CS 9601 T W Lam 林德華

- Theory of Computation
  - Automata, languages & complexity ~50%
  - Advanced undergraduate/first-year graduate level
  - Recommended reference: Sipser's book

- Advanced algorithms
  - Online algorithms, online scheduling ~25%
  - Data structures for text indexing ~25%
  - Graduate level
  - References: research papers

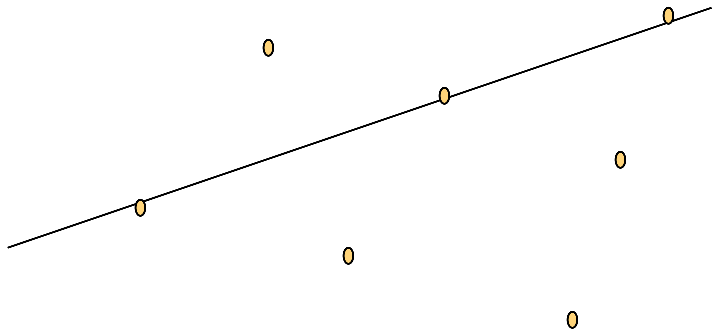
About TW Lam: www.cs.hku.hk/~twlam

Prerequisite

- undergraduate-level discrete math, data structures & algorithms
- Examples:
  - Logic, universal & existential quantification (for all, there exists), set theory, induction, proof by contradiction, counting, discrete probability, …
  - trees & graphs, graph algorithms, hashing, balanced search trees, string matching, dynamic programming, recursion, recurrence, greedy algorithms …

Are you ready for this course?

- Given a finite set of *n* points on the 2-d plane with the following property.
  - For any two points x, y in *A*, the line containing x and y must contain another point z in *A*.



- To prove: All points in *A* are on the same line.

All points in A on the same line !?

Is the following induction proof correct?

If not, where is the bug?

- By induction on the number of points in A.
  - Basis: $|A| = 3$. Trivial.
  - Assume the statement is true for $|A| = k >= 3$.
  - Consider the case when $|A|=k+1$.
    - Pick A' of k points of A. Let x be the remaining point.
    - Induction hypothesis: All points in A' on the same line.
    - Pick a point y in A', the x-y line must contain another point z in A'.
    - Thus, x, y & z are on the same line.
    - x and all points in A' on the same line.

there is something wrong within the second line

A simple observation

Consider a line L passing through 3 or more points, say, a, b, and c.

Let d be a point not on L.

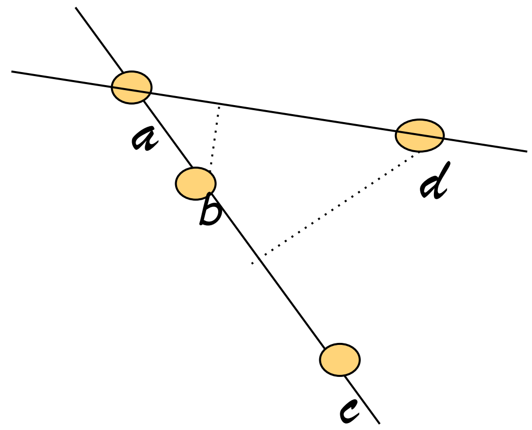
adj. 垂直的

Consider the perpendicular from d to L.

- at least 2 points, a and b, are on the same side.

Consider the line L' passing through d & a.

distance (b, L') < distance (d, L).



L, d => L', b => ……

Consider a line L passing through 3 or more points, say, a, b, and c.

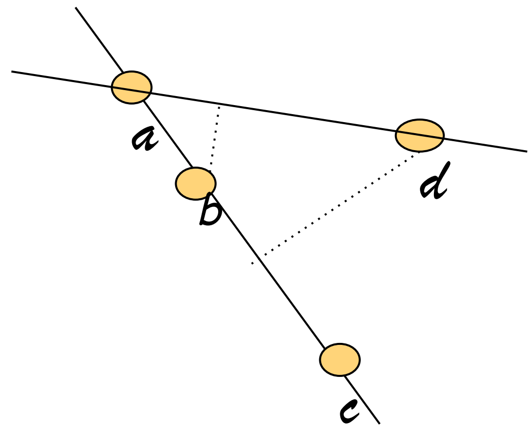
Let d be a point not on L.

Consider the perpendicular from d to L.

- at least 2 points, a and b, are on the same side.

Consider the line L' passing through d & a.

distance (b, L') < distance (d, L).



Summary: Line L + point d => Line L' + point b

And the point-line distance is getting smaller.

Repeat this process. One can keep reducing the distance.

Doesn't make sense ! Because A is finite.

Formal proof next page.

Proof by contradiction

Suppose for the sake of contradiction that the n points in A are not all on a line.

Consider any line L passing through 2 points A. Let d be a point in A not on L.

Since A is finite, we can choose L and d such that distance(L,d) is the smallest.

By the previous observation,

there exists a line L' passing through 2 points in A and a point b with distance (L',b) < distance (L,d).

This contradicts that distance (L,d) is the smallest.

$n^2 = O(n)$!? Does it make sense ?

Consider the following mathematical induction.

Base case: when $n=1$, $n^2 = n = O(n)$.

Induction step:

Assume that $n^2 = O(n)$ for any $n \geq 1$.

Consider the case for $n+1$:

$$(n+1)^2 = n^2 + 2n + 1 = O(n) + 2n + 1 = O(n).$$

Induction hypothesis

Anything wrong ?

