16.070

Introduction to Computers & Programming

Theory of computation: Sets, DFA, NFA

Prof. Kristina Lundqvist
Dept. of Aero/Astro, MIT

Set Theory

- A set is an **unordered collection** of objects. We use the notation $\{ob_1, ob_2, \ldots\}$ to denote a set where the ob_i are the objects in the set.

  eg: The set of all positive integers is $Z^+ = \{1, 2, 3, ...\}$

- The objects in a set are called the **elements** or **members** of the set. We say that a set **contains** its elements

  eg: 1, 2, 3, … are the elements of the set Z^+

- A set is defined in such general terms can cause problems. For this reason, this is called Naïve Set Theory.

Useful Sets

- The Set of **Natural** Numbers: $N = \{0, 1, 2, \ldots\}$
- The Set of **Integers**: $Z = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$
- The Set of **Positive Integers**: $Z^+ = \{1, 2, 3, \ldots\}$
- The Set of **Rational** Numbers:

  $$Q = \{p/q \mid p \text{ and } q \text{ are integers and } q \neq 0\}$$
- The Set of **Real** Numbers: $R = Q \cup Q'$
- A set with no members is called an **empty set** the symbol f is used to denote the empty set.

  - What is $\{f\}$?

Subset and Equivalence

- The set A is called a subset of the set B if and only if every element of A is also an element of B. The notation $A \subseteq B$ is used to indicate that A is a subset of B.

  Restated: $A \subseteq B$ iff $\forall x(x \in A \rightarrow x \in B)$

  eg: $\{1, 3, 5\} \subseteq \{1, 2, 3, 4, 5\}$ since every element in the first set is also a member of the second set

  eg: $\{6, 2, 4\} \subseteq \{4, 6, 2\}$. [In fact the two sets are equal.]

- Two sets A and B are equal if and only if $A \subseteq B$ and $B \subseteq A$. That is, when every member of A is also a member of B and when every member of B is also a member of A, then A and B have the same members. This is a very important technique that we use to prove that two sets are equal: show $A \subseteq B$ and show $B \subseteq A$.

n-tuples & Cartesian Product

- The ordered n-tuple $(a_1, a_2, \ldots, a_n)$ is the ordered collection that has a_1 as its first element, a_2 as its second element, …, and a_n as its nth element. Two ordered n-tuples are equal if and only if their first elements are equal, their second elements are equal, …, and their nth elements are equal.
- Let A and B be sets. The Cartesian product of A and B, denoted by $A \times B$ is the set of all ordered pairs (a, b) where $a \in A$ and $b \in B$. That is:

  $A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$

  Given: $A = \{1, 2\}$ and $B = \{a, b, c\}$

  $A \times B = \{(1, a), (1, b), (1, c), (2, a), (2, b), (2, c)\}$

Union & Intersection

- Let A and B be sets. The union of the sets A and B, denoted by $A \cup B$, is the set that contains those elements that are either in A or in B, or in both. That is,

  $$A \cup B = \{ x \mid x \in A \vee x \in B \}$$

  The union of {1, 3, 5} and {1, 2, 3} is {1, 2, 3, 5}

- _Let A and B be sets. The intersection of the sets A and B, denoted by $A \cap B$, is the set that contains those elements that are in both A and B. That is,

  $$A \cap B = \{ x \mid x \in A \wedge x \in B \}$$

  The intersection of {1, 3, 5} and {1, 2, 3} is {1, 3}

Functions

- Let A and B be sets. A mapping m from A to B is a subset of $A \times B$. We denote that m is a mapping from A to B by $m: A \Rightarrow B$

  Let A = {1, 2, 3} and B = {a, b, c}.

  $A \times B$ = {(1, a), (1, b), (1, c), (2, a), (2, b), (2, c), (3, a), (3, b), (3, c)}.
- m = {(1, a), (1, b), (2, a), (2, c)} is a mapping from A to B

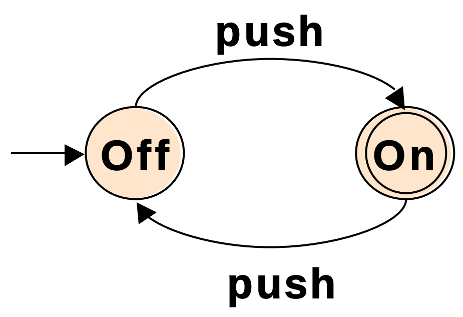
Kleene Star

- We can then define the Kleene Star A* of A as

$$A^* := \cup_{n \geq 0} A^n$$

Finite State Automata

- The FSA model seen so far is deterministic (DFA), exactly one transition for each given symbol and state.


$$\begin{tikzcd} \arrow[r] & \text{Off} \arrow[r, bend left, "push"] & \text{On} \arrow[l, bend left, "push"] \end{tikzcd}$$

- A **Model of Computation** consists of:
  - A set of **states**
  - An input **alphabet**
  - A **transition function** that maps input symbols and current states to a **next state**
  - A **start state**
  - **Accepting states**

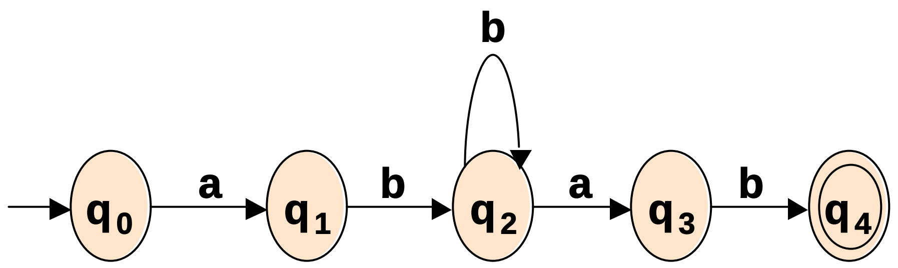
5-tuple $(Q, \Sigma, \delta, q_0, F)$

({On, Off}, {push}, {(On, push)→ Off, (Off, push)→ On}, Off, {On})

Formal Definition of Computation

- Let **M** = $(Q, \Sigma, \delta, q_0, F)$

- Let **w** = $w_1 w_2 \ldots w_n \in \Sigma^n$



- Then **M** accepts **w** ***iff*** there exists a sequence of states **$(r_0, r_1, \ldots, r_n)$ Î Q^n**

  1. $r_0 = q_0$
  2. $\delta(r_{i-1}, w_i) = r_i$ for all i = 1, 2, …, n
  3. $r_n \in F$

- We can formally define the Language **L(M)** accepted by automaton **M** as: **L(M) := { w Î** S* | **M** accepts **w }**

Operations on Languages

- We defined a (formal) language **L** over an alphabet S as a set of words: **L Ì S***

  - Let **A Ì** S* ***then***
    A := { w Î S* **| w Ï A }** ***or***
    A = S* \ **A**

  - Let **A, B Ì** S* be languages over the same alphabet.
    Then we define the:
    - **Intersection A Ç B** of **A** and **B** ***as***
      A Ç **B := {w Î** S* **| w Î A** Ù **w Î B }**
    - **Union A È B** of **A** and **B** ***as***
      A È **B := {w Î** S* **| w Î A** Ú **w Î B }**

Operations on Languages

- *Concatenation*

  Let **x_1, x_2** Î S^* ***then***

  - ***If*** **x_1** Î S^0**, i.e., x_1 =** e**,** ***then*** **x_1x_2 := x_2**
  - ***If*** **x_1** Î S^* \ **{**e**}, i.e., x_1 is not** the empty word;
    split **x_1** into a character **a** Î S and a word **x'$_1$** Î S^* **: x_1 = ax'$_1$**
    then: **x_1 x_2 = (ax'$_1$) x_2 = a(x'$_1$$x_2$)**

Example:

If **x_1 = a_1 a_2… a_n** ***and*** **x_2 = b_1 b_2 …b_m**
then **x_1 x_2 = a_1 a_2 … a_n b_1 b_2 …b_m**

Operations on Languages

- From the formal definition of **concatenation** we can derive its following two properties
  - **Associativity**: *If* **a, b, c** ∈ S^* are words over the same alphabet, *then* **a(bc) = (ab)c**
  - **Identity element**: *if* **a** ∈ S^* is a word, *then* **a = ea = ae**

- Two more operations on languages
  - Let **A, B** Ì S^* be languages over the same alphabet. Then we define the concatenation **AB** of **A** and **B** as **AB := {ab | a** Î **A** Ù **b** Î **B}**.
  - Let **A** Ì S^* be a language. Then we define the sets A^n recursively for all **n >= 0**:
    - $A^0 := \{e\}$
    - $A^{n+1} := A^nA$

In other words, A^n is the set of all words formed by taking any sequence $a_1, a_2, \ldots, a_n \in A$ of *n* words from A and concatenating them.

Closure of regular languages

- The claim is that applying any of these operations to a regular language creates another regular language; in other words, *the class of regular languages is* ***closed*** *under these operations*.

Nondeterministic finite state automata

- A finite state machine/automata whose transition function maps input symbols and states to a possibly empty set of next states. The transition may also map the null symbol (no input symbol needed) and states to next state.

- There are three differences between the transition function of an NFA and that of a DFA
  1. There can be *states with more than one arrow leaving for the same input symbol*
  2. There can be *states with no arrows leaving for an input symbol*
  3. There can be arrows labeled with the special symbol e (the null symbol)

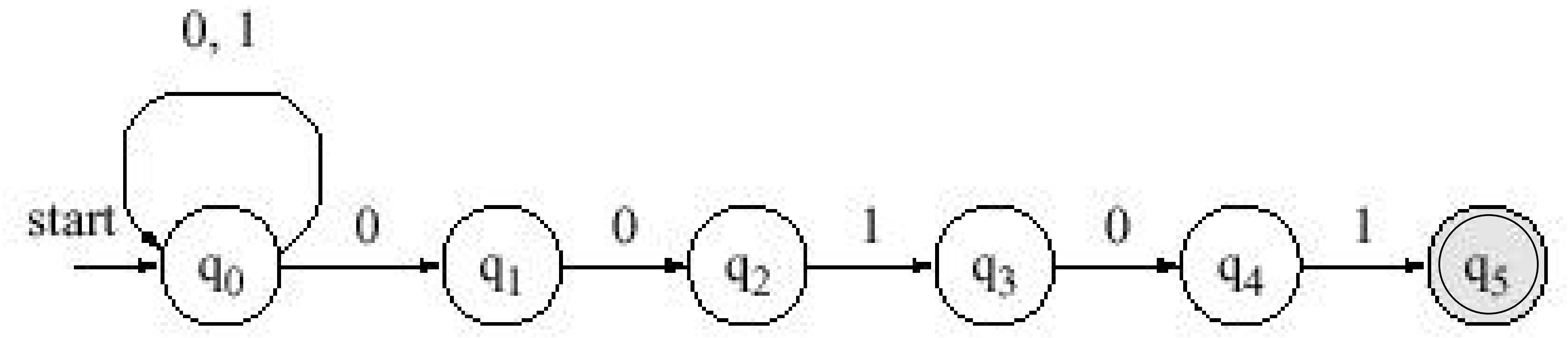
Non-Deterministic Languages

- Input string ***x*** is **accepted** by a nondeterministic FSA **if there is a set of transitions** (alternatively there is a path in the FSA graph) on input ***x*** that allows the NFA to reach an accepting state.

- The **language *L* recognized** by a nondeterministic FSA is the set of input strings accepted by it.

- Later we show that deterministic and nondeterministic FSAs recognize the same languages.

A Nondeterministic FSA

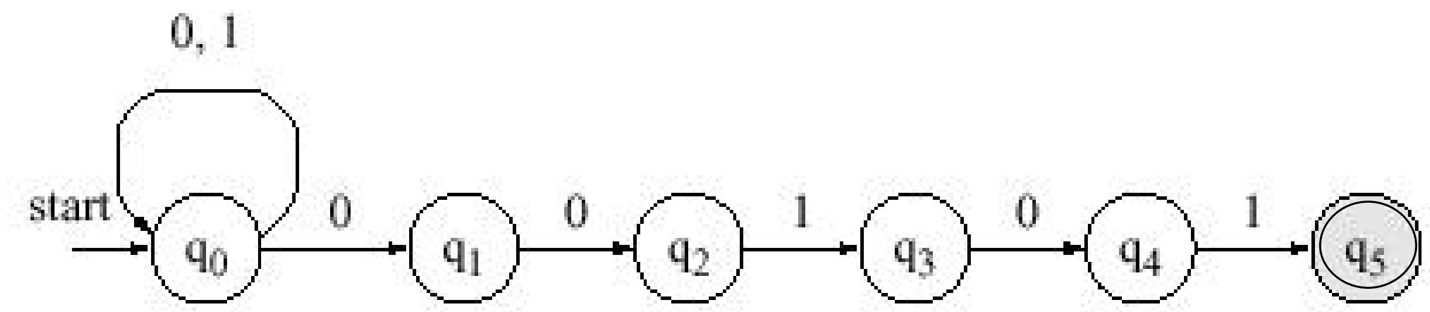


- Let 0,1 be input alphabet. If an NFA doesn’t have an edge labeled 0 (1) from state q, then 0 (1) is **rejected** at that state.

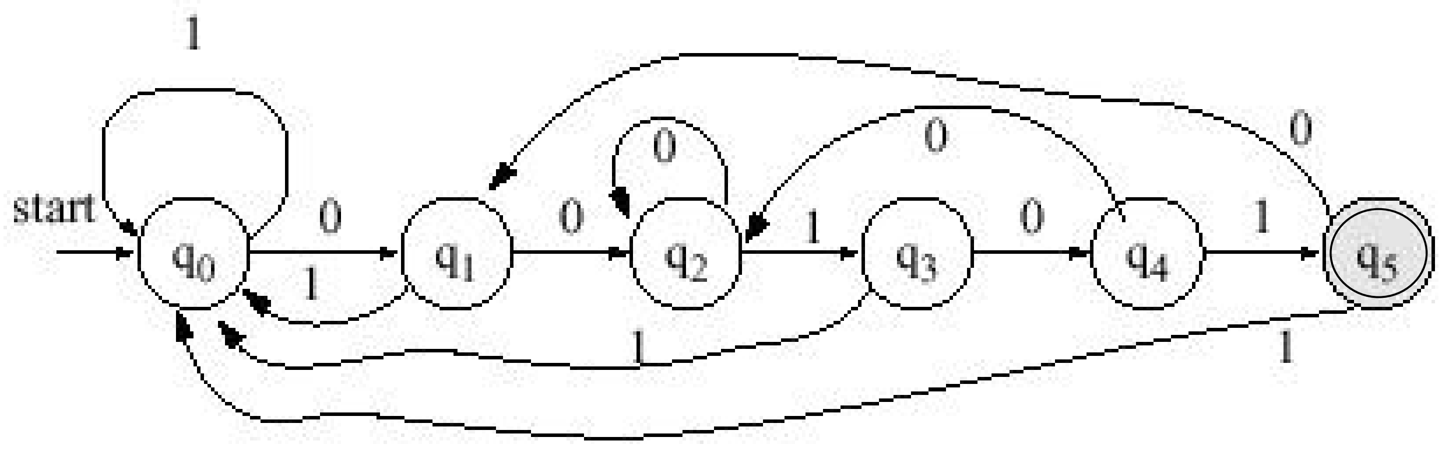
- Note that this machine accepts 00101, 000101, and 10100100101, among others.

- Clearly it accepts strings ending with 00101

A Nondeterministic FSA



- The equivalent DFA is shown below.



Equivalence of NFAs and DFAs

- By definition, every DFA is also an NFA; thus the class of DFAs is a subset of the class of NFAs.

  For the same reason L(DFA) $\subset$ L(NFA).

Equivalence of Regular Expressions and FSAs

- Earlier we have claimed that the class of languages that can be described by regular expressions is exactly the class of regular languages.

- *Proof*:
  - Show that the language L(R) generated by any regular expression R is accepted by some NFA M.
  - Show that the language L(M) accepted by any automaton M is generated by some regular expression R.