Kleene Theorem

Homework

- Homework #1 returned today
- Homework #2 due today

Homework

- Speaking of Homework
  - Here's Homework #3 (Due April 4)
    - From the textbook
      - 4.15b.c (No need to use proof of Kleene Theorem)
      - 4.16a,d
      - 4.29a
      - 4.35a,b (Do not simplify)
      - 4.38a (Must use proof of Kleene)

Before We Start

- Any questions?

Languages

- Recall.
  - What is a language?
  - What is a class of languages?

Regular Languages

- Regular languages
  - Means of defining: Regular Expressions
  - Machine for accepting: Finite Automata

Finite Automata

- A finite automaton (finite-state machine) is a 5-tuple $(Q, \Sigma, q_o, \delta, A)$ where
  - Q is a finite set (of states)
  - Σ is a finite alphabet of symbols
  - $q_o \in Q$ is the start state
  - $A \subseteq Q$ is the set of accepting states
  - δ is a function from Q x Σ to Q (transition function)

Did I happen to mention?

- Steven Cole Kleene
  - 1909-1994
  - b. Hartford, Conn.
  - PhD – Princeton (1934)
  - Prof at U of Wisc at Madison (1935 – 1979)
  - Introduced Kleene Star op
  - Defined regular expressions

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 (NDFA)
    - Nondeterministic Finite Automata with Λ transitions (NDFA-Λ)
  - Prove that FA, NDFA, and NDFA- Λ are equivalent w.r.t. the languages they accept
  - For a regular expression, build a NDFA- Λ that accepts the same language
  - For an FA build a regular expression that describes the language accepted by the FA.

Tonight

- Last time we defined these two FA variants:
  - Nondeterministic Finite Automata (NDFA)
  - Nondeterministic Finite Automata with Λ transitions (NDFA- Λ)
- In part 1 we show they are equivalent
- Questions?

Equivalence

- If L is a language over Σ^*, then the following 3 statements are equivalent:
  1. L is accepted by a FA
  2. L is accepted by a NDFA
  3. L is accepted by a NDFA-Λ

Equivalence

- How we will show this
  1. Given an NDFA that accepts L, create an FA that also accepts L
  2. Given an NDFA- Λ that accepts L, create an NDFA that also accepts L
  3. Given an FA that accepts L, create a NDFA- Λ that also accepts L.

Are we ready?

Step 1: NDFA->FA

- Given NDFA find FA
  - Let $M = (Q, \Sigma, q_0, A, \delta)$ be a NDFA then
    - There exists a FA, $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$
    - Such that $L(M) = L(M_1)$

Step 1: NDFA -> FA

- Basic idea
  - Recall that for a NDFA, δ: $Q \times \Sigma \rightarrow 2^Q$
  - Use the states of M_1 to represent subsets of Q.
  - If there is one state of M_1 for every subset of Q, then the non-determinism of M can be eliminated.
  - This technique, called subset construction, is a primary means for removing non-determinism from an NDFA.

Step 1: NDFA -> FA

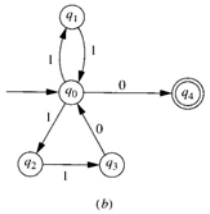
- Formal definition
  - $M = (Q, \Sigma, q_0, A, \delta)$ be a NDFA
  - We define FA, $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$
    - $Q_1 = 2^Q$
    - $q_1 = \{q_0\}$
    - For $q \in Q_1$ and $a \in \Sigma$,
    - $\delta_1(q,a) = \bigcup_{p \in q} \delta(p,a)$
    - $A_1 = \{q \in Q_1 \mid q \cap A \neq \varnothing\}$
  - Note that we need only include states on M_1 (subsets of Q) if the state is reachable.

