BBM402-Lecture 4: Regular expressions equivalence with NFAs, DFAs, closure properties of regular languages

Lecturer: Lale Özkahya

Resources for the presentation:
http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-045j-automata-computability-and-complexity-spring-2011/Syllabus/

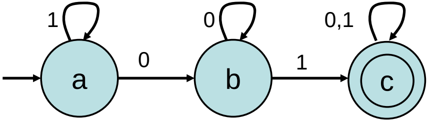
Closure under union

- Theorem: FA-recognizable languages are closed under union.
- Old Proof:
  - Start with DFAs M_1 and M_2 for the same alphabet Σ.
  - Get another DFA, M_3, with $L(M_3) = L(M_1) \cup L(M_2)$.
  - Idea: Run M_1 and M_2 "in parallel" on the same input. If either reaches an accepting state, accept.

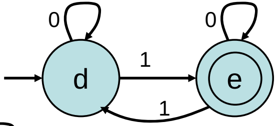
Closure under union

- Example:

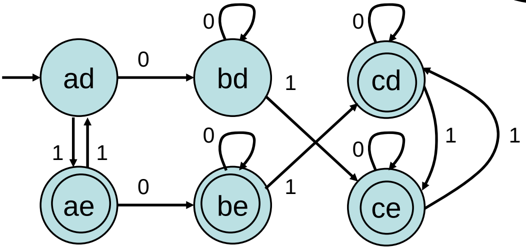
  M_1: Substring 01



  M_2: Odd number of 1s



  M_3:

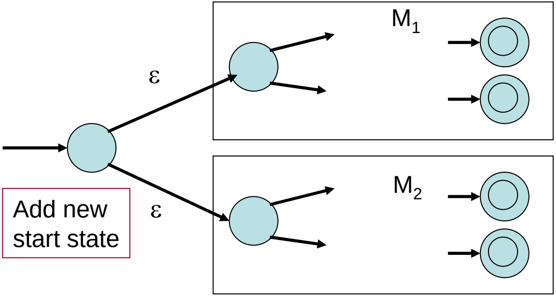


Closure under union, general rule

- Assume:
  - $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$
  - $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$
- Define $M_3 = (Q_3, \Sigma, \delta_3, q_{03}, F_3)$, where
  - $Q_3 = Q_1 \times Q_2$
    - Cartesian product, $\{(q_1,q_2) \mid q_1 \in Q_1 \text{ and } q_2 \in Q_2 \}$
  - $\delta_3 ((q_1,q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$
  - $q_{03} = (q_{01}, q_{02})$
  - $F_3 = \{ (q_1,q_2) \mid q_1 \in F_1 \text{ or } q_2 \in F_2 \}$

Closure under union

- Theorem: FA-recognizable languages are closed under union.
- New Proof:
  - Start with NFAs M_1 and M_2.
  - Get another NFA, M_3, with $L(M_3) = L(M_1) \cup L(M_2)$.



Use final states from M_1 and M_2.

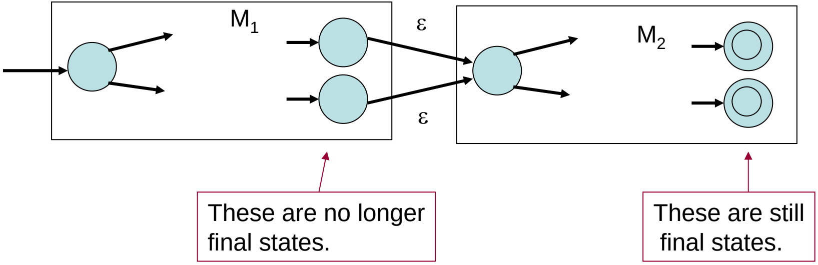
Closure under union

- Theorem: FA-recognizable languages are closed under union.
- New Proof: Simpler!

- Intersection:
  - NFAs don’t seem to help.
- Concatenation, star:
  - Now try NFA-based constructions.

