Kleene Theorem

Regular Languages

- Today we continue looking at our first class of languages: Regular languages
  - <u>Means of defining</u>: Regular Expressions
  - <u>Machine for accepting</u>: Finite Automata

Kleene Theorem

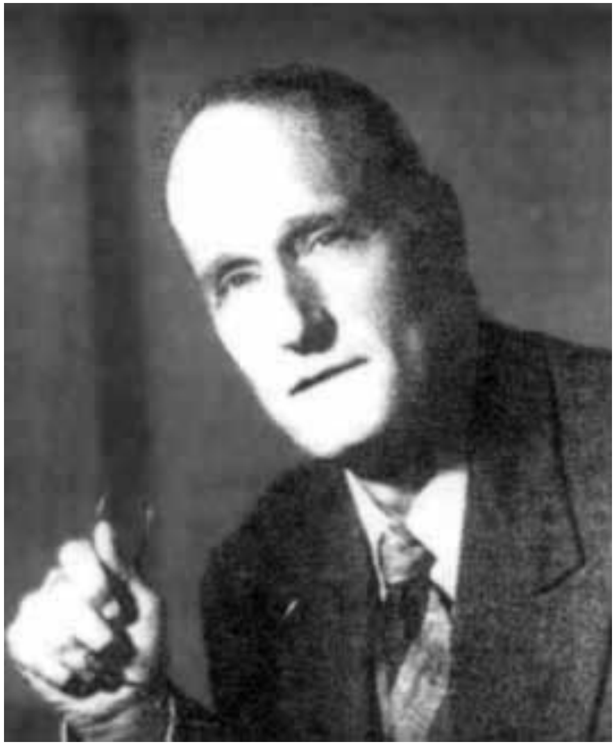
- Regular expressions (RE) and finite automata are equivalent (with respect to the languages they describe/accept)
  1. If R is a regular expression, there exists a DFA, M such that L(R) = L(M).
  2. For any DFA, M, L(M), the language accepted by the DFA can be described by a regular expression

Theory Hall of Fame

- Stephen Cole Kleene
  - 1909–1994
  - Born in Hartford, Conn.

  - PhD – Princeton (1934)
  - Prof at U. of Wisconsin at Madison (1935 – 1979)

  - Introduced Kleene Star op
  - Defined regular expressions
  - Proved equivalence with DFA



Proving Kleene Theorem

- Already completed:
  - We already showed the equivalence of DFA and NFA

- Left to do:
  - Given an RE, find a DFA that accepts the language described by the RE
    - Actually find an NFA
    - …Plus most of this is done!

  - Given a DFA, find an RE that describes the language accepted by the DFA

Part 1: RE -> DFA

- Since NFA are equivalent to DFA with respect to the class of languages they accept
  - We can, given an RE, build an NFA instead of a DFA that accepts the language described by the RE
  - We can always then convert that NFA to an equivalent DFA (using the subset construction)

Regular Expression Definition

- R is a regular expression if R equals

Base cases

1. $\varnothing$ (representing the empty language)
2. ε (representing the language $\{\varepsilon\}$)
3. a, for each $a \in \Sigma$, (representing the language $\{a\}$)

Recursive cases

4. $(R_1 \cup R_2)$ where R_1 and R_2 are regular expressions
5. $(R_1 R_2)$ where R_1 and R_2 are regular expressions
6. $(R_1)^*$ where R_1 is a regular expression

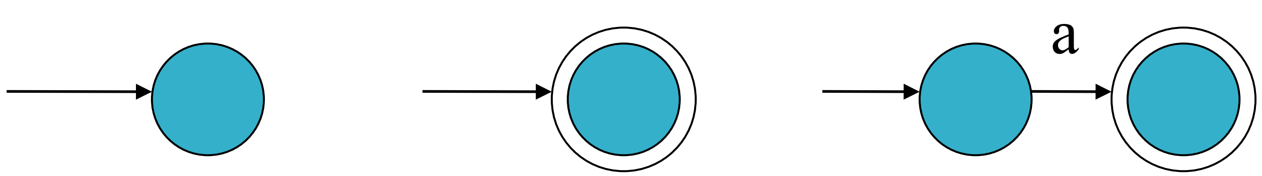
RE -> DFA

- We will prove by structural induction
  - Similar to mathematical induction, except instead of doing induction over integers, we will do induction over the structure
    - We will still start with base cases – basic regular expressions – and show that we can build NFA for them
    - And then we will do the inductive step.
      - Assume that given regular expressions R_1 and R_2 that describe languages L_1 and L_2, there exist NFAs, M_1 and M_2 that accept L_1 and L_2
      - Then we prove that for larger expressions that we build from R_1 and R_2 that we can still find an NFA that accepts the same language