Step 1: NDFA -> FA

- Algorithm for building M_1
  - Add $\{q_0\}$ to Q_1
  - While there are states of Q_1 whose transitions are yet to be defined
    - Let $q \in Q_1$
    - For each $a \in \Sigma$, determine the set of states, P, in M that are reachable from q on input a
    - If there is no state in Q_1 corresponding to P, add one.
    - Define δ_1 (q, a) = state in Q_1 corresponding to P
  - Define A_1 as any state in Q_1 that corresponds to a subset containing any of the final states of M

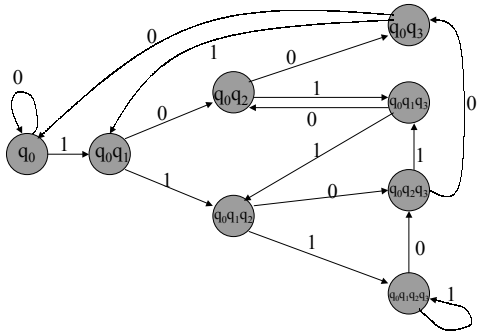
Step 1: NDFA -> FA

- Example



(b)

Step 1: NDFA -> FA



Step 1: NDFA -> FA

- Now we must show that M_1 accepts the same language as M
  - It can be shown (by structural induction) that for all $x \in \Sigma^*$
    - $\delta^*_1(q_1, x) = \delta^*(q_0, x)$

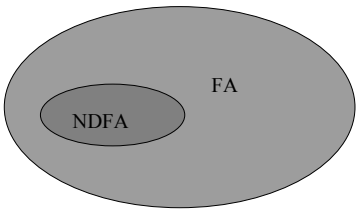
Step 1: NDFA -> FA

- Show that M and M_1 recognize the same language
  - x is accepted by M_1 iff $\delta^*_1(q_1, x) \in A_1$
  - x is accepted by M_1 iff $\delta^*(q_0, x) \in A_1$
  - By def of A_1,
    - x is accepted by M_1 iff $\delta^*(q_0, x) \cap A \neq \varnothing$
  - Thus
    - x is accepted by M_1 iff x is accepted by M

What have we shown

- In Step 1 we've shown:
  - Given a NDFA
    - There exists an FA that accepts the same language
    - Non-determinism can be removed from an NDFA by using a subset construction algorithm.
  - Questions?

What have we shown



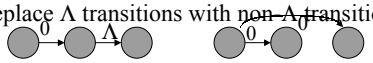
If $L \in$ NDFA then $L \in$ FA

Step 2: NDFA-Λ -> NDFA

- Given NDFA-Λ find NDFA
  - Let $M = (Q, \Sigma, q_0, A, \delta)$ be a NDFA-Λ then
    - There exists a NDFA, $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$
    - Such that $L(M) = L(M_1)$

Step 2: NDFA-Λ -> NDFA

- Basic idea
  - Recall that a NDFA-Λ is still non-deterministic
  - Replace Λ transitions with non-Λ transitions



  - Let $\delta_1(q,a)$ be the set of states reachable from q, reading symbol a, including those reachable via Λ transitions
    - Which is just $\delta^*(q,a)$

Step 2: NDFA-Λ -> NDFA

- Basic idea
  - Accepting states
    - If q_0 is not an accepting state of M but it is possible to get to an accepting state by just using Λ transitions, then q_0 must be added to the set of accepting states of M_1

Step 2: NDFA-Λ -> NDFA

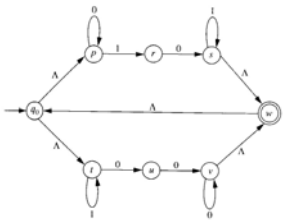
- Formal definition
  - M = (Q, Σ, q_0, A, δ) be a NDFA-Λ
  - We define NDFA, $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$
    - $Q_1 = Q$
    - $q_1 = q_0$
    - $\delta_1(q,a) = \delta^*(q,a)$
    - $A_1 = A \cup \{q_o\}$ if $\Lambda(\{q_o\}) \cap A \neq \varnothing$ in M
    - $= A$ otherwise

Step 2: NDFA-Λ -> NDFA

- Algorithm for constructing M_1
  - Set of states is the same as M
  - Start state is the same as M
  - Accepting states is the same as M
  - For each state, q, in M
    - compute the Λ closure
    - For each state in the Λ closure,p, and for each symbol a, add all elements of the Λ closure of δ (p,a) to the set $\delta_1(q, a)$
  - If you can get to to an accepting state in M from q_o by only using Λ transitions, add q_0 to A_1

