

CSL853: Complexity Theory

Ragesh Jaiswal, CSE, IIT Delhi

A quick recap. of Theory of Computation

Modeling Computation

Finite State Machines

Definition (Finite-state machine)

A finite-state machine $M = (S, I, O, f, g, s_0)$ consists of a finite set S of states, a finite input alphabet I , a finite output alphabet O , a transition function f that assigns to each state and input pair a new state, an output function g that assigns to each state and input pair an output, and an initial state s_0 .

Modeling Computation

Finite State Machines: Language recognition

- One of the most important applications of finite-state machines is in *language recognition*.

Definition (Vocabulary, language)

A *vocabulary* (or *alphabet*) V is a finite, nonempty set of elements called symbols. A word (or sentence) over V is a string of finite length of elements of V . The empty string or null string, denoted by λ , is the string containing no symbols. The set of all words over V is denoted by V^* . A *language* over V is a subset of V^* .

Modeling Computation

Finite State Machines: Language recognition

Definition (Finite-state automaton)

A *finite-state automaton* $M = (S, I, f, s_0, F)$ consists of a finite set S of *states*, a finite *input alphabet* I , a *transition function* f that assigns a next state to every pair of state and input (so that $f : S \times I \rightarrow S$), an *initial* or *start state* s_0 , and a subset F of S consisting of *final* (or *accepting states*).

- We can represent finite-state automata using either state tables or state diagrams. Final states are indicated in state diagrams by using double circles.

Modeling Computation

Finite State Machines: Language recognition

- Given a finite-state machine $M = (S, I, f, s_0, F)$, transition function f can be extended to a function $f : S \times I^* \rightarrow S$ in the following manner:
 - (i) $f(s, \lambda) = s$ for every state $s \in S$; and
 - (ii) $f(s, xa) = f(f(s, x), a)$ for all $s \in S$, $x \in I^*$, and $a \in I$.

Modeling Computation

Finite State Machines: Language recognition

Definition (Language recognition)

A string x is said to be *recognized* or *accepted* by the machine $M = (S, I, f, s_0, F)$ if it takes the initial state s_0 to a final state, that is, $f(s_0, x)$ is a state in F . The language recognized or accepted by the machine M , denoted by $L(M)$, is the set of all strings that are recognized by M . Two finite-state automata are called *equivalent* if they recognize the same language.

Modeling Computation

Finite State Machines: Nondeterministic finite-state automaton

Definition (Nondeterministic finite state automaton (NFA))

A *nondeterministic finite-state automaton* $M = (S, I, f, s_0, F)$ consists of a set S of states, an input alphabet I , a transition function f that assigns a set of states to each pair of state and input (so that $f : S \times I \rightarrow P(S)$), a starting state s_0 , and a subset F of S consisting of the final states.

- Example:

TABLE 2		
State	f	
	Input	
	0	1
s_0	s_0, s_1	s_3
s_1	s_0	s_1, s_3
s_2		s_0, s_2
s_3	s_0, s_1, s_2	s_1

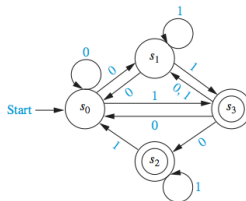


FIGURE 6 The Nondeterministic Finite-State Automaton with State Table Given in Table 2.

Modeling Computation

Finite State Machines: Nondeterministic finite-state automaton

Definition (Nondeterministic finite state automaton (NFA))

A *nondeterministic finite-state automaton* $M = (S, I, f, s_0, F)$ consists of a set S of states, an input alphabet I , a transition function f that assigns a set of states to each pair of state and input (so that $f : S \times I \rightarrow P(S)$), a starting state s_0 , and a subset F of S consisting of the final states.

- An NFA is said to recognize a string $x = x_1x_2...x_k$ if the following happens:
 - Let S_1 denote the set of states that the input symbol takes the machine from state s_0 .
 - Let $S_2^1, S_2^2, \dots, S_2^{|S_1|}$ denote the set of states to which x_2 takes the machine from each of the states in S_1 . Let S_2 be the union of all these sets.
 - Let $S_3, S_4, \dots, S_k$ be defined as in the previous step.
 - We say that x is accepted by the NFA iff S_k contains a final state.
- The language recognized by a nondeterministic finite-state automaton is the set of all strings recognized by this automaton.

Modeling Computation

Finite State Machines: Nondeterministic finite-state automaton

Theorem

If the language L is recognized by a nondeterministic finite-state automaton M_0 , then L is also recognized by a deterministic finite-state automaton M_1 .

Definition

A language L is called regular iff there is a DFA M such that $L(M) = L$.

- Are there languages that are not regular?
- Is the following language regular: $L_{01} = \{0^n 1^n | n \geq 1\}$?

Modeling Computation

Finite State Machines: Regular language

- Are there languages that are not regular?
- Is the following language regular: $L_{01} = \{0^n 1^n | n \geq 1\}$?

Theorem (Pumping lemma for regular languages)

Let L be a regular language. Then there exists a constant n (which depends on L) such that for every string w in L such that $|w| \geq n$, we can break w into three strings $w = xyz$, such that

- 1 $y \neq \lambda$.
- 2 $|xy| \leq n$.
- 3 For all $k \geq 0$, the string xy^kz is also in L .

