Session 6

The language accepted by a FA

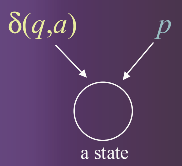
Transition Function δ

z	p_j	…	p_l
…	…	…	…
a	p_i	…	p_k
Σ \ Q	q_0	…	q_n

For any q of Q and $a \in \Sigma$, $\delta(q,a) = p$

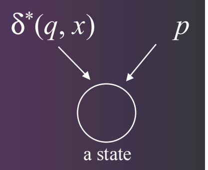
Extended Transition Function

- δ allows to move from one state to the next on an input symbol
- δ provides a mean to refer or name the next state in terms of the current state and an input symbol:
  - $\delta(q,a) = p$
- Reference relation:



Extended Transition Function

- It would be nice to have a way to refer or name the next state in terms of the current state and *an input string*:
  - $\delta^*(q, x) = p$
- The state we get to from a given state and an input string
- The extended transition function δ^* such a naming device!



Extended Transition Function δ^*

…	…	…	…
x_1	p_{01}	…	p_{n1}
$x_0 = \Lambda$	p_{00}	…	p_{n0}
Σ^* \ Q	q_0	…	q_n

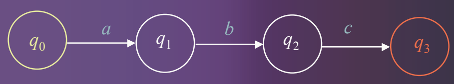
For any q of Q and $x \in \Sigma^*$, $\delta^*(q, x) = p$

Formal definition of δ^*

- Let M $= (Q, \Sigma, q_0, A, \delta)$ a FA
- We define the function $\delta^*: Q \times \Sigma^* \rightarrow Q$ as follows:
  - For any $q \in Q$, $\delta^*(q, \Lambda) = q$
  - For any $q \in Q$ and any $y \in \Sigma^*$ and a symbol $a \in \Sigma$

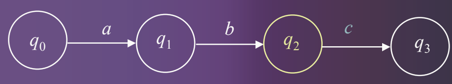
$$\delta^*(q, ya) = \delta(\delta^*(q, y), a)$$

Operation of δ^*



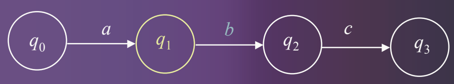
- $\delta^*(q_0, abc) = q_3$

Operation of δ^*



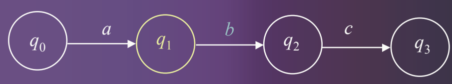
- $\delta^*(q_0, abc) = \delta(\delta^*(q_0, ab), c)$

Operation of δ^*



- $\delta^*(q_0, abc) = \delta(\delta^*(q_0, ab), c)$
  $= \delta(\delta(\delta^*(q_0, a), b), c)$

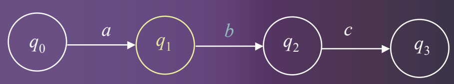
Operation of δ^*



- $\delta^*(q_0, abc) = \delta(\delta^*(q_0, ab), c)$
  $= \delta(\delta(\delta^*(q_0, a), b), c)$
- But δ^* is not defined for strings of a single character:

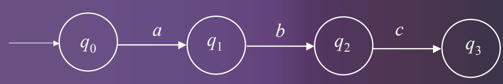
  $\delta^*(q, ya) = \delta(\delta^*(q, y), a)$

Operation of δ^*



- $\delta^*(q_0, abc) = \delta(\delta^*(q_0, ab), c)$
  $= \delta(\delta(\delta^*(q_0, a), b), c)$
  $= \delta(\delta(\delta^*(q_0, \Lambda a), b), c)$
- But … we take advantage of the identity of concatenation

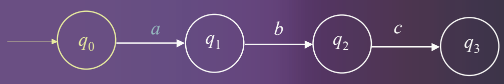
Operation of δ^*



- $\delta^*(q_0, abc) = \delta(\delta^*(q_0, ab), c)$
- $= \delta(\delta(\delta^*(q_0, a), b), c)$
- $= \delta(\delta(\delta^*(q_0, \Lambda a), b), c)$
- $= \delta(\delta(\delta(\delta^*(q_0, \Lambda), a), b), c)$
- Using the definition of δ^*:

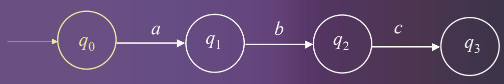
  $\delta^*(q, ya) = \delta(\delta^*(q, y), a)$

Operation of δ^*



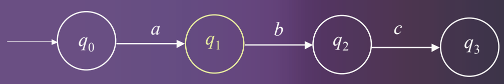
- $\delta^*(q_0, abc) = \delta(\delta^*(q_0, ab), c)$
- $= \delta(\delta(\delta^*(q_0, a), b), c)$
- $= \delta(\delta(\delta^*(q_0, \Lambda a), b), c)$
- $= \delta(\delta(\delta(\delta^*(q_0, \Lambda), a), b), c)$
- Nothing unusual: With the empty string we get to the initial state!

