Algorithms for regular languages

Algorithms

- We have seen many algorithms.
  - These algorithms form the basis of many proofs.
  - They construct one computational mechanism from another
- The algorithms have been presented
  - In the text
  - In the homework and exercises
  - In lectures in class

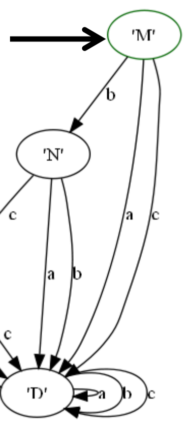
Characterizing algorithms

- The algorithms come in several classes
  1. Closure algorithms
     1. Some operation forms a new computational mechanism from an old mechanism but in the same class
        1. Union, intersection, complement, reversal, prefix, concatenation, etc.
  2. Inclusion algorithms
     1. Every computational mechanism from some class has an equivalent mechanism in another class
        1. DFA $\subseteq$ NFA
        2. RegExp $\subseteq$ NFA
        3. GNFA $\subseteq$ NFA
        4. NFA $\subseteq$ DFA

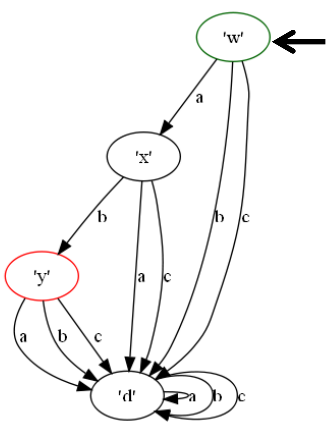
Union is closed over DFA

- Key ideas
  - Product (pair) construction
  - Any pair with a final state is final
  - Remove unreachable states

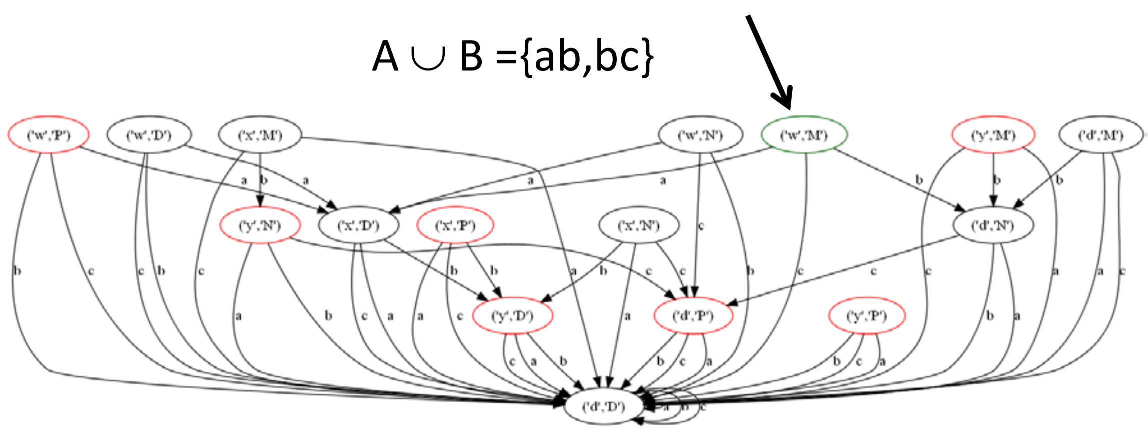
B={bc}



A={ab}



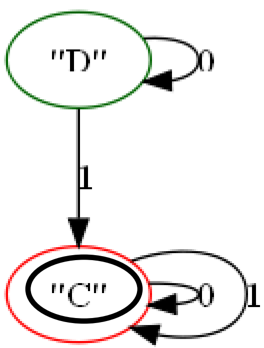
A ∪ B ={ab,bc}



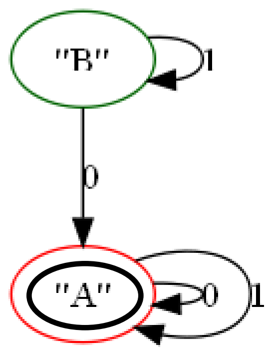
Intersection is closed over DFA

- Key ideas
  - Product (pair) construction
  - Only pairs with both left and right as final are final
  - Remove unreachable states

