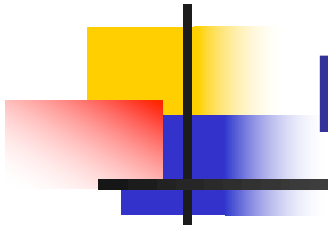
DFA to Regular Expression Conversion

Featured Event

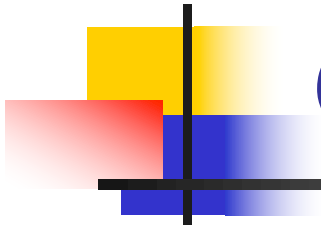


Reg Exp Equivalence with Finite Automata

- Regular expressions and finite automata are equivalent in their descriptive power
- This is surprising because FA and regular expressions outwardly are different
- However, any regular expression can be converted into an automata that recognizes the languages it describes, and vice versa

Theorem:

A language is regular if and only if some regular expression describes it.



One direction of the theorem

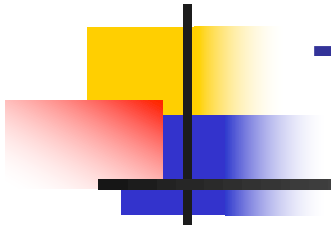
Lemma 1:

If a language is described by a regular expression, then it is regular.

Proof Idea:

Say that we have a regular expression R describing some language A. We show how to convert R into an NFA recognizing A. We know if an NFA recognizes A then A is regular.

DONE!



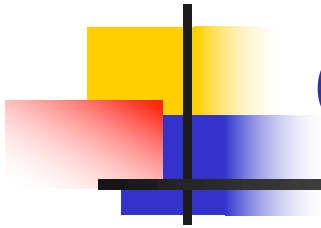
The other direction of the theorem

Lemma 2:

If a language is regular, then it is described by a regular expression.

Proof Idea:

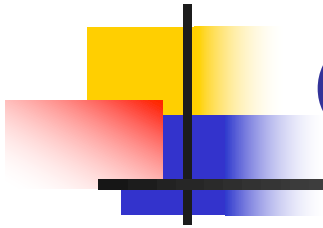
We need to show that if a language A is regular, a regular expression describes it. Because A is regular, it is accepted by a DFA. We describe a procedure for converting DFAs into equivalent regular expressions.



Generalized NFA (GNFA)

- We break the procedure of converting DFAs to regular expressions into two parts:
  - First, convert DFA into GNFA
  - Then, convert GNFA into regular expression

