Kleene Theorem I

Regular Languages

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

Kleene Theorem

- A language L over Σ is regular iff there exists an FA that accepts L.
  1. If L is regular there exists an FA M such that L = L(M)
  2. For any FA, M, L(M) is regular

     L(M), the language accepted by the FA can be expressed as a regular expression.

Proving Kleene Theorem

- Approach
  - Define 2 variants of the Finite Automata
    - Nondeterministic Finite Automata (NFA)
    - Nondeterministic Finite Automata with ε transitions (ε -NFA)
  - Prove that FA, NFA, and ε -NFA are equivalent w.r.t. the languages they accept
  - For a regular expression, build a ε -NFA that accepts the same language
  - For a DFA build a regular expression that describes the language accepted by the DFA.

Proving Kleene Theorem

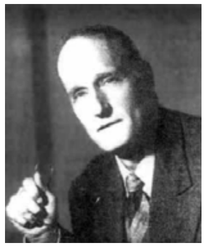
- We already showed the equivalence of DFA, NFA, and ε -NFA
- Left to do
  - Given a RE, find a DFA that accepts the language described by the RE
    - Actually find a ε -NFA
  - Given a DFA, find an RE that describes the language accepted by the DFA

Proving Kleene Theorem

- Today:
  - Given a RE, find a DFA that accepts the language described by the RE
    - Actually find a ε -NFA
- Thursday:
  - Given a DFA, find a RE that describes the language accepted by the DFA

Theory Hall of Fame

- <u>Steven Cole Kleene</u>
  - 1909-1994
  - b. Hartford, Conn.
  - PhD – Princeton (1934)
  - Prof at U of Wisc at Madison (1935 – 1979)
  - Introduced Kleene Star op
  - Defined regular expressions
  - Anyone with a Theorem named after him/her gets in the THOF!



Pt 1: RE -> DFA

- Since ε -NFA are equivalent to DFA w.r.t the class of languages they accept
  - We can, given an RE, build an ε -NFA instead of an DFA that accepts the language described by the RE
  - We can always then convert that ε -NFA to an equivalent DFA (using the algorithms presented last week)

Regular Expression

- Recursive definition of regular languages / expression over Σ :
  1. $\varnothing$ is a regular language and its regular expression is $\varnothing$
  2. $\{\varepsilon\}$ is a regular language and ε is its regular expression
  3. For each $a \in \Sigma$, $\{a\}$ is a regular language and its regular expression is a

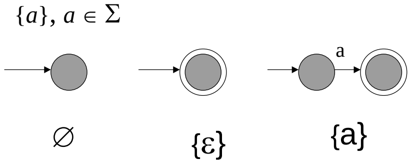
Regular Expression

4. If L_1 and L_2 are regular languages with regular expressions r_1 and r_2 then
   -- $L_1 \cup L_2$ is a regular language with regular expression $(r_1 + r_2)$
   -- L_1L_2 is a regular language with regular expression (r_1r_2)
   -- L_1^* is a regular language with regular expression (r_1^*)

Only languages obtainable by using rules 1-4 are regular languages.

RE -> DFA

- We will build our ε -NFA by structural induction:
  - Base case: Build an ε -NFA for $\varnothing$, $\{\varepsilon\}$, and $\{a\}$, $a \in \Sigma$



$\varnothing$ $\{\varepsilon\}$ $\{a\}$

RE -> DFA

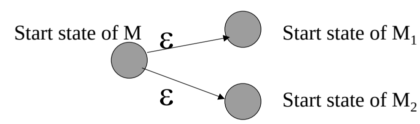
- Induction:
  - Assume R_1 and R_2 are regular expressions that describe languages L_1 and L_2. Then, by the induction hypothesis, there exists ε -NFA , M_1 and M_2 that accept L_1 and L_2
  - Create ε -NFA that accept the languages described by:
    - $R_1 + R_2$
    - $R_1 R_2$
    - R_1^*