Course Objectives

- To stimulate your interest in theoretical CS and to update you on some interesting research in theoretical CS.
- To give you a flavor of rigorous mathematical analysis in CS.
- I hope that at the end of this course, you would have better skills of analyzing problems & algorithms.

Challenge: try to follow my lecture (including notations, lemmas/theorems, proofs, etc) and raise questions when you have doubt.

Automata, Computability and Complexity

- Capabilities & limitations of computers; in particular, theory for explaining why some problems are so hard to solve.

- Decision problem: determine whether a number is prime; the answer is "yes" or "no"
- Computing a function: find the maximum matching of a graph.

(a) No algorithm can solve the problem.

(b) Algorithms exist, but they take too much time (or memory).

Basics (Formal language)

- An alphabet, usually denoted by Σ or Γ, is a set of symbols.
  - E.g., $\Sigma = \{0, 1\}$; $\Sigma = \{a,b,c,d,\ldots,x,y,z\}$.
- A string over an alphabet is a sequence of symbols from that alphabet.
  - E.g., 10111001 is a string over the alphabet {0,1};
  - "computers" is a string over the alphabet {a,b,.., y,z}.
- The length of a string is the number of symbols in the string.
  - The length of "computers" is 9.
- The null string or empty string is a string of length 0.

Σ^* denotes the set of all possible strings over the alphabet Σ , including the empty string.

Σ^i, where $i \geq 1$, denotes the set of strings of length exactly i.

E.g., $\Sigma = \{ 0, 1\}$, and $\Sigma^2 = \{ 00, 10, 11, 01\}$

A **language** L over an alphabet Σ is a set of strings over Σ.
I.e., $L \subseteq \Sigma^*$.

- E.g., $\Sigma = \{a,b,c,d,\ldots,x,y,z\}$;
  L_1 = {algorithms, complexity, computer, PC, unix};
  $L_2 = \{ w \in \Sigma^* \mid w \text{ contains an "a"} \}$
- E.g., $\Sigma = \{ 0, 1\}$;
  $L_3 = \{ w \in \Sigma^* \mid w \text{ is a prime binary number} \}$

Languages versus decision problems

- Decision problem: Given a binary string x, determine whether x is prime.
- Language acceptance problem:

  Let $L = \{ w \in \Sigma^* \mid w$ is a prime binary number $\}$.

  Given a binary string x, determine whether x is an element of L.

- Note that $x \in L$ if and only if x is prime.

In general, any decision problem can be formulated as a <u>language acceptance problem</u>.

- Let P be any decision problem; assume the input is a string over an alphabet Σ.

  The answer to P with respect to any input $\boldsymbol{w} \in \Sigma^*$ is either "YES" or "NO".

- The corresponding language L is

  $\{\boldsymbol{w} \in \Sigma^* \mid$ the answer to P w.r.t. input $\boldsymbol{w}$ is "YES" $\}$.

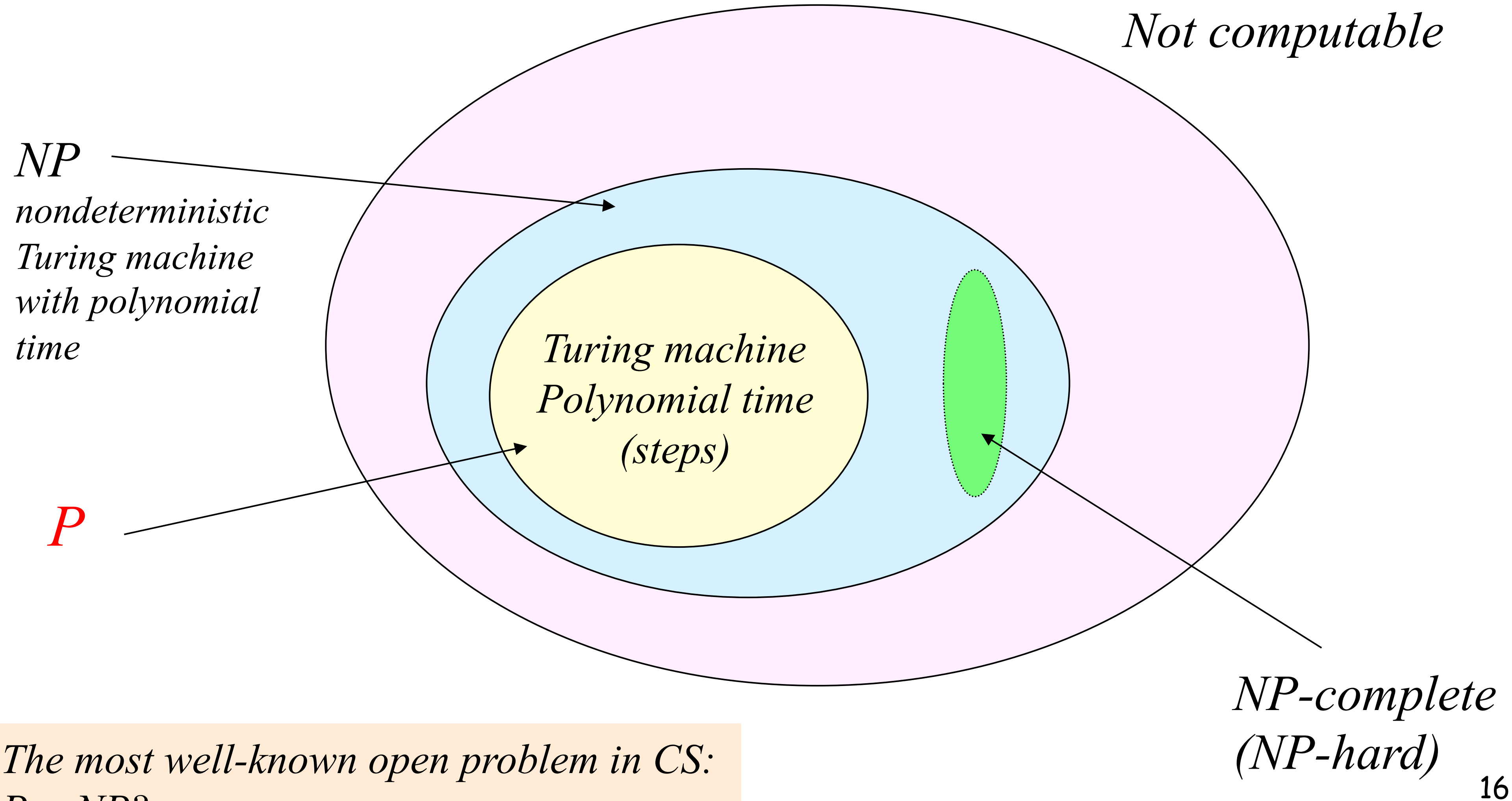
  NB. L includes all positive instances of P.