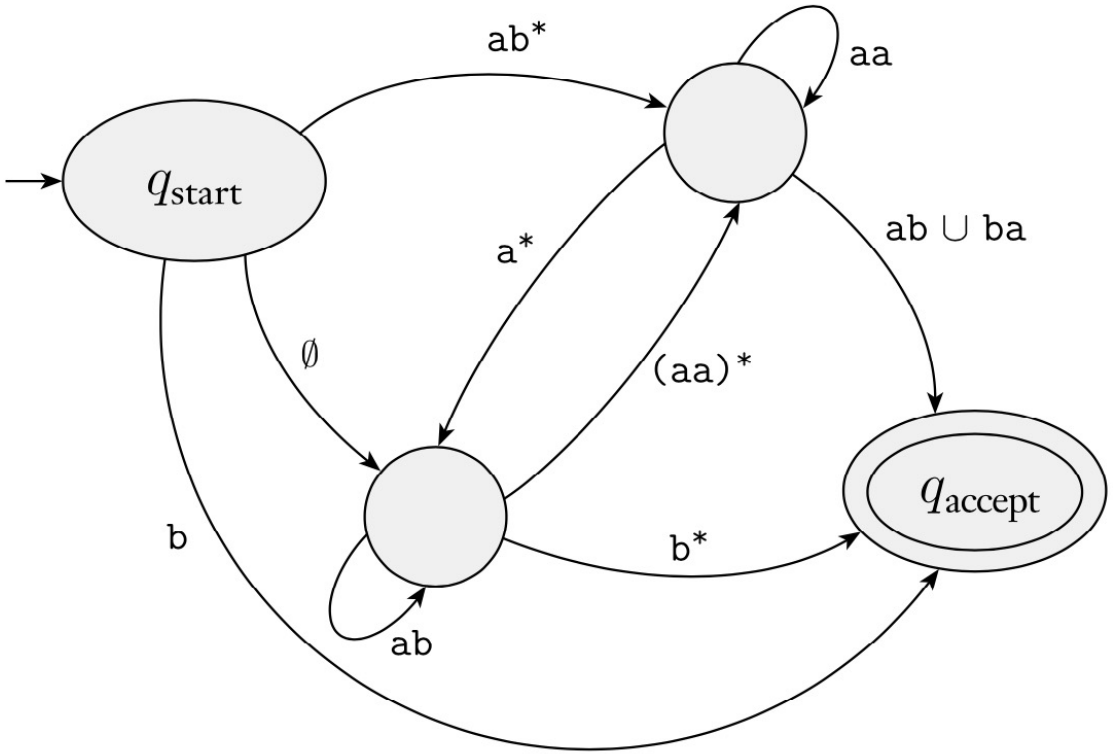
- What are **GNFA**s?
  - GNFAs are NFAs wherein the transition arrows may have regular expressions as labels, instead of only members of the alphabet or ε.
  - The GNFA reads blocks of symbols from the input, not necessarily just one symbol at a time as in an ordinary NFA.



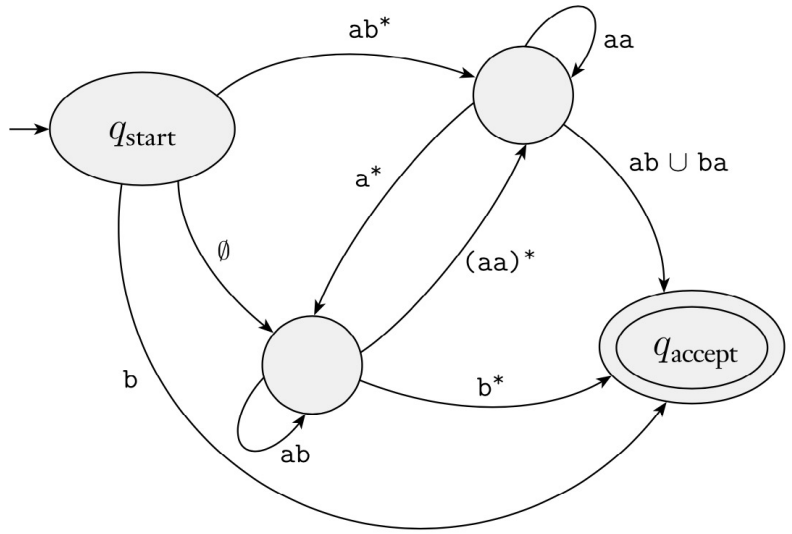
GNFAs (cont'd)

- The GNFA moves along a transition arrow connecting two states by reading a block of symbols from the input, which themselves constitute a string described by the regular expression on that arrow.
- A GNFA is nondeterministic and so may have several different ways to process the same input string.
- It accepts its input if its processing can cause the GNFA to be in an accept state at the end of the input.

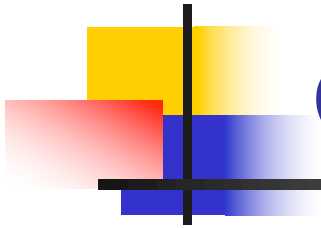
Example



Special form GNFA



- For convenience we require that GNFAs always have a special form that meets the following conditions:
  - The start state has transition arrows going to every other state but no arrows coming in from any other state.
  - There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.
  - Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.