RE -> DFA

- Induction Hypothesis:
  - $L_1 = L(M_1)$ where $M_1 = (Q_1, \Sigma, q_1, \delta_1, F_1)$
  - $L_2 = L(M_2)$ where $M_2 = (Q_2, \Sigma, q_2, \delta_2, F_2)$
    - Assume Q_1 and Q_2 are disjoint
- Will build
  - $M_u = (Q_u, \Sigma, q_u, \delta_u, F_u)$ $L(M_u) = L_1 + L_2$
  - $M_c = (Q_c, \Sigma, q_c, \delta_c, F_c)$ $L(M_c) = L_1L_2$
  - $M_k = (Q_k, \Sigma, q_k, \delta_k, F_k)$ $L(M_k) = L_1^*$

RE -> DFA: Union

- Basic idea
  - Using ε transitions, create a "branch" where the machine can either following one branch (representing one RE) or the other branch (representing the other RE)



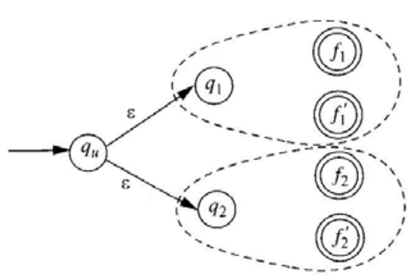
RE -> DFA: Union

- Basic idea
  - If a string is accepted by either of the existing Ms, it will be accepted by the new M.
    - The set of accepting states of M will include each of the accepting states from M_1 and M_2.

RE -> DFA: Union

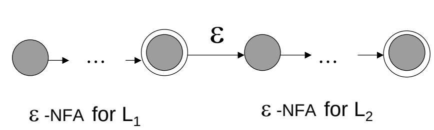
- Let's formalize this:
  - $M_u = (Q_u, \Sigma, q_u, \delta_u, F_u)$
  - $Q_u = Q_1 \cup Q_2 \cup \{q_u\}$
  - $F_u = F_1 \cup F_2$
  - Transition function: δ_u
    - $\delta_u(q_u, \varepsilon) = \{q_1, q_2\}$
    - $\delta_u(q_u, a) = \emptyset$ for all $a \in \Sigma$
    - $\delta_u(q, a) = \delta_1(q, a)$ if $q \in Q_1$
    - $\delta_u(q, a) = \delta_2(q, a)$ if $q \in Q_2$

RE -> DFA: Union



RE -> DFA: Concatenation

- Basic idea
  - Build M to start at the start state of M_1 and from any accepting state of M_1 move directly to the start state of M_2 via a ε transition.



RE -> DFA: Concatenation

- Basic idea
  - After being accepted by the first machine, a string will immediately be tested on the 2nd machine
    - The set of accepting states of the new M will be the same as that of the 2nd machine.

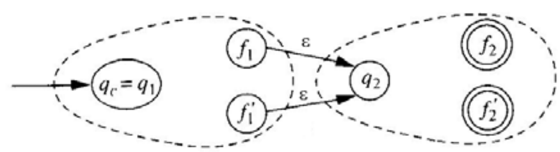
RE -> DFA: Concatenation

- Let's formalize this:
  - $M_c = (Q_c, \Sigma, q_c, \delta_c, F_c)$
  - $Q_c = Q_1 \cup Q_2$
  - $Q_c = q_1$
  - $F_c = F_2$

RE -> DFA: Concatenation

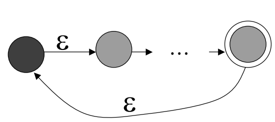
- Let's formalize this:
  - Transition function δ_c :
    - $\delta_c (q, a) = \delta_1 (q, a)$ if $q \in Q_1$
    - $\delta_c (q, a) = \delta_2 (q, a)$ if $q \in Q_2$
    - For all $q \in F_1$, $\delta_c (q, \varepsilon) = \delta_1 (q, \varepsilon) \cup \{q_2\}$