Step 2: NDFA-Λ -> NDFA

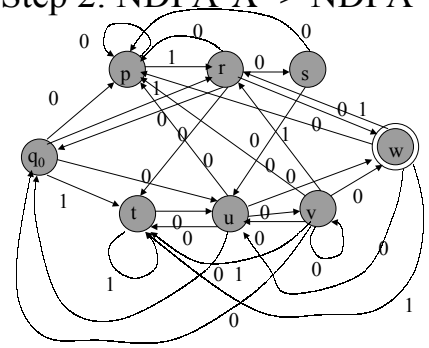
- Example



Step 2: NDFA-Λ -> NDFA

State	Λ closure
q_0	{q_0, p, t }
p	{p}
r	{r}
s	{s, w, q_0, p, t }
t	{t}
u	{u}
v	{v, w, q_0, p, t }
w	{w, q_0, p, t}

Step 2: NDFA-Λ -> NDFA



Step 2: NDFA-Λ -> NDFA

- Now we must show that M_1 accepts the same language as M
  - Can be shown (using structural induction) that for all $x \in \Sigma^*$
    - $\delta^*_1(q, x) = \delta^*(q, x)$

Step 2: NDFA-Λ -> NDFA

- Show that M and M_1 recognize the same language
  - x is accepted by M_1 iff $\delta^*_1(q_0, x) \cap A \neq \varnothing$
  - x is accepted by M_1 iff $\delta^*(q_0, x) \cap A \neq \varnothing$
  - Thus,
    - x is accepted by M_1 iff x is accepted by M
    - We showed this true for $|x| > 0$

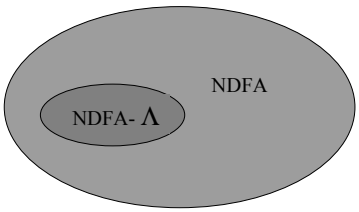
Step 2: NDFA-Λ -> NDFA

- Show that M and M_1 recognize the same language
  - If $|x| = 0$, I.e. $x = \Lambda$
    - If Λ accepted by M then $\Lambda(q_o)$ contains an accepting state, in which case we've added q_o to the set of accepting states of M_1
    - Thus Λ is accepted by M_1
    - If Λ rejected by M, then q_o is not an accepting state. Since the set of accepting states of M_1 is the same as M (except for the special case above), then $q_.$ will not be in the set of accepting states for M_1.
    - This Λ is rejected by M_1

Where we are

- In Step 2 we've shown:
  - Given an NDFA- Λ
    - There exists an NDFA that accepts the same language
    - Λ transitions can be removed from an NDFA- Λ by replacing Λ transitions with non- Λ transitions in an already non-deterministic NDFA.

Where we are

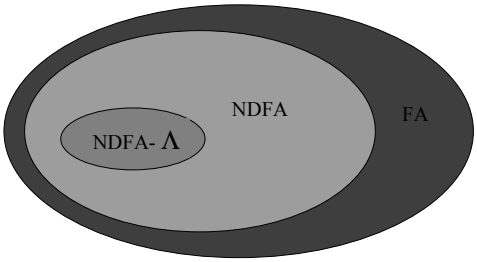


If L ∈ NDFA- Λ then L ∈ NDFA

Where we are

- We've shown that:
  - If L is accepted by an NDFA- Λ it is also accepted by an NDFA
  - If L is accepted by an NDFA it is also accepted by an FA
  - So if L is accepted by an NDFA- Λ, it is accepted by an FA.
- We still need to show:
  - If L is accepted by an FA, it is also accepted by an NDFA- Λ
- Questions?

Where we are



Step 3: FA-> NDFA-Λ

- Given FA find NDFA-Λ
  - Let M = (Q, Σ, q_0, A, δ) be a FA then
    - There exists a NDFA -Λ, $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$
    - Such that $L(M) = L(M_1)$

Step 3: FA-> NDFA-Λ

- Basic idea
  - Since FAs are more restrictive than NDFA-Λs, any FA is essentially an NDFA-Λ that doesn't take advantage of:
    - Non-determinism
    - Λ-transitions
  - Must consider details of transition functions

Step 3: FA-> NDFA-Λ

- Formal definition
  - M = (Q, Σ, q_0, A, δ) be a FA
  - We define NDFA-Λ, $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$
    - $Q_1 = Q$
    - $q_1 = q_0$
    - $A_1 = A$
    - $\delta_1(q, \Lambda) = \varnothing$ for all $q \in Q$
    - $\delta_1(q, a) = \{\delta(q, a)\}$ for all $q \in Q, a \in \Sigma$

Step 3: FA-> NDFA-Λ

- Now we must show that M_1 accepts the same language as M
  - Can show (using structural induction) that for all $x \in \Sigma^*$
    - $\delta^*_1(q, x) = \{\delta^*(q, x)\}$

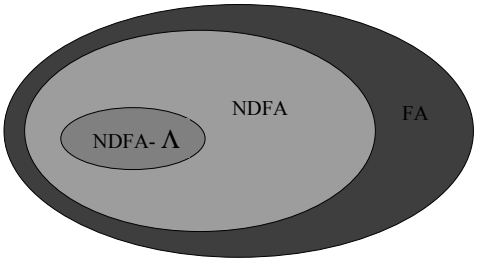
Step 3: FA-> NDFA-Λ

- Show that M and M_1 recognize the same language
  - x is accepted by M_1 iff $\delta^*_1(q_0, x) \cap A \neq \varnothing$
  - x is accepted by M_1 iff $\{\delta^*(q_0, x)\} \cap A \neq \varnothing$
  - That's the same as saying
    - $\delta^*(q_0, x) \in A$ or
    - x is accepted by M
  - Thus
    - x is accepted by M_1 iff x is accepted by M

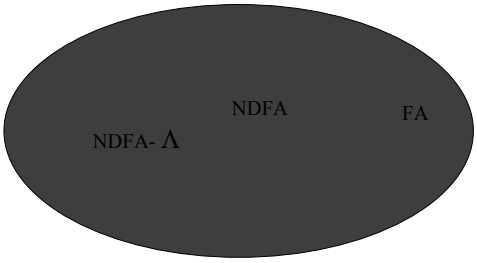
Equivalence!

- What we have shown
  1. Given an NDFA that accepts L, create an FA that also accepts L
  2. Given an NDFA- Λ that accepts L, create an NDFA that also accepts L
  3. Given an FA that accepts L, create a NDFA-Λ that also accepts L.
- All 3 are equivalent

Equivalence!



Equivalence!

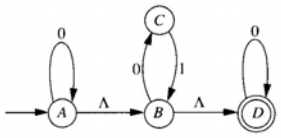


Let's try an example

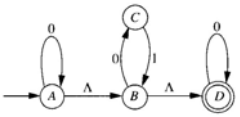
- Find a FA for the regular expression
  - $0^*(01)^*0^*$

Example: RE -> NDFA-Λ

- $0^*(01)^*0^*$

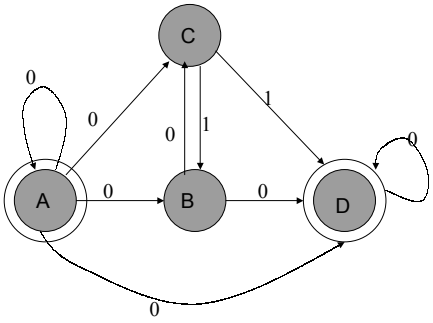


Example: NDFA-Λ -> NDFA

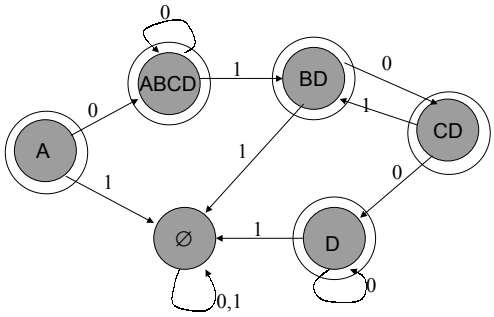


State	Λ closure
A	{A, B, D}
B	{B, D}
C	{C}
D	{D}

Example: NDFA-Λ -> NDFA



Example: NDFA-> FA



Summary

- If L is a language over Σ^*, then the following 3 statements are equivalent:
  1. L is accepted by a FA
  2. L is accepted by a NDFA
  3. L is accepted by a NDFA-Λ

Summary

- How we showed this
  1. Given an NDFA that accepts L, create an FA that also accepts L (subset construction)
  2. Given an NDFA- Λ that accepts L, create an NDFA that also accepts L (Λ-transition elimination)
  3. Given an FA that accepts L, create a NDFA- Λ that also accepts L.

After the break

- Now we have all the tools needed to prove Kleene's Theorem directly
- Will do this after the break

Kleene's Theorem

- To prove Kleene's Theorem, we will have two show two things:
  1. Given a Regular Expression, there is a FA that accepts the language described by the regular expression
  2. Given a FA, the language accepted by the FA can be expressed by a regular expression.

Pt 1: RE -> FA

- Since NDFA-Λs are equivalent to FAs w.r.t the class of languages they accept
  - We can, given an RE, build an NDFA-Λ instead of an FA that accepts the language described by the RE
  - We can always then convert that NDFA-Λ to an equivalent FA (using the algorithms presented in the first half)

Regular Expression

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

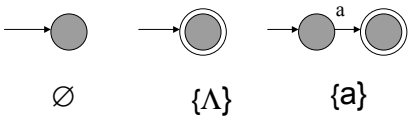
Regular Expression

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.

Pt 1: RE -> FA

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



Pt 1: RE -> FA

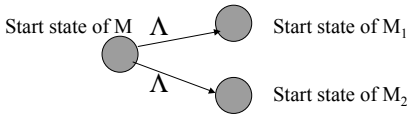
- Induction:
  - Assume R_1 and R_2 are regular expressions that describe languages L_1 and L_2. Then, by the induction hypothesis, there exists NDFA-Λ, M_1 and M_2 that accept L_1 and L_2
  - Create NDFA-Λ that accept the languages described by:
    - $R_1 + R_2$
    - $R_1 R_2$
    - R_1^*

Pt 1: RE -> FA

- Induction Hypothesis:
  - $L_1 = L(M_1)$ where $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$
  - $L_2 = L(M_2)$ where $M_2 = (Q_2, \Sigma, q_2, A_2, \delta_2)$
    - Assume Q_1 and Q_2 are disjoint
- Will build
  - $M_u = (Q_u, \Sigma, q_u, A_u, \delta_u)$ $L(M_u) = L_1 + L_2$
  - $M_c = (Q_c, \Sigma, q_c, A_c, \delta_c)$ $L(M_c) = L_1 L_2$
  - $M_k = (Q_k, \Sigma, q_k, A_k, \delta_k)$ $L(M_k) = L_1^*$

Pt 1: RE -> FA: Union

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



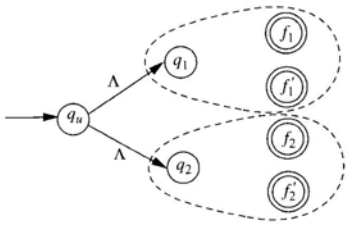
Pt 1: RE -> FA: Union

- 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.

Pt 1: RE -> FA: Union

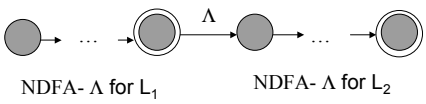
- Let's formalize this:
  - $M_u = (Q_u, \Sigma, q_u, A_u, \delta_u)$
  - $Q_u = Q_1 \cup Q_2 \cup \{q_u\}$
  - $A_u = A_1 \cup A_2$
  - Transition function: δ_u
    - $\delta_u (q_u, \Lambda) = \{q_1, q_2\}$
    - $\delta_u (q_u, a) = \varnothing$ for all $a \in \Sigma$
    - $\delta_u (q, a) = \delta_1 (q, a)$ if $q \in Q_1$
    - $\delta_u (q, a) = \delta_2 (q, a)$ if $q \in Q_2$

Pt 1: RE -> FA: Union



Pt 1: RE -> FA: 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 Λ transition.



Pt 1: RE -> FA: 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.

