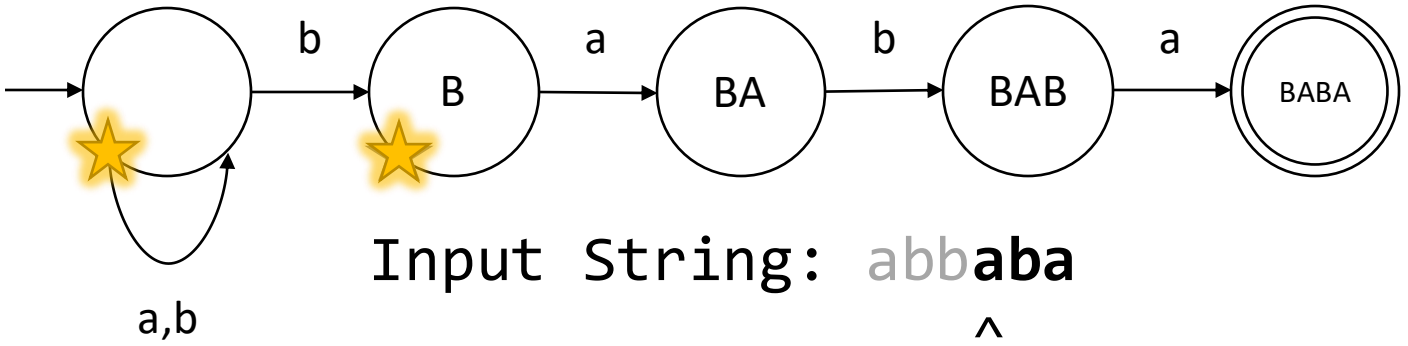
- Let's consider this NFA.



- How do we tell whether it recognizes "abbaba"?
- From the "makes all possible choices" perspective, we view the NFA as occupying multiple states at once. If any state is accepting at the end, we accept.

Language Recognition Example

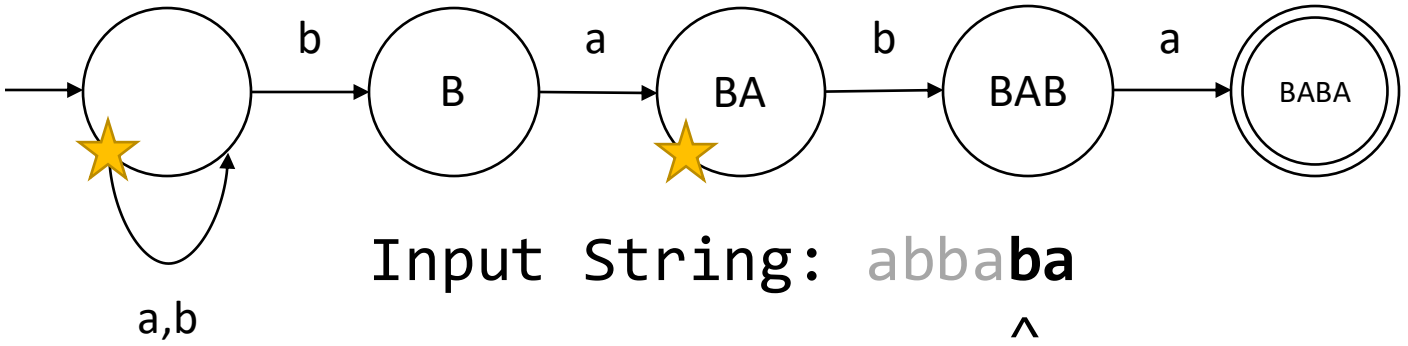
- Let's consider this NFA.



- How do we tell whether it recognizes "abbaba"?
- From the "makes all possible choices" perspective, we view the NFA as occupying multiple states at once. If any state is accepting at the end, we accept.

Language Recognition Example

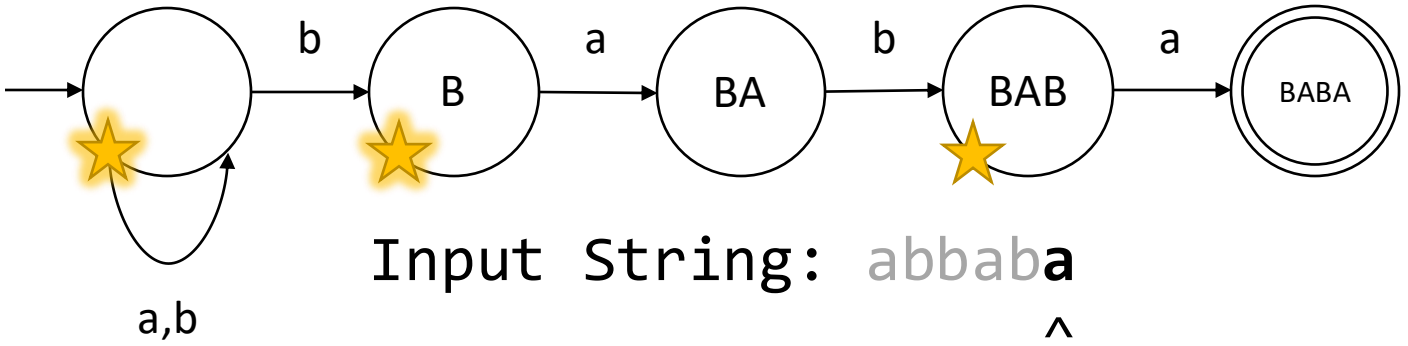
- Let's consider this NFA.