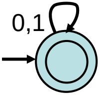
Closure under concatenation

- $L_1 \circ L_2 = \{ x\, y \mid x \in L_1 \text{ and } y \in L_2 \}$
- Theorem: FA-recognizable languages are closed under concatenation.
- Proof:
  - Start with NFAs M_1 and M_2.
  - Get another NFA, M_3, with $L(M_3) = L(M_1) \circ L(M_2)$.

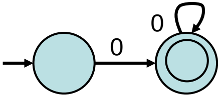


Closure under concatenation

- Example:
  - $\Sigma = \{ 0, 1\}$, $L_1 = \Sigma^*$, $L_2 = \{0\} \{0\}^*$.
  - $L_1 L_2$ = strings that end with a block of at least one 0
  - M_1:

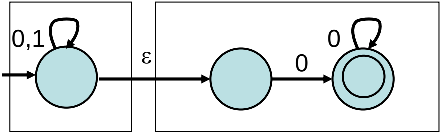


  - M_2:



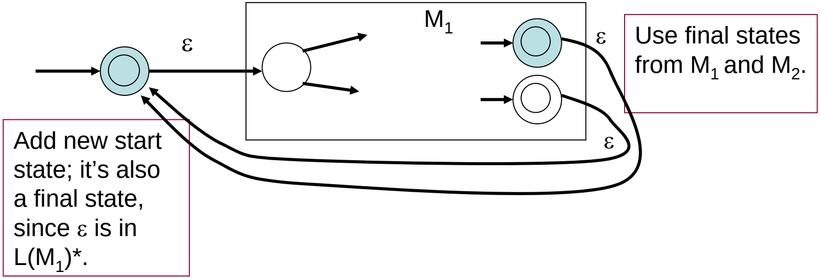
NFAs

  - Now combine:



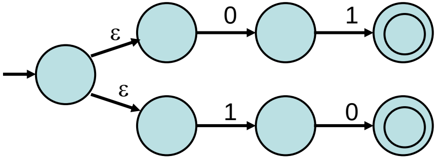
Closure under star

- $L^* = \{ x \mid x = y_1\, y_2\, \ldots\, y_k$ for some $k \geq 0$, every y in $L \}$
  $= L^0 \cup L^1 \cup L^2 \cup \ldots$
- Theorem: FA-recognizable languages are closed under star.
- Proof:
  - Start with FA M_1.
  - Get an NFA, M_2, with $L(M_2) = L(M_1)^*$.

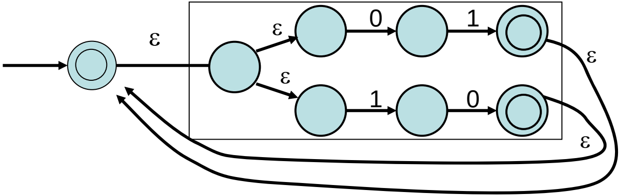


Closure under star

- Example:
  - $\Sigma = \{ 0, 1\}$, $L_1 = \{ 01, 10 \}$
  - $(L_1)^*$ = even-length strings where each pair consists of a 0 and a 1.
  - M_1:



  - Construct M_2:

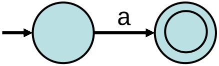


Languages denoted by regular expressions

- The languages denoted by regular expressions are exactly the regular (FA-recognizable) languages.
- Theorem 1: If R is a regular expression, then L(R) is a regular language (recognized by a FA).
- Proof: Easy.
- Theorem 2: If L is a regular language, then there is a regular expression R with L = L(R).
- Proof: Harder, more technical.

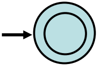
Theorem 1

- Theorem 1: If R is a regular expression, then L(R) is a regular language (recognized by a FA).
- Proof:
  - For each R, define an NFA M with L(M) = L(R).
  - Proceed by induction on the structure of R:
    - Show for the three base cases.
    - Show how to construct NFAs for more complex expressions from NFAs for their subexpressions.
  - Case 1: R = a
    - L(R) = { a }



Accepts only a.