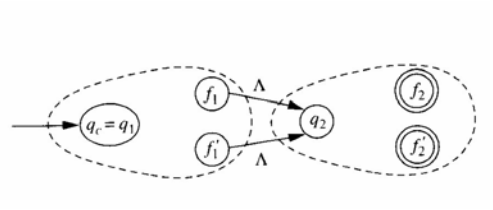
Pt 1: RE -> FA: Concatenation

- Let's formalize this:
  - $M_c = (Q_c, \Sigma, q_c, A_c, \delta_c)$
  - $Q_c = Q_1 \cup Q_2$
  - $Q_c = q_1$
  - $A_c = A_2$

Pt 1: RE -> FA: Concatenation

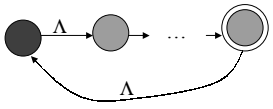
- Let's formalize this:
  - Transition function δ_c :
    - $\delta_c (q, a) = \delta_1 (q, a)$ if $q \in Q_1$
    - $\delta_c (q, a) = \delta_2 (q, a)$ if $q \in Q_2$
    - For all $q \in A_1$, $\delta_c (q, \Lambda) = \delta_1 (q, \Lambda) \cup \{q_2\}$

Pt 1: RE -> FA: Concatenation



Pt 1: RE -> FA: Kleene Star

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



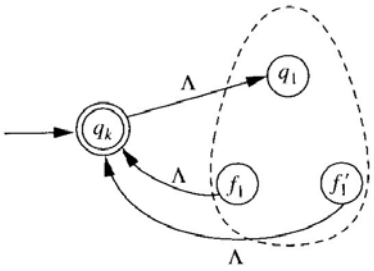
Pt 1: RE -> FA: Kleene Star

- 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 Λ transition.

Pt 1: RE -> FA: Kleene Star

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

Pt 1: RE -> FA: Kleene Star

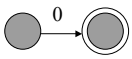


Pt 1: RE -> FA: Example

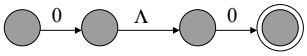
- Let's try an example
  - Create an NDFA-Λ for the regular expression:
    - $(00 + 1)^*(10)^*$

Pt 1: RE -> FA: Example

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

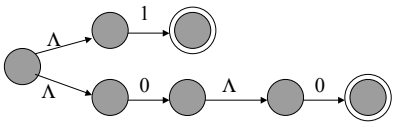


✓$(\underline{00} + 1)^*(10)^*$



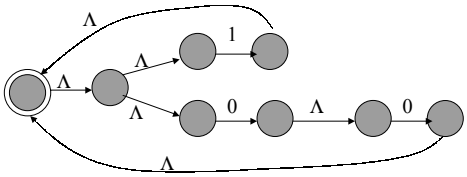
Pt 1: RE -> FA: Example

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



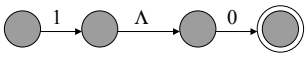
Pt 1: RE -> FA: Example

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

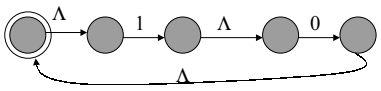


Pt 1: RE -> FA: Example

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

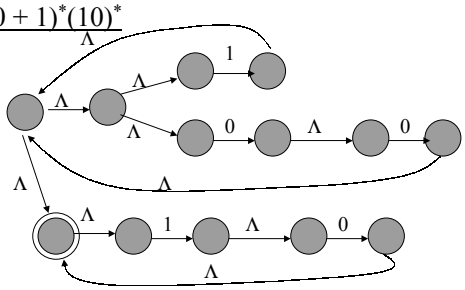


✓$(00 + 1)^*(\underline{10})^*$



Pt 1: RE -> FA: Example

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



Pt 1: RE -> FA: Summary

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

Pt 2: FA -> RE

- Given a FA, M, there is a regular expression R that described the language accepted by M.

Pt 2: FA -> RE

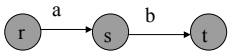
- Basic idea
  - This proof is surprisingly much trickier than the first
  - We will consider a path of states taken by the machine on a given input:
    - Will develop a recursive definition for a path from one state to another using the operations +, concatenation, Kleene Star
    - This will be sufficient, since the language accepted by the FA is merely the union of paths starting from the start state leading to accepting states, and that Regular Languages are closed under union (by definition).

