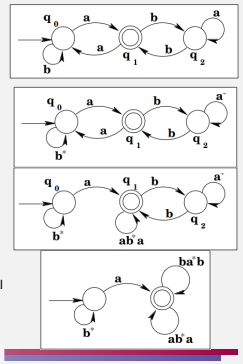
Finite-State Automata and Regular Languages

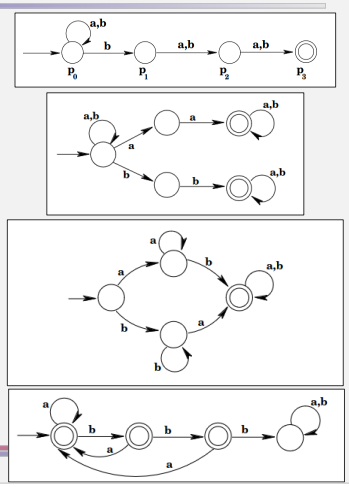
Theorem 3.4. *Every language that is accepted by a DFA or an NFA is generated by a regular expression.*

- a strategy (not a proof or algorithm)
  - if the DFA/NFA has more than one final state, build an equivalent NFA with a single final state
  - repeatedly replace sequences of transitions with single transitions labeled with the corresponding regular expressions until only start and final states remain
    - replace self loops
    - replace simple loops with self-loops, dropping no longer needed transitions and states
      - "simple loop" is a cycle where the middle state(s) are not final and have only a single transition in and out (excluding self loops)
    - replace simple and length 2 paths with transitions, dropping no longer needed transitions and states
      - "simple path" is a path where the middle state(s) are not final and have only a single transition in and out (excluding self loops)
    - combine parallel transitions
  - read the result



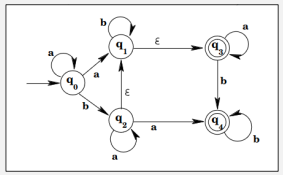
Finite-State Automata and Regular Languages

- if the DFA/NFA has more than one final state, build an equivalent NFA with a single final state
- repeatedly replace sequences of transitions with single transitions labeled with the corresponding regular expressions until only start and final states remain
  - replace self loops
  - replace simple loops with self-loops, dropping no longer needed transitions and states
    - "simple loop" is a cycle where the middle state(s) are not final and have only a single transition in and out (excluding self loops)
  - replace simple and length 2 paths with transitions, dropping no longer needed transitions and states
    - "simple path" is a path where the middle state(s) are not final and have only a single transition in and out (excluding self loops)
  - combine parallel transitions
- read the result



Finite-State Automata and Regular Languages

- if the DFA/NFA has more than one final state, build an equivalent NFA with a single final state
- repeatedly replace sequences of transitions with single transitions labeled with the corresponding regular expressions until only start and final states remain
  - replace self loops
  - replace simple loops with self-loops, dropping no longer needed transitions and states
    - "simple loop" is a cycle where the middle state(s) are not final and have only a single transition in and out (excluding self loops)
  - replace simple and length 2 paths with transitions, dropping no longer needed transitions and states
    - "simple path" is a path where the middle state(s) are not final and have only a single transition in and out (excluding self loops)
  - combine parallel transitions
- read the result



Finite-State Automata and Regular Languages

Theorem 3.5. *The intersection of two regular languages is a regular language.*

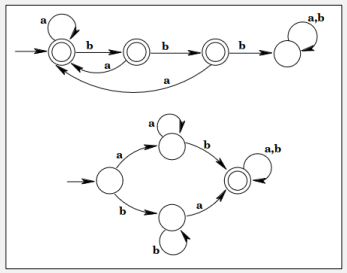
The union of two regular languages is a regular language.
The concatenation of two regular languages is a regular language.
The complement of a regular language is a regular language.
The Kleene closure of a regular language is a regular language.

- sketch of proof
  - build a DFA or NFA accepting each language using Theorem 3.3
  - build a DFA or NFA which accepts the union/concatenation/Kleene closure
    - utilize the construction in Theorem 3.3
  - or build a DFA or NFA which accepts the complement
    - for a DFA M, M' accepting the complement of M's language is M with the final and non-final states switched

    or build a DFA or NFA which accepts the intersection
    - the idea is to build M' to trace through DFAs M_1 and M_2 simultaneously – only accept strings which end up in final states for both machines
  - use Theorem 3.4 to conclude that there is a regular expression corresponding to that DFA/NFA

Finite-State Automata and Regular Languages

- for a DFA M, M' accepting the complement of M's language is M with the final and non-final states switched
  - must have all transitions specified, no omit-trap-state shortcuts!



Finite-State Automata and Regular Languages

- for two DFAs M_1, M_2, M accepting $M_1 \cap M_2$ contains pairs of states from M_1 and M_2
  - the idea is to move through M_1, M_2 simultaneously and only accept strings which end in final states in both machines