RE -> DFA

- Base Cases: Build an NFA for regular expressions: ∅, ϵ, and a, a ∈ Σ

∅	{ϵ}	{a}	Language described by the regular expression



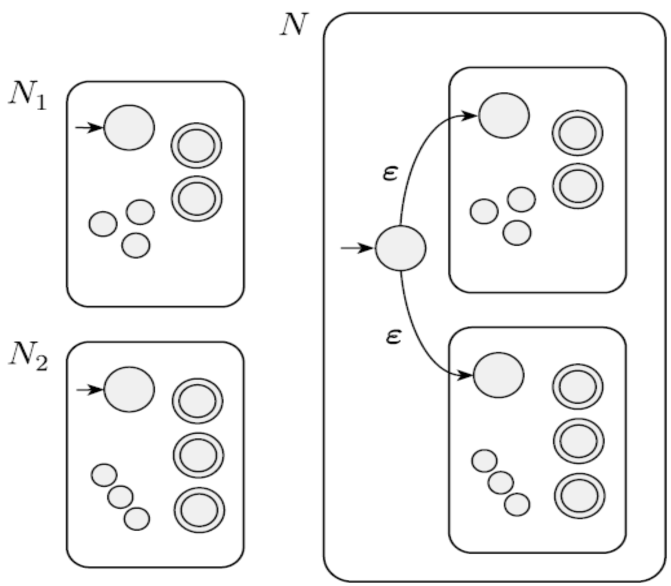
NFA recognizing the language described by the regular expression

RE -> DFA

- Induction:
  - Assume R_1 and R_2 are regular expressions that describe languages L_1 and L_2. Then, by the induction hypothesis, there exist NFA, M_1 and M_2, that accept L_1 and L_2
  - Then we must prove that we can create NFA that accept the languages described by:
    - $R_1 + R_2$ (Union)
    - $R_1 R_2$ (Concatenation)
    - R_1^* (Kleene Star)

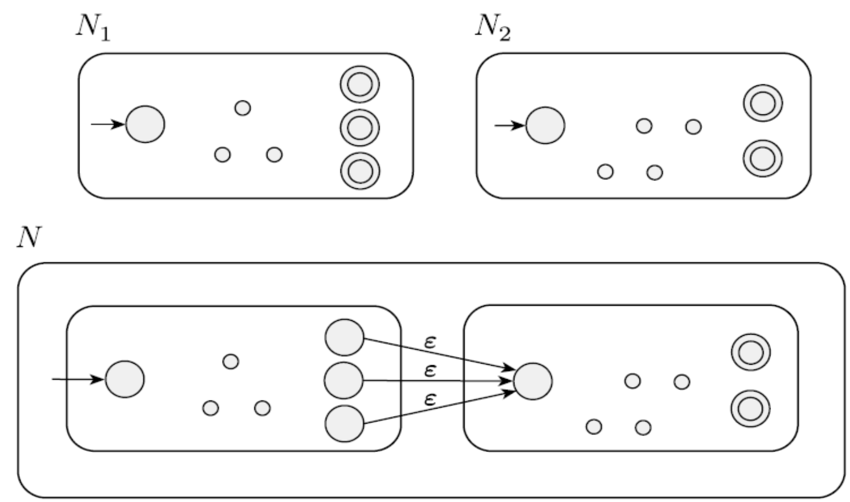
RE -> DFA

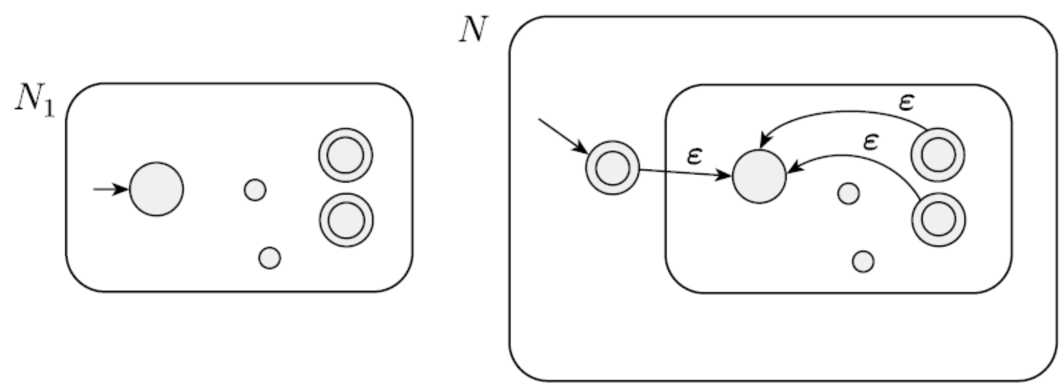
- Which we did already!



Union

Concatenation





Kleene Star

RE -> DFA

- So that proves one direction.
  - If we start with a regular expression, we know that there exists an NFA that accepts the same language.
  - And since NFA and DFA are equivalent, we could build a DFA accepting the same language.

- This shows that the class of languages that DFA can represent is at least as large as the class of languages that regular expressions can represent

Example

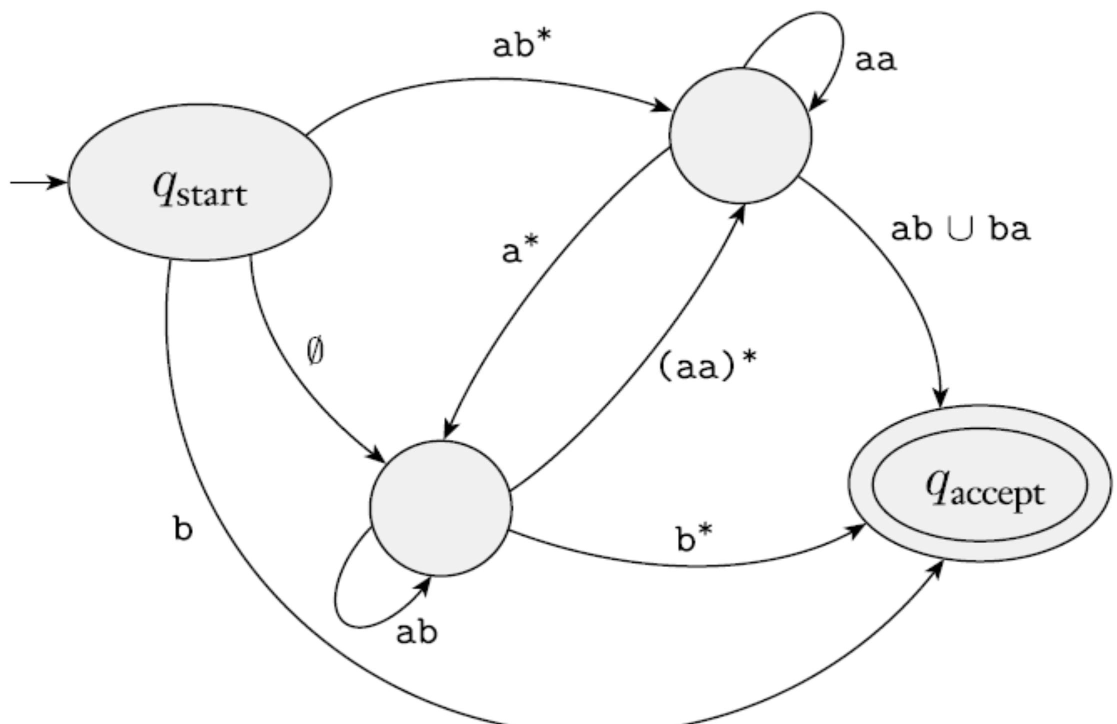
- Convert (a ∪ b)*ab to an NFA

DFA -> RE

- Given a DFA, M, there is a regular expression R that describes the language accepted by M.
  - We will construct the RE using a new type of FA, the generalized nondeterministic finite automata (GNFA).
  - We will convert the DFA into an equivalent GNFA
  - We will manipulate the GNFA until the final RE can be read directly from the GNFA

Generalized Nondeterministic Finite Automata (GNFA)

- Generalized Nondeterministic Finite Automata (GNFA) are NFA whose edges are regular expressions.



Generalized Nondeterministic Finite Automata (GNFA)

- Special conditions:
  - Start state has transitions ***to*** every other state but has no transitions ***from*** other states.
  - Only one final state…which is not the start state. Has transitions ***from*** every other state but has no transitions ***to*** other states.
  - Each other state has a single transition to every other state (including itself)

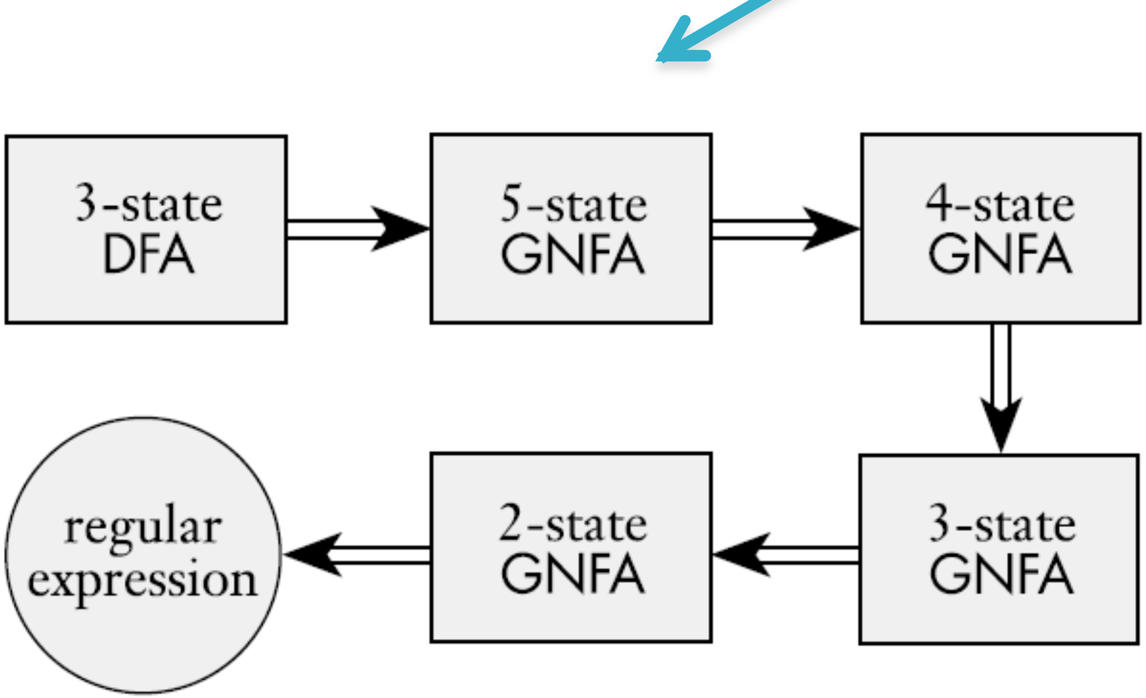
Basic Idea to Convert DFA to RE

- Convert the DFA to an equivalent GNFA (which will have two extra states associated with it)
- Use algorithm designed for GNFA to reduce the number of states in the GNFA one at a time, until reaching just two states
  - Must make sure that the reduced GNFA is equivalent at each step
  - The remaining states will be just the start state and the final state
  - The expression along the arrow from the start state to the final start will be the regular expression

Basic Idea to Convert DFA to RE

Given a 3-state DFA to start with:

Add extra start state
and final state



From DFA to GNFA

- Add new start state, add ϵ–transition from new start state to original start state.
- Add new final state, add ϵ–transitions from old final states (which are no longer final states) to new final state.
- For transitions with multiple labels, replace with union of symbols
- Add Ø transitions between all states where no transitions originally existed.

From DFA to GNFA

- Let's convince ourselves that this hasn't changed the language (strings we accept) at all
  - Adding the new start state
    - One branch ϵ-transitions into the usual start state
    - The other branch stays in the new start state, but this new start state is not an accepting state, and dies out upon any symbol being read, so it won't change the strings that are accepted.

