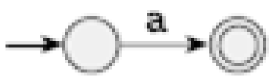
Finite Automata Vs RE

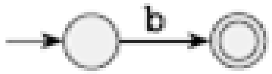
Example 1: Regular Expressions to NFA

- Find NFA for (ab ∪ a)*
- Start with NFAs for strings of just a and b
- Concatenate NFAs with ε to get ab
- Next union with new start state
- Lastly star previous NFA by connecting accept states to start state.

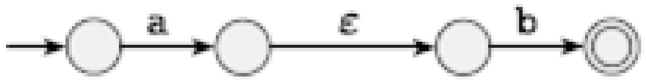
a



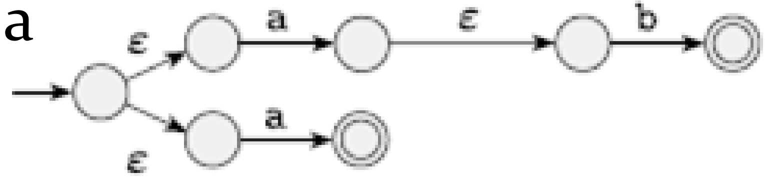
b



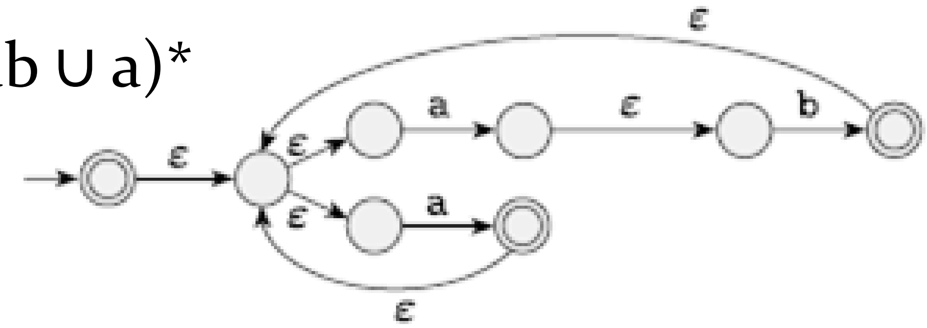
ab



ab∪a



(ab ∪ a)*



Example 2: Regular Expressions to NFA

- Find NFA for (a ∪ b)*aba
- Start with NFAs for a and b
- Union with new start state
- Star by connecting accepts with start state
- Concatenate multiple times for aba
- Concatenate from all accept states to aba

a

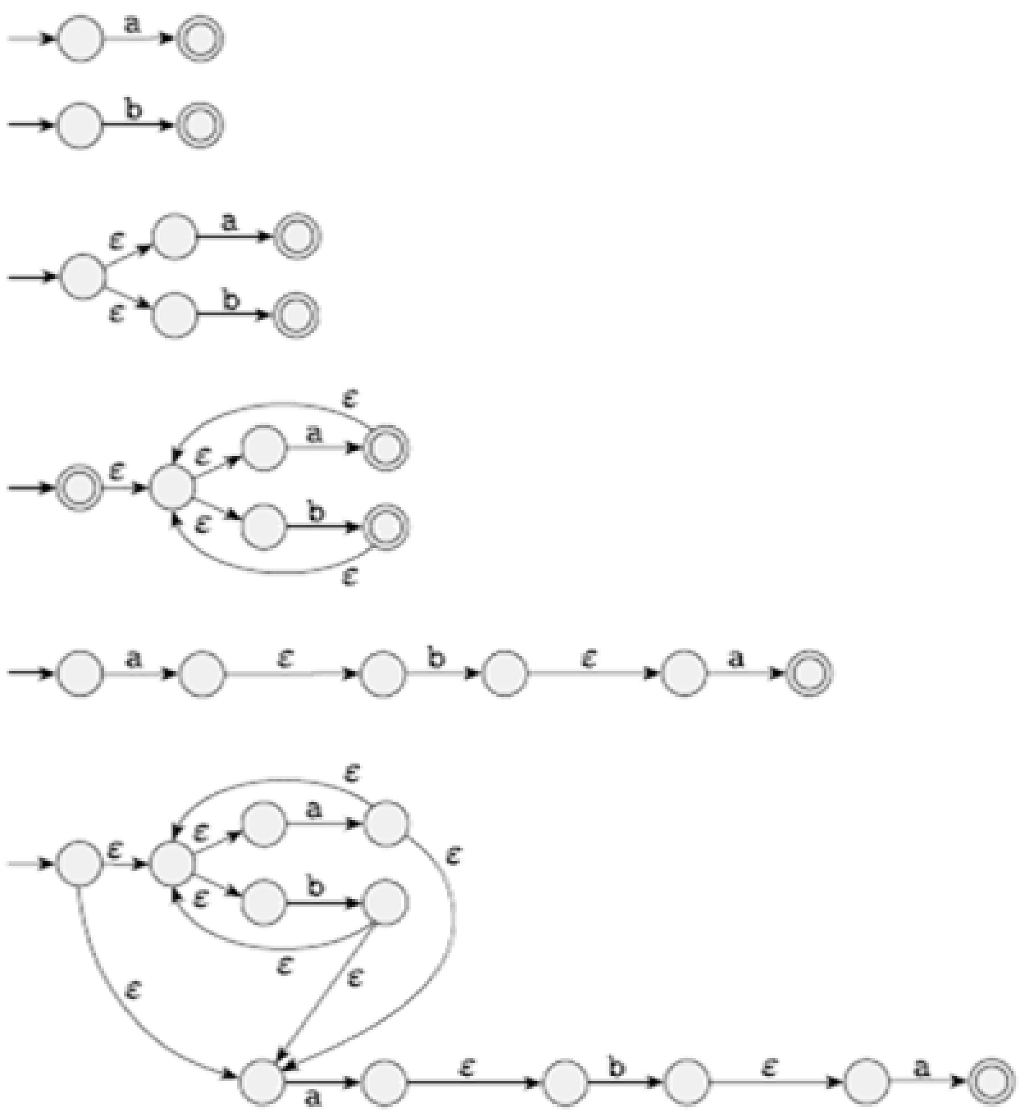
b

a∪b

(a ∪ b)*

aba

(a ∪ b)*aba



Theorem

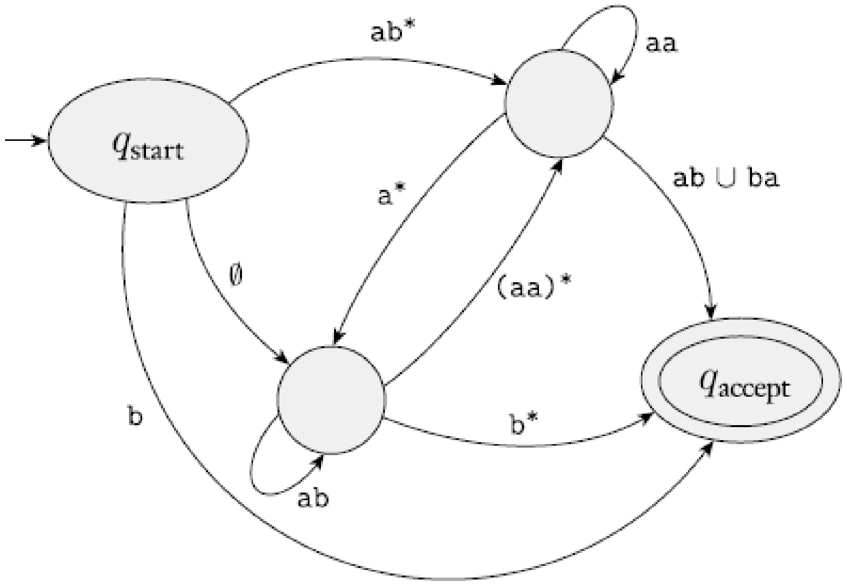
- Theorem: If L is a regular language, then there is a regular expression R with L = L(R).
  - Theorem shows relationship from opposite direction
  - Allows a finite automate to be converted to a regular expression

- Generalized nondeterministic finite automaton (GNFA)
  - NFAs with any regular expressions as transition arrows instead of just the alphabet and ε
    - $\delta(q_i,q_j) = R$
  - Can read a block of symbols instead of just individual symbols

- Formal definition changes from NFA
  - $q_o = q_{start}$, $q_k = q_{accept}$,$q_{start} \neq q_{accept}$
  - For every pair of states starting from q_{start} to q_{accept} we get a regular expression
    - R = set of all regular expressions over the alphabet
    - Regular expressions can be combined

GNFA Restrictions

- For convenience, require GNFAs to always have the following conditions
  1. Start state has transition arrows going to every other state
     1. but no arrows coming in from any other state.
  2. Only a single accept state
     1. Arrows from every other state
     2. No arrows going to any other state
     3. Must be different from start state. (not a single state FA)
  3. All other states must be arrows going to each other
     1. Must also have a loop to itself.

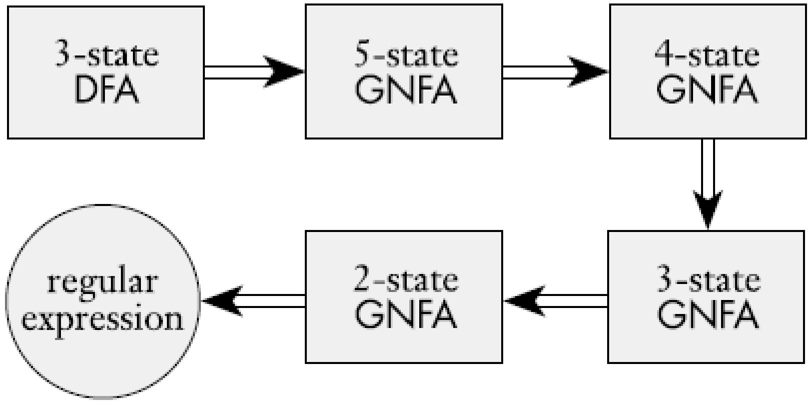


Formal Definition for GNFA

- A GNFA can be formally defined as a 5-tuple $(Q,\Sigma,\delta,q_{start},q_{accept})$, where:
  - Q is a finite set of states
  - Σ is a finite set (alphabet) of input symbols
  - $\delta: (Q-\{q_{accept}\}) \times (Q-\{q_{start}\}) \rightarrow \mathscr{R}$ is the **transition function**
    - $\mathscr{R}$ = collection of all regular expressions over the alphabet Σ
    - Transition any state except accept state to any state except start state is made by any regular expression
  - q_{start}, is the start state
  - q_{accept}, is the accept state

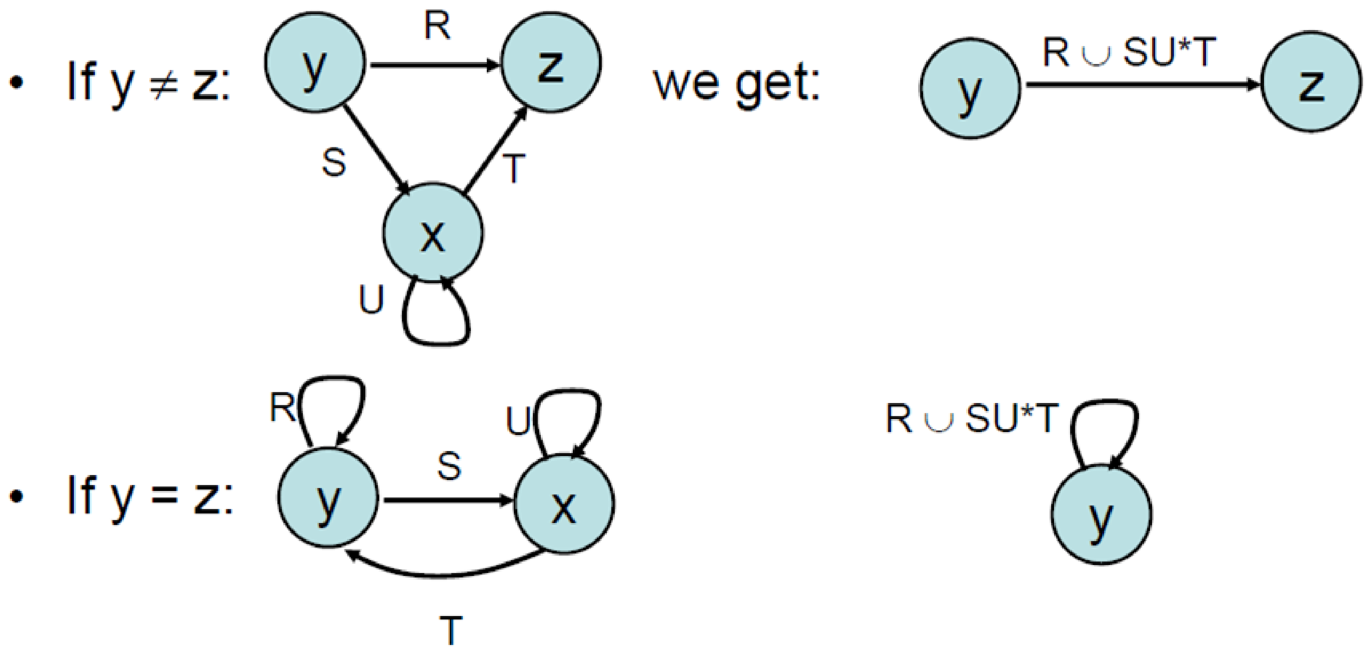
Convert DFA to GNFA to Regular Expression

- DFA to GNFA
  - Add a new start state with an ε arrow to the old start state
  - Add a new accept state with an ε arrow from the old accept state
  - Replace arrows with multiple labels or multiple directed arrows between the same nodes with a single arrow labelled with the union of the previous labels
  - Add arrows labelled with $\emptyset$ between states without arrows.
    - Does not change language because $\emptyset$ can never be used.
- Reduce k-state GNFA to k-1 states
  - Repeat until k = 2
    - Single arrow from start state to accept state
- Transition arrow label is the **regular expression**.



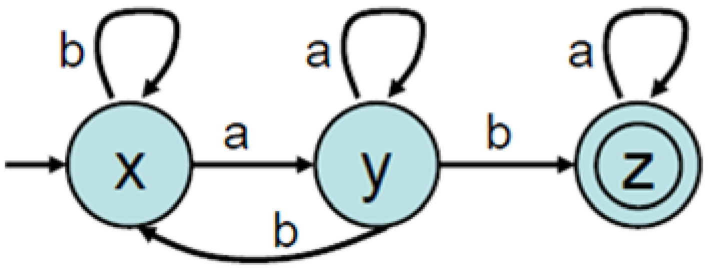
Reducing Number of States for GNFA

- Select a state to remove that is <u>not</u> the q_{start} or q_{accept}
  - Remove state and consolidate transition arrows pass through removed state
  - Combine regular expressions of consolidated transition arrows
- To remove a state x, consider every pair of other states, y and z, including y=z
- New label for edge (y,z) is the union of two expressions:
  - What was there before, and
  - One for paths through (just) x

