Advanced Research Topics in Computational Complexity Theory

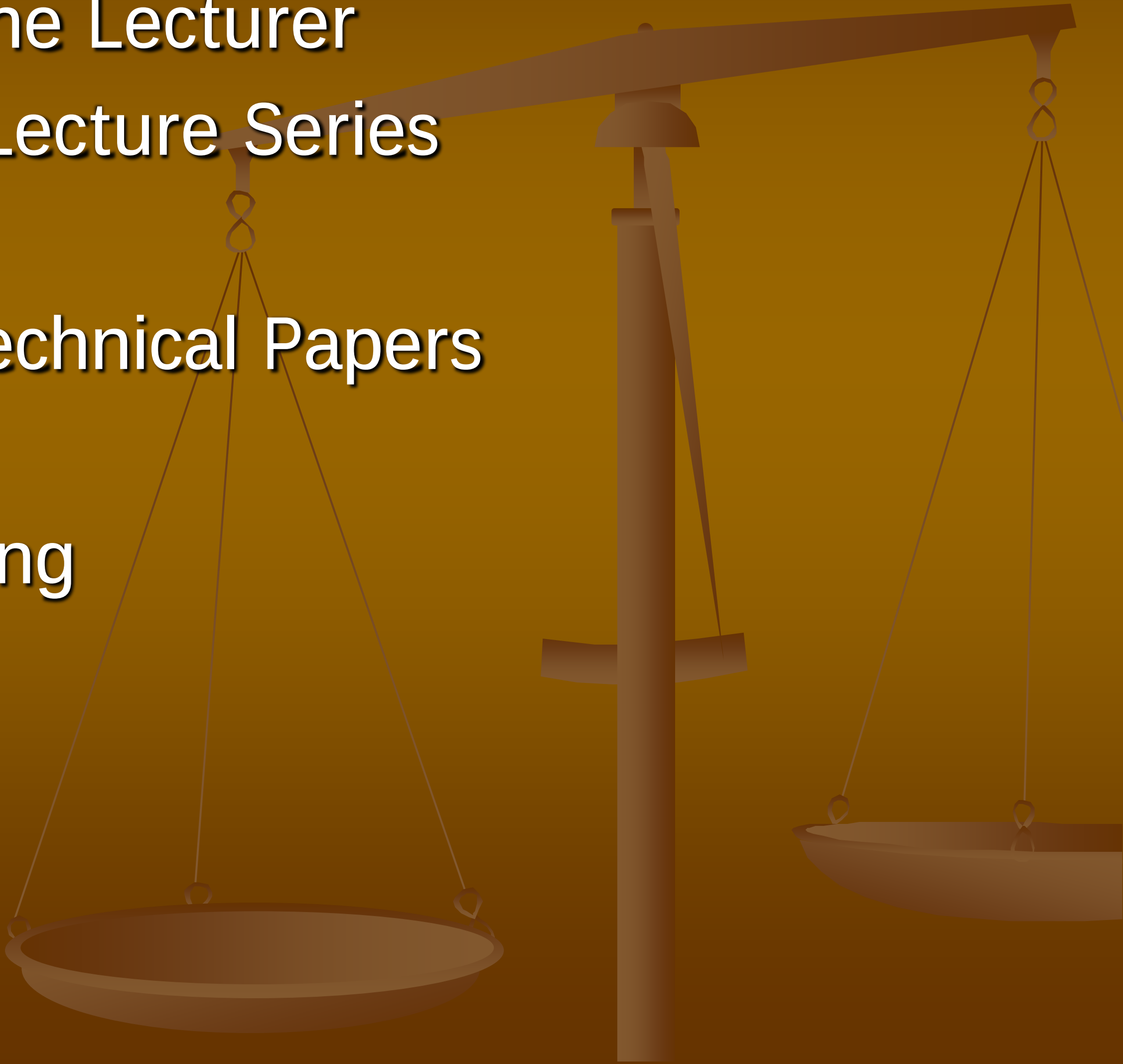
— Introduction to T. Yamakami's Work —

Dr. Tomoyuki Yamakami

First Semester 2018
University of Fukui

Basic Information on This Lecture Series

1. Introduction of the Lecturer
2. Purpose of This Lecture Series
3. Course Schedule
4. Where to Find Technical Papers
5. YouTube Videos
6. Suggested Reading



Lecturer

Dr. Tomoyuki Yamakami

August 2017

Affiliation:

Faculty of Engineering
University of Fukui, Japan

Research Topics:

- quantum computing
- languages and finite automata
- optimization and approximation
- constraint satisfaction problems
- cryptographic systems
- fuzzy computing
- logic, etc.

- Twitter ↪ http://twitter.com/tomoyamakami/
- YouTube ↪ https://www.youtube.com/user/tomoyukiyamakami

Purpose of This Lecture Series

- This lecture series is designed to introduce fundamental features of computational complexity theory to graduate students who have just started their study on complexity-theoretic issues, by explaining and also giving a clear pointer to a number of results obtained by Tomoyuki Yamakami since 1982.
- This lecture series also aims at providing unsolved questions, mostly related to T. Yamakami's work, which the audience may try to tackle for his/her theses.
- Topics of this lecture series can be summarized by four basic categories: complexity-theoretical issues based on (1) Turing machines, (2) cryptographic systems, (3) finite automata and their variants, and (4) quantum machines.

Course Schedule: 16 Weeks

Subject to Change

- Week 1: Basic Computation Models
- Week 2: NP-Completeness, Probabilistic and Counting Complexity Classes
- Week 3: Space Complexity and the Linear Space Hypothesis
- Week 4: Relativizations and Hierarchies
- Week 5: Structural Properties by Finite Automata
- Week 6: Stype-2 Computability, Multi-Valued Functions, and State Complexity
- Week 7: Cryptographic Concepts for Finite Automata
- Week 8: Constraint Satisfaction Problems
- Week 9: Combinatorial Optimization Problems
- Week 10: Average-Case Complexity
- Week 11: Basics of Quantum Information
- Week 12: BQP, NQP, Quantum NP, and Quantum Finite Automata
- Week 13: Quantum State Complexity and Advice
- Week 14: Quantum Cryptographic Systems
- Week 15: Quantum Interactive Proofs
- Week 16: Final Evaluation Day (no lecture)

Course Schedule: 16 Weeks

Subject to Change

- Week 1: Basic Computation Models
- Week 2: NP-Completeness, Probabilistic and Counting Complexity Classes
- Week 3: Space-Bounded Complexity and the Linear Space Hypothesis
- Week 4: Relativizations and Hierarchies
- Week 5: Structural Properties by Finite Automata
- Week 6: Stype-2 Computability, Multi-Valued Functions, and State Complexity
- Week 7: Cryptographic Concepts for Finite Automata
- Week 8: Constraint Satisfaction Problems
- Week 9: Combinatorial Optimization Problems
- Week 10: Average-Case Complexity
- Week 11: Basics of Quantum Information
- Week 12: BC Automata
- Week 13: Qu
- Week 14: Quantum Cryptographic Systems
- Week 15: Quantum Interactive Proofs
- Week 16: Final Evaluation Day (no lecture)

classical complexity theory

Course Schedule: 16 Weeks

Subject to Change

- Week 1: Basic Computation Models
- Week 2: NP-Completeness, Probabilistic and Counting Complexity Classes
- Week 3: Space-Bounded Complexity and the Linear Space Hypothesis
- Week 4: Relativizations and Hierarchies
- Week 5: Structural Properties by Finite Automata
- Week 6: Stype-2 Computability, Multi-Valued Functions, and State Complexity
- Week 7: Cryptographic Concepts for Finite Automata
- Week 8: Cor
- Week 9: Cor
- Week 10: Average-Case Complexity
- Week 11: Basics of Quantum Information
- Week 12: BQP, NQP, Quantum NP, and Quantum Finite Automata
- Week 13: Quantum State Complexity and Advice
- Week 14: Quantum Cryptographic Systems
- Week 15: Quantum Interactive Proofs
- Week 16: Final Evaluation Day (no lecture)

quantum complexity theory

Where to Find Technical Papers

- This lecture series is based on numerous papers written by Tomoyuki Yamakami over the years. Those papers can be found at the following websites.

- DBLP Computer Science Bibliography
  http://dblp.uni-trier.de/pers/hd/y/Yamakami:Tomoyuki
- arXiv.org (public domain)
  https://arxiv.org/find/quant-ph,grp_cs/1/au:+Yamakami_Tomoyuki/0/1/0/all/0/1?per_page=100

December 2015

YouTube Videos I

- Most materials in this lecture series were presented in numerous conferences and universities (in English) by Tomoyuki Yamakami.
- Some of these public talks have been video-recorded and the videos were edited and then uploaded to YouTube.
- Please search on YouTube site using the following keywords:

(*) YouTube search keywords:

Tomoyuki Yamakami conference playlist

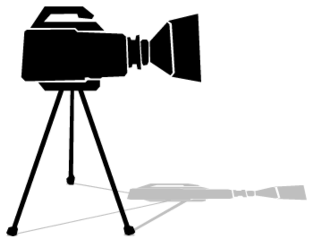
Conference talk video
Egham, UK 2017

YouTube Videos II

- Moreover, there are a few invited talk videos (in English) currently available on YouTube.
- Invited Talks (in English)
  - Invited Talk 2013 – Liverpool, UK
  - Invited Talk 2013 – Lisbon, Portugal
  - Dagstuhl Seminar – Public Talk 2013 – Schloss Dagstuhl, Germany

(*) YouTube search keywords:

Tomoyuki Yamakami invited talk playlist



Invited talk video
Lisbon, Portugal 2013

Suggested Reading

- Through this lectures, we will not use specific textbooks. However, I advice the audience to read standard textbooks on computational complexity theory and languages and automata theory.
- For example:
  - Theory of Computational Complexity. D. Du and K. Ko. Wiley Interscience, 2000.
  - Introduction to Automata Theory, Languages, and Computation. J. E. Hopcroft, R. Motwani, and J. D. Ullman. Addison-Wesley, 2nd edition, 2001.
  - Foundations of Cryptography: Basic Tools. O. Goldreich. Cambridge University Press, 2001.
  - An Introduction to Kolmogorov Complexity and Its Applications. M. Li and P. Vitányi. Springer-Verlag, 1997.
  - Quantum Computation and Quantum Information. M. A. Nielsen and I. L. Chuang. Cambridge University Press, 2000.

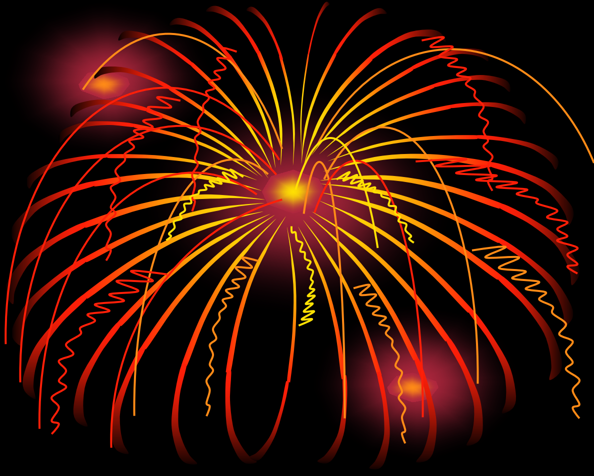
1st Week

Basic Computation Models

Synopsis.