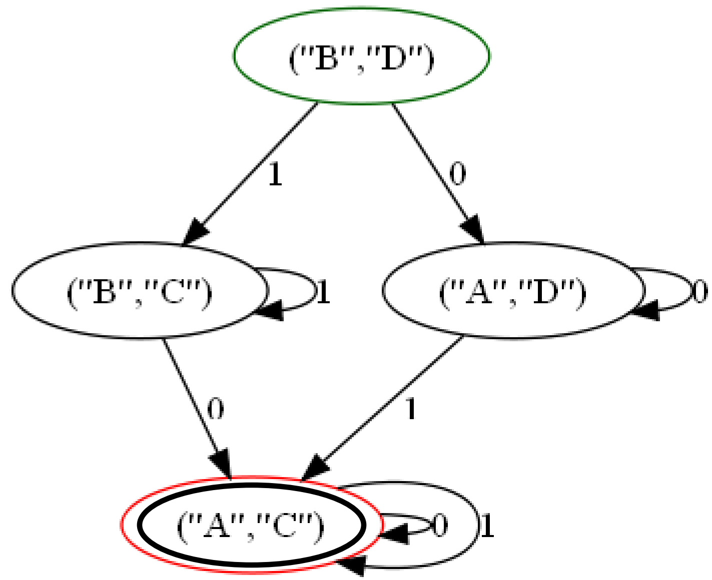
Contains a "1"



Contains a "0"



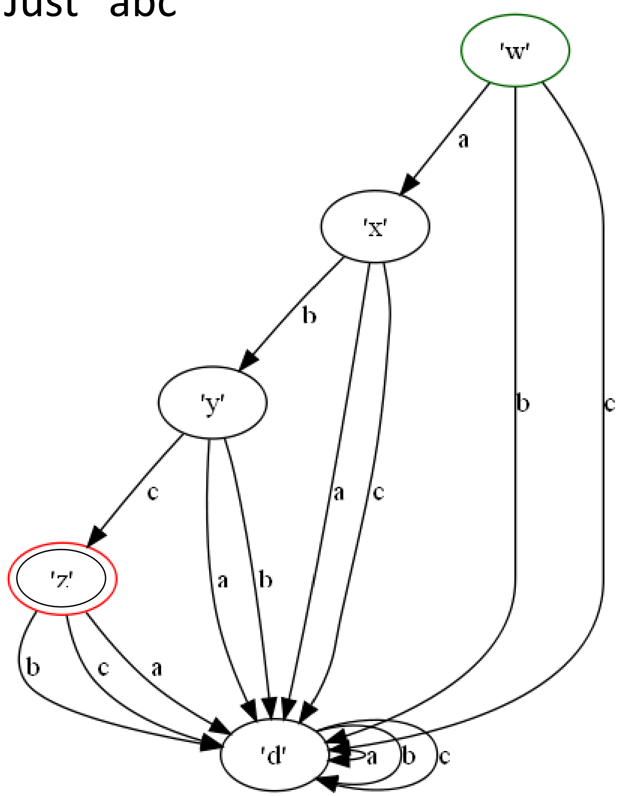
Contains both a "1" and a "0"