Pt 2: FA -> RE

- Some definitions:
  - $M = (Q, \Sigma, q_o, A, \delta)$
  - L(p, q)
    - Set of all strings that, on input, takes you from state p to state q.
    - $\{x \in \Sigma^* \mid \delta^*(p,x) = q\}$

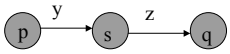
Pt 2: FA -> RE

- More definitions:
  - In creating the RE, we will actually consider paths that start at one state, ends at another, and goes through even another.



Pt 2: FA -> RE

- More definitions:
  - For a string x: we say that x represents a path from p to q going through s if:
    - There are non-null strings y,z such that
    - x = yz
    - δ^* (p, y) = s
    - δ^* (s, z) = q



Pt 2: FA -> RE

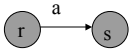
- Even more definitions
  - We will re-label each of the states of M to correspond with the integers 1…n, where n is the number of states
  - L(p,q, r)
    - Set of strings representing paths starting from p, ending at q, and going through no state numbered higher than r.

Pt 2: FA -> RE

- Finally
  - L(p, q, n) =
    - The set of strings represented by paths starting at p, ending at q, going through no state higher than n
    - L (p, q)
- We will show that
  - L(p, q, n) is regular
  - Will do so via induction on n
  - Questions?

Pt 2: FA -> RE

- Base step
  - Show L(p, q, 0) is regular
  - L(p, q, 0) =
    - Set of all strings represented by paths from p to q going through no state number higher than 0.
    - In other words, consists of paths of a single transition
    - This is clearly regular since this set is finite



Pt 2: FA -> RE

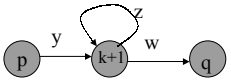
- Induction
  - Assume that L(p, q, k) is regular
  - L(p, q, k) =
    - Set of all strings represented by paths starting at p, ending at q, going through no state numbered higher than k.
  - Must show that L(p, q, k+1) is regular

Pt 2: FA -> RE

- Let's consider L(p, q, k+1)
  - Set of all strings represented by paths starting at p, ending at q, going through no state numbered higher than k+1.
  - For an $x \in L(p, q, k+1)$, there are two ways this can occur:
    - The path bypasses state k+1 altogether. In this case $x \in L(p, q, k)$
    - The path can start at p, get to state k+1, loop from k+1 back to k+1 (0 or more times), then finally get to q

Pt 2: FA -> RE

- Rewrite x as yzw where
  - y is a string represented by the path from p to k+1
  - z is a string represented by the looping from k+1 to itself
  - w is a string represented by the path from k+1 to q

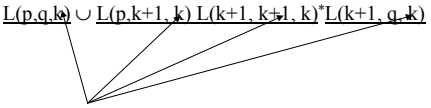


Pt 2: FA -> RE

- Take special note
  - y does not go through any state higher than k, so $y \in L(p, k+1, k)$
  - w does not go through any state higher than k, so $y \in L(k+1, q, k)$
  - A single loop of z does not go through any state higher than k, so $z \in L(k+1, k+1, k)^*$

Pt 2: FA -> RE

- Putting it all together
- $x \in L(p,q,k) \cup L(p,k+1,k)\, L(k+1,k+1,k)^* L(k+1,q,k)$



- Each of these are regular by the inductive hypothesis
- The set of strings x above describe L(p,q,k+1)
- We found an expression for L(p,q,k+1) using just union, concatenation, and Kleene star
- L(p,q,k+1) is regular.

Pt 2: FA -> RE

- Recursive Algorithm to find RE
  - Basic idea:
    - Calculate L(p,q,0)
    - Use recursive definition from proof to create L(p,q,k+1) from L(p,q,k)
    - Repeat until L(p,q,n) is calculated
    - Recall L(p,q,n) = L(p,q)
    - Take union of all $L(q_0, r)$ for all accepting states r.