- An algorithm that accepts (decides) correctly the elements of L also solves the problem P.

Modeling computation

- We will study three models of computation: finite automata, pushdown automata and Turing machines.
- They are very simple; their computation can be argued mathematically.
- Finite automata are primitive, modelling computers with very limited memory.
- Turing machines are more powerful and can model the computation of a PC or any computer.
- Based on Turing machines, we can easily study the limitation of computers, showing that some problems cannot be solved by computers.

Computability & Complexity



The most well-known open problem in CS:
P = NP?

Today's lecture

- A succinct review of finite automata, which is the simplest computational model.
- Key feature:
  - formal definitions
  - nondeterministic computation
  - limitation of finite automata.

Finite Automata

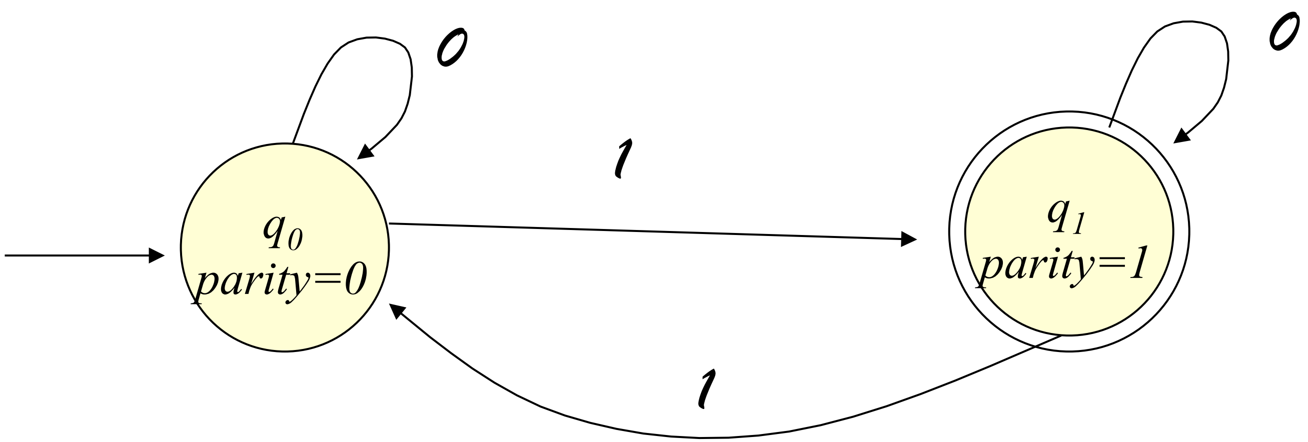
Input → **The control** → **Accept or Reject**

The control – At any time an automaton is in a particular **state**. In one step, it reads an input.

- Depending on what is read and the current state, the machine jumps to another state.
- The number of possible states is fixed in advance, i.e., independent of the input.
- The state transition (which state to jump) is pre-specified by a function (table).

Example

- A finite automaton for checking whether the input is a binary string with odd parity.



q_1 is the **final state**; final states are represented by double circles

E.g., *input = 11011 (stops at q_0; rejected)*
input = 11011001 (stops at q_1; accepted)

Formal definition

A finite automaton $M = (Q, \Sigma, f, q_0, F)$ consists of

- a finite set Q of states,
- a finite input alphabet Σ,
- a transition function $f: Q \times \Sigma \rightarrow Q$ that assigns a state (i.e., the next state) to each combination of state and input,
- a starting state q_0
- A set $F \subseteq Q$ of final states

What are the possible input symbols?

Computation

Given a finite automaton M, how does it operate?

First, M starts off in the starting state q_0 .

Assume that $w = x_1 x_2 x_3 \ldots x_n$ is the input, where each $x_i \in \Sigma$.

M reads the input symbols one by one.

- After reading x_1, M jumps to state $q = f(q_0, x_1)$.
- Next, M reads x_2 and jumps to state $q' = f(q , x_2)$.
- Next, M reads x_3 and jumps to state $q'' = f(q' , x_3)$.
- …
- Next, M reads x_n and jumps to state $q^{**} = f(q^* , x_n)$.
- The computation ends. If q^{**} is in F then w is said to be accepted (otherwise, rejected).

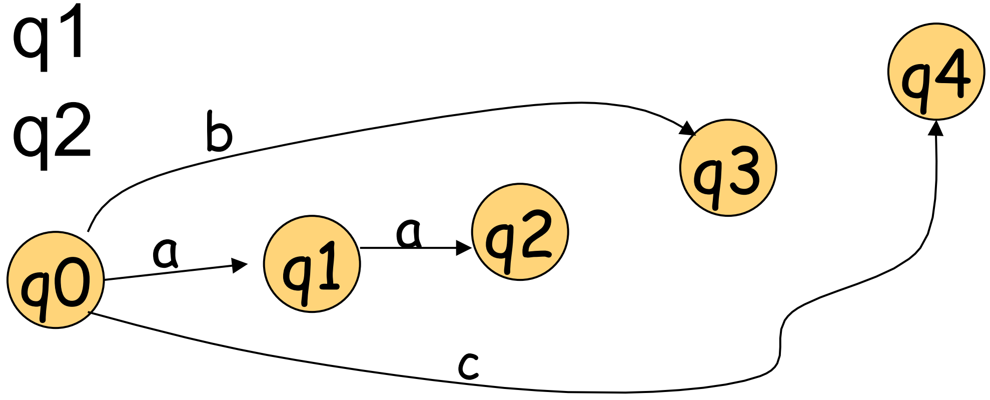
Transition function

- can be represented by a table or a diagram.

Example

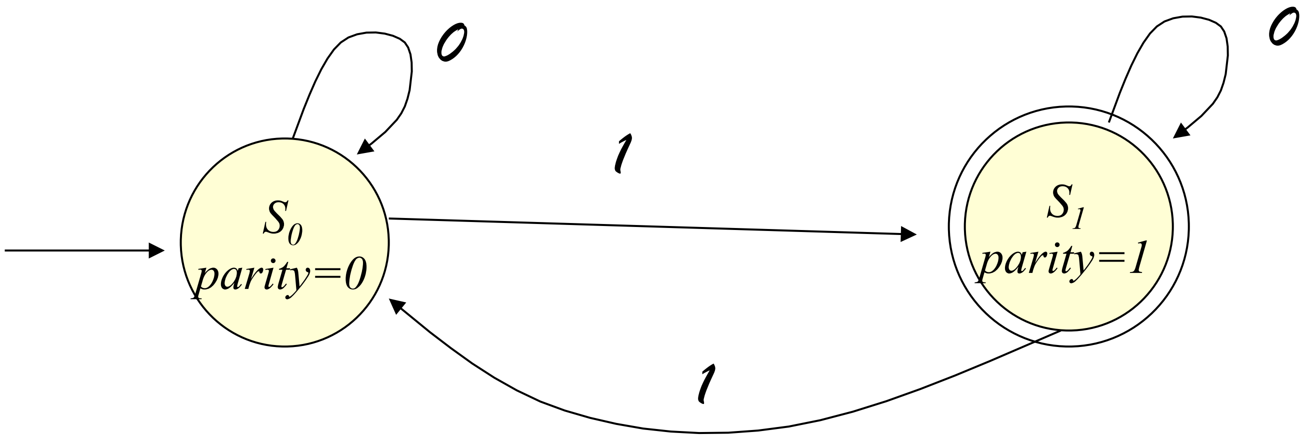
$Q = \{q0,q1,q2,q3,q4\}$
$\Sigma = \{a,b,c\}$

	q0	q1	q2	q3	q4
a	q1	q2	q2	q3	q0
b	q3	q1	q0	q4	q1
c	q4	q4	q4	q4	q2



Example

- A finite automaton for checking whether the input is a binary string with odd parity.



S_1 is the final state; final states are represented by **double circles**

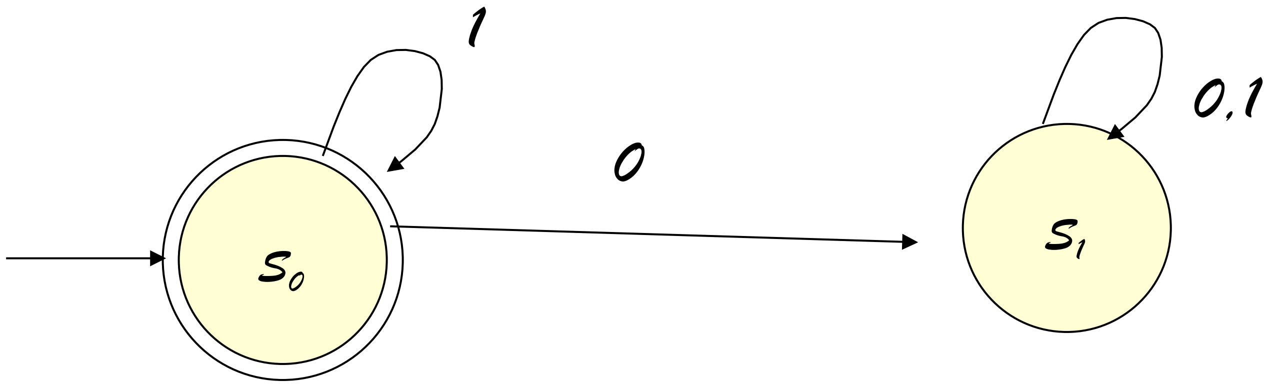
E.g., input = 11011 (stops at S_0; rejected)
input = 11011001 (stops at S_1; accepted)

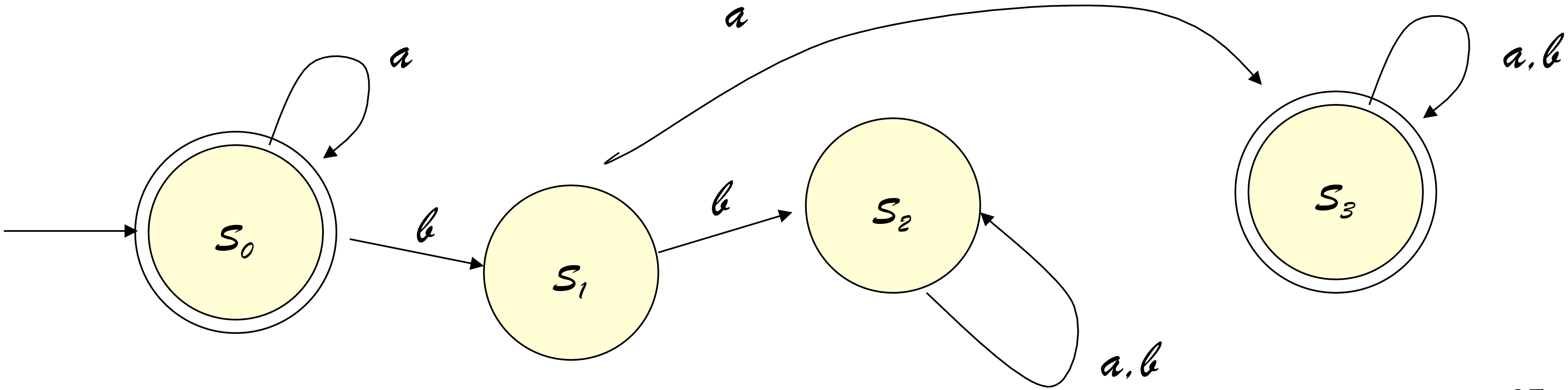
How to signal the acceptance of input?

- Some of the states in the automaton are marked as **final states**.
- If the automaton, after reading the entire input, stops at a final state, the input is said to be *accepted* (*Yes*);
- if the automaton stops at a non-final state, the input is said to be *rejected* (*No*).

More examples

- What are the inputs accepted by the following finite automata?





Definition of acceptance

Let $M = (Q, \Sigma, f, q_0, F)$ be a finite state automaton.

Let $w = x_1 x_2 \dots x_n$ be a string with n characters over Σ.

With w as the input, M will visit a sequence of states r_0, r_1, r_2, … r_n such that

- $r_0 = q_0$
- $r_{i+1} = f(r_i, x_{i+1})$ for $i = 0, 1, 2, \dots, n-1$

We say that M accepts w if r_n is in F. Otherwise, M rejects w.

Languages

- Recall that a language is a subset of strings over a certain alphabet.

- The language accepted (recognized) by finite state automaton M comprises all the input strings over Σ that are accepted by M.

  I.e., $L(M) = \{ w \mid M \text{ accepts } w \}$.

Nondeterministic finite automaton

DFA

- The finite automata discussed so far are deterministic in the sense that given any pair of state and input, an automaton goes to a unique state in the next step (because f is a function).
- A nondeterministic finite automaton is more flexible, allowing more than one possible next state.