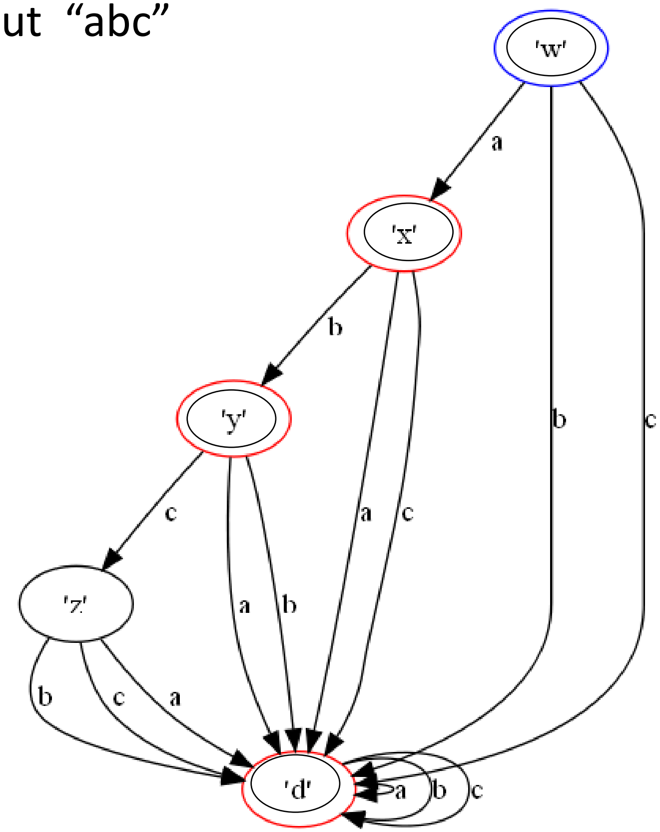
Complement is closed over DFA

- Key idea – Switch final and non final states.

Just “abc”

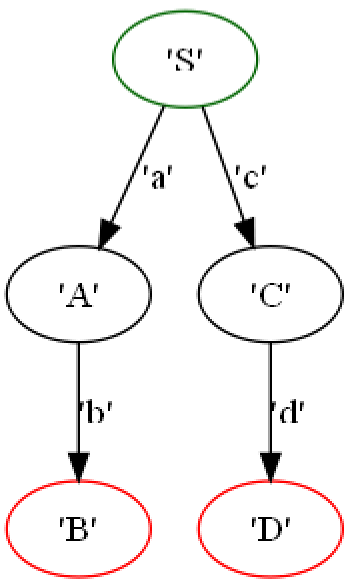


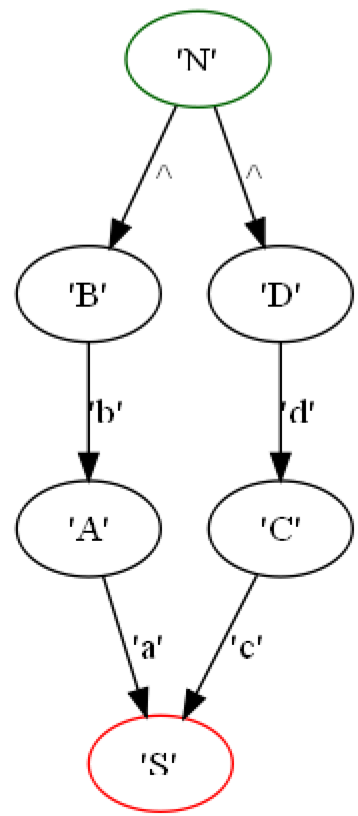
Anything but “abc”



Reversal is closed over NFA

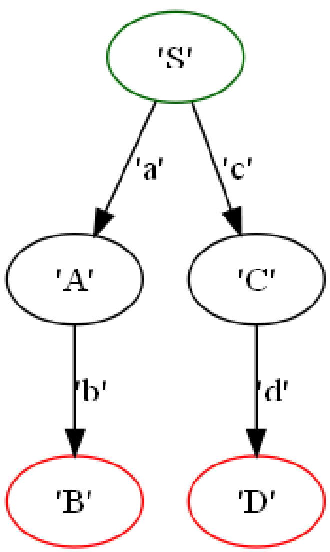
- Key ideas
  1. Reverse all arcs
  2. The old start state becomes the only new final state
  3. Add a new start state, and an ε-arc from it to all old final states.

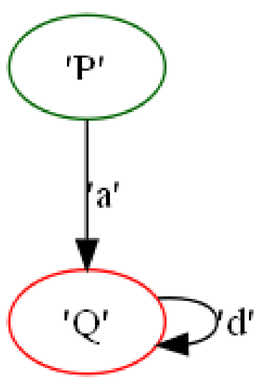


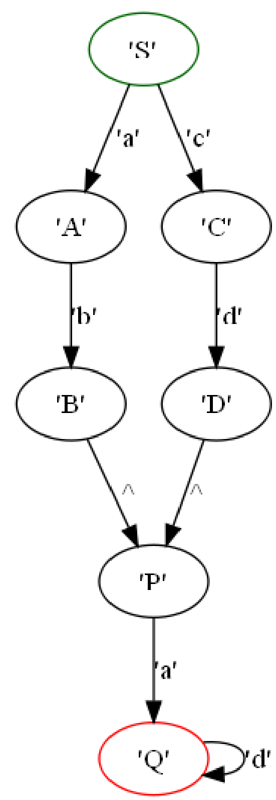


Concatenation is closed for NFA

- Key ideas
  - Union the states (you might have to rename them)
  - Add an ε-transition from each final state of the first to the start state of the second.