- Formal Languages
- Automata and Turing Machines
- Pushdown Automata
- 1-FLIN and FP
- Recursive Functions

April 1, 2018. 23:59



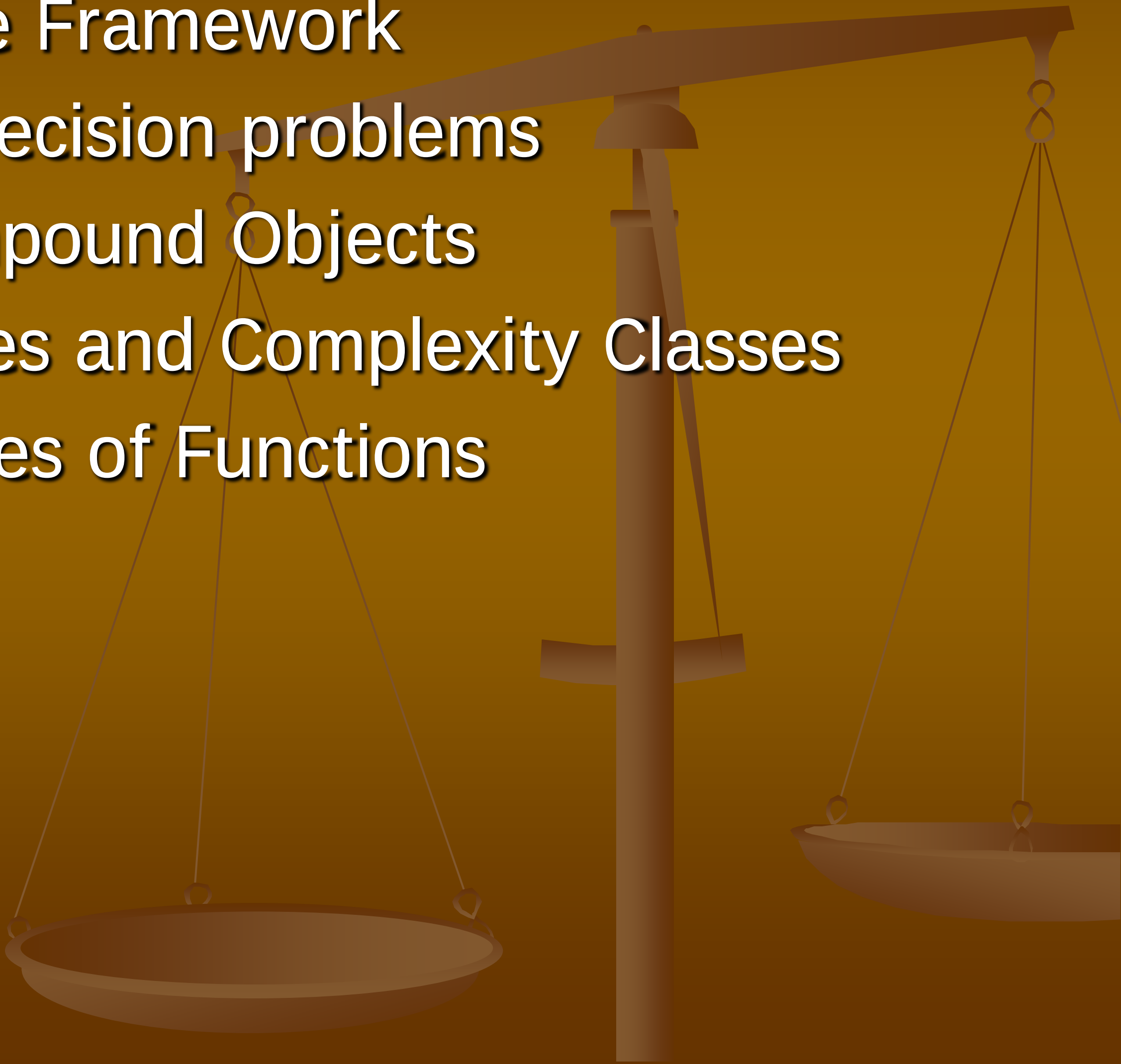
Main References by T. Yamakami

- K. Tadaki, T. Yamakami, J. C. H. Lin. Theory of one-tape linear-time Turing machines. Theoretical Computer Science 411(1): 22-43 (2010)

March 2018

I. Formal Languages and Language Families

1. Formal Language Framework
2. Languages Vs. Decision problems
3. Encoding of Compound Objects
4. Language Families and Complexity Classes
5. Input/Output Sizes of Functions



Formal Language Framework I

- We review basics of theory of formal languages and automata.
- An alphabet Σ is a finite nonempty set of "symbols."
  - For example, $\Sigma = \{ 0,1 \}$ or $\Sigma = \{ a, b, c, ..., z \}$.
- A string over Σ is a finite sequence of symbols in Σ.
- Given a string x, $|x|$ denotes the length of x.
- A language L over Σ is any set of strings over Σ.
  - For example, if $\Sigma = \{ 0,1 \}$, then the following set L is the language of binary representations of prime numbers:

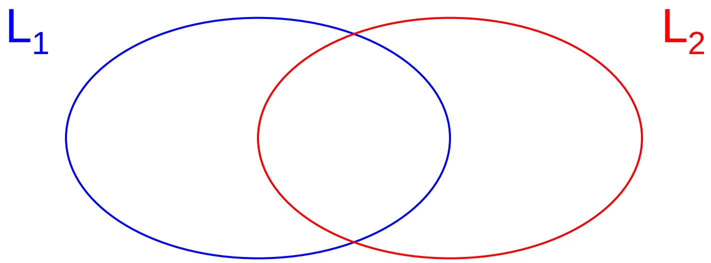
$$L = \{ 10, 11, 101, 111, 1011, ... \}.$$

- We denote the empty string by λ (or ε) with $|\lambda| = 0$.
- Let $\Sigma^n = \{ x \in \Sigma^* \mid |x| = n \}$ and $\Sigma^{\leq n} = \{ x \in \Sigma^* \mid |x| \leq n \}$.

Formal Language Framework II

- We denote the empty language by $\varnothing$.
- The language of all strings over Σ is denoted Σ^*.
  - For example, if $\Sigma = \{ 0,1 \}$, then

    $$\Sigma^* = \{ \lambda, 0, 1, 00, 01, 11, 000, 001, \dots \}.$$
- Every language L over Σ is a subset of Σ^*.
- We can perform a variety of operations on languages.
- Set-theoretic operations, such as union, intersection, difference, disjoint union, etc., follow directly from the set-theoretic definitions.

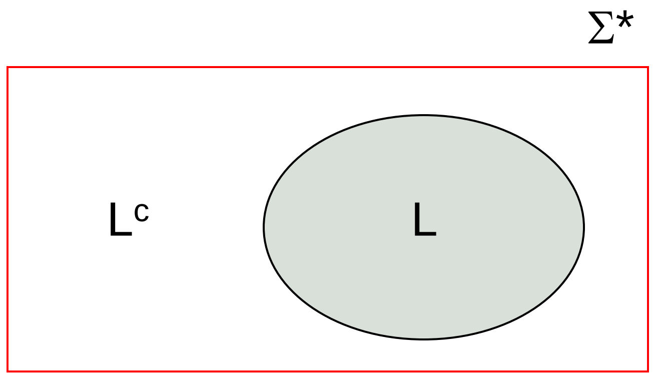


$$L_1 \cap L_2 = \left\{ x \in \Sigma^* \mid x \in L_1 \wedge x \in L_2 \right\}$$

$$L_1 \cup L_2 = \left\{ x \in \Sigma^* \mid x \in L_1 \vee x \in L_2 \right\}$$

$$L_1 - L_2 = \left\{ x \in \Sigma^* \mid x \in L_1 \wedge x \notin L_2 \right\}$$

Formal Language Framework III



- We define the complement of L by $L^c = \Sigma^* - L$.
- The concatenation L_1L_2 of two languages L_1 and L_2 is the language defined as

$$L_1L_2 = \{ xy \mid x \in L_1 \text{ and } y \in L_2 \}.$$

- For each $k \geq 1$, we define L^k as the language obtained by concatenating L to itself k times; that is,

$$L^1 = L, \quad L^2 = LL, \quad L^3 = LLL, \ldots\ldots$$

- The closure (or Kleene star) of a language L is the language

$$L^* = \{ \lambda \} \cup L \cup L^2 \cup L^3 \cup L^4 \cup \ldots\ldots.$$

Languages vs. Decision Problems I

- We want to solve the following decision problems.

Decision Problem

- instance: input to the problem
- solution: YES (1) or NO (0)

(or question: does a prescribed property hold?)

- Example:
  - instance: a positive integer n
  - question: Is n a prime number?
- In order to solve such a problem, we represent instances of the problem in a way that the program understands.
- Hence, we need to encode a compound object (such as polygons, graphs, functions, ordered pairs) as a binary string by combining the representations of its constituent parts.

Languages vs. Decision Problems II

- The set of instances for any decision problem Q is simply the set Σ*, where Σ = { 0,1 }.
- Since Q is entirely characterized by those problem instances that produces a 1 (YES) answer, we can view Q as a language L over Σ, where

  L = { x∈Σ* | Q(x) = 1 }.

$$\boxed{x \in L \Leftrightarrow Q(x) = 1 \ \text{(or } Q \text{ answers YES)}}$$

- Example:

(*) Decision Problem

- ➢ instance: a binary string of the form u#v
- ➢ solution: YES if |u|=|v|; NO otherwise

(*) Language

- ➢ { w | ∃u,v∈{0,1}*[w=u#v ∧ |u|=|v|] }

$\}$ We identify languages with decision problems.

Encoding of Compound Objects

- An encoding of a set S of objects is a mapping e( ) from S to the set of binary strings.
- Example 1: We encode natural numbers N = { 0,1,2,3,4...} as the strings { 0,1,10,11,100,... }. Using this encoding, e(0)=0, e(1)=1, e(2)=10, e(3)=11, e(4)=100.
- Example 2: Keyboard characters are encoded into the ASCII code using 7 bits. E.g., the encoding of letter “A” is 1000001.

- (*) In the rest of this lecture series, we assume that all objects are encoded into strings over some reasonable alphabet.
- (*) Therefore, we assume that our problems have the set of strings over some reasonable alphabet as its instance set.

Language Families and Complexity Classes

- We informally define a complexity class as a set of languages, membership in which is determined by a complexity measure, such as running time, of an algorithm that determines whether a given string x belongs to language L.
- Hence, a complexity class is used as a synonym for "family of languages."

- The complement of a language L over alphabet Σ is the difference $\Sigma^* - L = \{ x \in \Sigma^* \mid x \notin L \}$ and is denoted by L^c.
- Given a complexity class C, the class of all complements of languages in L, $\{ L^c \mid L \in C \}$, is denoted by co-C.

Input/Output Sizes of Functions

- Let Σ_1 and Σ_2 be two alphabets.
- A function $\Sigma_1 \rightarrow \Sigma_2$ is polynomially bounded (or simply, p-bounded) if there exists a positive polynomial p such that, for all $x \in \Sigma_1^*$, $|f(x)| \leq p(|x|)$ holds.
- That is, output size cannot be too large compared to input size.
- A function $\Sigma_1 \rightarrow \Sigma_2$ is polynomially honest (or simply, p-honest) if there exists a positive polynomial p such that, for all $x \in \Sigma_1^*$ and $y \in \Sigma_2^*$, if $f(x) = y$, then $|x| \leq p(|y|)$.
- That is, output size cannot be too small compared to input size.

II. Finite (State) Automata

1. 1-Way Finite (State) Automata
2. Acceptance and Rejection for 1dfa's
3. Determinism vs. Nondeterminism
4. 1-Way Nondeterministic Finite Automata
5. 2-Way finite (State) Automata
6. Transition Functions of 1npda's
7. Configurations and Computation
8. Complexity Classes REG and CFL
9. Formal Definition of 1dpda's

Why Finite (State) Automata?

- Finite (state) automata are one of the simplest computational models to execute (prescribed) algorithms.
- Finite automata are machines with no memory device.
- Finite automata are also known as "constant-space" Turing machines (with read-only input tapes and rewritable work tapes using only constant space)
- There are numerous variants of finite automata in use.
- In particular, we introduce 1-way/2-way variants of finite automata.
- We also consider deterministic/nondeterministic variants of finite automata.

1-Way Finite (State) Automata I

- A 1-way deterministic finite (state) automaton (1dfa) M is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where
  - Q is a finite set of inner states,
  - $q_0 \in Q$ is the initial state (or start state),
  - $F \subseteq Q$ is a set of final (or accepting) states,
  - Σ is a finite (input) alphabet, and
  - δ is the transition function mapping $Q\times(\Sigma\cup\{¢,\$\})$ to Q.
- The finite automaton begins in state q_0 and reads the characters of its input string one at a time.
- δ is viewed as a “program” of a machine.

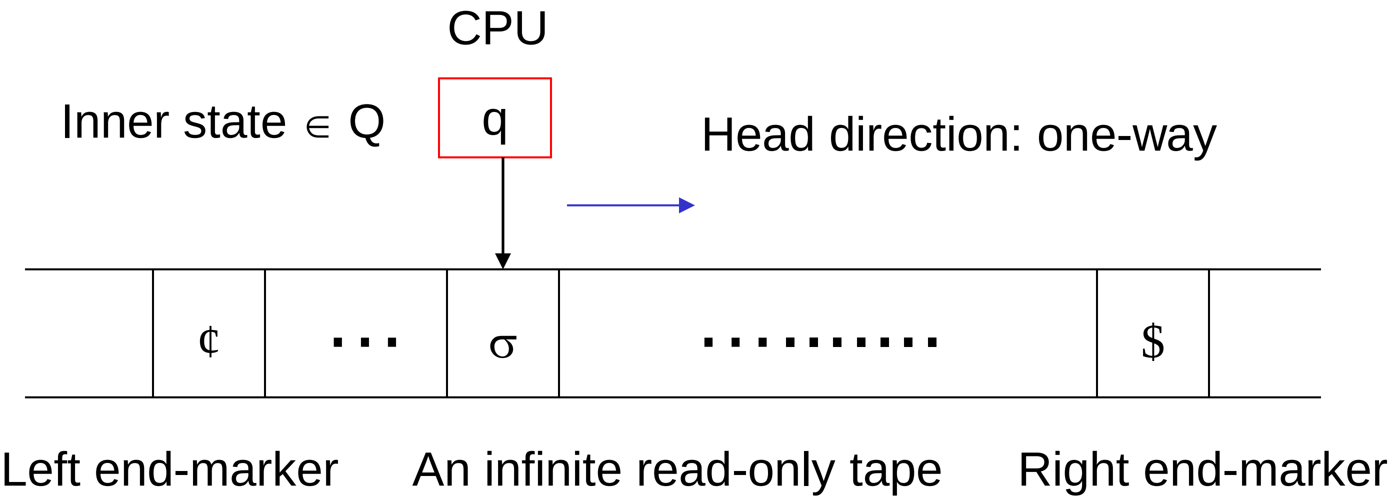
1-Way Finite (State) Automata II

A “standard” model of 1-way one-head finite (state) automaton (or simply, 1dfa) is shown as follows.

$M = (Q,\Sigma,\{\cent,\$\},\delta,q_0,F)$

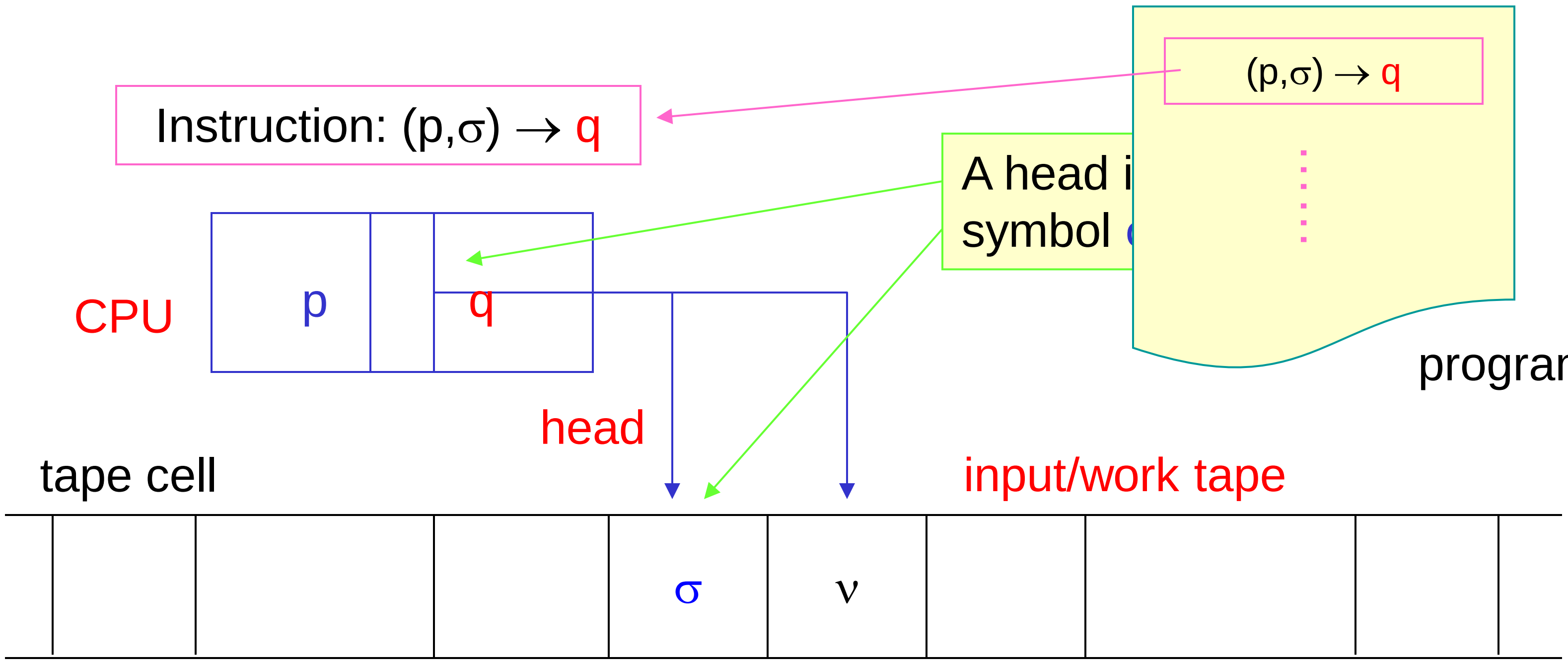
L(M) = set of strings accepted by M

Q = set of inner states
Σ = input alphabet
δ : transition function
q_0 : initial state
F = set of final states



1-Way Finite (State) Automata III

- A finite automaton consists of a tape, a tape head, and a CPU.
- An input is initially written on the input tape.
- The automaton scans a tape symbol, follows a program, changes the inner state of the CPU, and then moves the head.

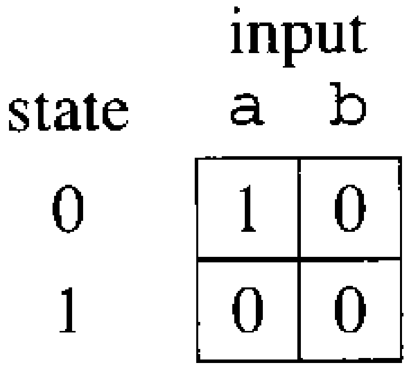


Acceptance and Rejection for 1dfa's

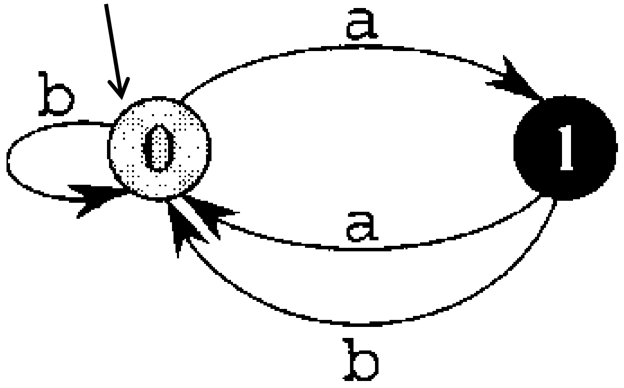
- An automaton starts scanning ¢ in the initial state q_0.
- If the automaton is in inner state q and reads input character σ, it moves from state q to state $\delta(q,\sigma)$.
- The tape head always moves to the right.
- Whenever its current state q is a member of F, the machine M is said to have accepted the input string.
- An input that is not accepted is said to be rejected.
- The automaton halts if either it enters a final state or reaches the right endmarker $.
- A finite automaton M induces a function φ, called the final-state function, from Σ^* to Q such that $\varphi(w)$ is the state in which M ends up after scanning the entire string w.
- Thus, M accepts w iff $\varphi(w) \in F$.

Three Different Descriptions of 1dfa's

A simple two-state finite automaton with state set Q = {0,1}, initial state q_0 = 0, input alphabet Σ = {a,b}, final state set F = {1}



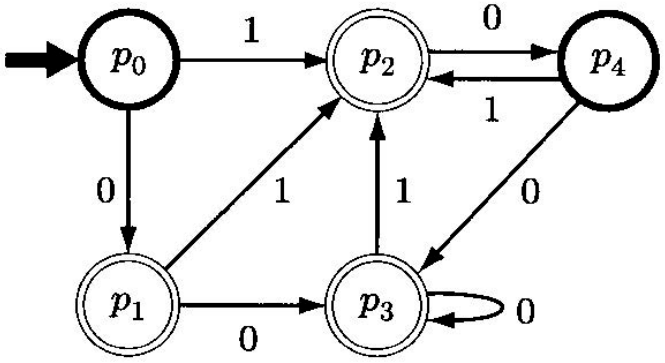
	input	
state	a	b
0	1	0
1	0	0

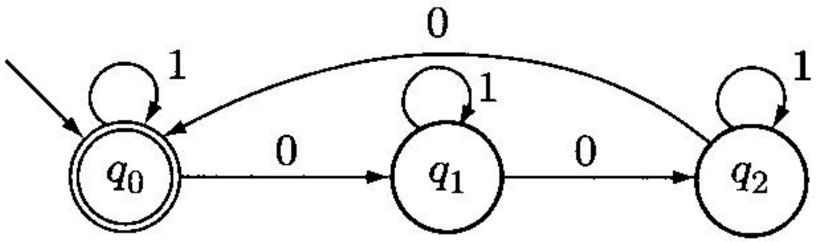


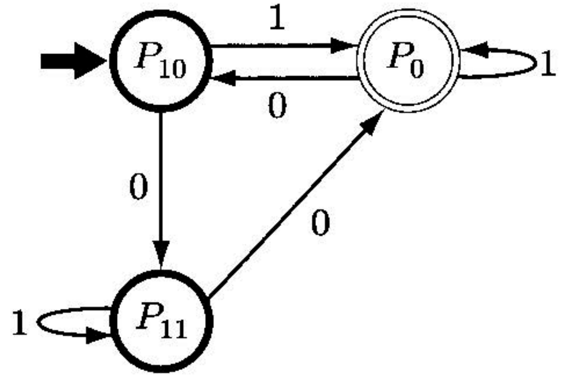
$\delta(0,a) = 1, \delta(0,b) = 1,$
$\delta(1,a) = 0, \delta(1,b) = 0$

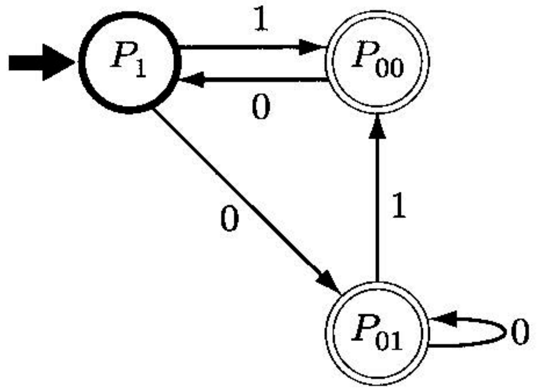
Three ways to express the same finite automaton

Examples of 1dfa’s







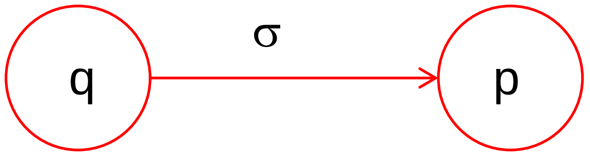


Determinism vs. Nondeterminism

- Nondeterminism is a natural extension of determinism, representing “choices of next moves”.
- 1-way deterministic finite automaton (1dfa)

$$\delta : Q\times(\Sigma\cup\{¢,\$\})\rightarrow Q$$

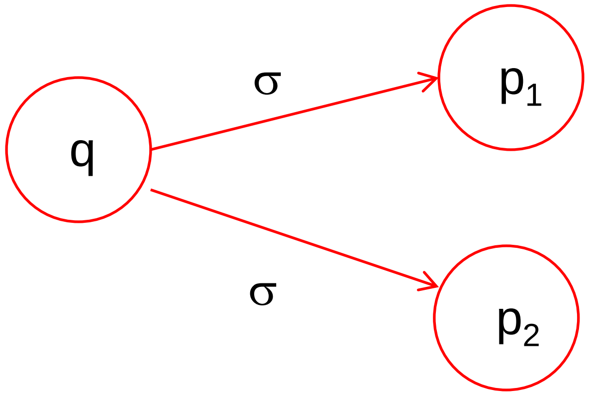
$$\boxed{\delta(q,\sigma) = p}$$



- 1-way nondeterministic finite automaton (1nfa)

$$\delta : Q\times(\Sigma\cup\{\lambda,¢,\$\})\rightarrow 2^Q$$

$$\boxed{\delta(q,\sigma) = \{ p_1, p_2 \}}$$



1-Way Nondeterministic Finite Automata

- A 1-way nondeterministic finite (state) automaton (1nfa) M is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where
  - Q is a finite set of inner states,
  - $q_0 \in Q$ is the initial state (or start state),
  - $F \subseteq Q$ is a distinguished set of final states,
  - Σ is a finite (input) alphabet,
  - δ is a function: $(Q\text{-}F)\times(\Sigma\cup\{\lambda,¢,\$\})\rightarrow\wp(Q)$, called the transition function of M (with λ-transitions).
- $\wp(Q)$ $(=2^Q)$ denotes the power set (a set of all subsets) of Q.
- A 1nfa begins in state q_0, reads the characters of its input string one at a time, and moves its tape head.
- Final states are considered as halting (accepting) states.