Operation of δ^*



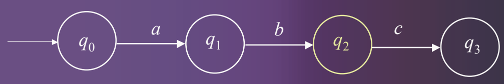
- $\delta^*(q_0, abc) = \delta(\delta^*(q_0, ab), c)$
- $= \delta(\delta(\delta^*(q_0, a), b), c)$
- $= \delta(\delta(\delta^*(q_0, \Lambda a), b), c)$
- $= \delta(\delta(\delta(\delta^*(q_0, \Lambda), a), b), c)$
- $= \delta(\delta(\delta(q_0, a), b), c)$
- Using base condition: $\delta^*(q, \Lambda) = q$
- Now, we have reduce δ^* to a composition of δs

Operation of δ^*



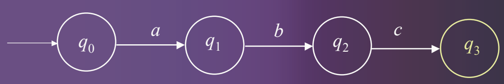
- $\delta^*(q_0, abc) = \delta(\delta^*(q_0, ab), c)$
- $= \delta(\delta(\delta^*(q_0, a), b), c)$
- $= \delta(\delta(\delta^*(q_0, \Lambda a), b), c)$
- $= \delta(\delta(\delta(\delta^*(q_0, \Lambda), a), b), c)$
- $= \delta(\delta(\delta(q_0, a), b), c)$
- $= \delta(\delta(q_1, b), c)$

Operation of δ^*



- $\delta^*(q_0, abc) = \delta(\delta^*(q_0, ab), c)$
- $= \delta(\delta(\delta^*(q_0, a), b), c)$
- $= \delta(\delta(\delta^*(q_0, \Lambda a), b), c)$
- $= \delta(\delta(\delta(\delta^*(q_0, \Lambda), a), b), c)$
- $= \delta(\delta(\delta(q_0, a), b), c)$
- $= \delta(\delta(q_1, b), c)$
- $= \delta(q_2, c)$

Operation of δ^*



- $\delta^*(q_0, abc) = \delta(\delta^*(q_0, ab), c)$
- $= \delta(\delta(\delta^*(q_0, a), b), c)$
- $= \delta(\delta(\delta^*(q_0, \Lambda a), b), c)$
- $= \delta(\delta(\delta(\delta^*(q_0, \Lambda), a), b), c)$
- $= \delta(\delta(\delta(q_0, a), b), c)$
- $= \delta(\delta(q_1, b), c)$
- $= \delta(q_2, c)$
- $= q_3$

A consequence of δ^*

- Composition of strings transitions:

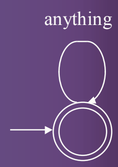
$$\delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$$

Acceptance by a FA

- Let $M = (Q, \Sigma, q_0, A, \delta)$ be a FA
- $x \in \Sigma^*$ is *accepted* by M if $\delta^*(q_0, x) \in A$
- If a string is not accepted, it is rejected by M
- The language *accepted* or *recognized* by M is the set
  - $\mathbf{L}(M) = \{x \in \Sigma^* \mid x \text{ is accepted by } M\}$
- If L is any language over Σ, L is accepted, or recognized, by M if and only if $L = \mathbf{L}(M)$

But …

- The definition does not say that L is accepted by M if every string is accepted by M
- In this case, M could also accept other things!



- In order to accept a language a FA has to
  - Accept all strings in L
  - Reject all strings in its complement

THEOREM:

A L language over the alphabet Σ is regular if and only if there is an FA with input alphabet Σ that accepts L

Kleene'sTheorem:

- If this is true (and it is), regular expressions and FA are equivalent representations: if a *RE* *denotes* a language, there is a FA that accepts such a language!
  - *RE* are declarative representations
  - FA are procedural representations

Distinguishing strings

- Let L be a language in Σ^*, and $x \in \Sigma^*$
- Let L/x a set of strings such that:
  - $L/x = \{z \in \Sigma^* \mid xz \in L\}$
- Two strings x and y are distinguishable with respect to L if
  - $L/x \neq L/y$
- Any string z such that $xz \in L$ and $yz \notin L$ or vice versa, is said to distinguish x and y with respect to L
- If $L/x = L/y$, x and y are indistinguishable (or equivalent) with respect to L

Distinguishing strings

- To show that x and y are distinguishable with respect to L
  - Find z such that $xz \in L$ BUT $yz \notin L$ OR
  - Find z such that $yz \in L$ BUT $xz \notin L$
  - So z is in L/x or L/y but not in the other!
- Example:
  - $L = \{w \mid |w| \text{ is even}\}$
  - Any string z of an even length will distinguish the strings of even length in L, from the strings of odd length, not in L
  - $L/x = \{z \in \Sigma^* \mid xz \in L\}$
  - i.e. If $|x|$ is odd, then $|xz|$ is also odd ($|z|$ is even), and $xz \notin L$

Distinguishing strings

- Lemma:
  - Suppose $L \subseteq \Sigma^*$ and $M = (Q, \Sigma, q_0, A, \delta)$ is a FA recognizing L.
  - *If* x and y are two strings in Σ^* that are distinguishable with respect to L, then

    $\delta^*(q_0, xz) \neq \delta^*(q_0, yz)$
  - One is state is accepting but not the other!

Proof of lemma

- One is accepting, but not the other:
  - $\delta^*(q_0, xz) \neq \delta^*(q_0, yz)$
- The state δ^* takes us from q_0 on xz and on yz are:
  - $\delta^*(q_0, xz) = \delta^*(\delta^*(q_0, x), z)$
  - $\delta^*(q_0, yz) = \delta^*(\delta^*(q_0, y), z)$
- But if z is a distinguishing string:
  - $\delta^*(\delta^*(q_0, x), z) \neq \delta^*(\delta^*(q_0, y), z)$
- Then
  - $\delta^*(q_0, x) \neq \delta^*(q_0, y)$
- This is: x and y take M from q_0 to different states!

THEOREM

- Suppose $L \subseteq \Sigma^*$ and for some n, there are n strings in Σ^*, any two of which are distinguishable with respect to L.
- Then, every FA recognizing L must have at least n states.

Proof of Theorem

- Suppose $x_1, x_2, \ldots, x_n$ are n strings, any two of which are distinguishable with respect to L
- Then, if $M = (Q, \Sigma, q_0, A, \delta)$ is any FA with fewer than n states then, by the pigeonhole principle, the states
  - $\delta^*(q_0, x_1)$
  - $\delta^*(q_0, x_2)$
  - ….
  - $\delta^*(q_0, x_n)$

  are not all distinct, so
  - For some $i \neq j$, $\delta^*(q_0, x_i) = \delta^*(q_0, x_j)$
- But since x_i and x_j are distinguishable with respect to L, it follows from the lemma that M cannot recognize L

Showing languages not to be regular!

- The theorem about distinguishing strings offers a potential way to show that a language is not regular
- If there is an infinite number of strings, any two distinguishable with respect to L, any FA accepting L must have an infinite number of states
- But this is what a FA is not allowed to have!
- So, no FA can accept a language consisting of an infinite set of distinguishable strings!

A language that cannot be accepted by a FA

- The language *pal* over $\Sigma = \{0, 1\}$ cannot be accepted by a FA and therefore is not regular
  - Palindrome: *madamimadam*
  - Palindromes: Λ, 0, 1, 0110, 11011
- Proof: We show that any two strings x and y in $\{0, 1\}^*$ are distinguishable with respect to *pal*
- Case 1: $|x| = |y|$ and let $z = x^r$ (e.g. if $x = 0101$ $y = 1010$)
  - Then $xz = xx^r$ which is a pal: (e.g. $z = x^r = 1010$)

    $xz = 01011010$
  - But yz is not

    $yz = 10101010$
  - $z = x^r$ distinguishes x from y

A language that cannot be accepted by a FA

- Case 2: $|x| \neq |y|$ and assume $|x| < |y|$ (e.g. x = 010 y = 10101)
  - Let $y = y_1y_2$ where $|y_1| = |x|$ (e.g. y_1 = 101 y_2 = 01)
  - We look for a z such that $xz \in pal$ but $yz \notin pal$
    - Any z of the form $z = ww^rx^r$ satisfies $xz \in pal$
    - Consider that: $xz = xww^rx^r$ (e.g. xPx^r = 010P010)
    - To make sure that $yz \notin pal$, $w \neq y_2$ but $|w| = |y_2|$
    - Let's chose w = 10
  - Then:

    $xz = xww^rx^r$ (e.g. $xwwx^r$ = 0101001010)
  - But

    $yz = y_1y_2z = y_1y_2ww^rx^r$ (e.g. 101011001010)
  - $z = ww^rx^r$ distinguishes x from y

Distinguishing strings

- How many different classes are there in a language?
- For $L = \{w \mid |w| \text{ is even}\}$ we need just one z
  - any z of even length will do for all strings in the language!
  - In this language there are just two classes
- For the language pal of all palindromes over $\Sigma = \{0, 1\}$
  - we need a different z for each palindrome
  - there is an infinite number of palindromes
  - There is an infinite number of classes!
- If there are n classes of strings, we need n z's to distinguish them!

The languages accepted by FA