NFA

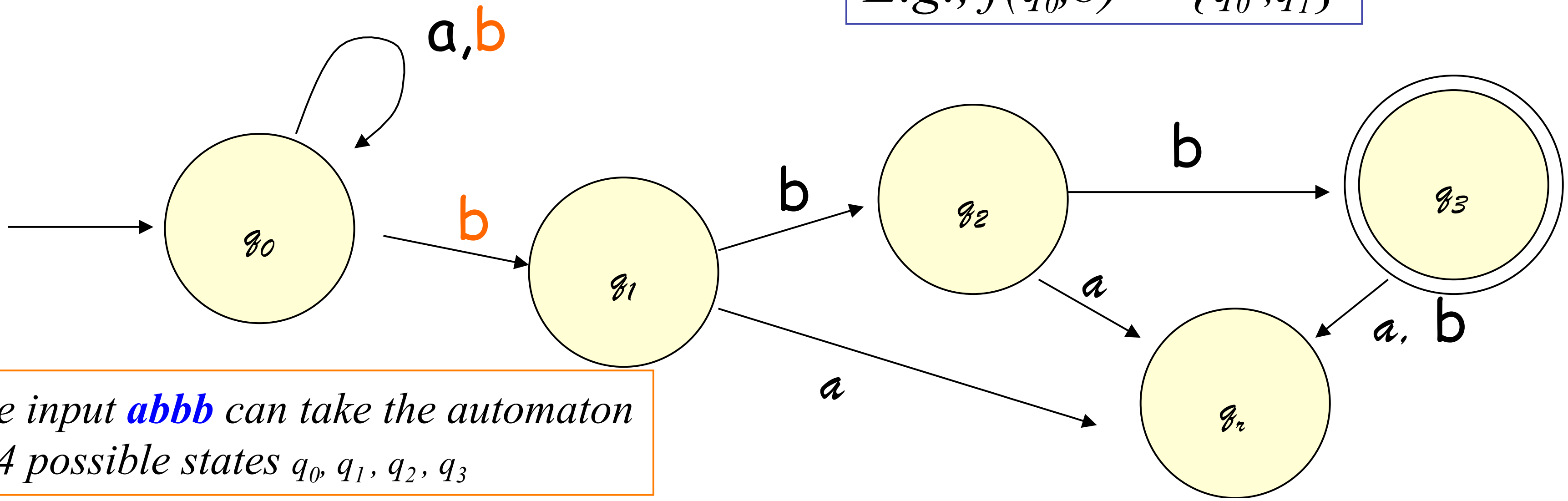
Nondeterministic finite automaton

DFA

- The finite automata discussed so far are deterministic in the sense that given any pair of state and input, an automaton goes to a unique state in the next step (note that f is a function).
- A nondeterministic finite automaton is more flexible, allowing more than one possible next state.

NFA

E.g., $f(q_0, b) = \{q_0, q_1\}$



*The input **abbb** can take the automaton to 4 possible states* q_0, q_1, q_2, q_3

NFA

Formal definition

A nondeterministic finite automaton $M = (Q, \Sigma, f, q_0, F)$ consists of

- a finite set Q of states, an input alphabet Σ,
- a transition function ***f*** that assigns **a set of states** to each pair of state and input
- a starting state q_0, and a set of **F** $\subseteq$ Q of final states.

NFA

Formal definition

A nondeterministic finite automaton $M = (Q, \Sigma, f, q_0, F)$ consists of

- a finite set Q of states, an input alphabet Σ,
- a transition function *f* that assigns **a set of states** to each pair of state and input
- a starting state q_0, and a set of **F** $\subseteq$ Q of final states.

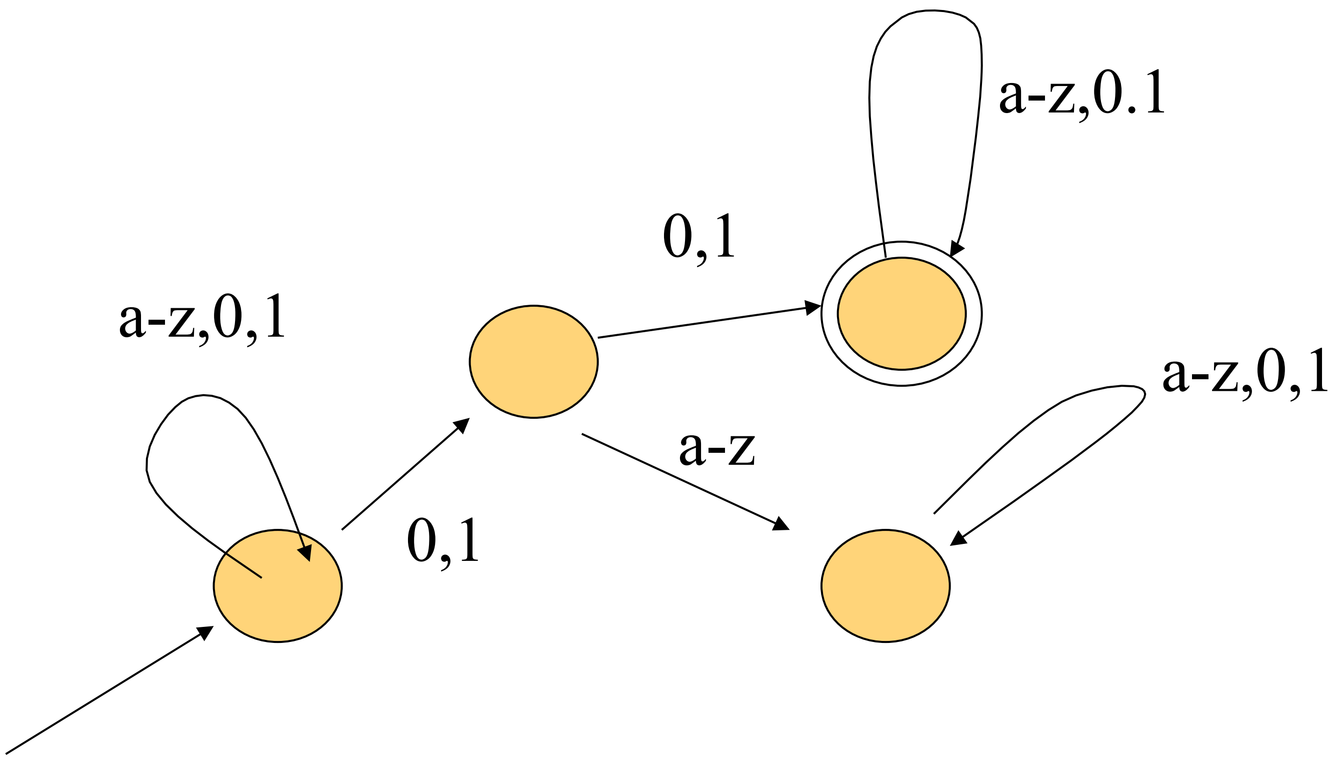
Given an input *x*, M can ***take different sequence of moves, stopping at different states***. Some of these states may be final and others may not.

We say that *x* is accepted if, among all the states at which M can stop, there is one in **F**.

An NFA also defines a language, which comprises all the input strings it accepts.