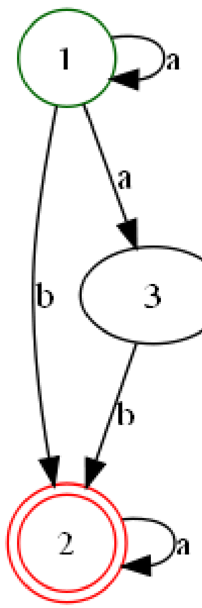


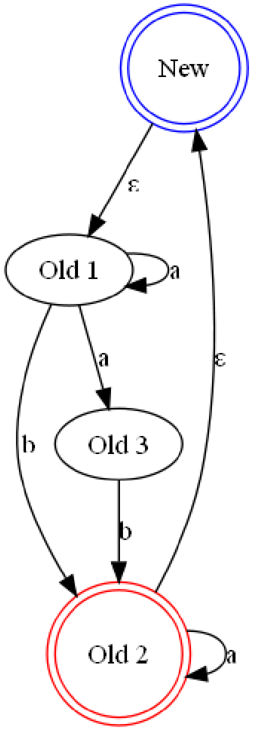




Kleene star is closed over NFA

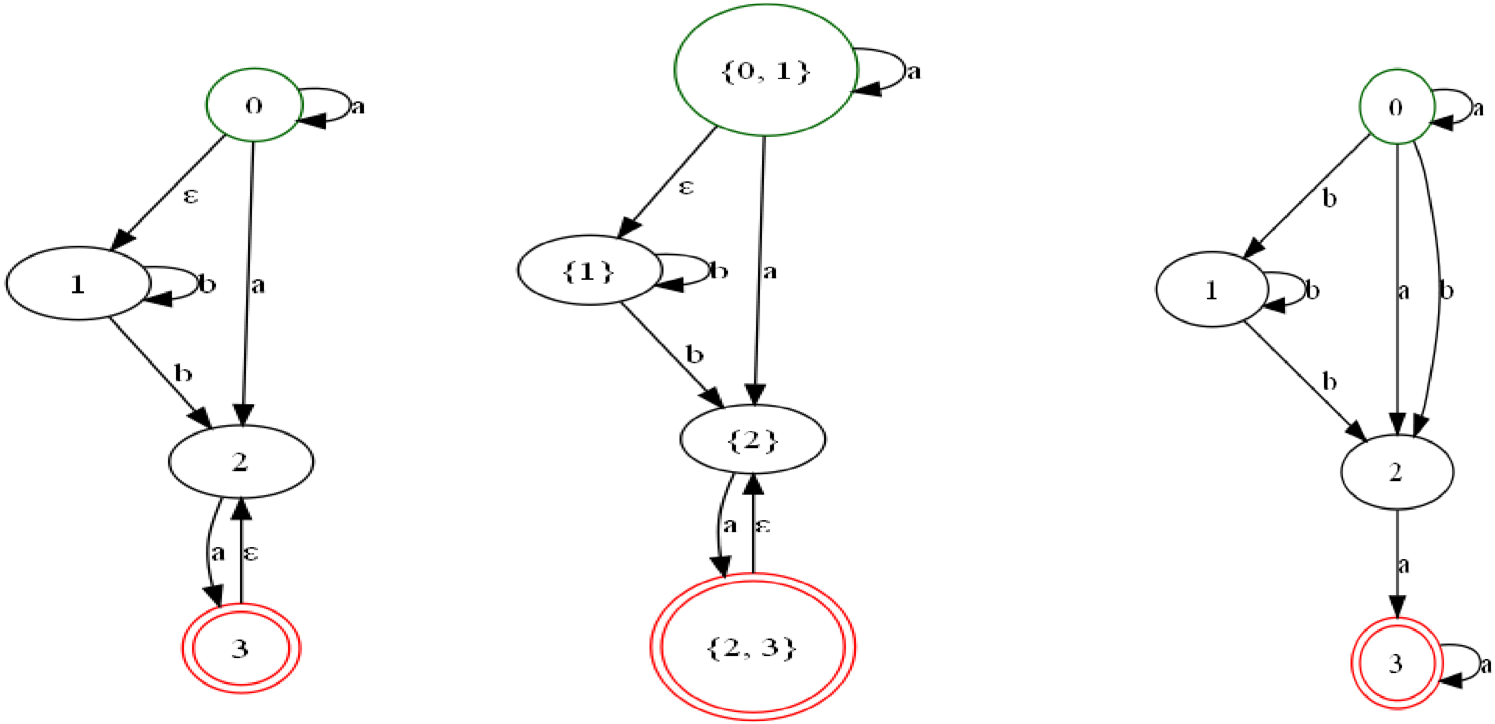
- Key ideas
  - Add a new state.
  - Make it the start state in the new NFA.
  - Add an ε-arc from this state to the old start state.
  - Add ε-arcs from every final state to this new state.





ε-NFA ⊆ NFA

- Key ideas
  - Compute e-closure for each state
  - Use these sets-of-states as states in a new NFA
  - Compute transitions by unioning transitions from individual states in the set of states on the old transition function



DFA $\subseteq$ NFA

- Key idea
  - Make a new transition function that returns a singleton set!
  - Everything else remains the same!

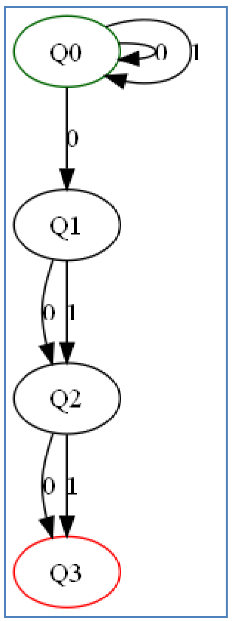
A DFA is a quintuple **A** = (**Q,S,T,q_0,F**) where
- **Q** is a set of ***states***
- **S** is the **alphabet** (of *input symbols)*
- **T**: **Q** $\times$ S $\rightarrow$ **Q** is the ***transition function***
- **q_0** $\in$ **Q** -- the ***start state***
- **F** $\subseteq$ **Q** -- ***final states***

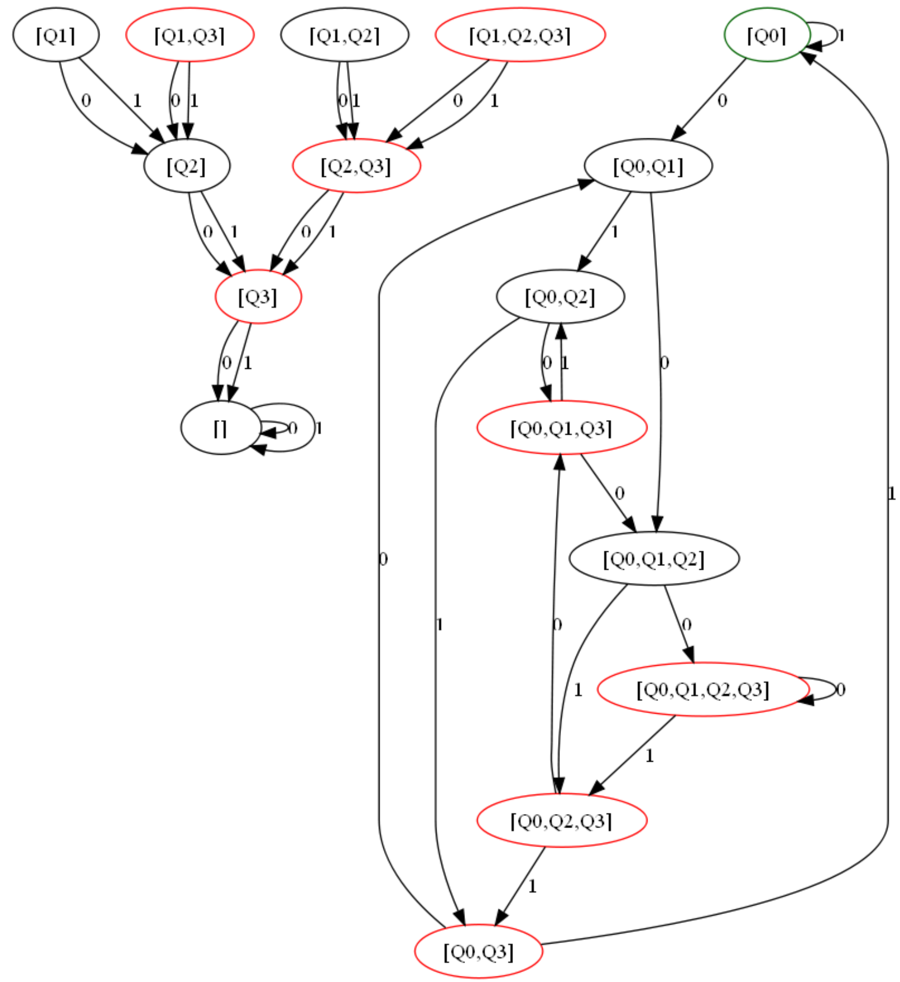
An NFA is a quintuple **A= (Q,S,T,q_0,F)** , where
- **Q** is a set of ***states***
- **S** is the **alphabet** (of *input symbols)*
- **T**: **Q** $\times$ S $\rightarrow$ P(**Q**) is the ***transition function***
- **q_0** $\in$ **Q** -- the ***start state***
- **F** $\subseteq$ **Q** -- ***final states***

```
dfaToNfa (DFA states alphabet trans start accept)
       = (NFA states alphabet delta start accept)
  where delta s c = [trans s c]
```

NFA ⊆ DFA

- Key ideas
  - Use subset construction
  - Remove unreachable states





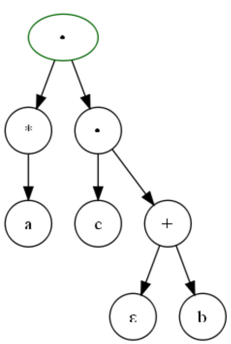
RegExp ⊆ NFA

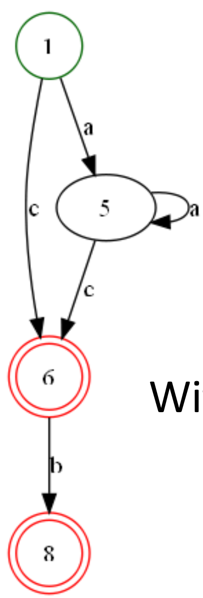
- Key ideas
  1. Decompose a RegExp into its parts
  2. Small parts make simple DFAs
  3. Combine smaller parts by merging with new transitions, and or new states.
  4. One can proceed top down or bottom up
  5. Remove ε-transitions

RegExp
a*c(ε + b)

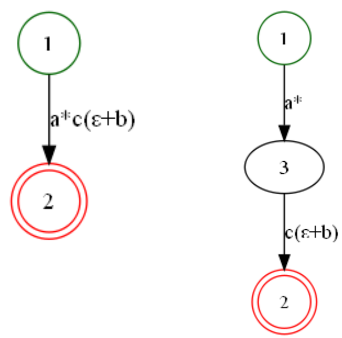
Parenthesized
(a*)(c(ε + b))

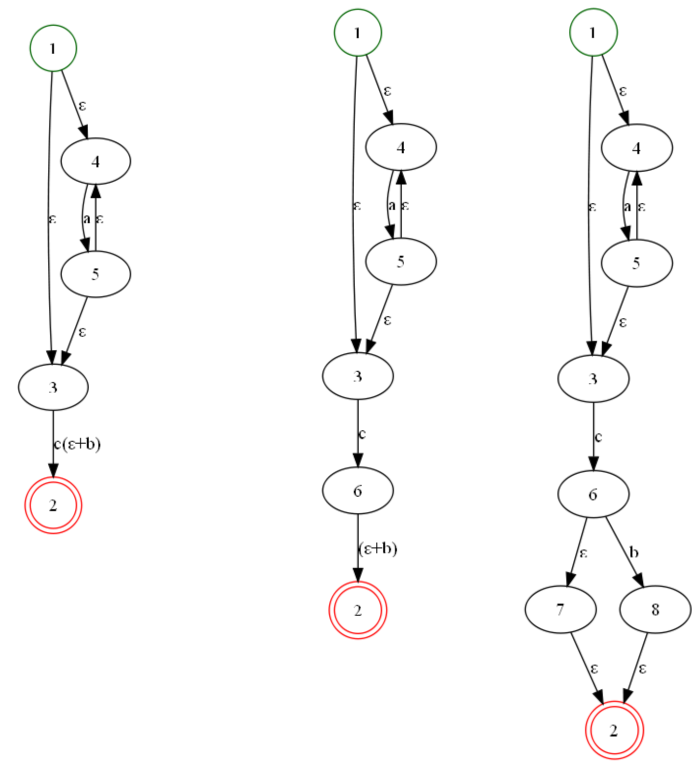
Tree





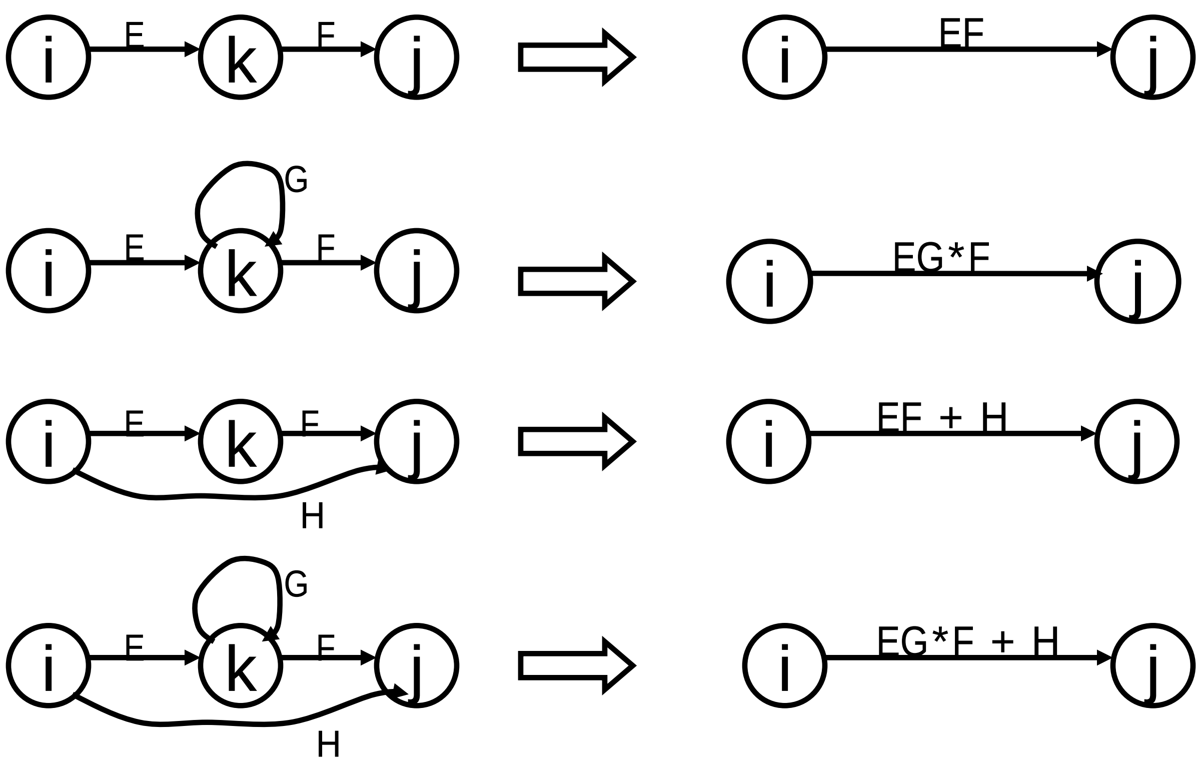
With ε-transitions removed

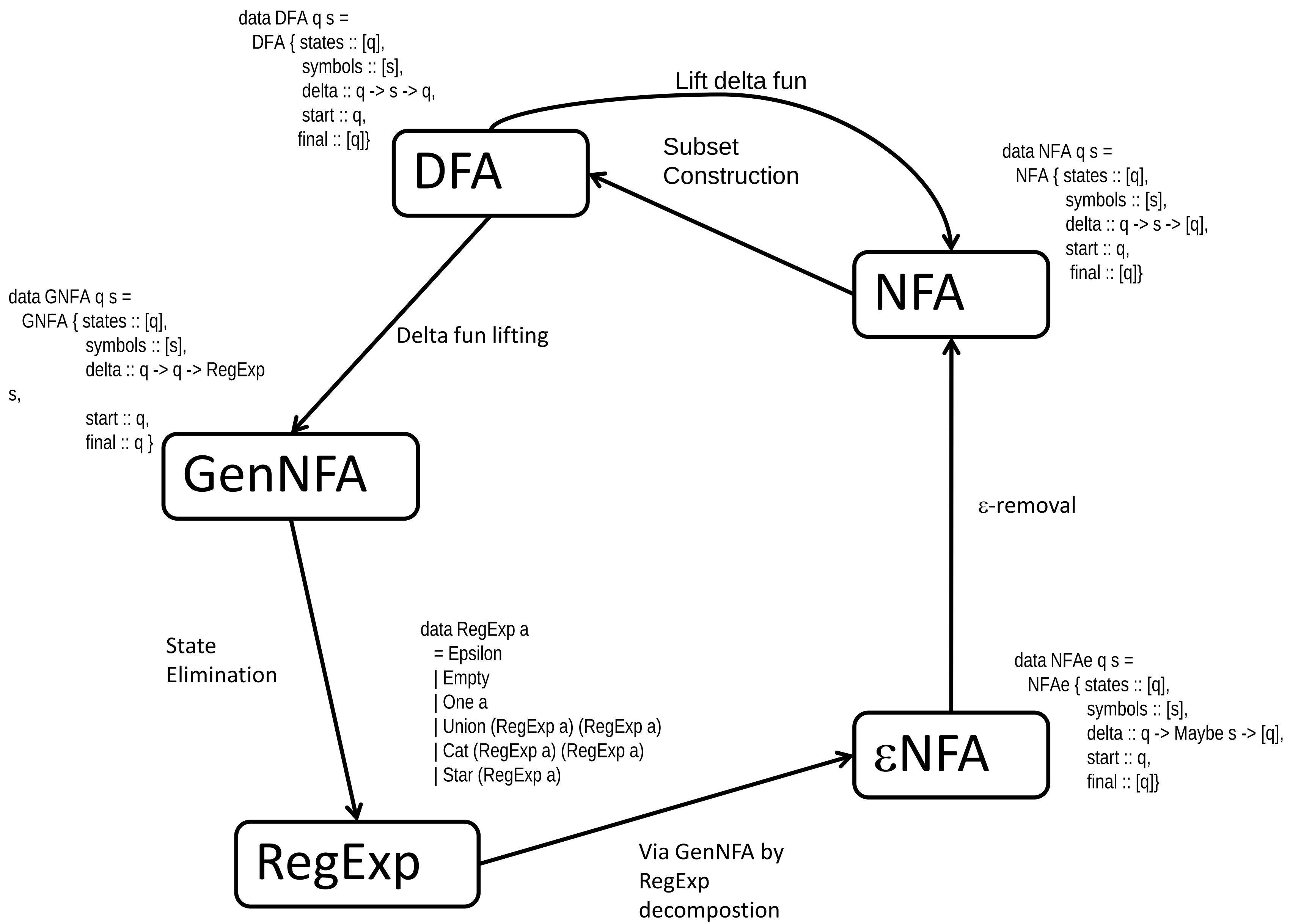




NFA $\subseteq$ RegExp

- Key ideas
  - Use GNFA construction (arcs labelled with RegExp)
  - Remove one state at a time




data DFA q s =
DFA { states :: [q],
symbols :: [s],
delta :: q -> s -> q,
start :: q,
final :: [q]}
DFA
Lift delta fun
Subset
Construction
data NFA q s =
NFA { states :: [q],
symbols :: [s],
delta :: q -> s -> [q],
start :: q,
final :: [q]}
NFA
data GNFA q s =
GNFA { states :: [q],
symbols :: [s],
delta :: q -> q -> RegExp
s,
start :: q,
final :: q }
Delta fun lifting
GenNFA
ε-removal
State
Elimination
data RegExp a
= Epsilon
| Empty
| One a
| Union (RegExp a) (RegExp a)
| Cat (RegExp a) (RegExp a)
| Star (RegExp a)
data NFAe q s =
NFAe { states :: [q],
symbols :: [s],
delta :: q -> Maybe s -> [q],
start :: q,
final :: [q]}
εNFA
RegExp
Via GenNFA by
RegExp
decompostion