Configurations

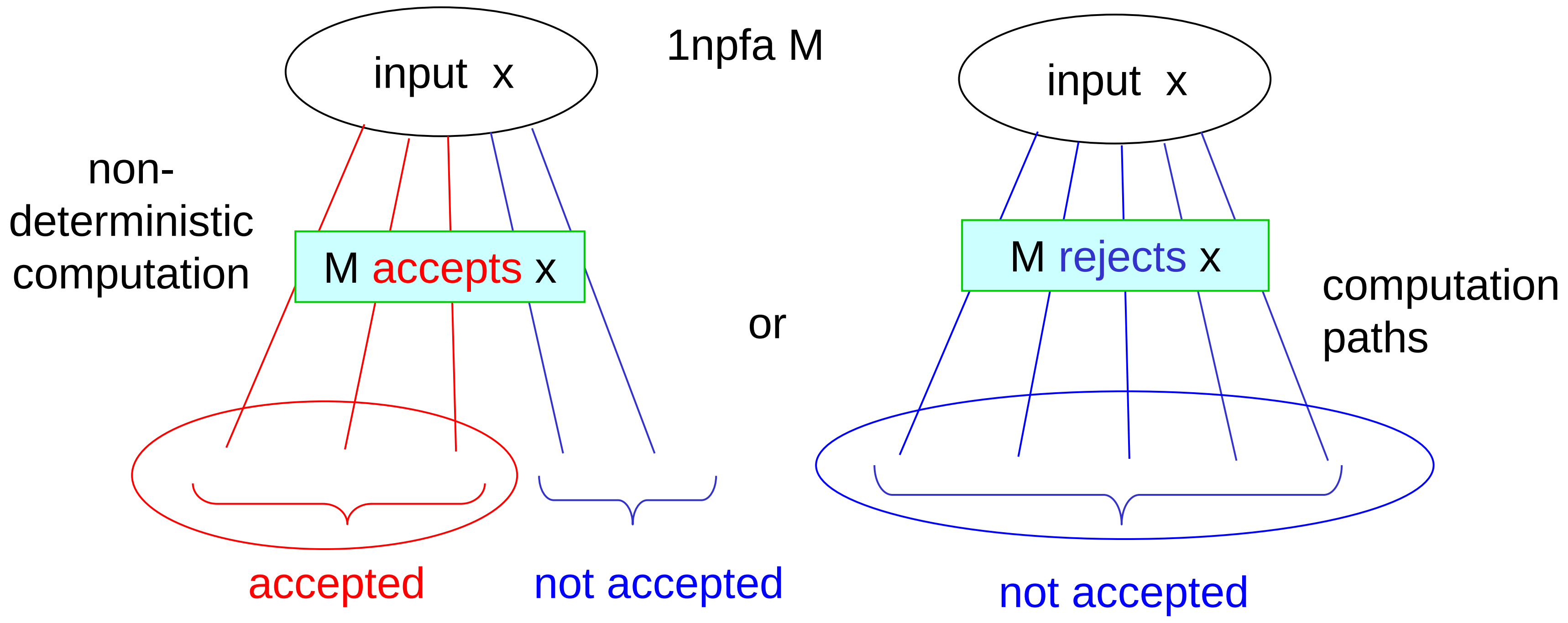
- Consider a machine (such as 1dfa and 1npda) M.
- A configuration is, roughly, an instantaneous description of the machine's current internal situation.
  - In the case of 1dfa M, a configuration of M on input x is of the form (q,z) with $q \in Q$ and $z \in \Sigma^*$, which describes that M is in state q and z is the unread substring of the input x.
  - In the case of 1npda M, a configuration of M on input x is of the form (q,z,w), which states that M is in state q with unread substring z of x and stack content w.
- The initial configuration of M on input x is a configuration (q_0,x) and (q_0,x,Z) for 1dfa and 1npda, respectively.
- A final (or halting) configuration is a configuration with a final state or the unread string becomes empty in case of 1-way head move.

Computation

- A computation of M on input x is a series $(c_0,c_1,c_2,...)$ of configurations such that
  - c_0 is the initial configuration of M on x and
  - for each $i \geq 0$, c_{i+1} is obtained from c_i by making a single step of M.
- Moreover, when M halts (either M enters a final state or it reads up all input symbols), the computation must end with a final configuration.

Acceptance and Rejection for 1nfa's

- If a 1nfa's current state q is a member of F along a certain computation path, the machine M is said to have accepted the input string.
- An input that is not accepted is said to be rejected.



2-Way Finite (State) Automata

- If we allow a finite automaton to move its tape head in both directions as well as make the tape head stay still (called a stationary move), then we obtain a computational model of 2-way finite (state) automaton (or simply, 2dfa) with two endmarkers.

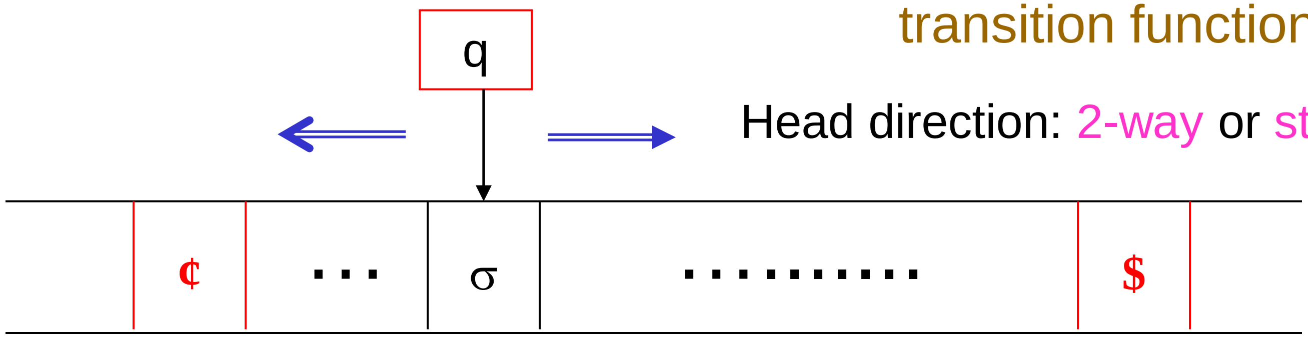
$M = (Q,\Sigma,\{\cent,\$\},\delta,q_0, Q_{acc},Q_{rej})$

Σ = input alphabet

$Q_{acc} \cup Q_{rej} \subseteq Q$

δ : a probabilistic transition function

Inner state $q \in Q$



Head direction: 2-way or stationary

End-marker　　Infinite read-only input tape　　End-marker

Formal Definition of 2dfa's

A 2dfa M = $(Q,\Sigma,\{¢,\$\},\delta,q_0, Q_{acc},Q_{rej})$ has a read-only input tape and a transition function δ of the form:

$$\delta : (Q - Q_{halt}) \times \breve{\Sigma} \rightarrow Q \times D$$

$\breve{\Sigma} = \Sigma \cup \{ ₵, \$ \}$

D = { -1, 0, +1 }

$Q_{halt} = Q_{acc} \cup Q_{rej}$

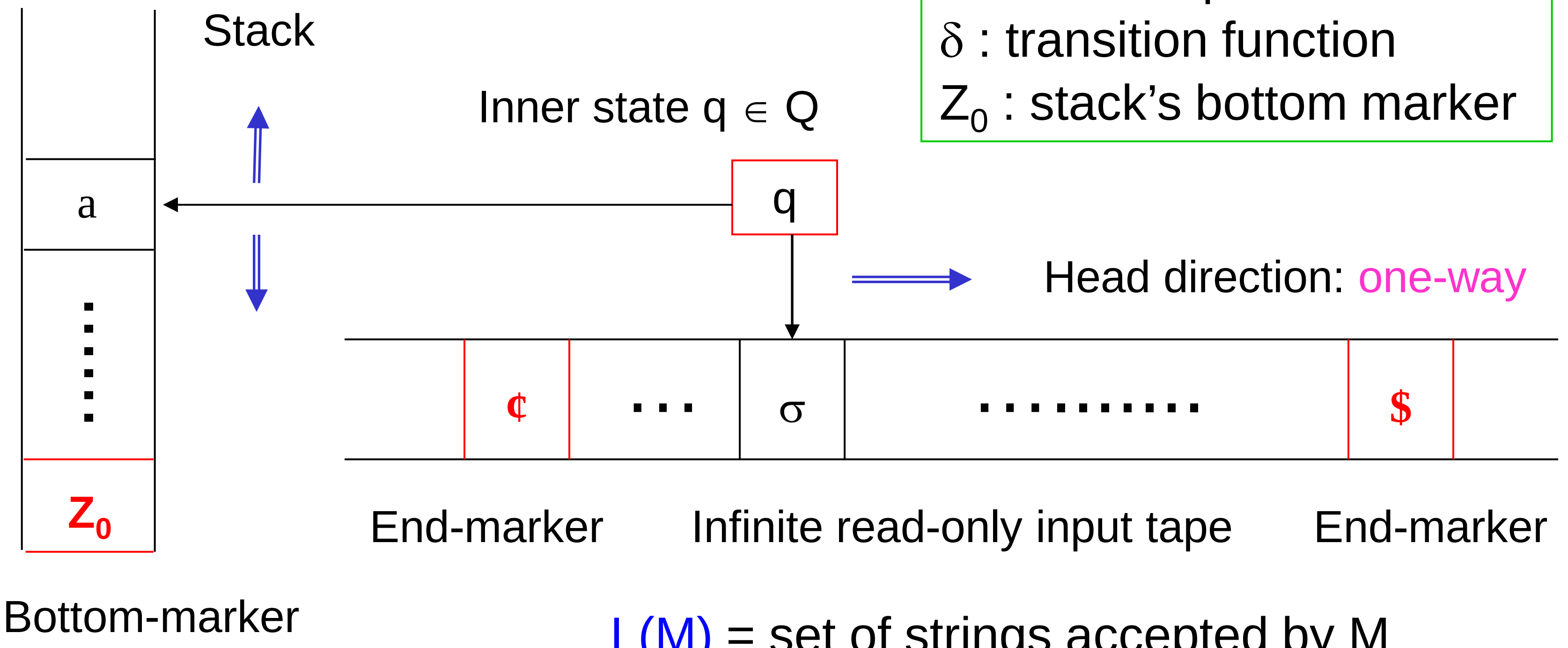
- Endmarker condition:
  - ✓ No tape head should move out of the region marked between ¢ and \$.
- Acceptance and Rejection:
  - ✓ When a 2dfa enters an accepting state (in Q_{acc}) and a rejecting state (in Q_{rej}), then the 2dfa halts and accepts and rejects a given input, respectively.

1-Way Pushdown Automata

Let us review a model of 1-way nondeterministic pushdown automaton (or 1npda).

$M = (Q,\Sigma,\{\cancel{c},\$\},\Theta,\delta,q_0,Z_0,F)$

Q,q_0,F are the same
Σ = input alphabet
Θ = stack alphabet
δ : transition function
Z_0 : stack's bottom marker



L(M) = set of strings accepted by M

Transition Functions of 1npda's

- The arguments of a transition function δ of a 1npda are the current state of the control unit, the current input symbol, and the current symbol on top of the stack.
- The result is a set of pairs (q,u), where q is the next state of the control unit and u is a string which is put on top of the stack in place of the single symbol there before.
- The second argument of δ may be λ, indicating that a move that does not consume an input symbol is possible. Such a move is called a λ-move (or λ-transition).
- No 1npda can remove Z_0 from the stack at any step.
- A 1npda may have several choices for its move.
- The use of λ-move is crucial for pushdown automata to exercise their full computational power.

Formal Definition of 1npda's

A 1npda $M = (Q,\Sigma,\{\lambda, ¢, \$\},\delta,q_0, Q_{acc},Q_{rej})$ has a read-only input tape, a stack and a transition function δ of the form:

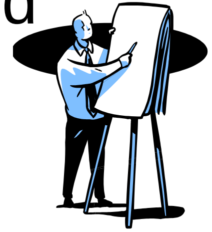
$$\delta : (Q - Q_{halt}) \times (\check{\Sigma} \cup \{\lambda\}) \times \Theta \to \wp(Q \times \Theta^*)$$

$\check{\Sigma} = \Sigma \cup \{ ¢, \$ \}$

$D = \{ -1, 0, +1 \}$

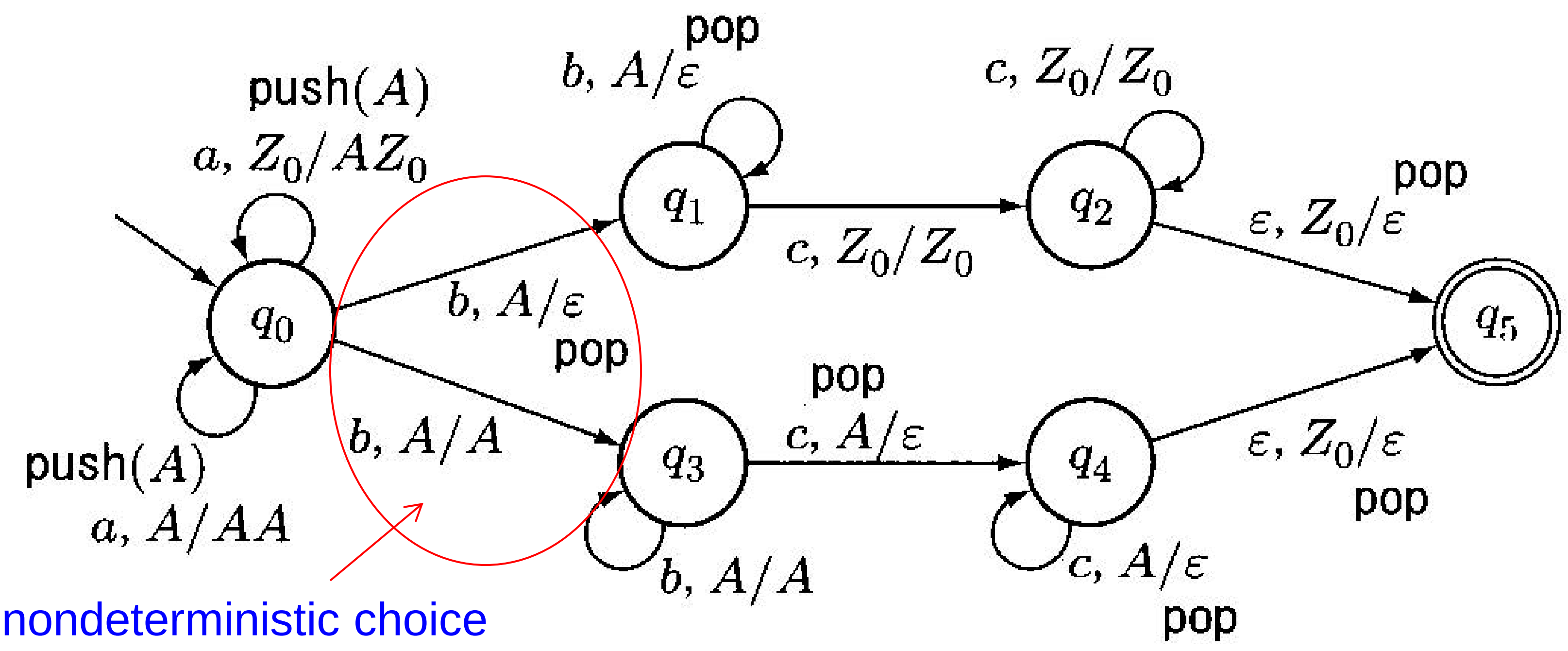
$Q_{halt} = Q_{acc} \cup Q_{rej}$

- Stack usage:
  - ✓ A 1npda scans only the topmost stack symbol together with/without each input symbol, including ¢ and \$.
- Acceptance and Rejection:
  - ✓ When a 1npda enters an accepting state (in Q_{acc}) and a rejecting state (in Q_{rej}), then the 1npda halts and accepts and rejects a given input, respectively.



Examples of 1npda's

M: 1npda



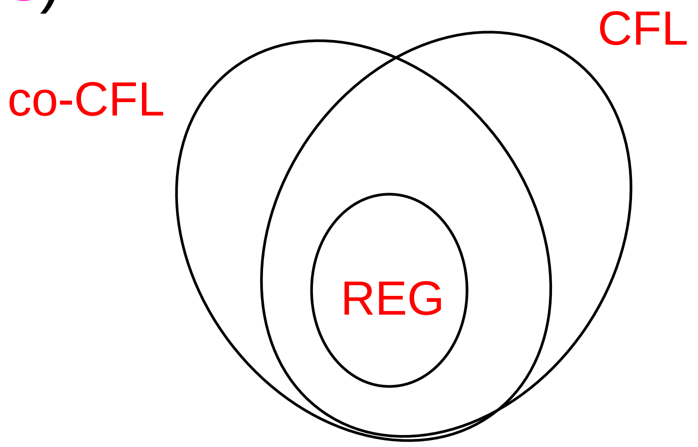
$L(M) = \{ a^n b^m c^n \mid n,m \geq 1 \} \cup \{ a^n b^n c^m \mid n,m \geq 1 \}$

Complexity Classes: REG and CFL

- We use the following abbreviations for machines.
  - ✓ 1dfa = 1-way deterministic finite automaton
  - ✓ 1npda = 1-way nondeterministic pushdown automaton

- REG = collection of all languages recognized by 1dfa's (i.e., regular languages)
- CFL = collection of all languages that are recognized by 1npda's (i.e., context-free languages)
- co-CFL = $\{ L^c \mid L \in \text{CFL} \}$

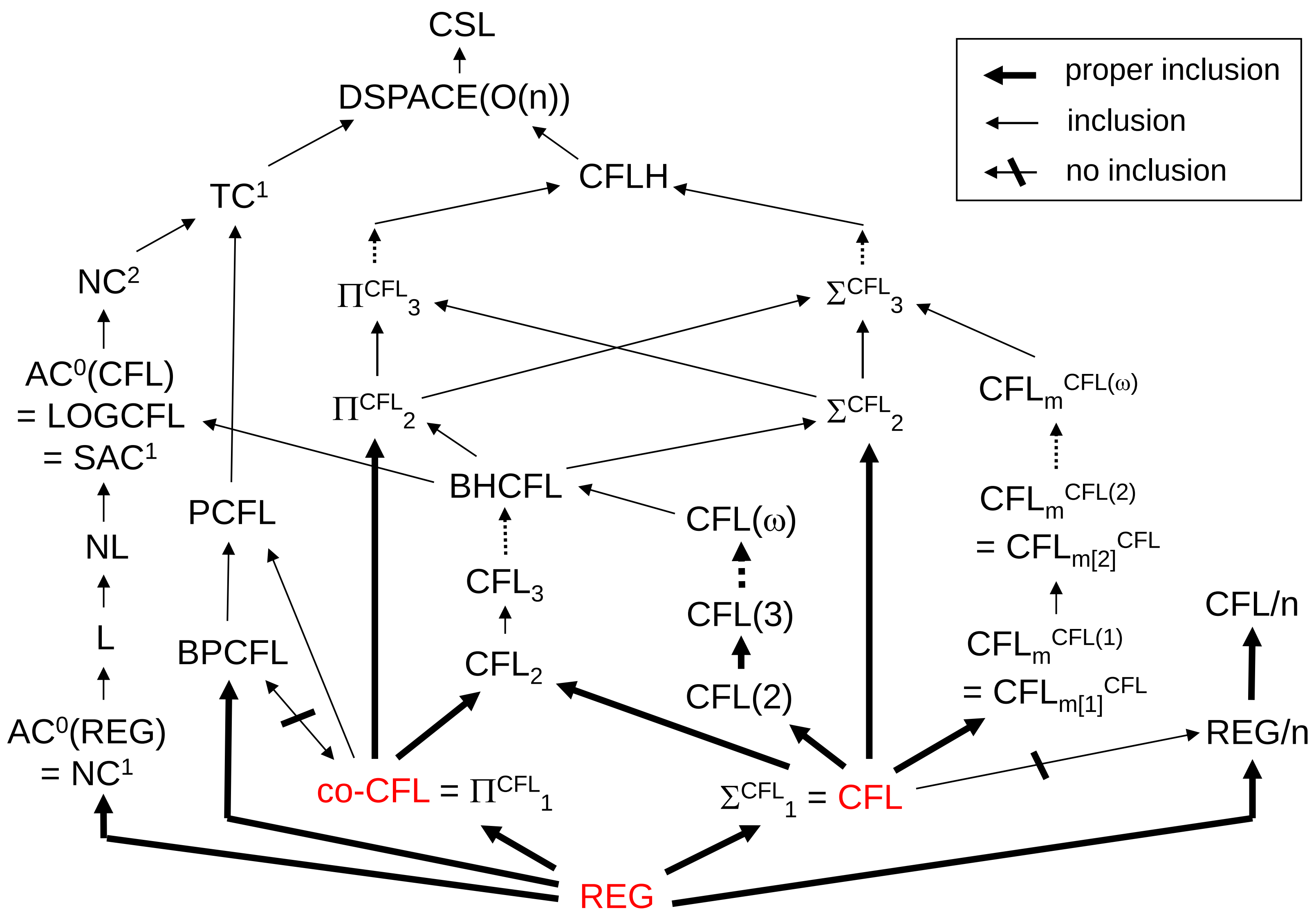
- For the separations among the above language families, see the next slide.



Known Facts on REG, CFL, and co-CFL

- Here is a short list of well-known facts on 1-way finite automata.
- (Claim) REG $\subseteq$CFL but REG $\neq$ CFL.
  - ✓ Upal = $\{a^n b^n \mid n \geq 1\} \in$ CFL – REG
  - ✓ Pal = $\{ x \in \{0,1\}^* \mid x^R = x \}$ is in CFL – REG.
- The above non-regularity results are obtained by the pumping lemma for 1dfa's.
- (Claim) CFL $\neq$ co-CFL.
  - ✓ Diff = $\{ a^i b^j c^k \mid i \neq j \text{ or } j \neq k \}$ is in CFL but not in co-CFL.
- The pumping lemma was proposed by Bar-Hillel, Perles, and Shamir (1961).

Inclusion Relations among Language Families



Formal Definition of 1dpda's

A 1dpda M = ($Q,\Sigma,\{\lambda, ¢, \$\},\delta,q_0, Q_{acc},Q_{rej}$) has a read-only input tape, a stack and a transition function δ of the form:

$$\delta : (Q - Q_{halt}) \times (\breve{\Sigma} \cup \{\lambda\}) \times \Theta \rightarrow \wp(Q \times \Theta^*)$$

$Q_{halt} = Q_{acc} \cup Q_{rej}$

$\breve{\Sigma} = \Sigma \cup \{ ₵, \$ \}$

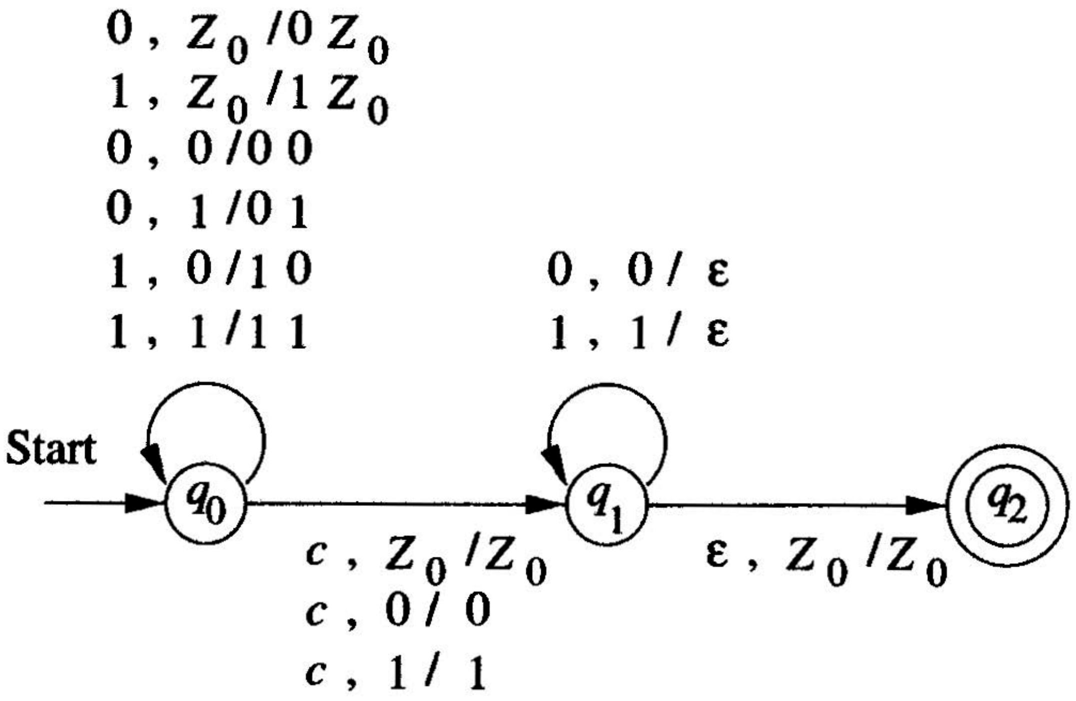
D = { -1, 0, +1 }

- Deterministic requirement:
  - ✓ A deterministic transition function must satisfy the following condition:
    (1) The output set of δ contains at most one element, and
    (2) At every step, the next move of M is uniquely determined, including λ-moves.
- DCFL = collection of all languages recognized by 1dpda's

Examples of 1dpda's

- The marked even-length palindrome:

  Mark-Pal = { wcw^R | w∈ {0,1}* }.
- Here is a 1dpda that recognizes Mark-Pal.