Pt 2: FA -> RE

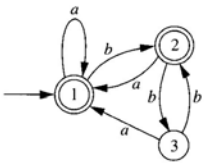
- Recursive Algorithm – Base cases :
  - L(p,p,0)
    - The set of all strings of a single character a, such that there is a transition from p to p on input a.
    - Λ
    - $\{a \in \Sigma \mid \delta(p,a) = p\} \cup \{\Lambda\}$
  - L(p,q,0)
    - The set of all strings of a single character a such that there is a transition from a p to q on input a
    - $\{a \in \Sigma \mid \delta(p,a) = q\}$

Pt 2: FA -> RE

- Recursive Algorithm – Recursion
- L(p,q,k+1) =
  - $L(p,q,k) \cup L(p,k+1,k)\,L(k+1,k+1,k)^{*}L(k+1,q,k)$

Pt 2: FA -> RE

- Let's look at an example



Pt 2: FA -> RE

- The set of all x accepted by this FA is:
  - Set of all strings starting at state 1, ending in states 1 or 2, going through no state higher than state 3.
    - $L(1,1,3) \cup L(1,2,3)$
  - We will construct these starting from base case.

Pt 2: FA -> RE

- L (p, q, 0)

	r(p,1,0)	r(p,2,0)	r(p, 3, 0)
1	$a + \Lambda$	b	$\varnothing$
2	a	Λ	b
3	a	b	Λ

Pt 2: FA -> RE

- L(p,q,1) =
  - $L(p,q,0) \cup L(p,1,0)\,L(1,1,0)^{*}L(1,q,0)$
- Let's try a few
  - L (1,1,1) =
    - $L(1,1,0) + L(1,1,0)L(1,1,0)^{*}L(1,1,0)$
    - $(a + \Lambda) + (a + \Lambda)(a + \Lambda)^{*}(a + \Lambda)$
    - a^{*}

Pt 2: FA -> RE

- Let's try a few
  - L (1,2,1) =
    - $L(1,2,0) + L(1,1,0)L(1,1,0)^{*}L(1,2,0)$
    - $b + (a + \Lambda)(a + \Lambda)^{*}b$
    - $a^{*}b$
  - L (3,2,1) =
    - $L(3,2,0) + L(3,1,0)L(1,1,0)^{*}L(1,2,0)$
    - $b + a(a + \Lambda)^{*}b$
    - $a^{*}b$

Pt 2: FA -> RE

- L(p,q,1) =
  - $L(p,q,0) \cup L(p,1,0)\, L(1,1,0)^* L(1,q,0)$

	r(p,1,1)	r(p,2,1)	r(p, 3, 1)
1	a^*	a^*b	$\varnothing$
2	aa^*	$\Lambda + aa^*b$	b
3	aa^*	a^*b	Λ

Pt 2: FA -> RE

- L(p,q,2) =
  - $L(p,q,1) \cup L(p,2,1)\, L(2,2,1)^* L(2,q,1)$

	r(p,1,2)	r(p,2,2)	r(p,3,2)
1	$a^*(baa^*)^*$	$a^*(baa^*)^*b$	$a^*(baa^*)^*bb$
2	$aa^*(baa^*)^*$	$(aa^*b)^*$	$(aa^*b)^*b$
3	$aa^* + a^*(baa^*)(baa^*)^*$	$a^*b(aa^*b)^*$	$\Lambda + a^*b\,(aa^*b)^*b$

Pt 2: FA -> RE

- L(p,q,3) =
  - $L(p,q,2) \cup L(p,3,2)\, L(3,3,2)^* L(3,q,2)$
  - We only need L(1, 1, 3) and L (1,2,3)
  - L(1,1,3) =
    - $L(1,1,2) + L(1,3,2)L(3,3,2)^*L(3,1,2)$
    - $a^*(baa^*)^* + a^*(baa^*)^*bb(\Lambda + a^*b\,(aa^*b)^*b)^*(aa^* + a^*(baa^*)\,(baa^*)^*)$
  - You get the idea?

Pt 2: FA -> RE

- What have we found?
  - Given an FA, we can find express the language accepted by the FA as a regular expression
  - I never claimed that this regular expression would be pretty!

Summary

- Pt 1
  - Given a regular language we built an NDFA-Λ that accepts the language
- Pt 2
  - Given a FA, we constructed a regular expression that describes the language accepted by the FA

Summary

- The proof of Kleene Theorem is complete!
- Questions

Next time

- Finding the FA with the minimal number of states.
- Look at the Question:
  - Are there languages that are not regular?