- How do we tell whether it recognizes "abbaba"?
- From the "makes all possible choices" perspective, we view the NFA as occupying multiple states at once. If any state is accepting at the end, we accept.

Language Recognition Example

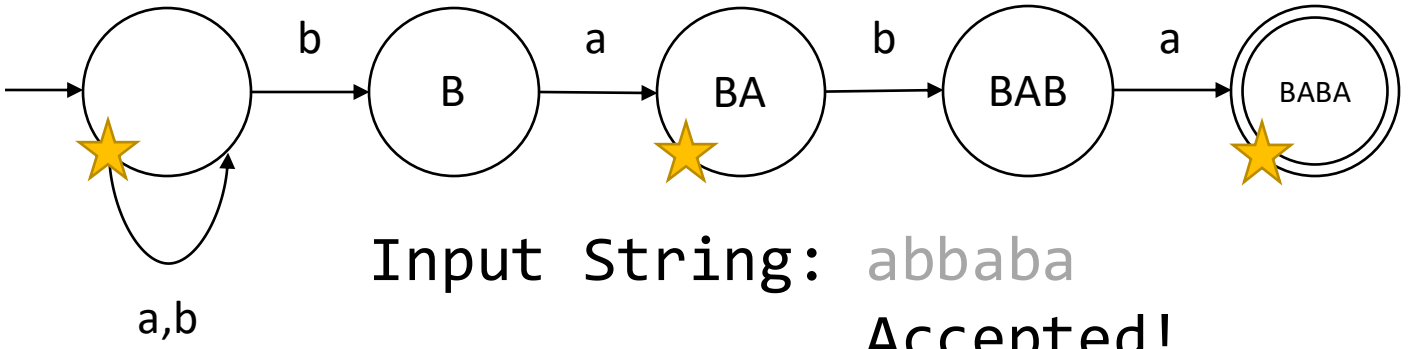
- Let's consider this NFA.



- How do we tell whether it recognizes "abbaba"?
- From the "makes all possible choices" perspective, we view the NFA as occupying multiple states at once. If any state is accepting at the end, we accept.

Language Recognition Example

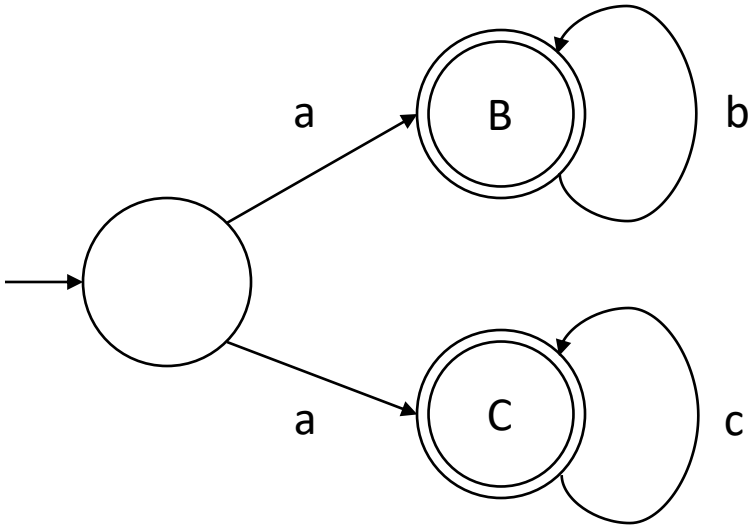
- Let's consider this NFA.



- How do we tell whether it recognizes "abbaba"?
- From the "makes all possible choices" perspective, we view the NFA as occupying multiple states at once. If any state is accepting at the end, we accept.

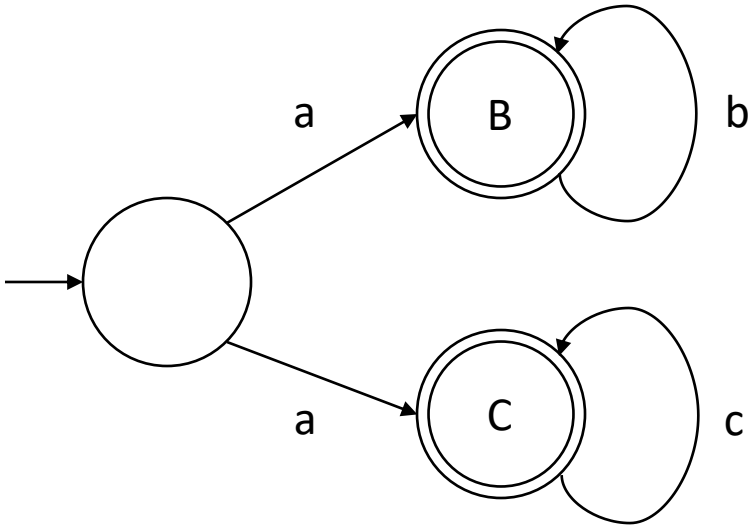
More Nondeterminism?!

- Here's an NFA corresponding to the regular expression ab*|ac*.
- What if we want an NFA for (ab*|ac*)*?



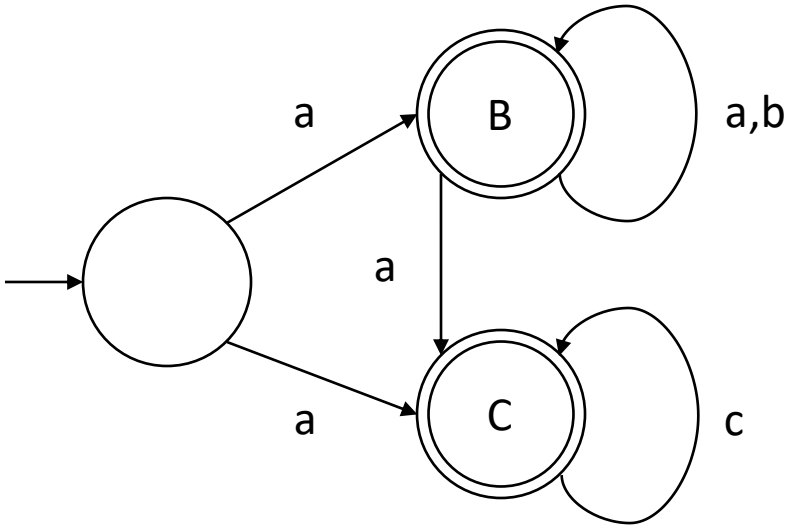
More Nondeterminism?!

- Let's try to turn this into an NFA for (ab*|ac*)*.
- If in state B, and the next input is "a", it could be the start of an "ab*" word OR a "ac*" word.



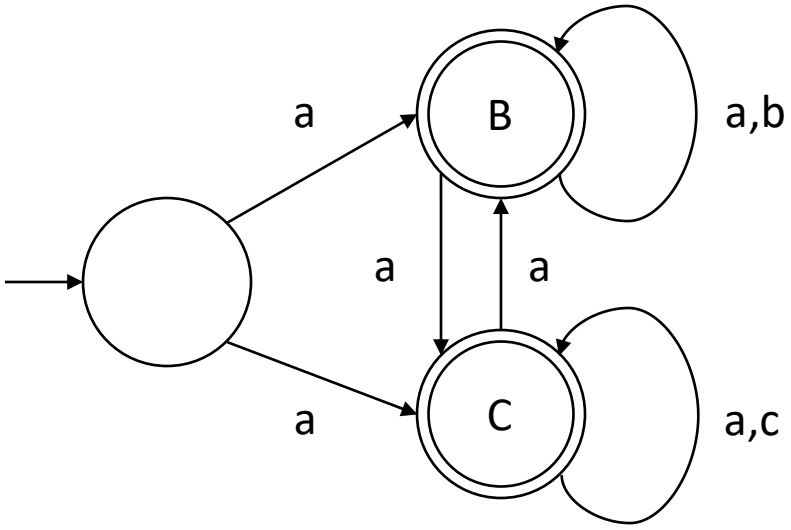
More Nondeterminism?!

- Let's try to turn this into an NFA for (ab*|ac*)*.
- If in state B, and the next input is "a", it could be the start of an "ab*" word OR a "ac*" word. So we could go back to B or go to C.



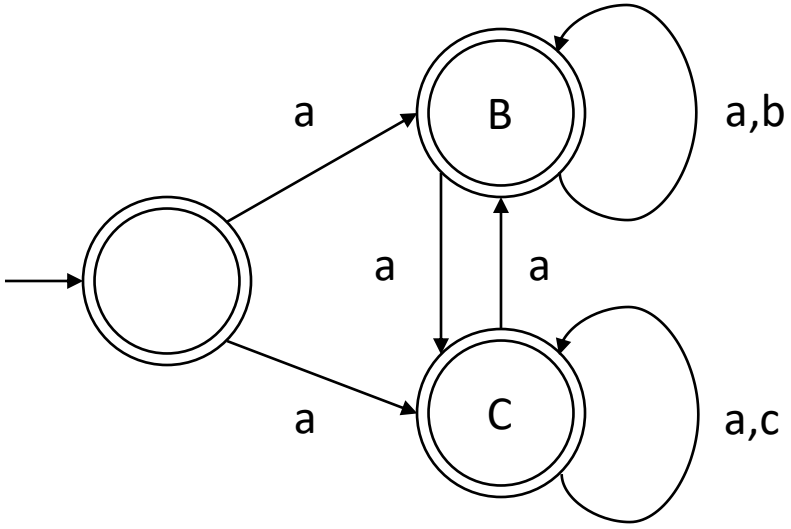
More Nondeterminism?!

- Let's try to turn this into an NFA for (ab*|ac*)*.
- Similarly, if in state C, if we see an "a", we can should either go to B or stay in C.



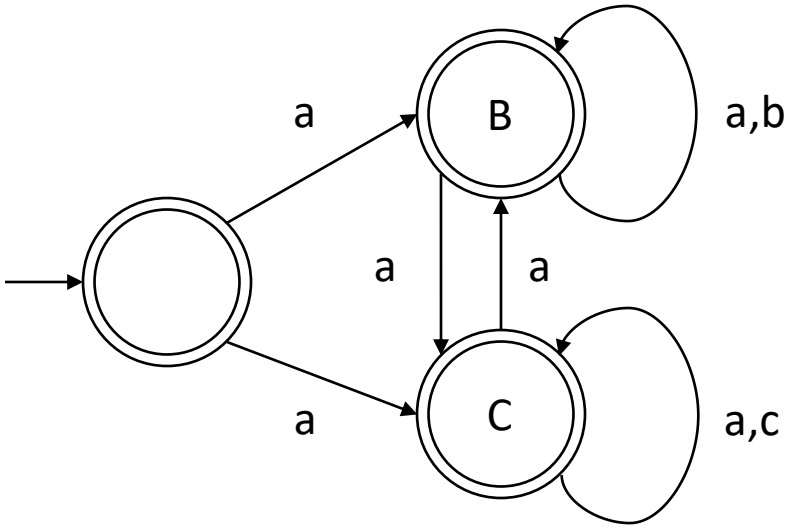
More Nondeterminism?!

- Let's try to turn this into an NFA for (ab*|ac*)*.
- Because the star of a language always contains the empty string, we need to make the initial state accepting as well.



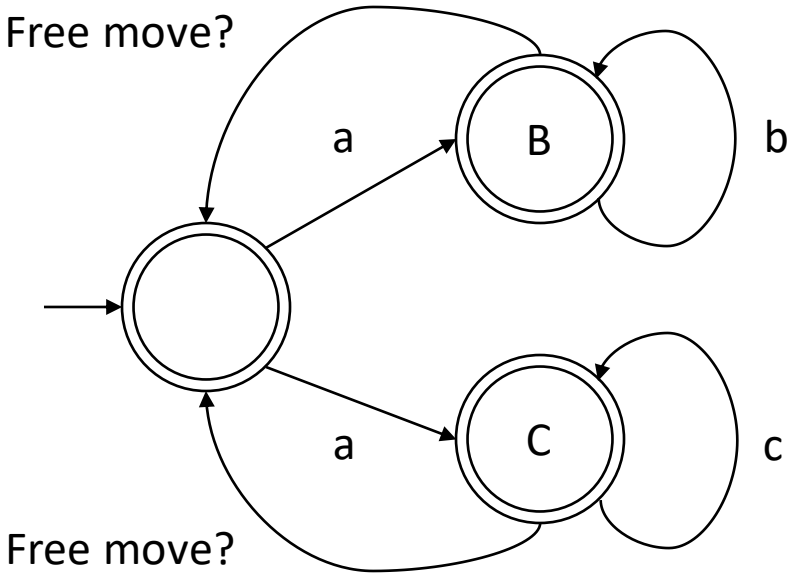
More Nondeterminism?!

- Let's try to turn this into an NFA for (ab*|ac*)*.
- This works (we don't need to add any more transitions) but it's hard to tell that it works and it's a little complicated.



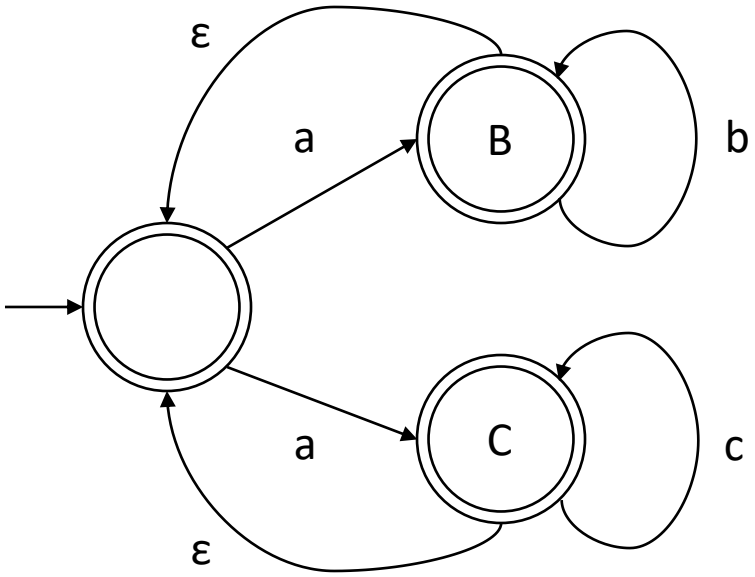
More Nondeterminism?!

- Let's try to turn this into an NFA for (ab*|ac*)*.
- It would be nice if we could express this idea: "Any time we're in an accepting state, we can take a *free move* back to the initial state."



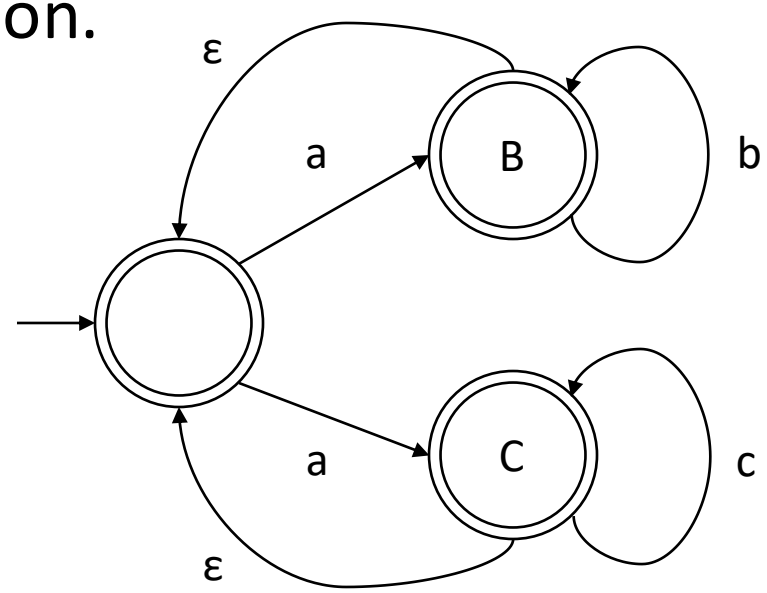
NFAs with ε-transitions

- We will use symbol ε to denote a “free move”.
  - ε (epsilon) is sometimes used as a symbol for the *empty string*.



NFAs with ε-transitions

- An *ε-transition* is a transition that is *optional to take*, and can be taken *without consuming input*.
- This is inherently nondeterministic since we have a choice of whether to take the ε-transition.



Recognition with ε-transitions

- In the "always makes the correct choice" perspective, a string is accepted if there is some possible path to an accepting state.
- An ε-transition is a "free move" you can take without reading a character. These must be considered in your paths.
- In the "makes all possible choices" perspective, you must consider all possible ε-transitions you can take before reading a character.
- Given a set of states S, the **ε-closure** of S is the set of all states you can reach by following a sequence of zero or more ε-transitions from states in S.

NFA Recognition with ε-transitions

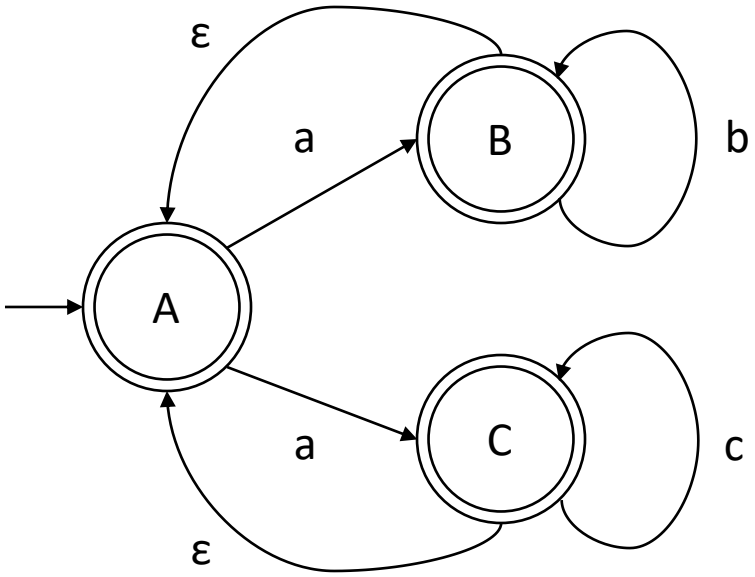
- Let's trace through the NFA recognition process with the NFA below and the string abac.

Input string:
abac

Occupied states:
{A}

Current step:
None

Next step:
Apply ε-closure



NFA Recognition with ε-transitions

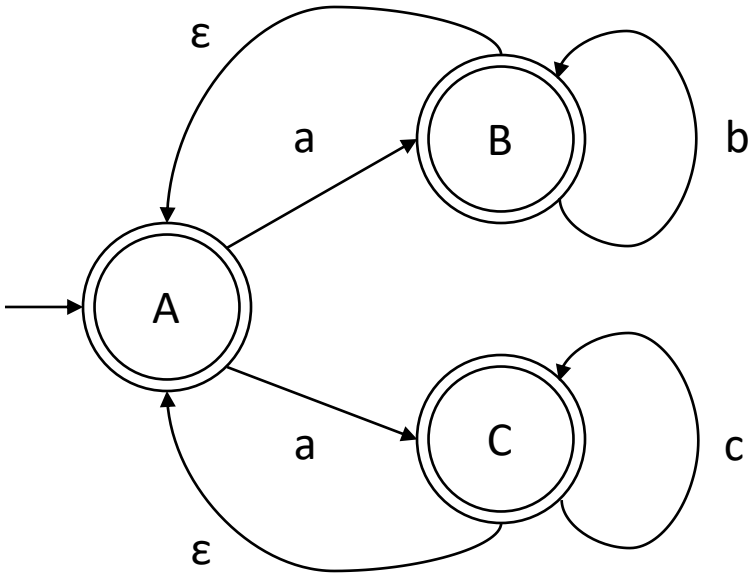
- Let's trace through the NFA recognition process with the NFA below and the string abac.

Input string:
abac

Occupied states:
{A} (no change)

Current step:
Apply ε-closure

Next step:
Read symbol **a**



NFA Recognition with ε-transitions

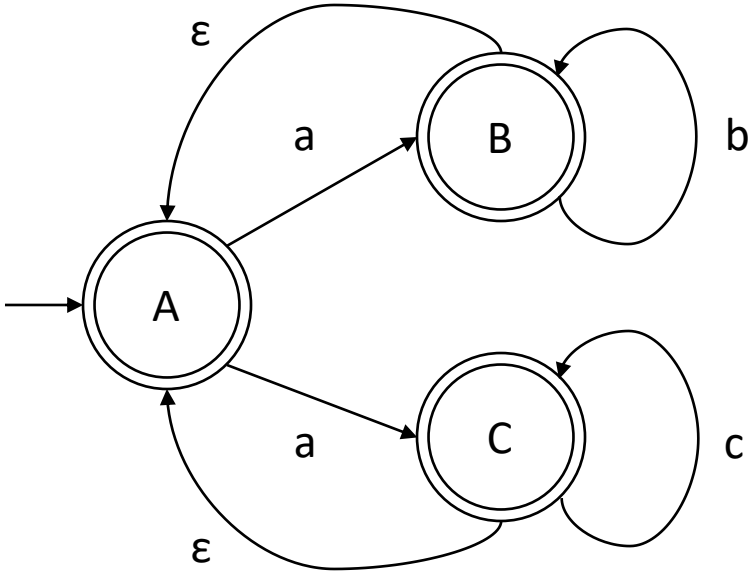
- Let's trace through the NFA recognition process with the NFA below and the string abac.

Input string:
ab**a**c

Occupied states:
{B,C} previously {A}

Current step:
Read symbol a

Next step:
Apply ε-closure



NFA Recognition with ε-transitions

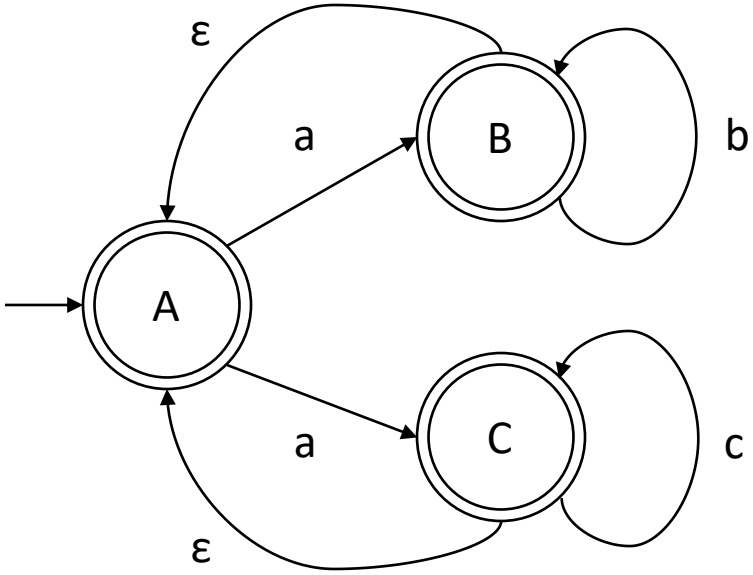
- Let's trace through the NFA recognition process with the NFA below and the string abac.

Input string:
a**b**ac

Occupied states:
{A,B,C} previously {B,C}

Current step:
Apply ε-closure

Next step:
Read symbol **b**



NFA Recognition with ε-transitions

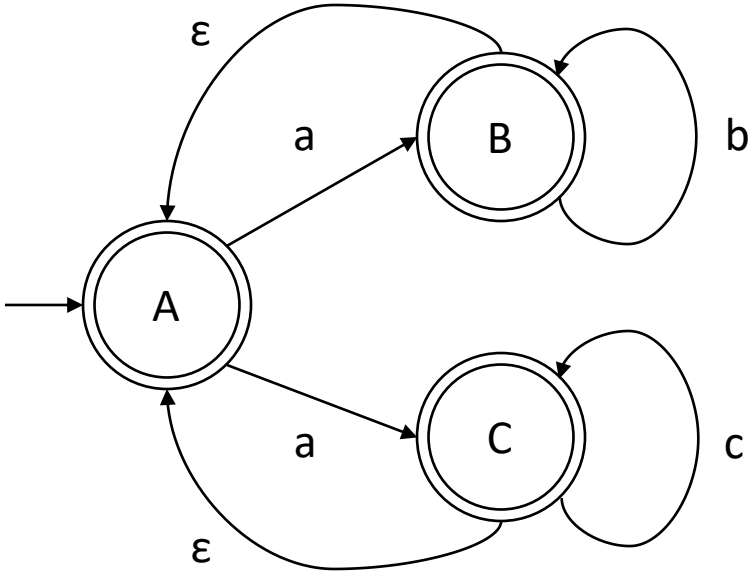
- Let's trace through the NFA recognition process with the NFA below and the string abac.

Input string:
a**b**a**c**

Occupied states:
{B} previously {A,B,C}

Current step:
Read symbol b

Next step:
Apply ε-closure



NFA Recognition with ε-transitions

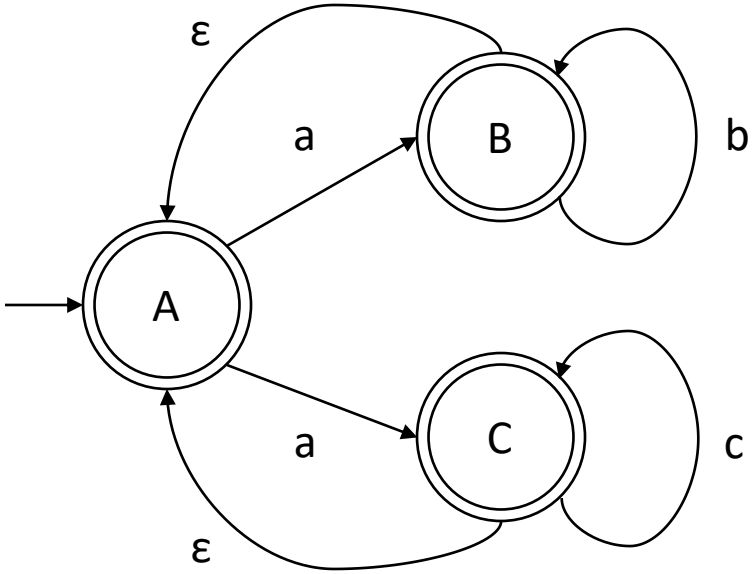
- Let's trace through the NFA recognition process with the NFA below and the string abac.

Input string:
ab**a**c

Occupied states:
{A,B} previously {B}

Current step:
Apply ε-closure

Next step:
Read symbol a



NFA Recognition with ε-transitions

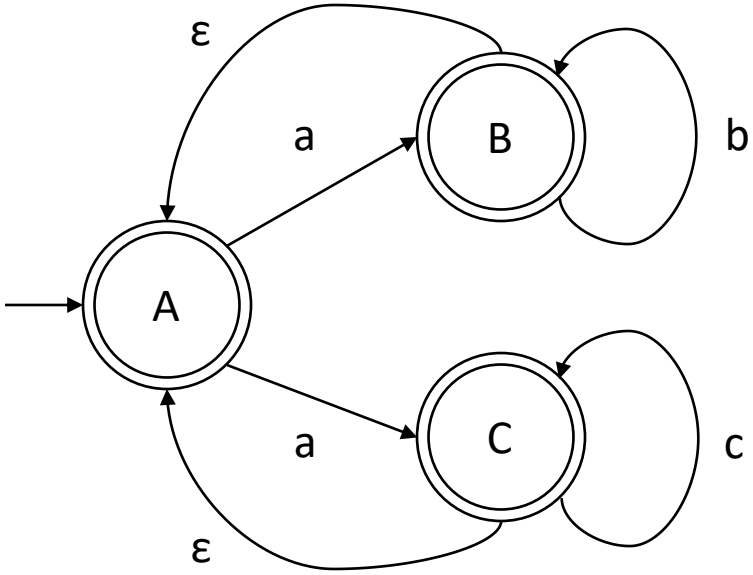
- Let's trace through the NFA recognition process with the NFA below and the string abac.

Input string:
ab**a**<u>**c**</u>

Occupied states:
{B,C} previously {A,B}

Current step:
Read symbol a

Next step:
Apply ε-closure



NFA Recognition with ε-transitions

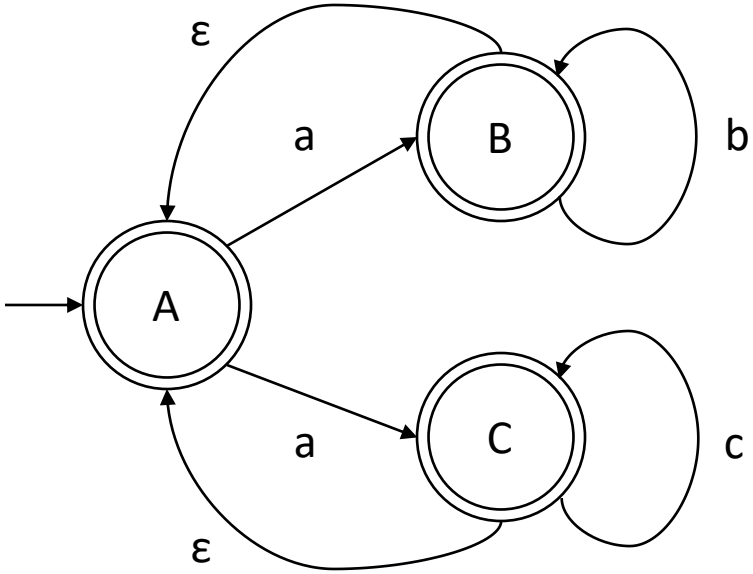
- Let's trace through the NFA recognition process with the NFA below and the string abac.