III. Turing Machines

1. Languages Accepted by Algorithms
2. Turing Machines
3. Time and Space Complexity
4. One-Tape Linear-Time Turing Machines
5. Polynomial-Time Algorithms
6. Complexity Classes P, NP, co-NP
7. Alternating Finite Automata
8. Alternating Turing Machines

Languages Accepted by Algorithms

- An “algorithm” is an informal term for “mechanical procedure.”
- We say that an algorithm A accepts a string $x \in \Sigma^*$ if, given input x, the algorithm’s output A(x) is 1.
- Similarly, an algorithm A rejects a string if A(x) = 0.

- The language accepted by algorithm A is the set of strings $L = \{ x \in \Sigma^* \mid A(x) = 1 \}$; that is, the set of strings that the algorithm accepts.
- A language L is decided (or recognized) by algorithm A if every string in L is accepted by A and every string not in L is rejected by A.

Acceptance vs. Decision

- A language L is accepted in time t(n) by algorithm A if
  1. it is accepted by A and
  2. If, in addition, for any length-n string $x \in L$, algorithm A accepts x in time t(n).
- A language L is decided (or recognized) in time t(n) by algorithm A if, for any length-n string $x \in \Sigma^*$, the algorithm correctly decides whether $x \in L$ in time t(n).

- To accept a language L $\rightarrow$ A accepts every string in L.
- To decide (or recognize) a language L $\rightarrow$ A accepts and rejects every string in Σ^* correctly.

Time and Space Complexity

- Let L be a problem and let s,t be functions from N to N.
- We say that an algorithm solves a (decision) problem in time O(t(n)) if, when it is provided a problem instance x of length n=|x|, the algorithm can produce the solution in O(t(n)) time.
- Similarly, we define the notion of solving in space O(s(n)).
- The problem L is of time complexity t(n) if (i) there exists an algorithm that solves L in time t(n) for all length-n inputs.
- The problem L is of space complexity s(n) if (i) there exists an algorithm that solves L in space s(n) for all length-n inputs.

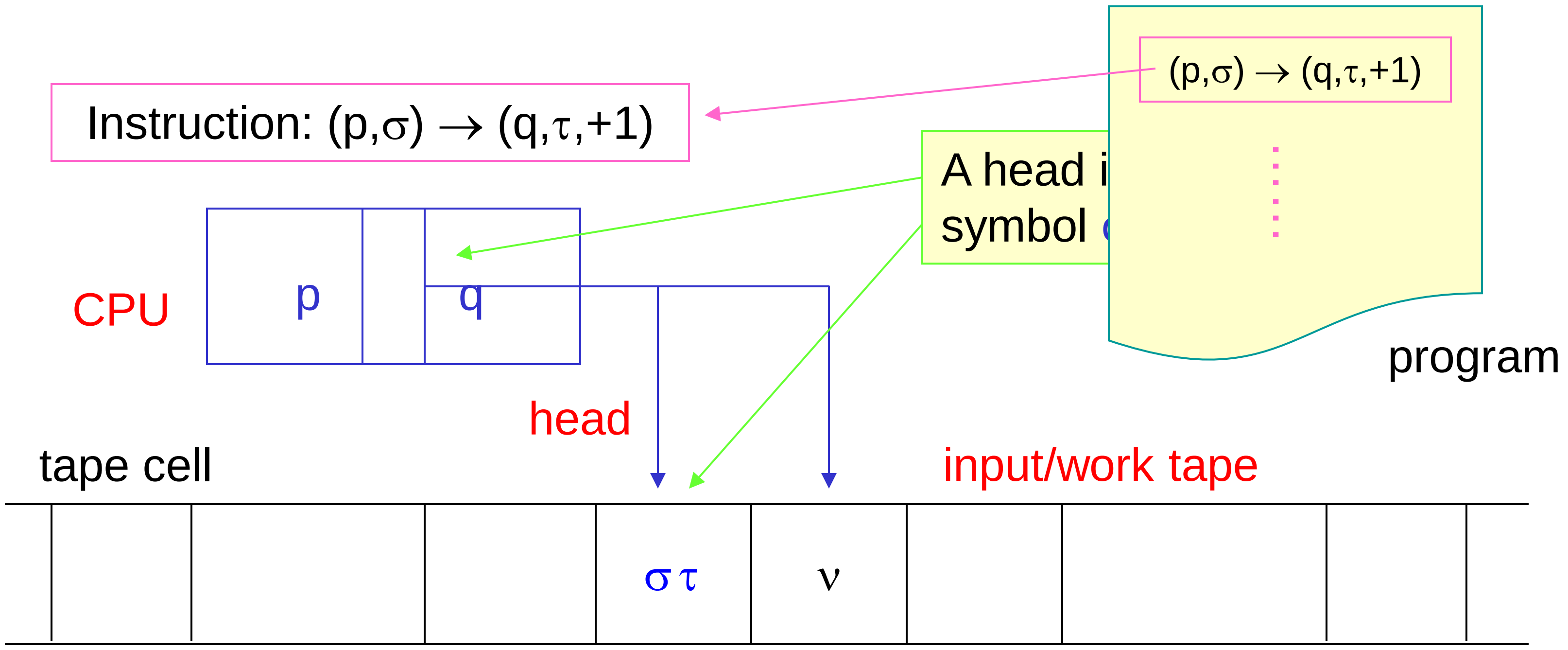
Turing Machines I

- In 1936, Alan Turing introduce the notion of Turing machine to realize “mechanical procedures.”
- This notion had become a blueprint of the existing computers.

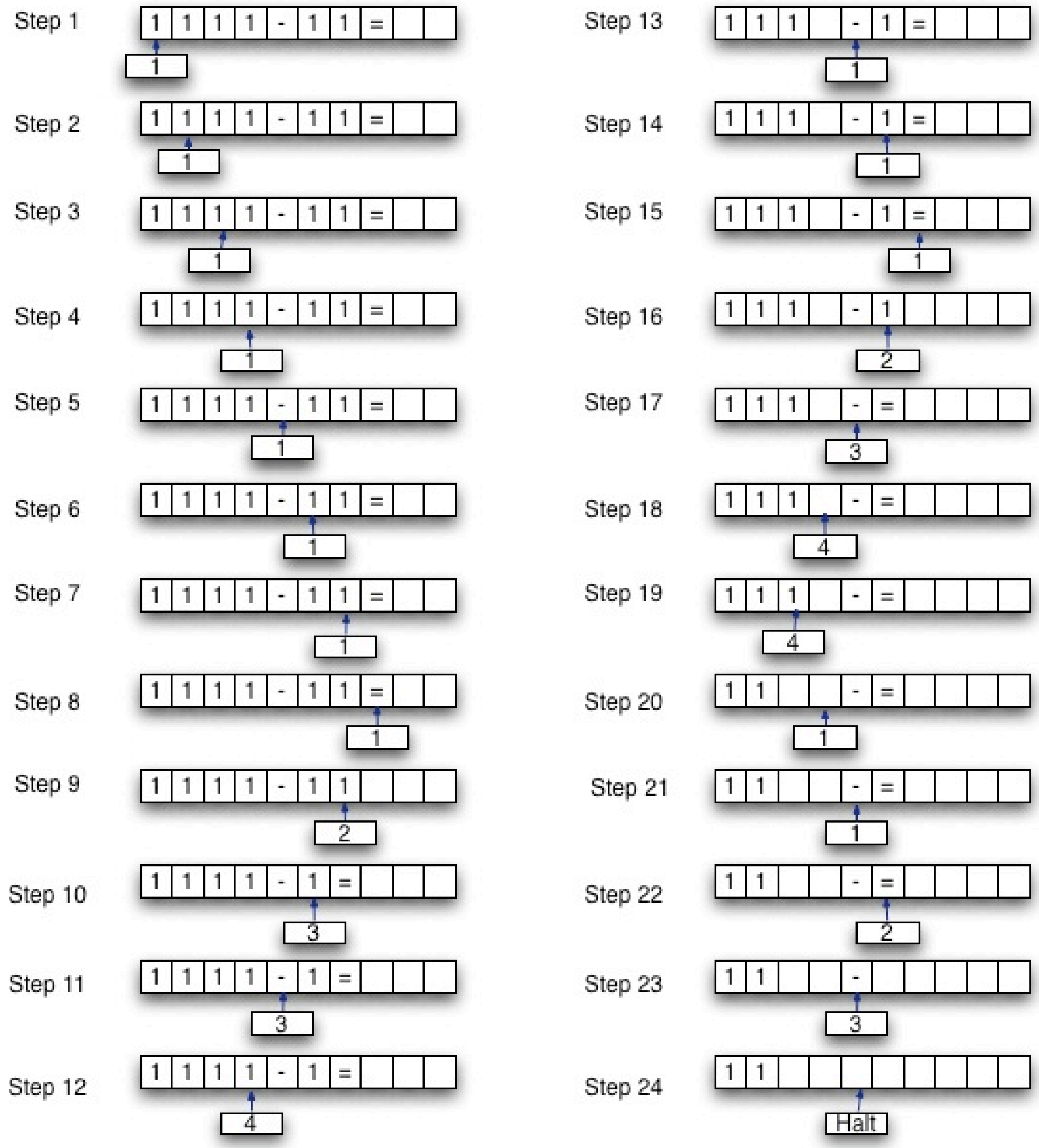
- “On Computable Numbers, with an Application to the Entscheidungsproblem,” 1936.

Turing Machines II

- A Turing machine consists of tape(s), tape head, and CPU.
- An input string is initially written on the input/work tape.
- The machine scans a symbol on the tape, follows a program, rewrites the tape symbol and the inner state of the CPU, and then moves the head.



a computation
=
a series of configurations
like these ones



One-Tape Turing Machines I

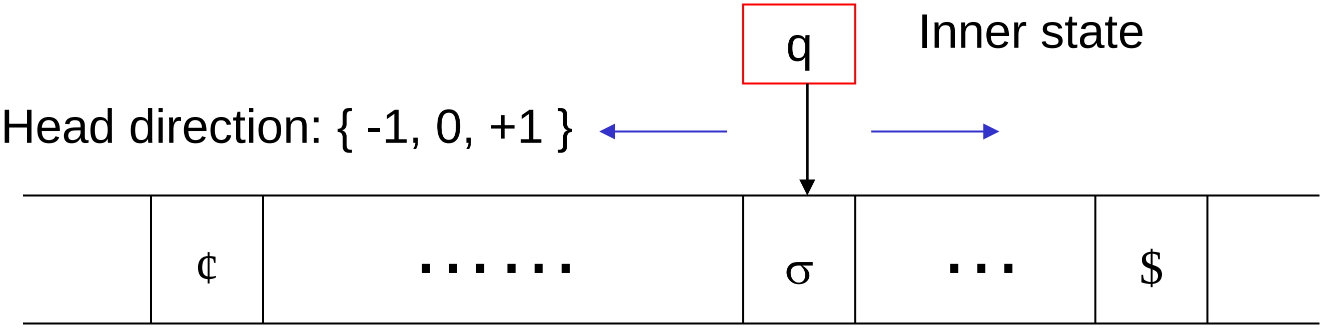
- We consider one-tape two-way (one-head) Turing machines.
- A one-tape two-way (one-head) Turing machine M is defined as follows.