From DFA to GNFA

- Let's convince ourselves that this hasn't changed the language (strings we accept) at all
  - Adding the new final state
    - All of the old final states are no longer final states, and have ϵ-transitions to the new final state
    - When an old final state is reached, it immediately branches to the new final state. So any string that was accepted previously will still be accepted.
    - The new branch at the new final state dies out upon reading any symbol, however, so it won't add any new strings to the language

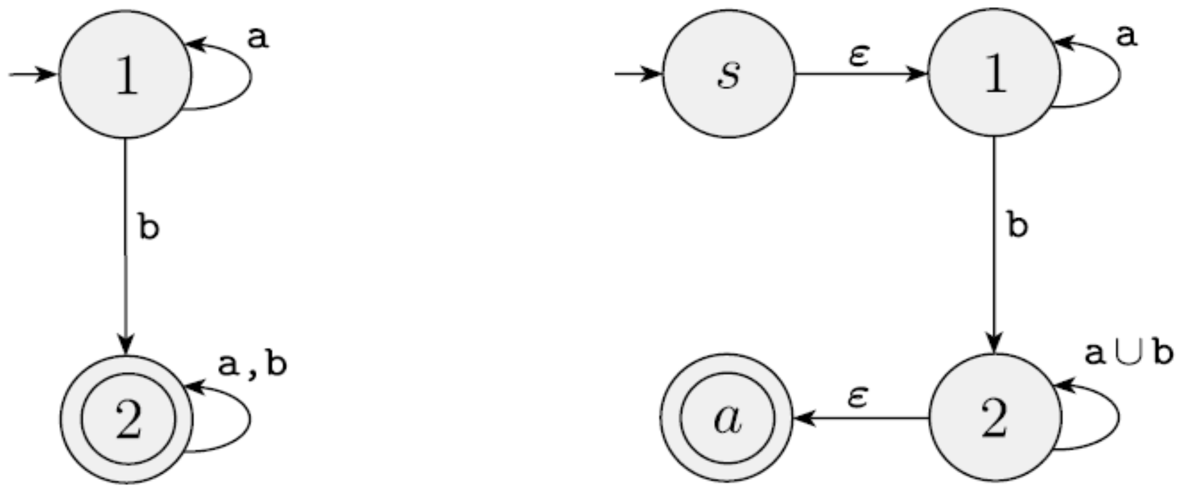
From DFA to GNFA

- Let's convince ourselves that this hasn't changed the language (strings we accept) at all
  - Transitions with multiple labels replaced with union expression
    - The union expression allows transition along that arrow for the same set of symbols

From DFA to GNFA

- Let's convince ourselves that this hasn't changed the language (strings we accept) at all
  - Adding arrows with Ø symbols
    - It is impossible to travel across an arrow with a Ø symbol, so no new strings will be accepted.

From DFA to GNFA



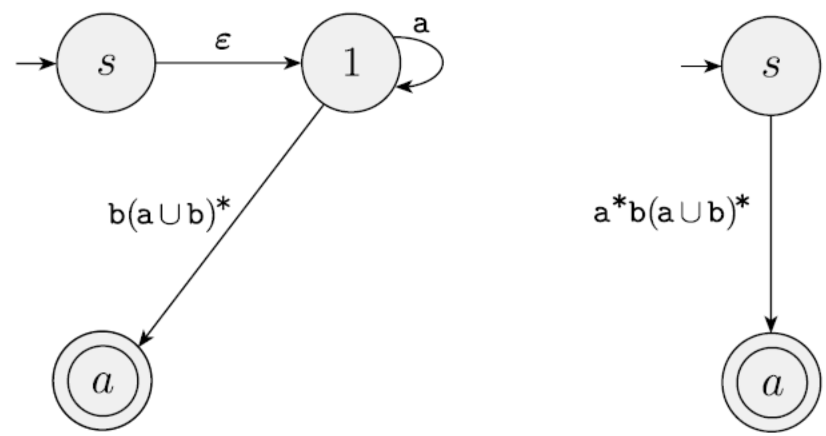
(a) (b)

Note that this figure skips putting in all of the Ø transitions

GNFA

- Once you have a GNFA,
  - Repeatedly "rip" states, then "repair"
  - Until you come up with a 2 state GNFA

- On a 2 state GNFA, the RE will be on the only transition (which will be from start to final state)