RE -> DFA: Concatenation



RE -> DFA: Kleene Star

- Basic idea
  - Create a new start state
    - Go from new start state to original start state via a ε transition
    - Go from any accepting state back to the new start state via a ε transition



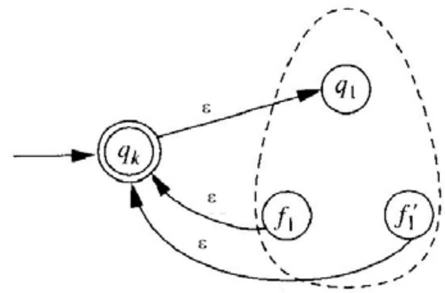
RE -> DFA: Kleene Star

- Basic idea
  - Make new start state the accepting state.
  - Note that you can get from any excepting state to the new start state via a ε transition.

RE -> DFA: Kleene Star

- Let's formalize this:
  - $M_k = (Q_k, \Sigma, q_k, \delta_k, F_k)$
  - $Q_k = Q_1 \cup \{q_k\}$
  - $F_k = \{q_k\}$
  - Transition function δ_k
    - $\delta_k (q, a) = \delta_1 (q, a)$ if $q \in Q_{1,}$
    - $\delta_k (q_k, \varepsilon) = \{q_1\}$
    - $\delta_u (q_u, a) = \varnothing$ for all $a \in \Sigma$
    - For all $q \in A_1$, $\delta_k (q, \varepsilon) = \delta_1 (q, \varepsilon) \cup \{q_k\}$

RE -> DFA: Kleene Star

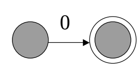


RE -> DFA: Example

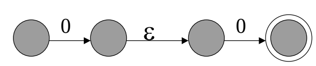
- Let's try an example
  - Create an ε-NFA for the regular expression:
    - $(00 + 1)^*(10)^*$

RE -> DFA: Example

- $(\underline{0}0 + 1)^*(10)^*$

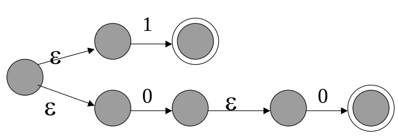


$(\underline{00} + 1)^*(10)^*$



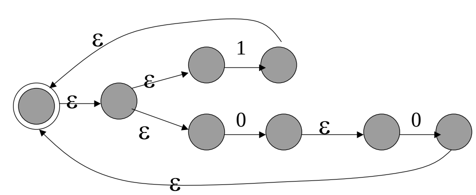
RE -> DFA: Example

- $(\underline{00 + 1})^*(10)^*$



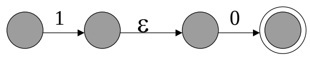
RE -> DFA: Example

- $\underline{(00 + 1)^*}(10)^*$

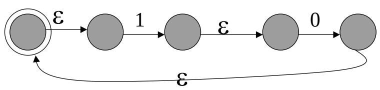


RE -> DFA: Example

- $(00 + 1)^*\underline{(10)}^*$

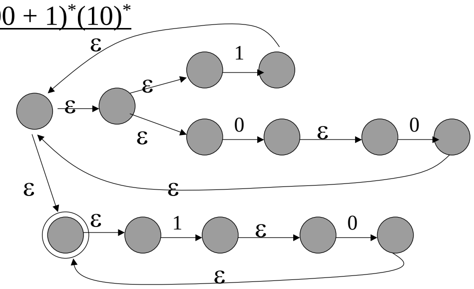


$(00 + 1)^*\underline{(10)^*}$



RE -> DFA: Example

- $\underline{(00 + 1)^*(10)^*}$



RE -> DFA: Summary

- What have we shown:
  - Given a language L described by a regular expression, we can build an ε -NFA that accepts L
  - Since ε -NFA are equivalent to DFAs, we can, if we wanted to, build an DFA to accept L.
  - Part 1 of the proof is complete.
  - Questions?

Next Time

- Kleene II
  - DFA -> RE
  - Problem Session