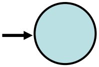
  - Case 2: R = ε
    - L(R) = { ε }



Accepts only ε.

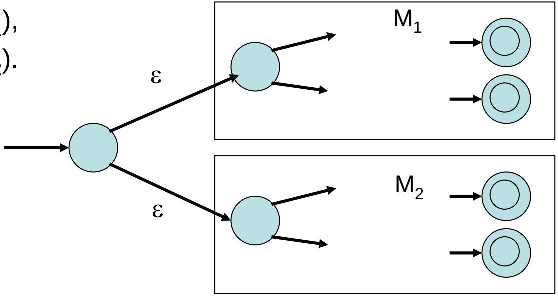
Theorem 1

- Theorem 1: If R is a regular expression, then L(R) is a regular language (recognized by a FA).
- Proof:
  - Case 3: $R = \emptyset$
    - $L(R) = \emptyset$



Accepts nothing.

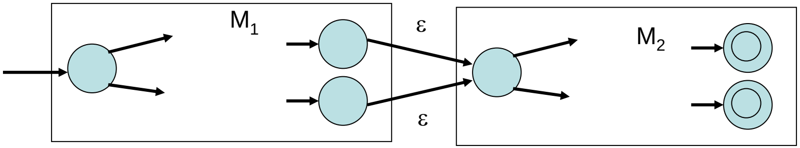
  - Case 4: $R = R_1 \cup R_2$
    - M_1 recognizes $L(R_1)$,
    - M_2 recognizes $L(R_2)$.
    - Same construction we used to show regular languages are closed under union.



Theorem 1

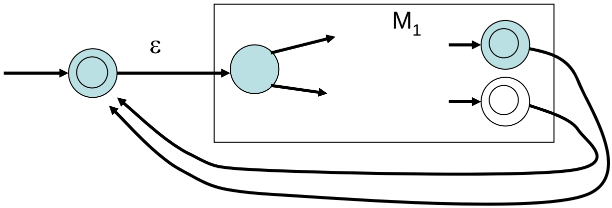
- Theorem 1: If R is a regular expression, then L(R) is a regular language (recognized by a FA).
- Proof:
  - Case 5: $R = R_1 \circ R_2$
    - M_1 recognizes $L(R_1)$,
    - M_2 recognizes $L(R_2)$.

    - Same construction we used to show regular languages are closed under concatenation.

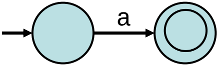


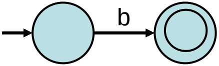
Theorem 1

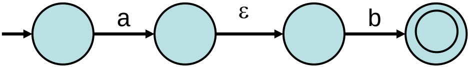
- Theorem 1: If R is a regular expression, then L(R) is a regular language (recognized by a FA).
- Proof:
  - Case 6: $R = (R_1)^*$
    - M_1 recognizes $L(R_1)$,
    - Same construction we used to show regular languages are closed under star.

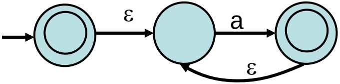


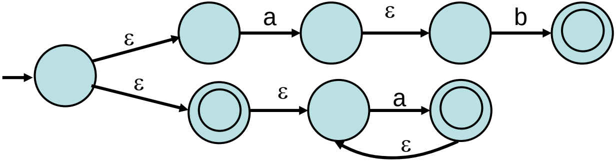
Example for Theorem 1

- L = ab ∪ a*
- Construct machines recursively:
- a: 

  b: 

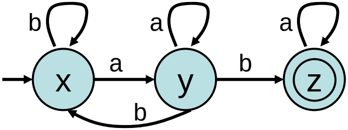
- ab: 

- a*: 