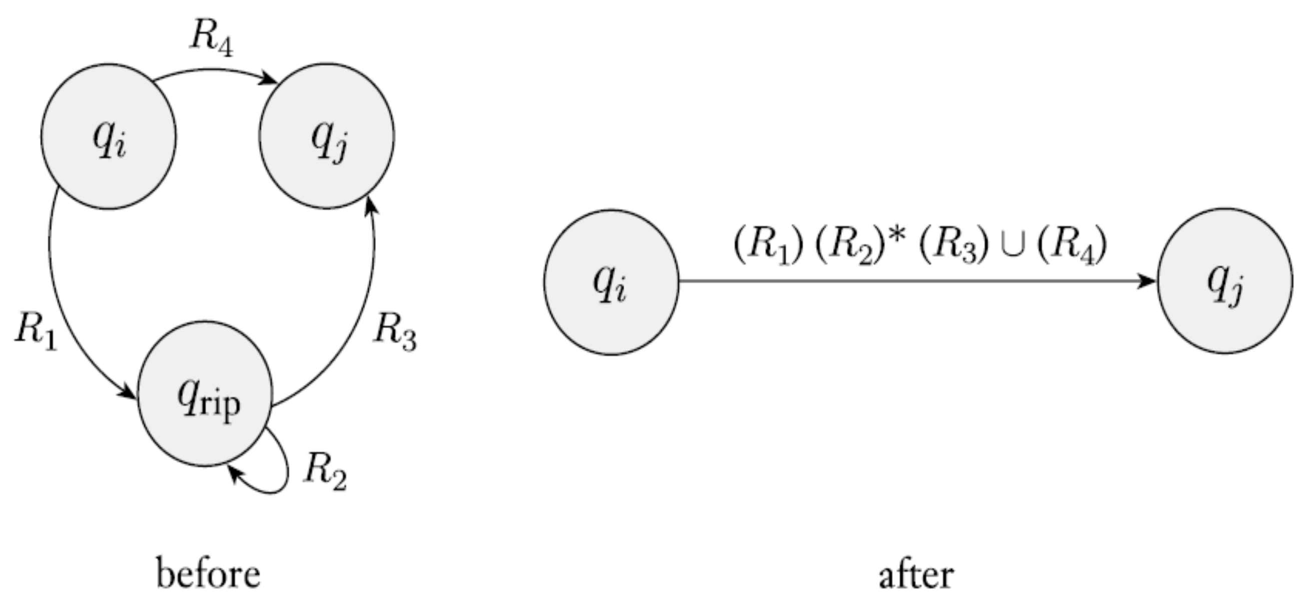
(c) (d)

GNFA

- Rip and repair
  - Remove a state (Not the new start or finish!)
  - Create new transitions to ensure that “repaired” machine still accepts the same language.
    - The transition between any two remaining states will be modified so that it accepts what it did originally, plus any expression that corresponds to traveling between the two states via the state being removed

GNFA

- Rip and repair



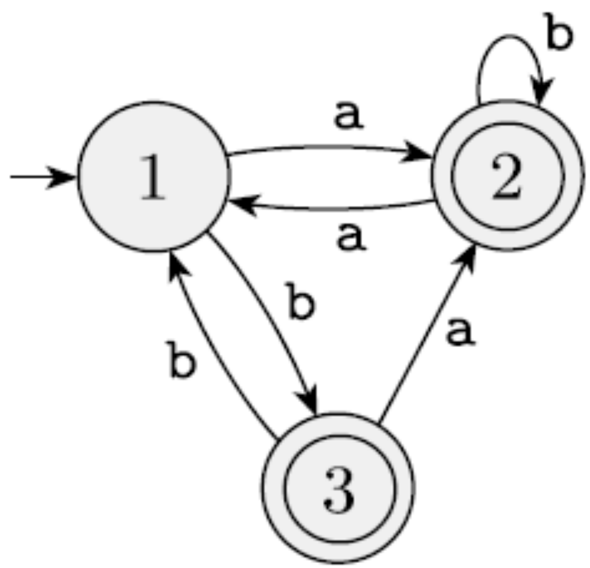
$$(R_1)(R_2)^*(R_3) \cup (R_4)$$

GNFA

- A note about Ø transitions…
  - $(R_1)(R_2)^*(R_3) \cup (R_4)$
  - If $R_4 = Ø$
    - Then the new expression will just be the expression corresponding to traveling through the state to be removed
  - If $R_1 = Ø$ or If $R_3 = Ø$
    - Concatenation with the empty set is the empty set, so no new expression will be added to the combined path
  - If $R_2 = Ø$
    - Kleene star generates ϵ so it is still possible to augment with direct path through q_{rip} (R_1R_3)

DFA -> RE: Example

- Example 1.68



(a(aa∪b)*ab∪b)((ba∪a)(aa∪b)*ab∪bb)*((ba∪a)(aa∪b)*∪ε) ∪a(aa∪b)*

DFA -> RE: Formal Proof

- Given a DFA, M, there exists a regular expression that describes the language L(M).