Example

- Design an NFA to accept the set of strings of lower-case letters or digits containing at least two consecutive digits.



NFA more powerful than DFA ?

- Is there a decision problem that can be solved by an NFA but not by a DFA?
- Is there a language that be accepted by an NFA but not by a DFA?
- The answer is NO.

Theorem Let M be any NFA accepting a language L. Then there exists a DFA M' that can accept exactly all strings of L.

Proof: Subset construction

- Consider any NFA $M = (Q, \Sigma, f, q_0, F)$. Let n be the number of states in M. Note that n is a constant.
- After reading some input symbols, M can possibly reach more than one state, more precisely, a certain subset of states.

  How many possible subsets of states M can reach?

Proof: Subset construction

- Consider any NFA $M = (Q, \Sigma, f, q_0, F)$. Let n be the number of states in M. Note that n is a constant.
- After reading some input symbols, M can possibly reach more than one state, more precisely, a certain subset of states.

  How many possible subsets of states M can reach?

  Answer: at most 2^n, which is also a constant.

Proof: Subset construction

- Consider any NFA $M = (Q, \Sigma, f, q_0, F)$. Let n be the number of states in M. Note that n is a constant.
- After reading some input symbols, M can possibly reach more than one state, more precisely, a certain subset of states.

  How many possible subsets of states M can reach?

  Answer: at most 2^n, which is also a constant.
- E.g., n=10. There are 1024 different subsets of states. No matter what is the input, the subset of states M can reach is one of these 1024 subsets.
- By definition, an input x is accepted by M means that one of the states at which M stops is a final state (i.e., in F).

DFA

We construct a DFA M' to simulate any given NFA M as follows:

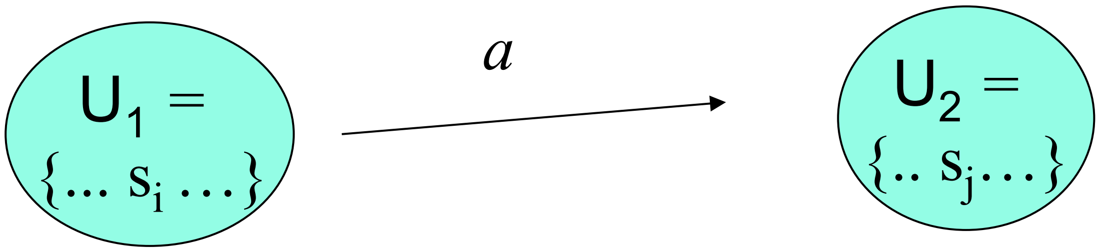
- Let $M = (Q, \Sigma, f, q_0, F)$, and let $Q = \{s_0, s_1, \ldots, s_{n-1}\}$ be the set of states of M.
- Define $M' = (Q', \Sigma, f', U_0, F')$ to be a DFA with 2^n "states", each "state" $\in Q'$ represents a subset of Q.
- Example. We use the symbol U to denote a state in Q'

$U_0 = \{s_0\}$ $U_1 = \{s_2, s_8\}$ $U_{135} = \{s_0, s_1, \ldots, s_{n-1}\}$

Intuitively, M' uses one state to memorize all the possible states that can be reached in M.

DFA

Let U_1 be a "state" of M'. For any a in Σ, how to define $f'(U_1, a)$?



$\boxed{f'(U_1, a) = U_2}$

M'

$f'(U_1, a) = U_2$ **<u>if and only if</u>**

U_2 represents the subset $\{ s_j \mid s_j \in f(s_i, a)$ for some state s_i in $U_1 \}$.

M

$\boxed{f(s_i, a) = \{\ldots s_j \ldots\}}$

M′ simulates M

- What is the starting state of M′ ? $U_0 = \{s_0\}$.
- Which are the final states of M′ ?

  All the states U in Q′ such that $U \subseteq Q$ and U contains a state in F.

Lemma. On any input x, M can reach a <u>subset</u> U of states

$\Leftrightarrow$ M′ can reach the state U.

NB. This can be proven using an induction on the length of x.

Corollary. M accepts x $\Leftrightarrow$ U contains a state in F

$\Leftrightarrow$ U is a final state of M′

$\Leftrightarrow$ M′ accepts x.

Limitation of finite automata

Let L be the set of strings aa…aaabb…bbbwhich contain the same number of a's and b's.

I.e., L = {ab, aabb, aaabbb, …. }.

Notation: Let a^i denote the string with i a's.

??? Construct a DNA or NFA to accept L.

In other words, we want a finite automaton to check the number of a's and b's.

No, such an automaton doesn't exist.

NB. Roughly speaking, DFA has no memory to store a counter.

Proof (by contradiction)

Suppose that there is a DFA M accepting L.

Assume that M has n states and the starting state is s_0. Note that n is a constant.

Consider the string $x = a^n b^n$. By definition, M should accept x.