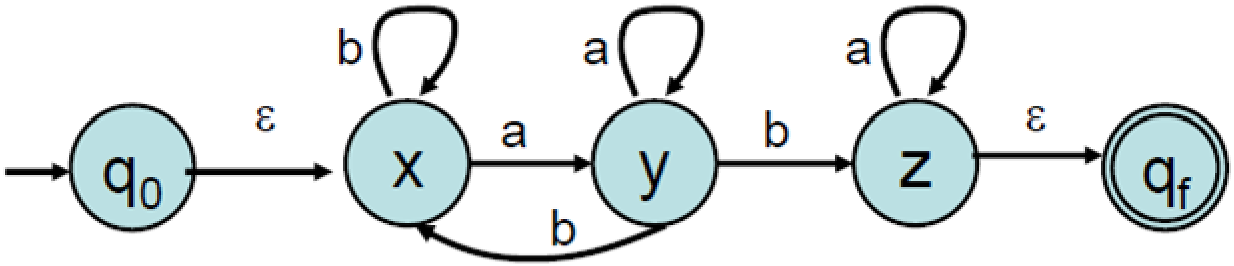


Proof of Theorem

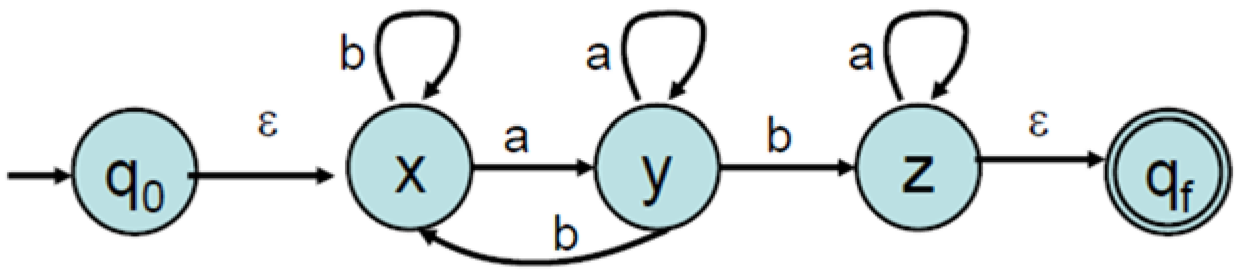
- Theorem: If L is a regular language, then there is a regular expression R with L = L(R)
- Proof
  - For each NFA M, define a regular expression R with L(R)=L(M)
    - Show with an example:



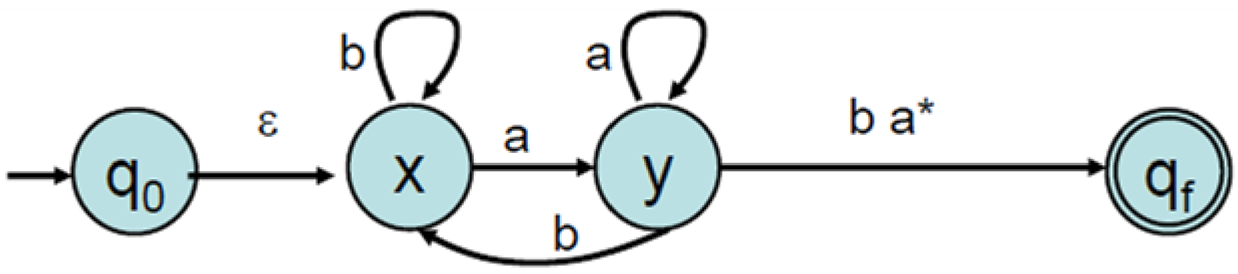
  - Convert to a special form with only one final state, no incoming arrows to start state, no outgoing arrows from final state



Proof of Theorem Continued

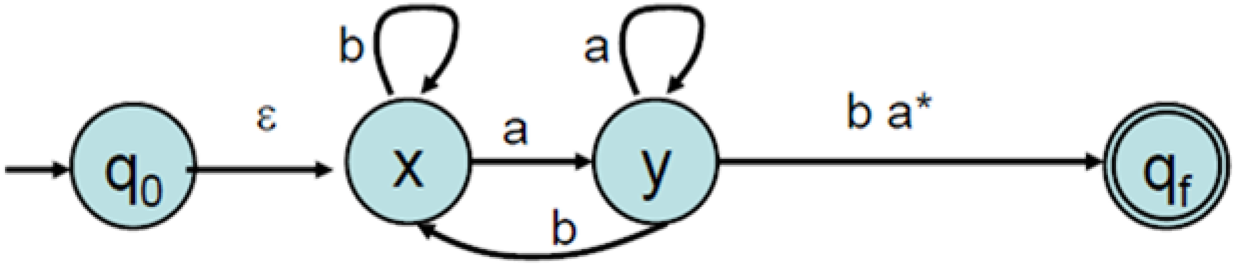


- Now remove states one at a time (any order), replacing labels of edges with more complicated regular expressions
- First remove z:

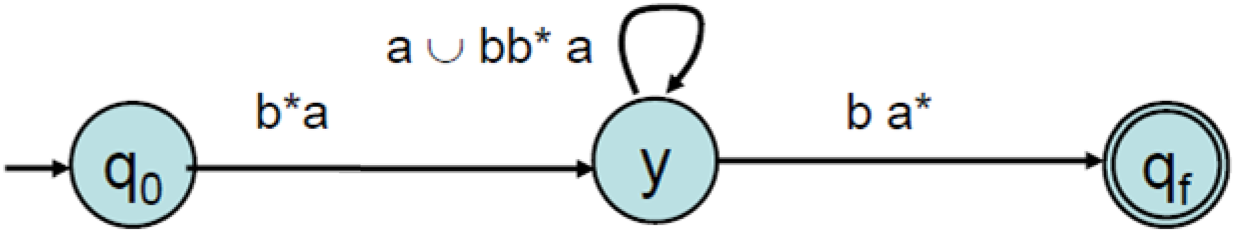


- New label ba* describes all strings that can move the machine from state y to state qf, visiting (just) z any number of times

Proof of Theorem Continued



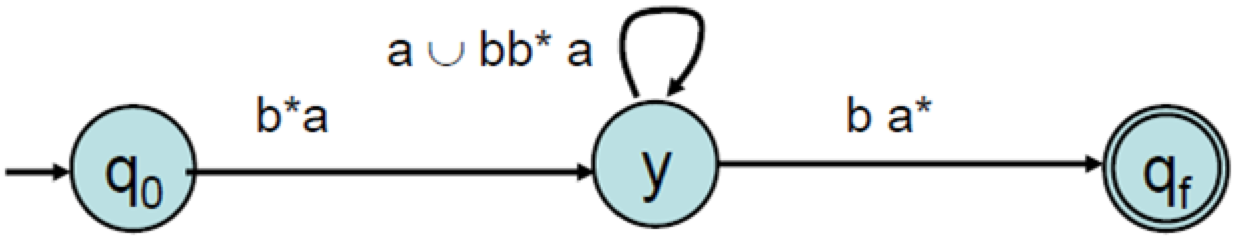
- Next remove x:



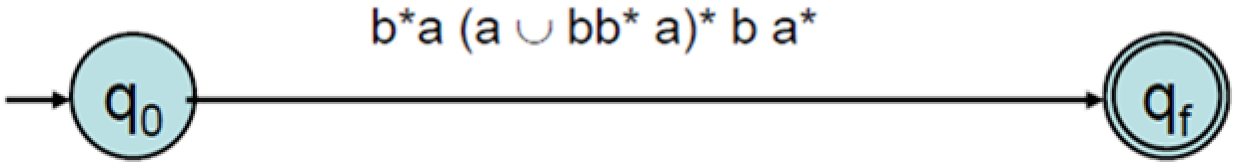
- New label b*a describes all strings that can move the machine from qo to y, visiting (just) x any number of times
- New label a ∪ bb*a describes all strings that can move the machine from y to y, visiting (just) x any number of times

Proof of Theorem Continued

- Last, remove y:

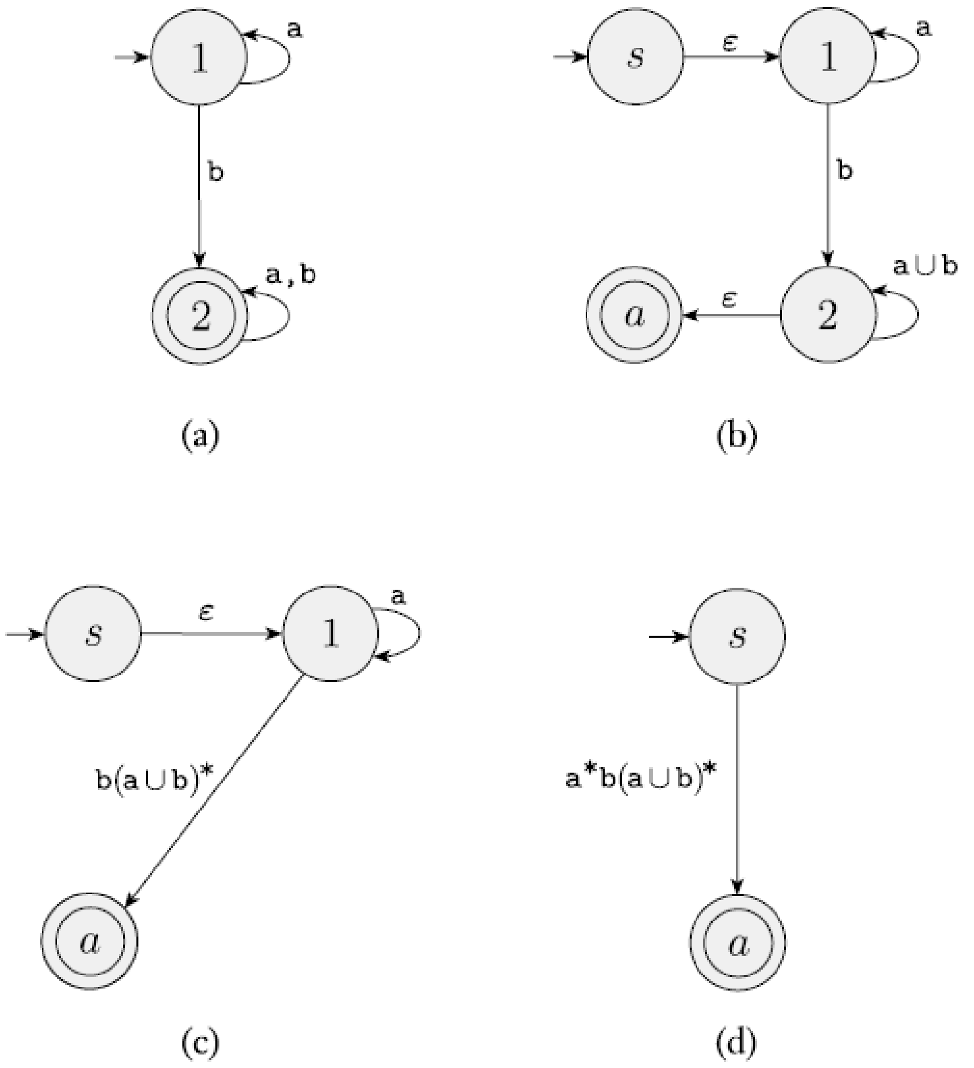


- New label describes all strings that can move the machine from qo to qf, visiting (just) y any number of times
- This final label is the equivalent regular expression



Example: 2 State DFA to Regular Expression

- Add new states
- Remove state original states one at a time
  - Example removes 2 then 1



Example: 3 State DFA to Regular Expression

- Add new states
- Remove state 1
- Remove state 2
- Remove state 3