Modeling Computation: Machine with more power than DFA/NFA

- DFA/NFA have limited amount of “memory”.
- Pushdown automata (PDA):
 - DFA with a *stack*.
 - On an input symbol this automaton can make a transition and push/pop symbols on the stack.
 - A string is said to be accepted if the automaton is in a final state and the stack is empty.
- Is there a PDA that recognizes $\{0^n 1^n | n = 0, 1, 2, \dots\}$?
- Are PDA more powerful than DFA/NFA?
- Fact: A language is recognized by a pushdown automata if and only if the language is generated by a *Context Free Grammar*.
- Fact: $\{0^n 1^n 2^n | n = 0, 1, 2, \dots\}$ is a language that cannot be recognized by a PDA.

Modeling Computation: Machine with more power than DFA/NFA

- DFA/NFA have limited amount of “memory”.
- Pushdown automata (PDA):
- Turing Machines:
 - DFA with a *tape(s)*.
 - The machine has a read/write head on the tape(s).
 - Depending on the current symbol read by the tape head, the machine makes a state transition, writes a symbol on the tape and moves left/right.
- Is there a Turing Machine that recognizes $\{0^n 1^n | n = 0, 1, 2, \dots\}$?

Turing Machines

k -tape Turing Machine

A k -tape Turing machine M may be described by a 3-tuple, (Γ, Q, δ) consisting of:

- A set Γ of the symbols that M 's tapes can contain.
 $\Gamma = \{\triangleright, \square, 0, 1\}$ and these are called *alphabets* of M . $\triangleright$ is called the *start* symbol and $\square$ is called the *blank* symbol.
- A set Q of M 's states. (Alternatively the set of states that M 's register can be in). Q contains a designated start state q_{start} and a designated halt state q_{halt} .
- A function $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^{k-1} \times \{L, S, R\}^k$ describing the transition rule. δ is called the *transition function*.

Modeling Computation: Turing machines

k -tape Turing Machine

A k -tape Turing machine M may be described by a 3-tuple, (Γ, Q, δ) consisting of:

- A set Γ of the symbols that M 's tapes can contain. $\Gamma = \{\triangleright, \square, 0, 1\}$ and these are called *alphabets* of M . $\triangleright$ is called the *start* symbol and $\square$ is called the *blank* symbol.
- A set Q of M 's states. (Alternatively the set of states that M 's register can be in). Q contains a designated start state q_{start} and a designated halt state q_{halt} .
- A function $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^{k-1} \times \{L, S, R\}^k$ describing the transition rule. δ is called the *transition function*.

Starting configuration:

- The input tape has $\triangleright$ in the leftmost tape cell, followed by the input, and followed by blank symbols everywhere else.
- All work/output tapes have $\triangleright$ in the leftmost tape cell and blank symbols everywhere else.
- All tape heads are at the left end of their respective tapes.
- The state of M is q_{start} .

Modeling Computation: Turing machines

k -tape Turing Machine

A k -tape Turing machine M may be described by a 3-tuple, (Γ, Q, δ) consisting of:

- A set Γ of the symbols that M 's tapes can contain. $\Gamma = \{\triangleright, \square, 0, 1\}$ and these are called *alphabets* of M . $\triangleright$ is called the *start* symbol and $\square$ is called the *blank* symbol.
- A set Q of M 's states. (Alternatively the set of states that M 's register can be in). Q contains a designated start state q_{start} and a designated halt state q_{halt} .
- A function $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^{k-1} \times \{L, S, R\}^k$ describing the transition rule. δ is called the *transition function*.

- Starting configuration:

- The input tape has $\triangleright$ in the leftmost tape cell, followed by the input, and followed by blank symbols everywhere else.
- All work/output tapes have $\triangleright$ in the leftmost tape cell and blank symbols everywhere else.
- All tape heads are at the left end of their respective tapes.
- The state of M is q_{start} .

- Transition: Transitions are executed using function δ .

- Halting: M halts off it enters the state q_{halt} . Moreover, after it enters q_{halt} , δ makes sure that no further modification to the tape is done.

Modeling Computation: Turing machines

Definition (Computing a function and running time)

Let $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ and let $T : \mathbb{N} \rightarrow \mathbb{N}$ be some functions, and let M be a Turing machine. We say that M computes f in $T(n)$ -time if for every $x \in \{0, 1\}^*$, if M is initialized to the start configuration on input x , then after at most $T(|x|)$ steps it halts with $f(x)$ written on its output tape. We say that M computes f if it computes f in $T(n)$ time for some function $T : \mathbb{N} \rightarrow \mathbb{N}$.

Definition (Time constructible functions)

We say that a function $T : \mathbb{N} \rightarrow \mathbb{N}$ is time constructible if $T(n) \geq n$ and there is a TM M that computes the function $x \rightarrow \text{Bin}(T(|x|))$ in time $T(n)$. ($\text{Bin}(T(|x|))$ denotes the binary representation of the number $T(|x|)$.)

Modeling Computation: Turing machines

- Consider the following function $PAL : \{0, 1\}^* \rightarrow \{0, 1\}$:

$$PAL(x) = \begin{cases} 1 & \text{if } x \text{ is a palindrome} \\ 0 & \text{otherwise} \end{cases}$$

Theorem

There is a TM M that computes the function PAL in $(3n)$ -time.

Modeling Computation: Turing machines

Alternate definitions

- There are other definitions of Turing Machine. So, why does the definition does not matter much as far our study of computation is concerned?
- For any function f and any TM M , as per a different definition, that computes f , there is a TM M' (as per our definition) that computes f *almost as efficiently* as M .
- Examples:

Theorem

For every $f : \{0, 1\}^* \rightarrow \{0, 1\}$ and time-constructible $T : \mathbb{N} \rightarrow \mathbb{N}$, if f is computable in time $T(n)$ by a TM M using alphabet Γ then it is computable in time $4 \log |\Gamma| T(n)$ by a TM M' using the alphabet $\{\triangleright, \square, 0, 1\}$.

End