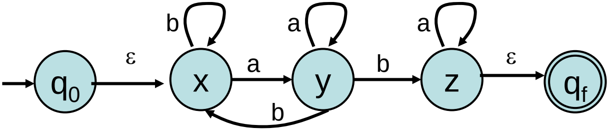
- ab ∪ a*: 

Theorem 2

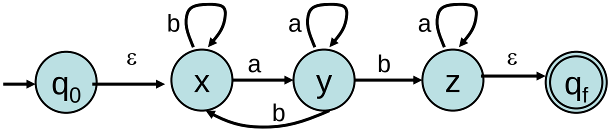
- Theorem 2: If L is a regular language, then there is a regular expression R with L = L(R).
- Proof:
  - For each NFA M, define a regular expression R with L(R) = L(M).
  - Show with an example:



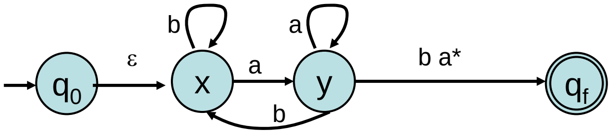
  - Convert to a special form with only one final state, no incoming arrows to start state, no outgoing arrows from final state.



Theorem 2

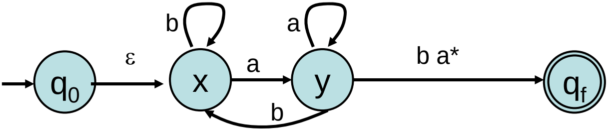


- Now remove states one at a time (any order), replacing labels of edges with more complicated regular expressions.
- First remove z:

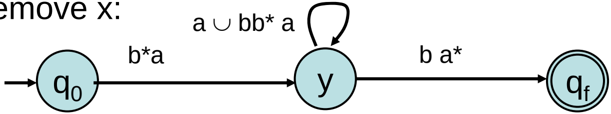


- New label b a* describes all strings that can move the machine from state y to state q_f, visiting (just) z any number of times.

Theorem 2

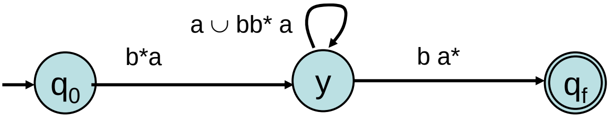


- Then remove x:

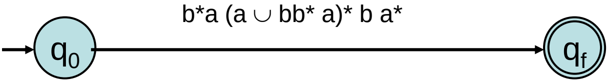


- New label b*a describes all strings that can move the machine from q_0 to y, visiting (just) x any number of times.
- New label a ∪ bb* a describes all strings that can move the machine from y to y, visiting (just) x any number of times.

Theorem 2



- Finally, remove y:



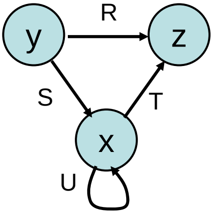
- New label describes all strings that can move the machine from q_0 to q_f, visiting (just) y any number of times.
- This final label is the needed regular expression.

Theorem 2

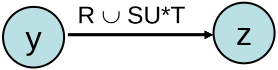
- Define a generalized NFA (gNFA).
  - Same as NFA, but:
    - Only one accept state, $\neq$ start state.
    - Start state has no incoming arrows, accept state no outgoing arrows.
    - Arrows are labeled with regular expressions.
  - How it computes: Follow an arrow labeled with a regular expression R while consuming a block of input that is a word in the language L(R).
- Convert the original NFA M to a gNFA.
- Successively transform the gNFA to equivalent gNFAs (recognize same language), each time removing one state.
- When we have 2 states and one arrow, the regular expression R on the arrow is the final answer:

Theorem 2

- To remove a state x, consider every pair of other states, y and z, including y = z.
- New label for edge (y, z) is the union of two expressions:
  - What was there before, and
  - One for paths through (just) x.
- If y $\neq$ z:



we get:



- If y = z: