# **Section 11.2.1 DFAs (Deterministic Finite Automata)**

## **Background**

Let the alphabet A = {a, b}.

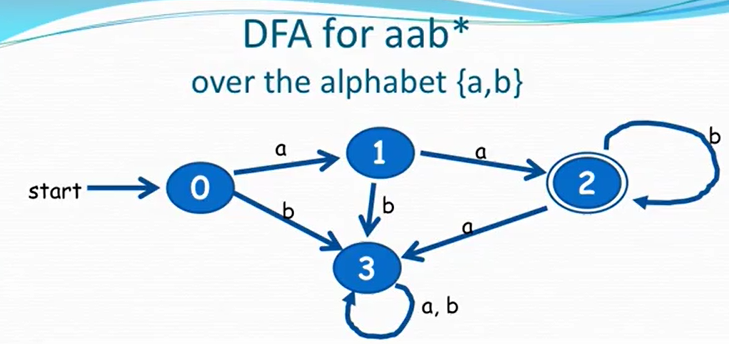
One regular language over A = {aa}{b}\*

Problem: Find a regular expression for {aa}{b}\*: aab\*

List some strings that are in aab\*: {aa, aab, aabb, aabbb, aabbbb, aabbbbb, …}

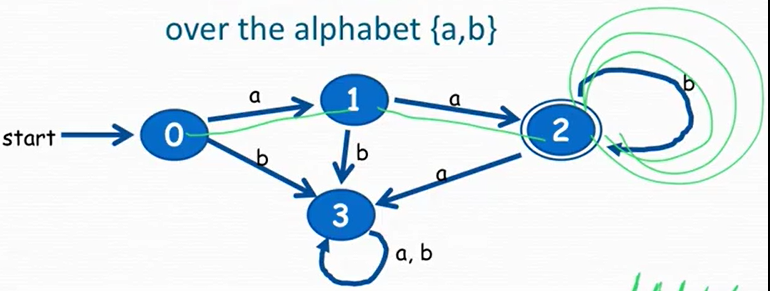
List some strings that are NOT in aab\*: {ba, bba, abb, b, ^}

**An example of a DFA**



Let’s try a string that is part of the regular language, aabbb.

* You go to the start (there’s only ONE start state for a DFA).
* See that the first symbol in the string is an a, so from state 0, follow the a arrow (“transition”) to state 1.
* The second symbol is an a, so from state 1, go to state 2.
* The third symbol is a b, so from state 2, stay at state 2.
* Do the loop again on state 2 (since there are many b’s.

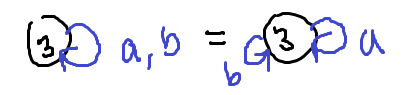


* Look at the state where you’re located at after going through the symbols in the string.
* If that state has double circles, that string is in the regular language.
* If that state doesn’t have double circles, that string is NOT in the regular language.

Let’s try the aaa string.

* First a, go from state 0 to state 1.
* Second a, go from state 1 to state 2.
* Third a, go form state 2 to state 3.
* State 3 does NOT have the double circle; thus it is NOT in the regular language.

## **Informal Definition of DFA over ɑβ A**



(An example is shown in the previous section.)

It’s a finite directed graph.

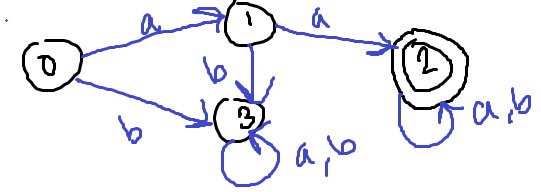
* Each node (state) emits one labeled edge (arrow or transition) for each symbol in A
* The graph has *one* special “start state”
* The graph has a *set* of “final states” (can have 0 to *n* nodes, but NOT infinite)
* Final states have the double circle
* If the DFA has no final states, the only language it accepts is the empty set ∅
* If the DFA has all final states, the language it accepts is A\*
* The “trash state” is the state you’ll never leave (Dr. Kay lingo) (example: state 3)

**Another example of DFA over alphabet {a, b}**

Problem: Change the DFA for aab\* (shown in the previous example) to be the DFA for the regular expression aa(a + b)\*.

The strings included in this regular expression: {aa, aaa, aab, aaaa, aaab, aaba, aabb, …}

* Notice that state 3 is a trash state
* You can have either a’s or b’s in the strings following aa
* You don’t want the a “outgoing” arrow going to the trash state since it’s included
* So the a gets added to the transition with b at state 2, making the new DFA like so:



Singular: Deterministic finite automaton

Plural: Deterministic finite automata

**Theorem**

The regular languages are exactly the same as the languages accepted by DFAs.

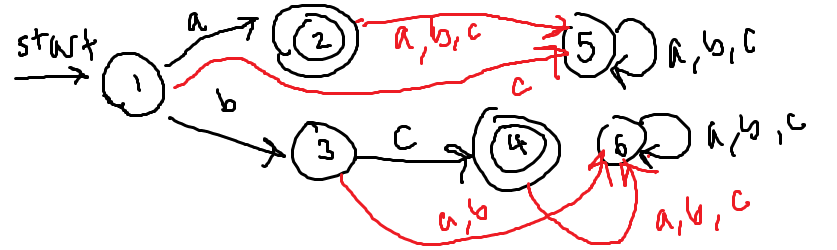
There IS an algorithm to turn a regular expression into a DFA, and the other way around too.

Remember, when creating DFAs, you NEED an out transition for every letter of the alphabet.

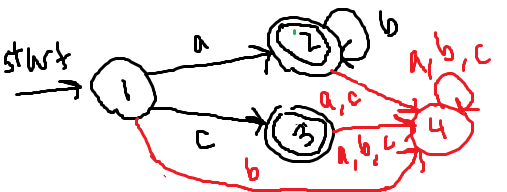
**Problem from the book #2: pg 767:** Use your wits to construct a DFA for each of the following regular expressions given that the alphabet is {a, b, c}: a + bc, ab\* + c, a\*bc\* + ac.

First, let’s write out some strings that are a part of the regular language so you have an idea of what to make: {a, bc}. Then, draw the start and all the final states appropriate for the language (in this case, the a, then the b, c sequence). Lastly, make a “trash state” for the out transitions of other letters in the alphabet. You can have multiple trash states for organization purposes (there is no limit to these), so you can draw DFAs in multiple ways.

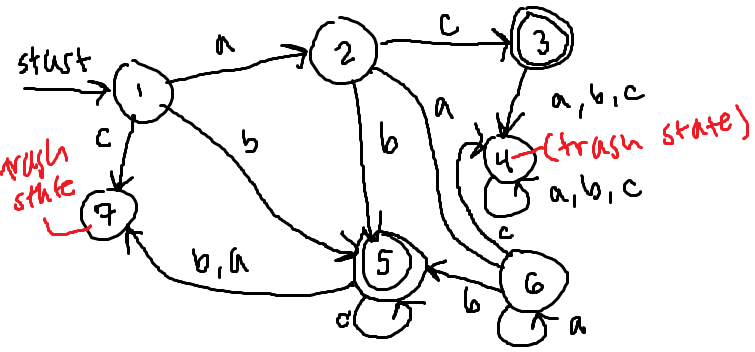
To check, make sure you check for what’s said in red in the text above, then test some strings that are in or not in the language. The red transitions are the transitions that will lead to a trash state, for better distinction. The 1 is used in the beginning so that the circle does not look double (can use any number for these).



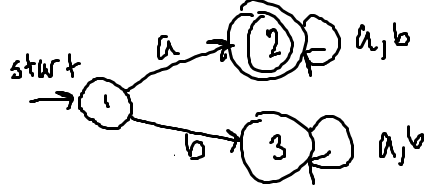
Let’s write out some strings in the language: {a, ab, abb, abbb, …}∪{c}. Then, make all the possibilities of these things first, then the other things for the trash state.



Let’s write out some strings in the language: {b, ab, bc, aab, abc, bcc, …}∪{ac}. You’re allowed exactly ONE out transition for each letter of the alphabet. It’s best to look at the video for this one.

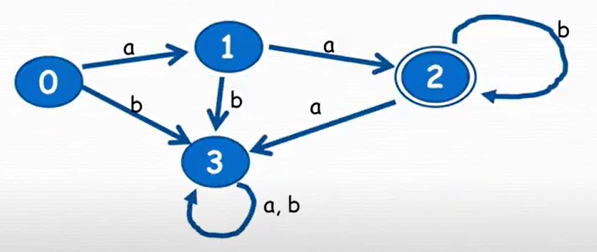


**Problem from the book:** Find a regular expression that accepts the same language as this DFA:

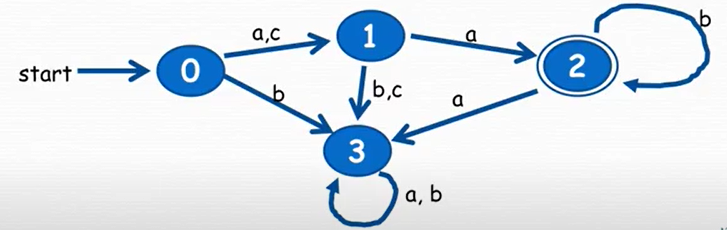


Notice that there is one trash state: state 3. This only occurs when the first symbol of the string is b. The final state, state 2, implies that the string must start with the symbol a and have any number of a’s and b’s after it. The valid strings can include {a, aa, ab, aaa, aab, aba, abb, …}. As a regular expression, this is a(a + b)\*.

**Problem:** Why aren’t these a DFA? (After every image will be the answer.)

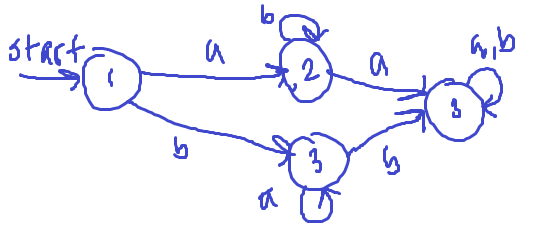
z

It’s missing a start arrow at the beginning. If the alphabet included more symbols, then this would be wrong as well because there wouldn’t be an out transition for every symbol.



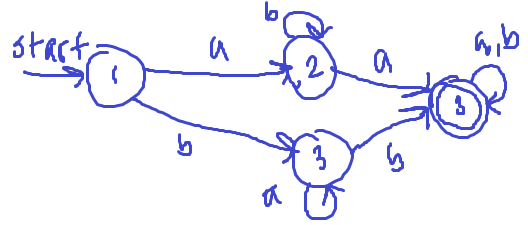
For states 2 and 3, there is no out transition for the symbol c, assuming the alphabet is over {a, b, c}.

**Problem:** Is this a DFA over the alphabet {a, b}? If so, what does it accept? If not, why not?



This is a DFA over the alphabet {a, b} since it has out transitions for each symbol; however, it only accepts the empty set ∅ since there’s no final state (double circle).

**Problem:** Find a regular expression that accepts the same language as this DFA:



Some strings that are valid according to this DFA include: {aaa, bba, aab, bbb, abaab, babab}. The top half of the DFA can represent the regular expression ab\*a(a + b)\*, while the bottom half can represent ba\*b(a + b)\*. Since they’re both valid, you can union them like so: ab\*a(a + b)\* + ba\*b(a + b)\*.

What is the alphabet of the same DFA? It’s {a, b}.

If you were given a regular expression, you may not be able to tell because maybe a symbol from the alphabet was not included in the language the expression represents. However, for a DFA, it is certain since there MUST be an out transition from every state for every symbol in the alphabet.

**Problem:** Are each of these legal DFAs over the alphabet {a, b}?



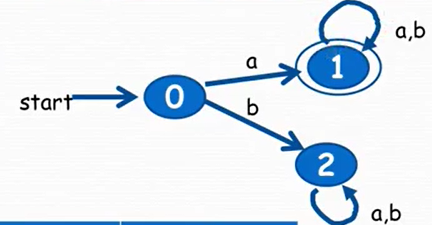
Yes. There is nothing wrong with labeling the states differently. The first one’s regular expression is (a + b)\*, and the second’s is the empty set ∅ (no final state). The first one even accepts lambda because even if you start with an empty string, it’s in the final state so it’s acceptable.

## **Becoming more formal about DFAs**

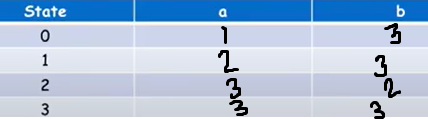
The edge is called a “state transition” in a DFA (see right):

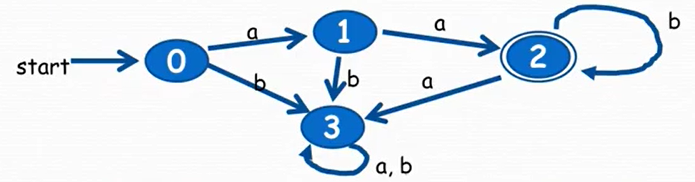
You can describe this as a function: T(i, a) = j, where “T” means transition.

You can also make a table (full state transition) since it’s tiring to write all the functions, where each value in the table is the state of its transition from the leftmost column:



**Problem:** Give the transition table for the following DFA (answer shown right):





If you have the transition function for a DFA, can you reconstruct the DFA? No because firstly, you don’t know which state you’re starting from. Secondly, you don’t know which states are final (“accept” states).

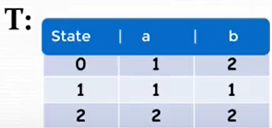
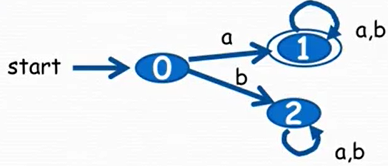
When a DFA runs, every step “consumes” a letter in your string.

**Formal Description of a DFA**

You need to explicitly state ALL 5 different pieces of information:

* Transition table
* SET of states (the leftmost column on the table)
* Alphabet (the topmost row on the table, excluding “State”)
* Start state
* SET of final states

An example of the formal description of a DFA below (with the transition table beside it):



States = {0, 1, 2} *Notice the curly braces! This is a SET!*

ɑβ = {a, b} *An alphabet is a set of symbols!*

Transition function = T *Notice that this is the table!*

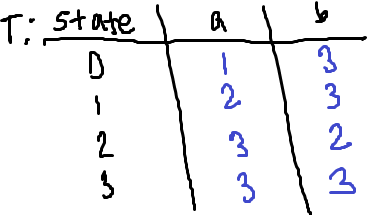
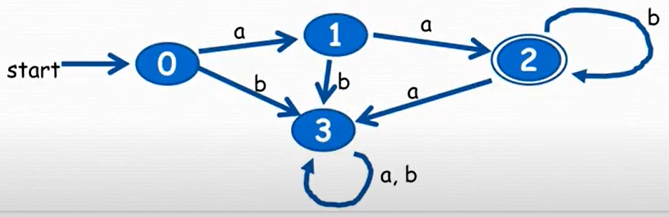
Start = 0 *The start state!*

Final = {1} *Notice the curly braces! This is a SET!*

You can show this in the form of a 5-tuple in the form (states, ɑβ, transition fn, start, final). This tuple is a LIST and NOT a set, so the *order matters*!! You can remember this as SATSF. T is always the third element in the tuple (so you can just memorize what’s to the left and right of it).

For example, using the same example above, you would have ({0, 1, 2}, {a, b}, T, 0, {1}).

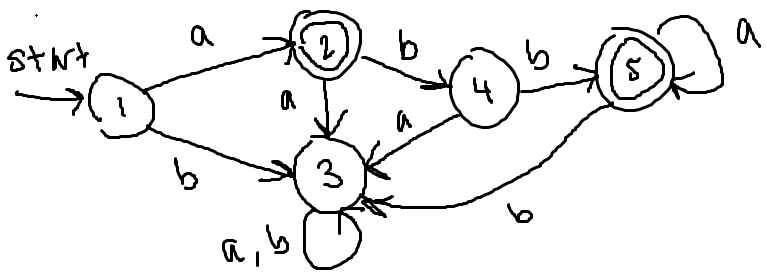
**Problem:** Formally describe the following DFA as a 5-tuple:

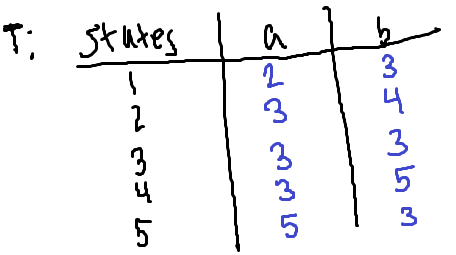


({0, 1, 2, 3}, {a, b}, T, 0, {2})

**Problem:** Formally describe the regular expression a + abba\* as a 5-tuple representing a DFA:

The process for this would be to draw the DFA, then write out the tuple according to it. Thus, the strings included in this language would be {a}∪{abb, abba, abbaa, abbaaa, …}, assuming that the alphabet is over {a, b}.

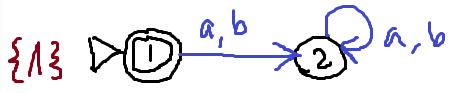


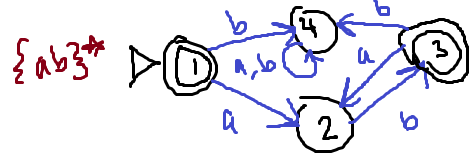
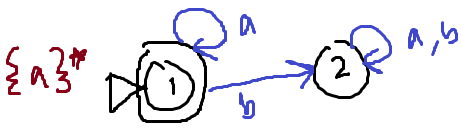


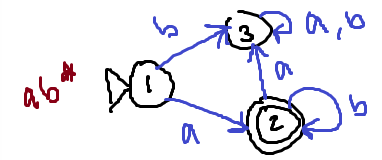
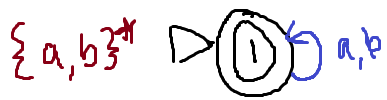
The tuple therefore is ({1, 2, 3, 4, 5}, {a, b}, T, 1, {2, 5}).

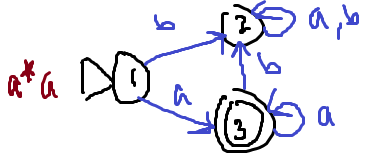
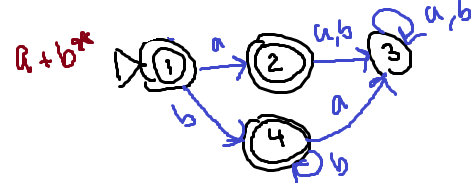
**Last Minute DFA Practice:** Create the following DFAs (all over the alphabet {a, b}) in order:

* (Do each one individually) Recognizes the languages ∅, {^}, {a}\*, {ab}\*, {a, b}\*
* (Do each one individually) Recognizes ab\*, (ab)\*, a + b\*, (a + b)\*, a\*a



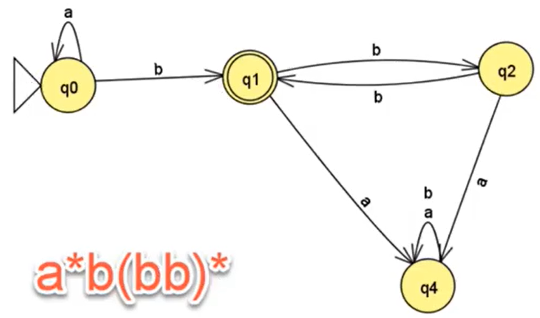




## **JFLAP Concerning DFAs**

**Creating DFAs and Processing Strings**

Example DFA:



Select “Finite Automaton”.

* First, you need to make states, so click on this icon (state creator)  and use your left mouse button to click, and it’ll create states.
  + You can change their sizes by using the “Automaton Size” slider at the bottom
* To pick a start state, click this icon (attribute editor)  . Then, right click on the start state and select “Initial”.
  + It won’t allow you to have more than one start state.
  + To make a final state, right click on the final state(s) and select “Final”
  + You can also use this to move around states
  + You can also use this to curve transitions (click on them, drag the circle)
  + You can change the name of a state by right clicking and typing the name
  + You can change the label of a state by right clicking and typing the name (like comments in your code)
  + You can change the visibility of these labels by right clicking on the white space
* To make an out transition b from q1 to q2, click this icon (transition creator)  and use the LMB (left mouse button) to click on q1 and drag to q2.
  + A text box will pop up and you can type in the symbol, then click enter.
  + To make a loop transition, click on the state you want a loop on and fill in text.
  + *Make sure to write the transitions separately (if you’re doing b, a, do b, then a)*
* To delete, click this icon (deleter)  and click on the states and transitions.

You can use JFLAP to step through strings. It’s kind of like a debugger!

Click “Input” at the top. You can select “Step by State”, and enter in your desired string.

* A window will appear at the bottom
* If the string entry is bolded, that means nothing has been processed yet
* It displays the state at the top
* You can click “Step” to start stepping through
* The symbol will unbold if that symbol has been processed

In this window, you can click “Trace” with the box of that string selected, and it shows you the step-through of the process with the states and unbold/bolded items.

You can also make a .txt (text) file and “Click to open an input file”. It takes whatever is the first string in that file and lets you step through that.

**Important Warning**

Do NOT enter transitions that have multiple symbols on it (like a, b). Do NOT put commas in your transitions. You won’t be able to progress further than that when you’re tracing. Make individual transitions for each symbol if multiple are required.

**Processing Multiple Strings**

Select “Input”, then “Multiple Run”.

Now, you can type a whole bunch of inputs in the text boxes in the right window. Then, you can select “Run Inputs” to run every input. You will get the result as either “reject” or “accept”.

If it’s saying that it’s blank, it’s the empty string.

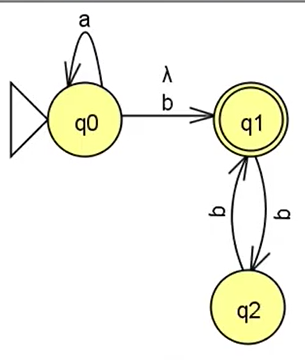
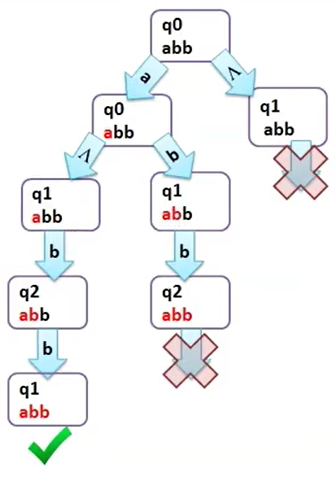
You can also select “Load Inputs” in that menu and load in a .txt (text) file. Make sure to remove the current inputs first if you don’t want them in your processing.

In the text file, separate the strings with white spaces (a new line counts too).

**How to add trash state to DFA**

Click on “Convert” at the top and select “Add Trap State to DFA”. Use the “state creator” tool and make a state. Next, click “Do All”. It makes a new file in a new window, and transitions are added to make it a DFA.

**NFAs Step by State**



You can step through a string similarly using “Input”, then “Step by State”. You can then use the “Step” button to go through each state, and the tracing will be shown in the bottom window.

If you click “Trace”, you can see a window that goes through each one and shows you the path that was taken for that string.

JFLAP takes into account the different paths that can be taken.

**Step by Closure**

Essentially, at every step, you ask if you can take any lambdas. If you can, you do. If you fail, you go back and take another path from there.

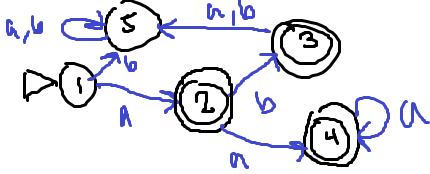
# 

# **Section 11.2.2 NFAs (Nondeterministic Finite Automata)**

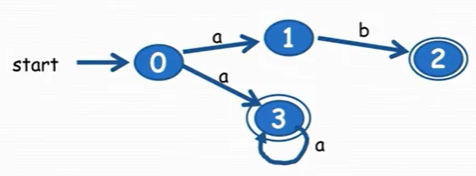
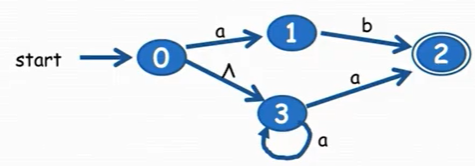
NFAs are like DFAs *but*:

* You can have 0, 1, or more out transitions for every symbol in the alphabet
  + If there’s no out transition you have in your string, it’s not in the language
* You can have ^ (lambda) transitions (that you can take whenever you want)
  + It means lambda jump, which means nothing in your string unless it’s directed to an accept state or something
* Sometimes easier to construct an NFA for a given regular expression than a DFA
* Accepts a string if there is ANY path that processes the whole string and then ends up at an accept state
  + For example, it may not accept the loop in the image, but it can use the right transition to get to the accept state

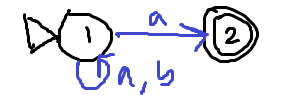
**Example:** Let’s draw a DFA for ab + a\*a so you can see how it compares:

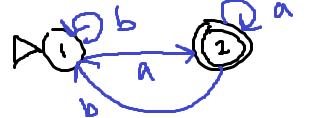


And the following are NFAs for the same regular expression:



**Problem:** So how would you make an NFA for the language (a + b)\*a?

* Of course, put your start state
* You can have as many a’s or b’s as you want
* At some point you’ll want an a, so make a transition to a final state (see right)

**Problem:** How would you make a DFA for (a + b)\*a?

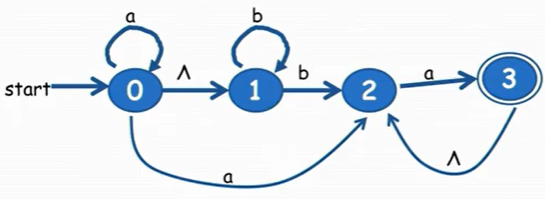
* The first state represents having as many b’s as you want
* The second represent having an a at the end of the string
* You go back to state 1 if you get a b somehow

## **Important Facts to Remember**

* Regular languages are sets of strings
* Every regular language can be represented by a regular expression
* Every regular expression represents a regular language
* Every regular language can have a DFA and NFA built to recognize it
* If a language is recognized by a DFA or NFA, then it’s regular

## **Problems**

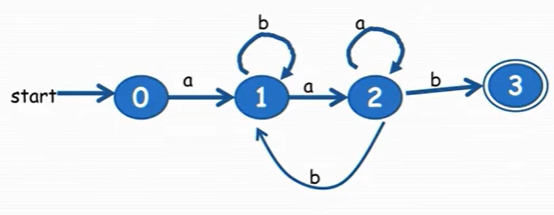
**Question:** Are all DFAs NFAs? Yes because a DFA is an NFA, which follows almost no rules.

**Question:** Are all NFAs DFAs? No because DFAs have a certain set of rules they must follow.

**Question:** Is the following NFA also a DFA? No because there are lambda transitions and states that don’t have out transitions for every symbol in the alphabet.

**Question:** Give a regular expression for the language it accepts.

a\*b\*ba\* + aa\* Notice that you can use the loops or any path to get these. So, part of it is a\*b\*ba due to one path to get to state 3. You could also get aa by going from 0, 2, 3. You could also get a\*b\*b, then go through the lambda transition. You could get to state 2 then 3 an infinite number of times when repeating a. Union these possibilities together to get the answer, but a better way to write this is the colored answer.

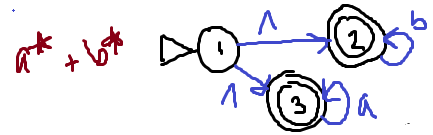
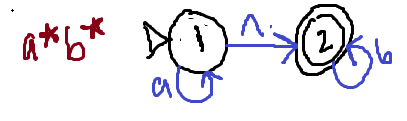


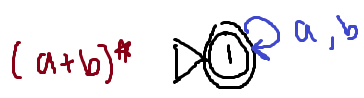
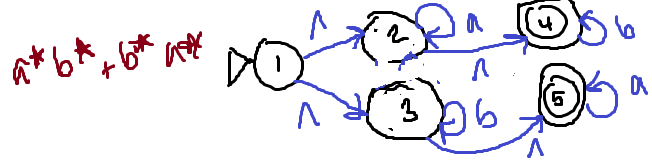
**Question:** Is the following NFA also a DFA? No because state 2 has two out transitions for the symbol b and state 0 has no out transition for b.

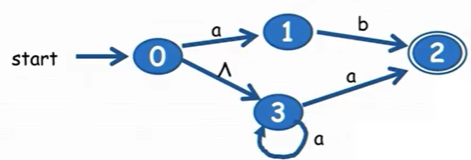
**Question:** Give a regular expression for the language it accepts.

ab\*aa\*(bb\*aa\*)\*b Let’s break it down: The first symbol must always be a and the last must always be b. So, there’s a --- b (the dashes means something is in between). The first sequence the dashes could be is the one without the arrow back at state 2, which is b\*aa\*. The second sequence the dashes could be is the one with the arrow back at state 2 (starting at state 2), which is bb\*aa\*. You can loop that second sequence 0 or more times. Thus, concatenate everything together to get the answer.

**Problem:** Construct an NFA for the following regular expressions: a\*b\*, a\* + b\*. a\*b\* + b\*a\*, (a + b)\*, (a + b)\*a (*this one is done already in the previous page*).







**Problem:** This NFA represents the regular expression ab + a\*a (same as previously).

The strings of this language include ab and 0 or more copies of a concatenated with a. You can write this as {ab}∪{a, aa, aaa, …}. Say you’re processing the string aaaa. You can either guess the path, OR…

* Start at state 0
* Take into account about lambda transitions from the start
* Note you can start from either state 0 or state 3 now, so either {0, 3}
* Process the first symbol in the string (a) and go from those beginnings to valid states
* After processing, record the 3 states you are after the first processing {1, 3, 2}
* Then take into account lambda transitions from those states
* Continue this for each symbol in the string while crossing out processes that don’t work
* The total process for this particular string is {0, 3}, {1, 3, 2}, {3, 2}, {3, 2}, {2}
* If you reach an end, then it is definitely in the NFA

**Question:** Why can’t you use the same formal description of DFA for NFAs? If you have an NFA, you can have 2 or even more ways to leave a state with a single symbol. A transition function (the T table) for DFAs won’t be able to uphold that.

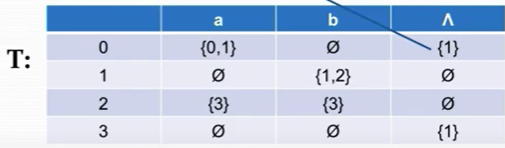
**Formal Description of an NFA**

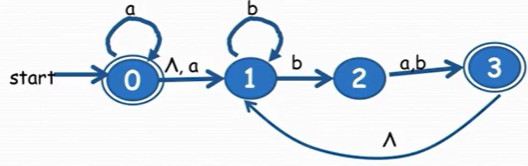
Same as DFA, just represent it as a 5-tuple like so: (states, alphabet, transition fn, start, final).

HOWEVER, the key differences:

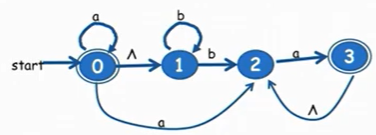
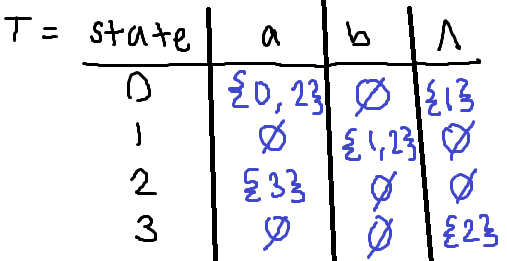
* The alphabet in the tuple does NOT contain ^ (lambda)
* All items in the transition fn (T table) are SETS, even if they contain one element
* The table can contain empty sets (∅) if no states have those symbol transitions
* ^ (lambda) is one of the columns in the table (ALWAYS, even if it’s just empty sets)

An example of an NFA whose tuple is ({0, 1, 2, 3}, {a, b}, T, 0, {0, 3}) and its table:





**Problem from the book #4: pg 768:** Write down the transition function for the following NFA (left picture):



(*Another problem on the next page.*)

**Problem:** Give a formal definition (5-tuple) for the NFA: ({0, 1, 2}, {a, c}, T, 0, {1})