Input string:
aba**c**

Occupied states:
{A,B,C} previously {B,C}

Current step:
Apply ε-closure

Next step:
Read symbol **c**



NFA Recognition with ε-transitions

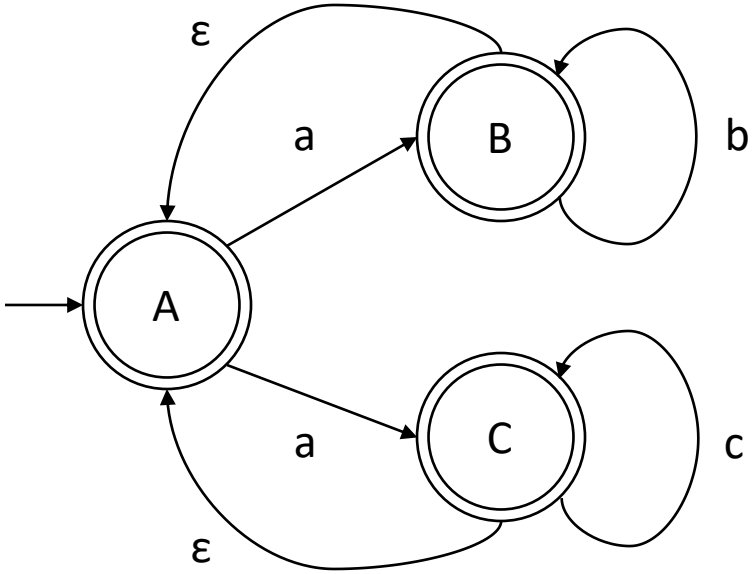
- Let's trace through the NFA recognition process with the NFA below and the string abac.

Input string:
aba**c**

Occupied states:
{C} previously {A,B,C}

Current step:
Read symbol c

Next step:
Apply ε-closure



NFA Recognition with ε-transitions

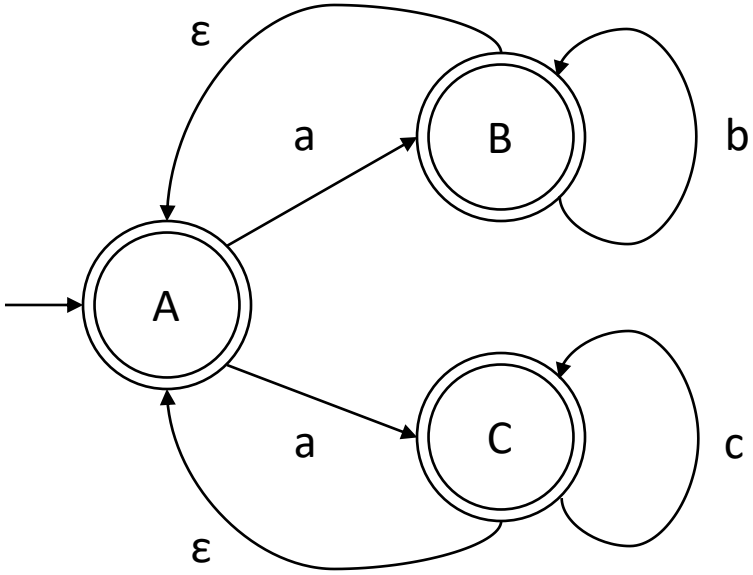
- Let's trace through the NFA recognition process with the NFA below and the string abac.

Input string:
abac

Occupied states:
{A,C} previously {C}

Current step:
Apply ε-closure

Next step:
Accept or reject



NFA Recognition with ε-transitions

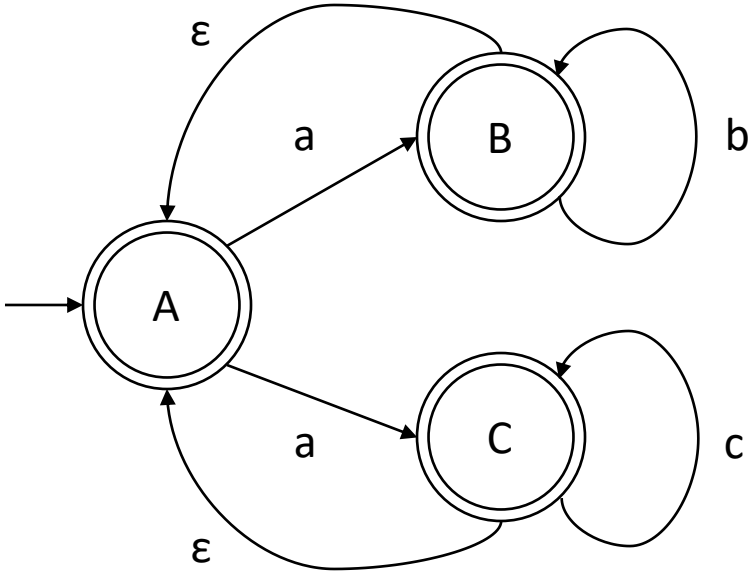
- Let's trace through the NFA recognition process with the NFA below and the string abac.

Input string:
abac

Occupied states:
{A,C}

Current step:
Accept or reject

We **accept** because **{A,C}** contains two accepting states.



The Recognition Power of NFAs

- NFAs give us extra convenience compared to DFAs, and ε-transitions add even more convenience.
- But can NFAs actually recognize any languages that DFAs cannot?
- We originally introduced DFAs as an alternative to regular expressions for specifying regular languages.
- It turns out that DFAs, NFAs (with ε-transitions) and regular expressions are all **equivalent in recognition power** in the sense that they all specify precisely the class of regular languages.
- We won't see a full proof of this but we'll see some of the ideas.