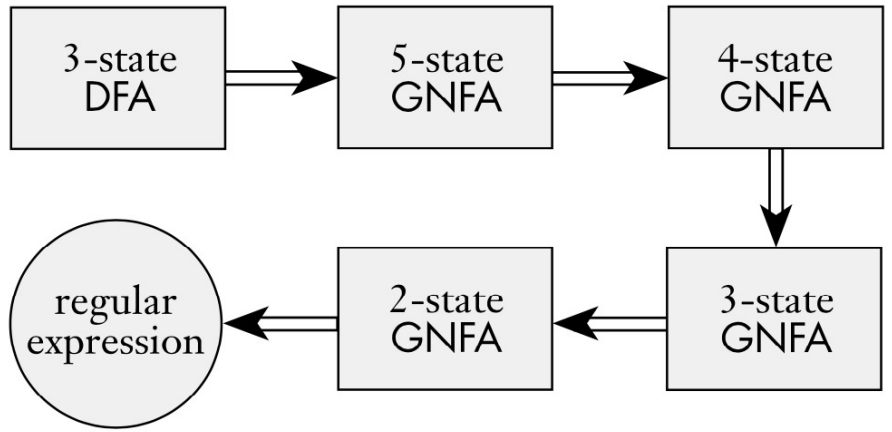
Converting a DFA into a GNFA in the special form

- Add a new start state with an ε arrow to the old start
- Add a new accept state with ε arrows from the old accept states
- If any arrows have multiple labels (or if there are multiple arrows going between the same two states in the same direction), replace each with a single arrow whose label is the union of the previous labels
- Add arrows labeled ∅ between states that had no arrows

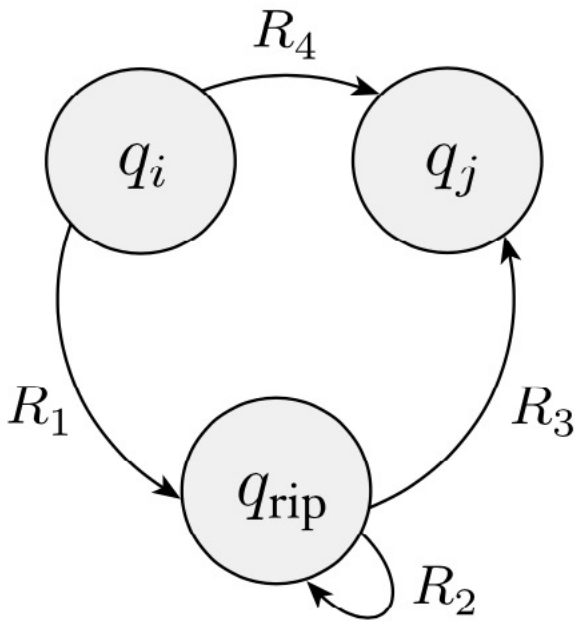
Converting a GNFA into a regular expression

- Say the GNFA has k states
- We know $k \geq 2$ since a GNFA must have a start state and an accept state and they must be different from each other

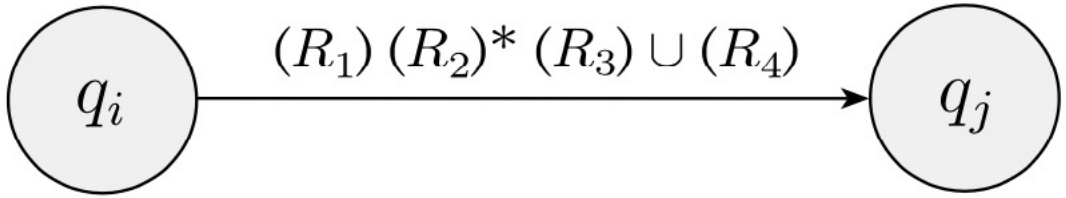
- If $k > 2$, we construct an equivalent GNFA with $k-1$ states
- The above step can be repeated on the new GNFA until it is reduced to 2 states
- If $k = 2$, the GNFA has a single arrow that goes from the start state to the accept state. The label of this arrow is the regular expression.