- Let's formally prove this.

Proof by Induction

- Steps to an inductive proof:
  1. Basis step:
     Show P(n) is true when $n=n_0$
  2. Induction hypothesis
     Assume that P(n) is true for some $k \geq n_0$
  3. Inductive step
     Prove P(n) is true for $n = k+1$ using the induction hypothesis.

We will inductively prove that the language accepted by the GNFA is equivalent as we rip states out

Formal Definition of GNFA

- A GNFA G = (Q, Σ, δ, q_{start}, q_{accept}) where
  - Q = set of states
  - Σ = input alphabet
  - δ = transition function (more on next slide)
  - q_{start} = start state
  - q_{accept} = final state

Formal definition of GNFA

- Definition of δ
  - $\delta : (Q - q_{accept}) \times (Q - q_{start}) \rightarrow R$
  - R = set of all regular expressions from alphabet Σ

DFA -> RE

- Given a DFA, $M = (Q_1, \Sigma, \delta_1, q_0, F)$, there exists a regular expression that describes the language L(M).

- Step 1 – convert DFA to GNFA
  - $M = (Q_1, \Sigma, \delta_1, q_0, F)$
  - $G = (Q, \Sigma, \delta, q_{start}, q_{accept})$
    - $Q = Q_1 \cup \{q_{start}, q_{accept}\}$

Step 1

- Now for the transition function, δ
- Recall:
  - $\delta_1: Q_1 \times \Sigma \rightarrow Q_1$
  - $\delta: (Q - q_{accept}) \times (Q - q_{start}) \rightarrow R$

- Regular transitions
  - For $q_1, q_2 \in Q_1$, let $B = \{a \in \Sigma \mid \delta_1(q_1, a) = q_2\}$
  - Then $\delta(q_1, q_2)$ is the regular expression corresponding to the union of $a_i \in B$
    - If $|B| = 0$: $\delta(q_1, q_2) = \emptyset$
    - If $|B| = 1$: $\delta(q_1, q_2) = a$
    - If $|B| > 1$: $\delta(q_1, q_2) = a_1 \cup \ldots \cup a_n$

Step 1

- Now for the transition function, δ
- Recall:
  - δ_1: Q_1 x Σ to Q_1
  - δ: $(Q - q_{accept})$ x $(Q - q_{start})$ to R

- Start state transitions
  - $\delta (q_{start}, q_0) = \epsilon$
  - For all $q \in Q_1$, $q \neq q_0$: $\delta (q_{start}, q) = \emptyset$
- Final state transitions
  - For all $q \in F$, $\delta (q, q_{accept}) = \epsilon$
  - For all $q \notin F$, $\delta (q, q_{accept}) = \emptyset$

Step 2 – Rip and Repair

- CONVERT (G)
  - returns a regular expression for GNFA G:

  - k = number of states in G
  - If k = 2 return $\delta(q_{start}, q_{accept})$
  - Choose $q_{rip} \in Q$, $q_{rip} \notin \{q_{start}, q_{accept}\}$
  - Construct G' = $(Q', \Sigma, \delta', q_{start}, q_{accept})$ where
    - $Q' = Q - \{q_{rip}\}$
    - For states $q_i \in Q - \{q_{accept}\}$, $q_j \in Q - \{q_{start}\}$
      - $\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4)$
      - Where
        - $R_1 = \delta(q_i, q_{rip})$, $R_2 = \delta(q_{rip}, q_{rip})$, $R_3 = \delta(q_{rip}, q_j)$, $R_4 = \delta(q_i, q_j)$

» Return CONVERT (G)=G'

Step 2 – Rip and Repair

- To Prove:
- For any GNFA G, L(G) = L(CONVERT(G))
  - For any GNFA, the language accepted by the GNFA is the same as the language accepted by the machine after it has been reduced all the way down to just a start and a final state

- Induction on k = number of states of G

- Will show true for all G with $k \geq 2$

Step 2 – Rip and Repair

- For any GNFA G, L(G) = L(CONVERT(G))

- BASIS step
  - For k = 2

  - If k = 2, the only 2 states are start and accept.
  - We don't have to reduce it at all, so
    - L(G) = L(CONVERT(G)) trivially

Step 2 – Rip and Repair

- For any GNFA G, L(G) = L(CONVERT(G))
- INDUCTION HYPOTHESIS
  - Assume true for n = k–1
  - L(G) = L(CONVERT(G)) if G has k–1 states
- INDUCTION
  - Show true for n = k
  - L(G) = L(CONVERT(G)) if G has k states.

Step 2 – Rip and Repair

- For any GNFA G, L(G) = L(CONVERT(G))
- Must show two things:
  - If w is accepted by G, it is accepted by CONVERT(G)
  - If w is accepted by CONVERT(G), it is accepted by G

Step 2 – Rip and Repair

- For any GNFA G, L(G) = L(CONVERT(G))
- Consider a single call of CONVERT
  - Converting from G (k states) to G' (k−1 states)
- If w is accepted by G, then when running w on G there is a sequence of states
  - $q_{start} q_1 q_2 \ldots q_{accept}$
- If none of these are q_{rip} then w is also accepted by G' by the same path

Step 2 – Rip and Repair

- For any GNFA G, L(G) = L(CONVERT(G)) (a single call of CONVERT)
- If w is accepted by G, then when running w on G there is a sequence of states $q_{start}q_1q_2\ldots q_{accept}$
- If one or more of these are q_{rip}
  - Isolate each occurrence of one or more consecutive q_{rip} states with q_i and q_j on either side. Each of these runs has a new regular expression in G' from q_i to q_j representing computation of G going from q_i to q_j but going through q_{rip}
    - $(R_1)\ (R_2)^*\ (R_3) \cup (R_4)$

Step 2 – Rip and Repair

- For any GNFA G, L(G) = L(CONVERT(G))=L(G')
- Consider a single call of CONVERT
- If w is accepted by G’ , then
  - Any transition between q_i and q_j represents a transition in G either going through q_{rip} or not.
  - G must also accept w.

Step 2 – Rip and Repair

- So what have we done

- For a single call of CONVERT,
  - Given a GNFA with k states
  - We constructed an equivalent GNFA with k-1 states.

- Look at the final line of the function

Step 2 – Rip and Repair

- CONVERT (G)
  - returns a regular expression for GNFA G:

  - k = number of states in G
  - If k = 2 return $\delta\ (q_{start}, q_{accept})$
  - Choose $q_{rip} \in Q$, $q_{rip} \notin \{q_{start}, q_{accept}\}$
  - Construct $G' = (Q', \Sigma, \delta', q_{start}, q_{accept})$ where
    - $Q' = Q - \{q_{rip}\}$
    - For states $q_i \in Q - \{q_{start}\}$, $q_j \in Q - \{q_{accept}\}$
      - $\delta'\ (q_i, q_j) = (R_1)\ (R_2)^*\ (R_3) + (R_4)$
      - Where
        - $R_1 = \delta\ (q_i, q_{rip})$, $R_2 = \delta\ (q_{rip}, q_{rip})$, $R_3 = \delta\ (q_{rip}, q_j)$, $R_4 = \delta\ (q_i, q_j)$

<u>Return CONVERT (G) = G'</u>

Step 2 – Rip and Repair

- So what have we done

- For a single call of CONVERT,
  - Given a GNFA with k states
  - We constructed an equivalent GNFA with k-1 states.

- Look at the final line of the function
  - The returned GNFA has k-1 states
  - By the inductive hypothesis, CONVERT will produce an equivalent GNFA

- We are done

Summary

- Part 1
  - Given a regular expression, R, we built an NFA that accepts the language R describes
    - This shows that the class of languages that DFA can represent is at least as large as the class of languages that regular expressions can represent
- Part 2
  - Given a DFA, we constructed a regular expression that describes the language accepted by the DFA
    - This shows that the class of languages described by regular expressions is at least as large as the class of languages that DFA can represent

Summary

- The proof of Kleene Theorem is complete!