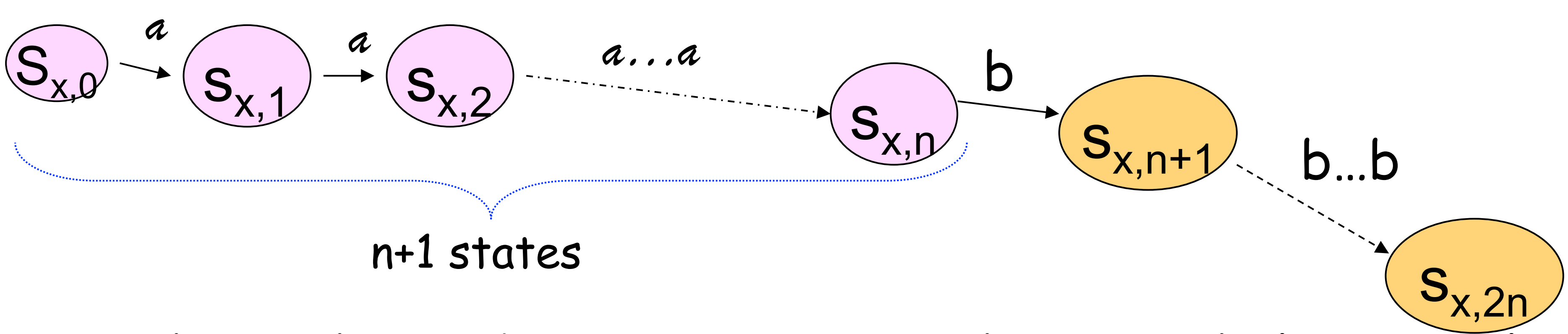
Denote the state of M after reading the 1st symbol of x as $s_{x,1}$.

And similarly, $s_{x,2}$, …, $s_{x,k}$ for the 2nd symbol, …, k-th symbol, respectively.

For convenience, we denote $s_{x,0} = s_0$.

The pigeonhole principle

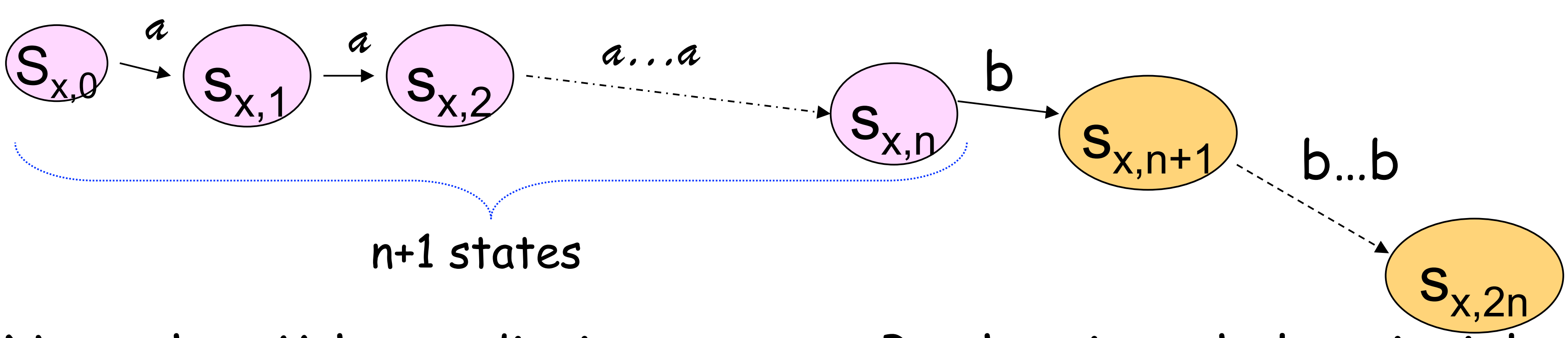
Consider the states $s_0, s_{x,1}, s_{x,2}, \ldots, s_{x,n}, s_{x,n+1}, \ldots, s_{x,2n}$.
Since M accepts x, $s_{x,2n}$ is a final state of M.



Note that M has n distinct states. By the pigeonhole principle, there exist $0 \leq j < k \leq n$ such that $s_{x,j} = s_{x,k}$

The pigeonhole principle

Consider the states s_o, $s_{x,1}$, $s_{x,2}$, …, $s_{x,n}$, $s_{x,n+1}$, …, $s_{x,2n}$.
Since M accepts x, $s_{x,2n}$ is a final state of M.



Note that M has n distinct states. By the pigeonhole principle, there exist $0 \le j < k \le n$ such that $s_{x,j} = s_{x,k}$

What can we conclude?

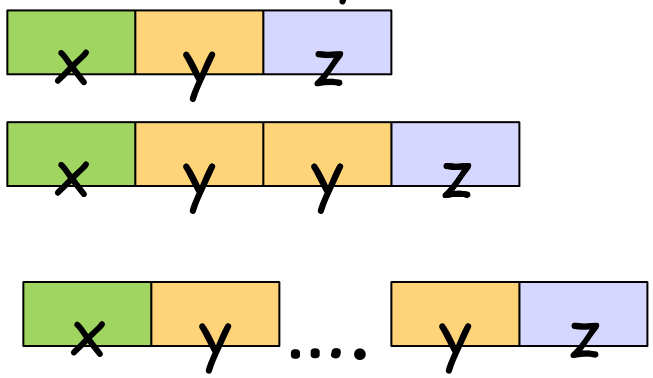
Let m = k-j. M accepts the string $a^{n-m}\, b^n$.

What about $a^{n+m}b^n$?

A contradiction occurs.

Pumping Lemma

Theorem Let L be a language that can be accepted by a DFA M with n states. For any string s in L of length at least n, s can be divided into three pieces, s = xyz such that



- $|y| > 0$,
- $|xy| \leq n$, and
- for all $i \geq 0$, xy^iz is in L.

Exercise: prove it yourself, or read the proof from somewhere (say, Sipser Chapter 1).