“Rip and Repair” – constructing an equivalent GNFA with one fewer state when k > 2



before



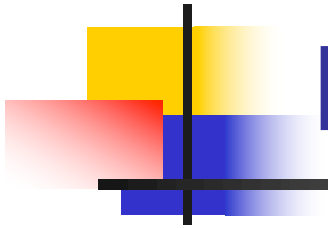
after

Formal definition of CNFGA

DEFINITION 1.64

A ***generalized nondeterministic finite automaton*** is a 5-tuple, $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$, where

1. Q is the finite set of states,
2. Σ is the input alphabet,
3. $\delta: \left(Q - \{q_{\text{accept}}\}\right) \times \left(Q - \{q_{\text{start}}\}\right) \longrightarrow \mathcal{R}$ is the transition function,
4. q_{start} is the start state, and
5. q_{accept} is the accept state.



Returning to the proof of Lemma 2…

- Let M be the DFA for language A
- We convert M to a GNFA G by adding a new start state and a new accept state and additional transition arrows as necessary
- We use the ***recursive*** procedure CONVERT(G), which takes a GNFA and returns an equivalent regular expression

CONVERT(G)

CONVERT(G):

1. Let k be the number of states of G.
2. If $k = 2$, then G must consist of a start state, an accept state, and a single arrow connecting them and labeled with a regular expression R. Return the expression R.
3. If $k > 2$, we select any state $q_{\text{rip}} \in Q$ different from q_{start} and q_{accept} and let G' be the GNFA $(Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$, where

$$Q' = Q - \{q_{\text{rip}}\},$$

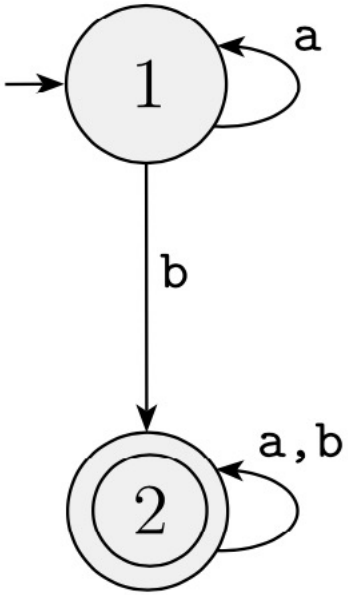
and for any $q_i \in Q' - \{q_{\text{accept}}\}$ and any $q_j \in Q' - \{q_{\text{start}}\}$, let

$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4),$$

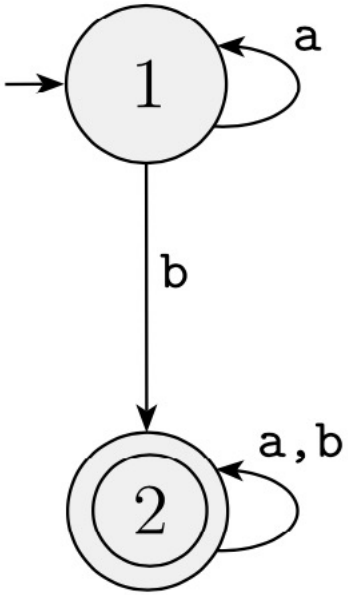
for $R_1 = \delta(q_i, q_{\text{rip}})$, $R_2 = \delta(q_{\text{rip}}, q_{\text{rip}})$, $R_3 = \delta(q_{\text{rip}}, q_j)$, and $R_4 = \delta(q_i, q_j)$.