$M = (Q,\Sigma,\{\cancel{¢},\$\},\delta,q_0,Q_{acc},Q_{rej})$ Σ = input alphabet

$\breve{\Sigma} = \Sigma \cup \{ ¢, \$ \}$

Deterministic case - $\delta: Q\times \breve{\Sigma} \rightarrow Q\times\Sigma\times\{-1,0,+1\}$

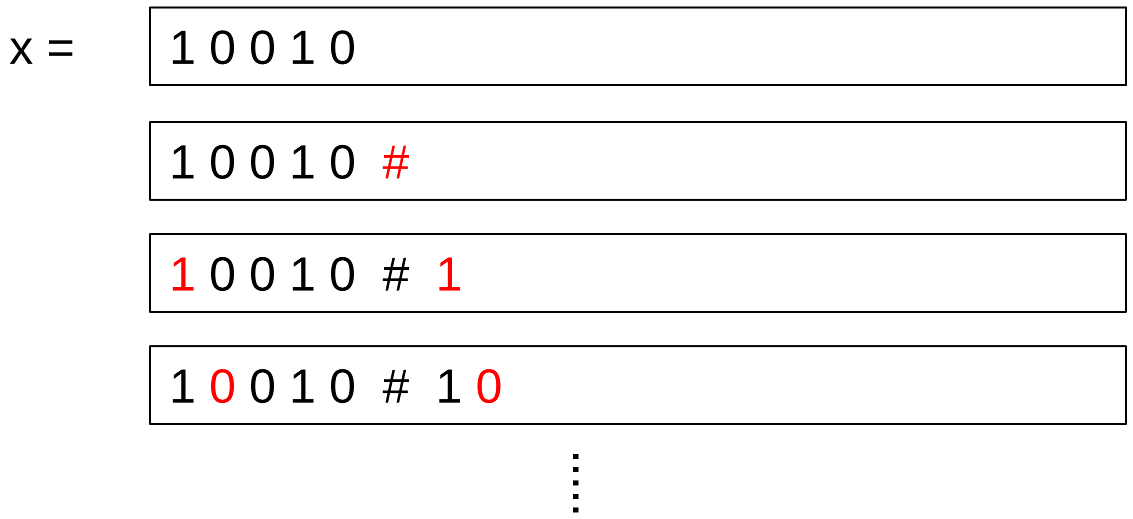
Nondeterministic case - $\delta: Q\times \breve{\Sigma} \rightarrow P(Q\times\Sigma\times\{-1,0,+1\})$



End-marker An infinite re-writable tape End-marker

Examples of Tape Head Moves

- Consider the following simple task.
- Task of modifying the current tape content
  - ✓ current tape content $x \in \Sigma^*$
  - ✓ produce x#x on an input/work tape
- To get this task done, a 1DTM requires $O(n^2)$ time by moving a tape head back and forth at most n times.

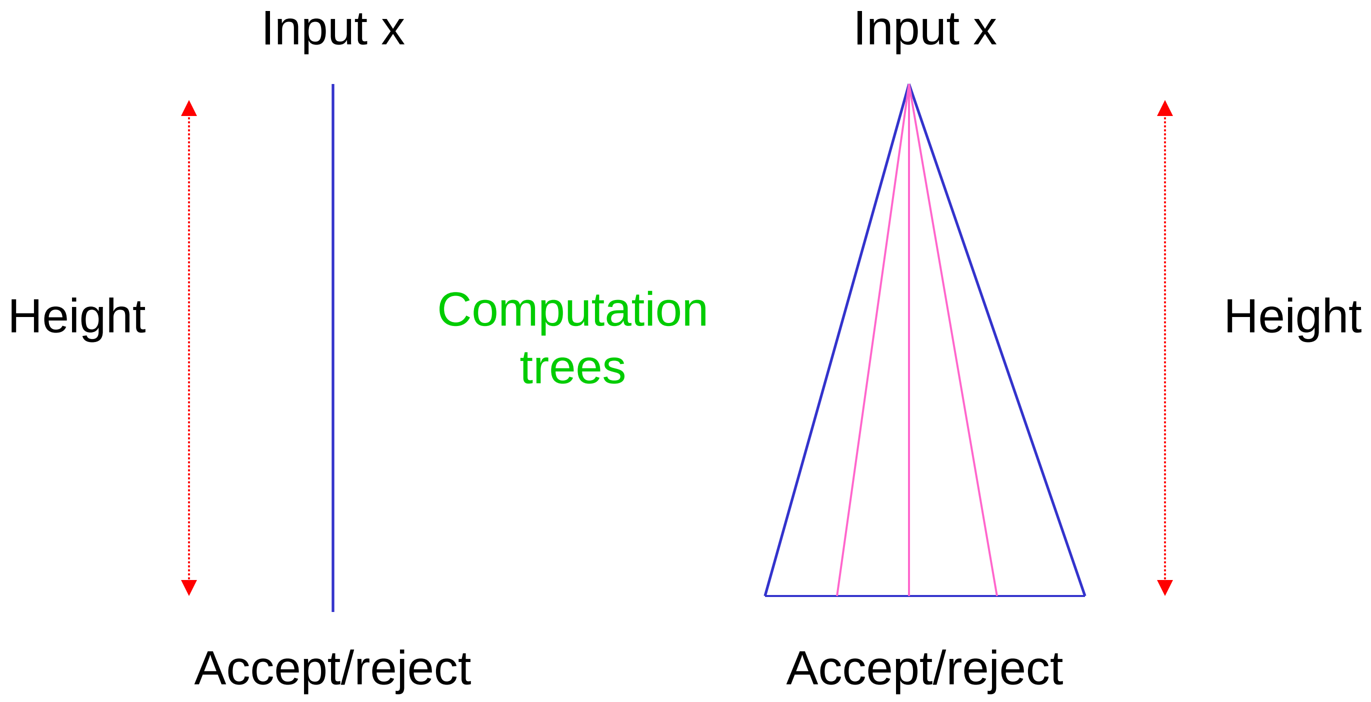


One-Tape Turing Machines II

- Major features
  - ✓ A tape of a one-tape Turing machine is used as an input/work tape, stretching to both the left and the right.
  - ✓ Initially, the tape is empty (i.e., filled with the blank symbols) except for an input string.
  - ✓ This machine can freely rewrite tape symbols, including ¢ and $.
  - ✓ This machine can use any blank space to remember any information necessary to carry out its computation.

Strong Definition for Running Time

- Michel (1991) proposed the notion of strong definition for running time of one-tape Turing machines.
- The running time is the length of the longest computation accepting/rejecting path.



One-Tape Linear-Time Turing Machines

- We use the following abbreviations:
  - 1DTM = one-tape deterministic Turing machine
  - 1NTM = one-tape non-deterministic Turing machine (using the strong definition for its running time)

- 1-DLIN = collection of all languages that are recognized by 1DTMs in linear time (i.e., O(n) time)
- 1-NLIN = collection of all languages that are recognized by 1DTMs in linear time

- NOTE: All accepting and/or rejecting computation paths of a 1NTM are always terminated within O(n) time.

One-Tape Linear-Time Classes

- The complexity classes 1-DLIn and 1-NLIN turn out to be closely related to finite automata.
- It is obvious that REG $\subseteq$ 1-DLIN $\subseteq$ 1-NLIN.

- Moreover, we can show that the computational power of 1NTMs is significantly limited when their running time is restricted to linear time.

- Collapse results
  - 1-DLIN = 1-NLIN = 1-DTIME(o(nlog(n))) = REG [Hennie65,Kobayashi85,Tadaki-Yamakami-LinL04]

Polynomial-Time Algorithms

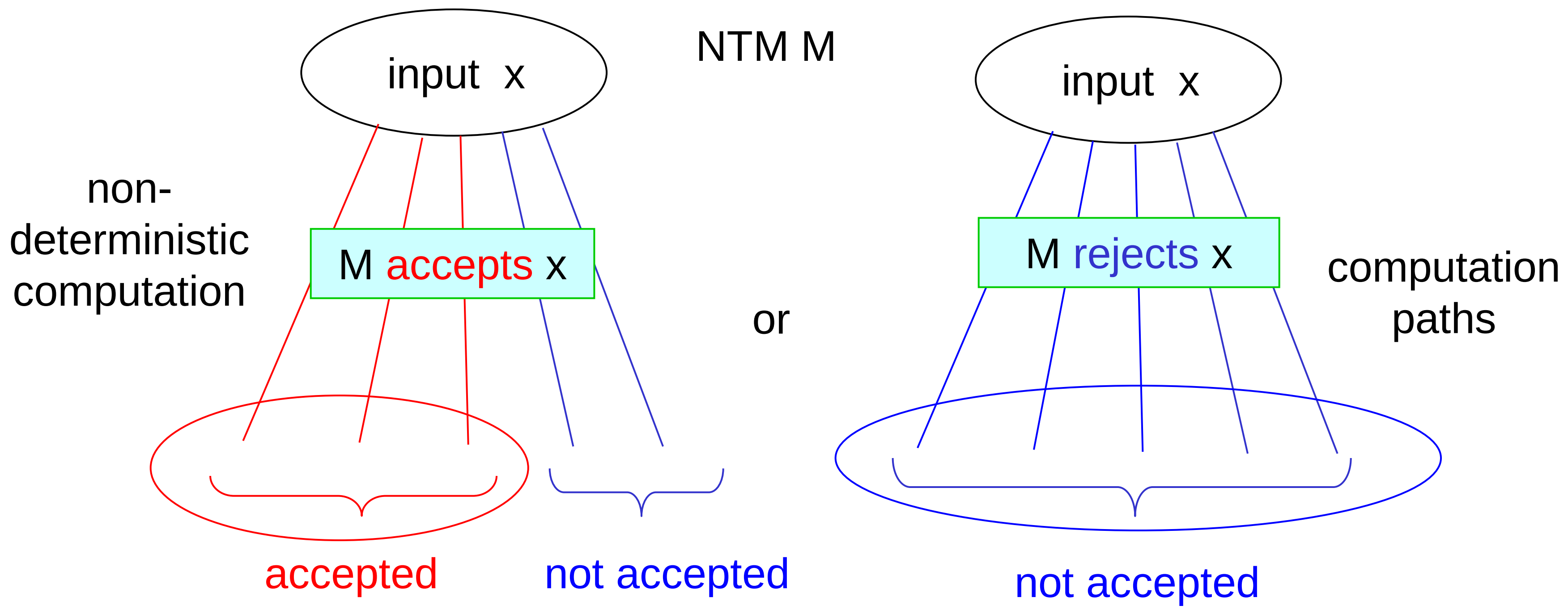
- Next, we are interested in polynomial time computation made by Turing machines.
- A polynomial-time algorithm takes inputs of size n and their worst-case running time is $O(n^k)$ for a certain fixed constant k.
- Question: Can all problems be solved by polynomial-time algorithms?
- Answer: No!
- Unfortunately, there are problems, such as Turing's "Halting Problem," that cannot be solved by any computer, no matter how much time we allow.

Polynomial-Time Solvable Problems

- Let t be a time-bounding function from N to N.
- We say that an algorithm (or a deterministic Turing machine) solves a (decision) problem in time O(t(n)) if, when it is provided a problem instance x of length n=|x|, the algorithm can produce the solution in O(t(n)) time.
- A (decision) problem is polynomial-time solvable if there exists an algorithm to solve it in time $O(n^k)$ for some constant k.

Acceptance vs Rejection for NTMs

- On input x, an NTM M is said to accept x if M enters an accepting state along a certain computation path.
- An input that is not accepted is said to be rejected.

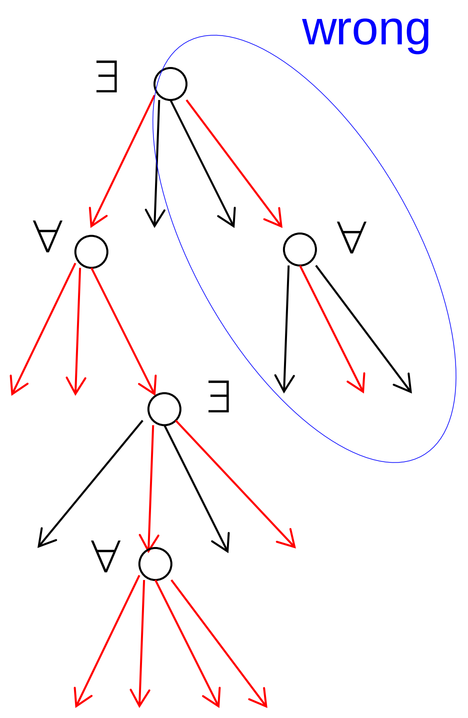


Complexity Classes P, NP, and co-NP

- "Polynomial time" means that the running time of a machine requires at most n^c steps for any input of length n, where c is a suitable constant, independent of n.
- We define three complexity classes associated with polynomial-time computability.
  - P = set of all decision problems solvable by deterministic Turing machines in polynomial time
  - NP = set of all decision problems solvable by nondeterministic Turing machines in polynomial time
  - co-NP = set of all complements of NP problems
- We will return to these complexity classes in Week 2.