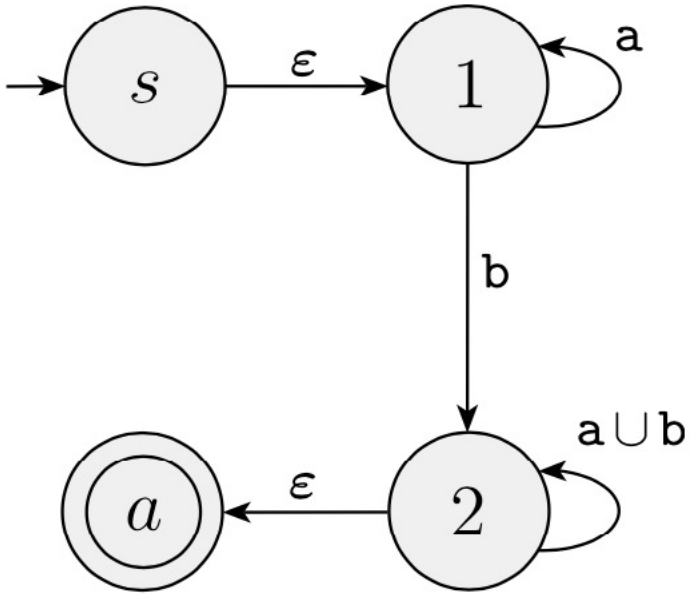
4. Compute CONVERT(G') and return this value.

Example 1

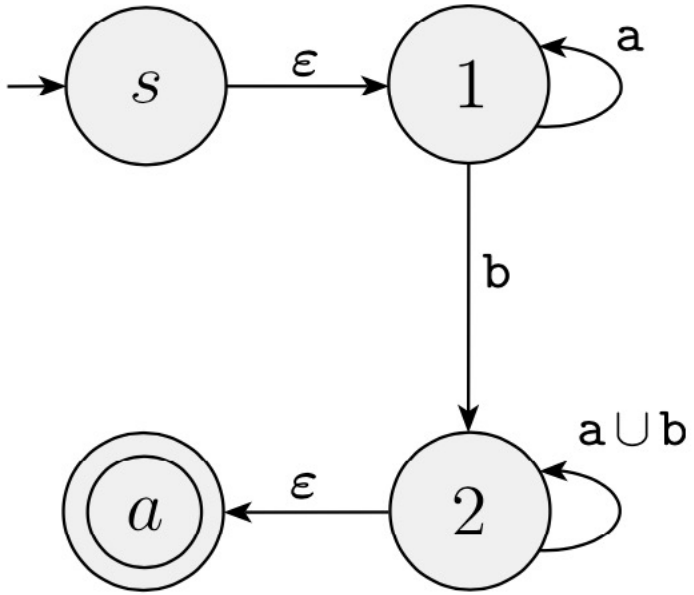


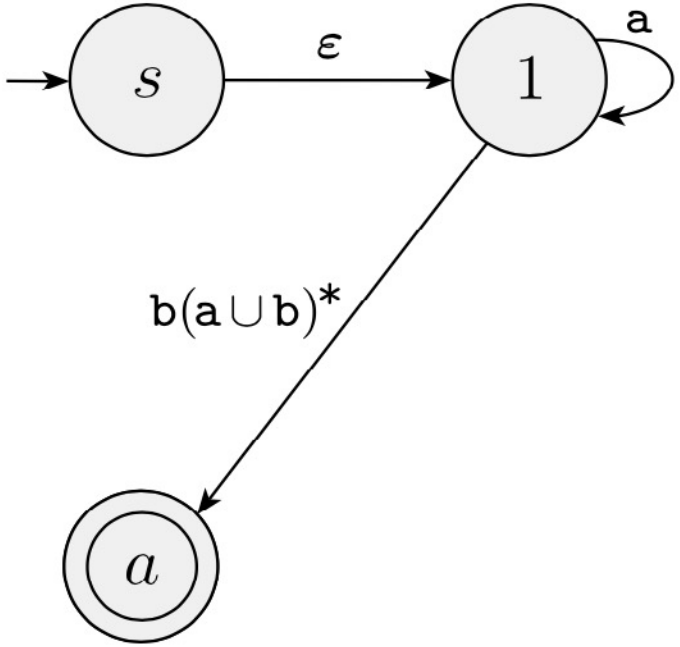
Example 1



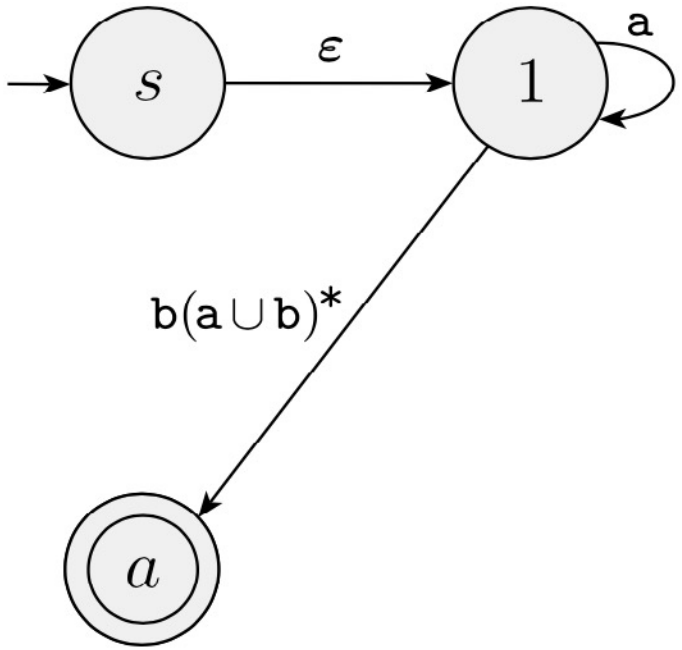


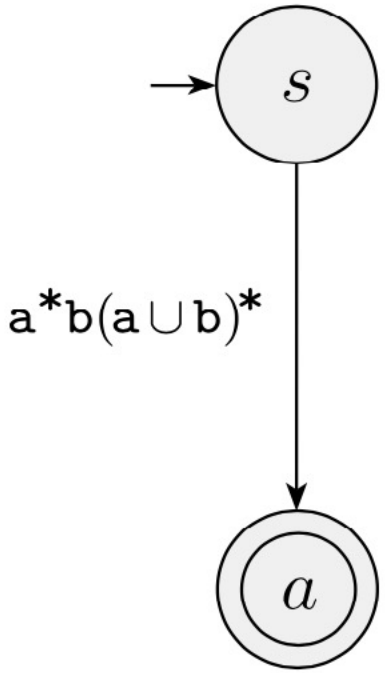
Example 1



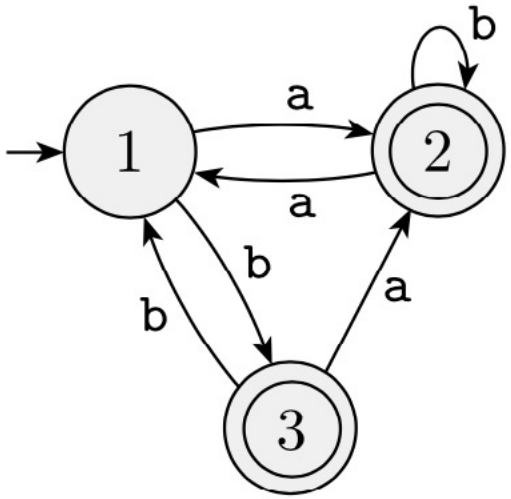


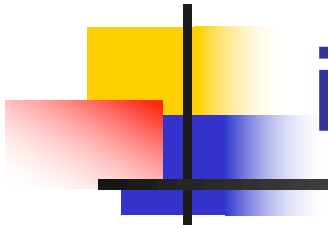
Example 1





Example 2





Will consider this example in next lecture on Wed

- Try it out yourself before we meet on Wed