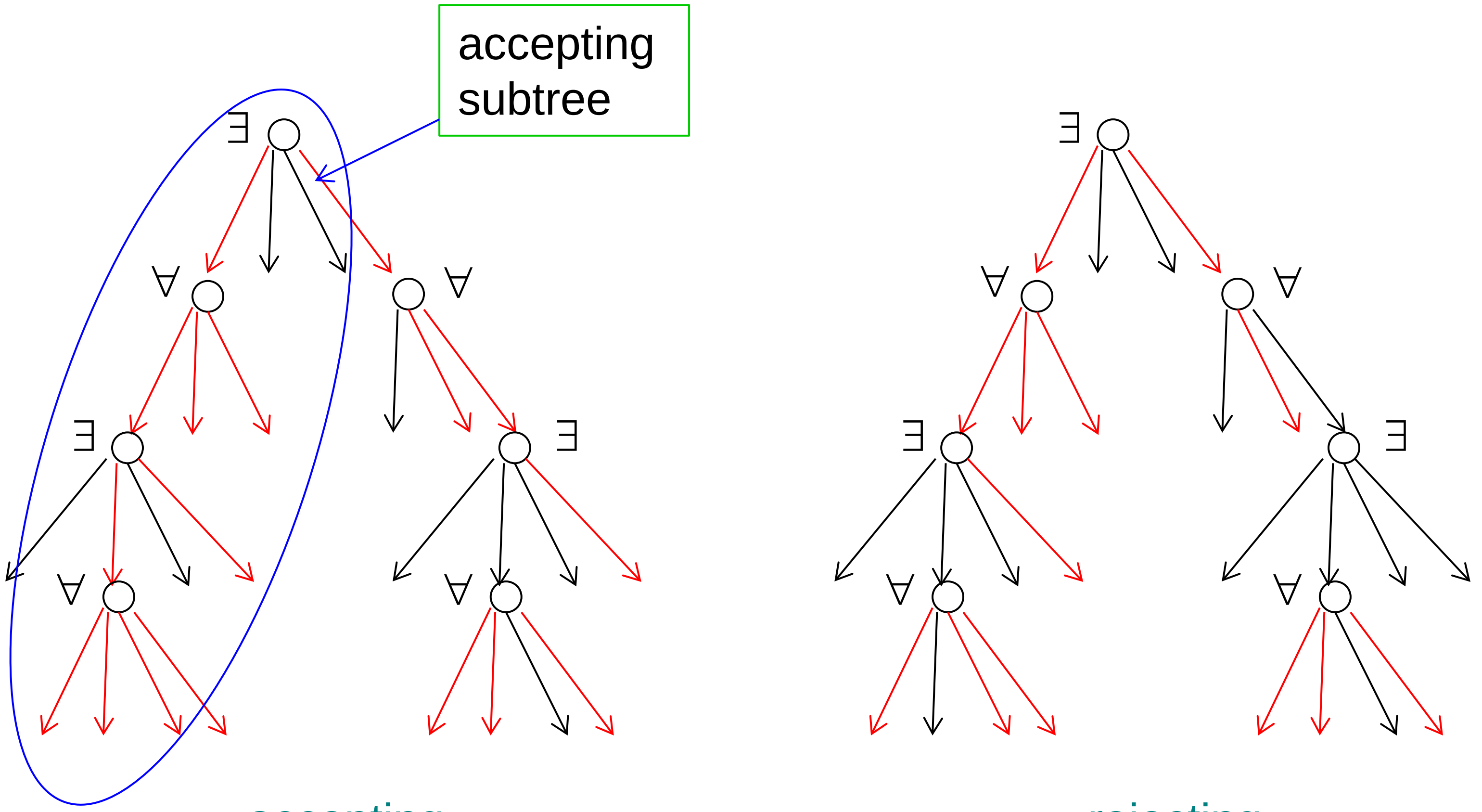
Alternating Finite (State) Automata I

- A 2-way alternating finite (state) automaton (2afa) is a generalization of a 2-way nondeterministic finite automaton (2nfa).
- A 2afa's inner states are partitioned into a set $Q_\exists$ of existential states and a set $Q_\forall$ of universal states.
- Roughly speaking, at each step, if the 2afa's inner state is existential, then it makes nondeterministically choose an accepting branch, and if its inner state is universal, then it checks that all branches are accepting.



bad computation tree

Alternating Finite (State) Automata II



accepting
computation tree

rejecting
computation tree

Alternating Turing Machines

- Similar to 2afa's, an alternating Turing machine (ATM) is also a generalization of a nondeterministic Turing machine (NTM).
- An ATM's inner states are partitioned into a set of existential states and a set of universal states.

- ATIME,SPACE(t(n),s(n)) = set of all decision problems solvable by ATMs in time at most t(|x|) using space at most s(|x|) on all inputs x.

- If there is no time bound t(n) or space bound s(n), we use the notation * (asterisk).

Relationships of ATMs to Other Machines

- There are close connections between alternating machines and deterministic machines.

- (Claim) [Chandra-Kozen-Stockmeyer (1981)]
  - ATIME,SPACE(*,log) = P
  - ATIME,SPACE(*,poly) = EXP
  - ATIME,SPACE(poly,*) = PSPACE

- (*) EXP is defined by exponential-time DTMs. PSPACE (polynomial space class) will be discussed extensively in Week 3.

VI. Functions and Function Classes

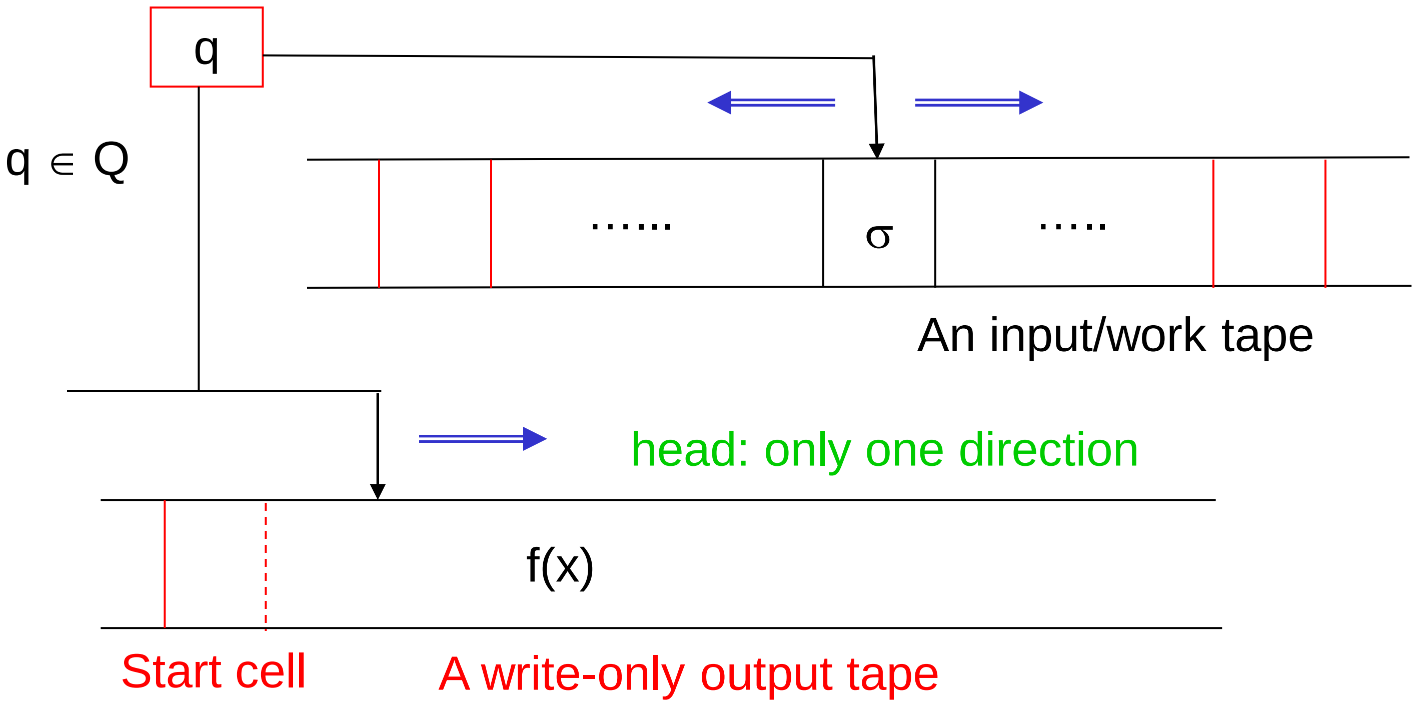
1. Decision Problems vs. Functions
2. Computing Functions Using Output Tapes
3. Function Class 1-FLIN
4. Polynomial-Time Computable Functions
5. Function Class FP
6. Recursive Functions

Decision Problems vs. Functions

- A decision problem (or a language) is a problem of determining whether an instance satisfies a certain property by accepting or rejecting the instance.
- In other words, a decision problem is to output a single bit; YES (1) or NO (0).
- In contrast, a function is to produce outcomes (or outputs) from each instance.
- By encoding "objects" into strings, we can handle any functions that map "objects" to "objects."
- Many problems are categorized as functions.
- For example, search problems, optimization problems, counting problems, etc.

Computing Functions Using Output Tapes

- Consider functions mapping from Σ* to Σ* (or N), where N={0,1,2,…} is the set of all natural numbers.
- To compute a function, we need to equip a machine with an extra write-only output tape whose tape head moves in one direction.



Polynomial-Time Computable Functions

- Let f be a function mapping Σ* to Σ* (or N).
- Let t be a time-bounding function from N to N.

- We say that a DTM M computes f in time O(t(n)) if, when M is provided a problem instance (or input) x of length n, M produces the outcome of the function on an output tape in O(t(n)) time.

- A function is said to be polynomial-time computable if there exists a DTM to compute it in time $O(n^k)$ for some constant k.

Function Class FP

- We introduce the function class, called FP.
- A function f: $\Sigma^* \to \Sigma^*$ (or N) is in FP $\Leftrightarrow$ there exists a polynomial-time DTM M such that, for every input $x \in \Sigma^*$,

  f(x) = outcome of M on x.
- Given a language A, χ_A denotes the characteristic function of A; that is,

$$\chi_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{otherwise} \end{cases}$$

- (Claim) For any language A, the following are equivalent.

1. $A \in P$.
2. $\chi_A \in FP$.

Output Convention for 1-Tape Machines

- Since a 1DTM M has only one input/work tape, we need to designate the same input tape as the output tape of the machine as well.
- To specify an “outcome” of the machine, we adopt the following convention.
- Output Convention
  - When the machine eventually halts with its output tape consisting only of a single block of non-blank symbols, say s, surrounded by the blank symbols, in a way that the leftmost symbol of s is written in the start cell, we consider s as the valid outcome of the machine.

Function Class 1-FLIN

- We pay attention to functions computed by 1-tape TMs, which treat the input/work tape as also an output tape.
- We say that a 1DTM M computes function f if, on any input x given onto an input/work tape, M writes down f(x) on the same tape, enters an accepting state, and halts.
- Similarly to 1-DLIN, we define its functional version.
- 1-FLIN = class of all functions computed by 1DTMs in linear time (i.e., O(n) time)

- Simple Example: A function f defined by
  - input: $x \in \{0,1,\#\}^*$
  - output: y_2 s.t. $\exists y_1,y_2 \in \{0,1\}^* \ \exists y_3 \in \{0,1,\#\}^* \ [\ x = y_1\#y_2\#y_3\ ]$

Recursive Functions

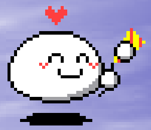
- If we do not place any bound on the running time of underlying DTMs (with output tapes) but we guarantee that the DTMs halt eventually, then we obtain “recursive” functions.
- Let f be a function mapping from Σ^* to Σ^* (or N).
- A function f is recursive (or computable) if there exists a DTM M such that, for any x, M produces f(x) on an output tape and then halts.
- A language L is recursive (or computable) if χ_L is a recursive function.

Non-Recursive Languages

- Consider the following decision problem.

- Halting Problem: HALT
  - instance: a (description) of a 1DTM M and an input string x
  - question: does M halt on input x?

- (Claim) HALT is not recursive.
- (Proof Idea) The proof is done by contradiction. This contradiction can be obtained by employing a sort of the Epimenides paradox: Epimenides was a Cretan who made one immortal statement "All Cretans are liars."



Thank you for listening

Q & A

I’m happy to take your question!

END