Theory of Computation Notes

William Traub

Contents

1 September 9

Definition 1. *A function $f : D \to R$ has domain D and range R. Each input $x \in D$ is mapped to exactly one output $f(x) \in R$.*

Example 1. *The function $add\colon \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ is defined by*

$$add(x, y) = x + y.$$

Goal of Computation

We focus on computing functions $f : \Sigma^* \to \{\text{accept}, \text{reject}\}$.

- **Domain:** strings over alphabet Σ.
- **Range:** Boolean $\{0, 1\}$ or $\{\text{accept}, \text{reject}\}$.

Why strings? Any input can be encoded as a string. Why booleans? Simplicity, while still capturing many interesting functions.

Functions as Languages

A language L over Σ is a subset of Σ^*. Example: $L = \{w \in \{0, 1\}^* : w \text{ ends with } 1\} = \{1, 01, 11, 001, 101, \dots\}$.

Equivalence between functions and languages:

$$f \leftrightarrow L \quad \text{where} \quad L = \{w : f(w) = \text{accept}\}.$$

Observation

Languages may be finite or infinite, but a "program" is always a finite description.

Finite Automata

A **deterministic finite automaton (DFA)** consists of:

- States (nodes).
- Transitions labeled by alphabet symbols.
- Unique start state q_0.
- Accept states (double circles).

Definition 2. *A DFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where:*

- *Q = finite set of states*
- *Σ = alphabet*
- *$\delta : Q \times \Sigma \to Q$ = transition function*
- *$q_0 \in Q$ = start state*
- *$F \subseteq Q$ = accepting states*

Definition 3. *The extended transition function $\delta^* : Q \times \Sigma^* \to Q$ is defined by:*

$$\delta^*(q, \epsilon) = q, \quad \delta^*(q, wa) = \delta(\delta^*(q, w), a).$$

2 September 12

Theorem 1. *If A is regular, then so is its complement A^c.*

Definition 4. *A nondeterministic finite automaton (NFA) is a tuple $M = (Q, \Sigma, \gamma, q_{start}, F)$ where transitions may be nondeterministic or labeled with ϵ.*

An NFA accepts w if there exists some computation path leading to an accept state.

3 September 19

Theorem 2. *If A, B are regular languages, then so is $A \cup B$.*

Theorem 3. *If A is a regular language, then A^* is regular.*

Regular Expressions

Definition 5. *A regular expression (RE) over Σ is defined inductively:*

- *Atomic: $\emptyset$, ϵ, or $a \in \Sigma$.*
- *If R_1, R_2 are REs, then so are:*

$$(R_1 \cup R_2), \quad (R_1 R_2), \quad (R_1^*).$$

Given regular expressions R and S, the following operations over them are defined to produce regular expressions:

- Concatenation (RS): denotes the set of strings that can be obtained by concatenating a string accepted by R and a string accepted by S (in that order). For example, let R denote {"ab", "c"} and S denote {"d", "ef"}. Then, (RS) denotes {"abd", "abef", "cd", "cef"}.
- Alternation (R|S) denotes the set union of sets described by R and S. For example, if R describes {"ab", "c"} and S describes {"ab", "d", "ef"}, expression (R|S) describes {"ab", "c", "d", "ef"}.
- Kleene Star (R*) denotes the smallest superset of the set described by R that contains ε and is closed under string concatenation. This is the set of all strings that can be made by concatenating any finite number (including zero) of strings from the set described by R. For example, if R denotes {"0", "1"}, (R*) denotes the set of all finite binary strings (including the empty string). If R denotes {"ab", "c"}, (R*) denotes {ε, "ab", "c", "abab", "abc", "cab", "cc", "ababab", "abcab", ...}.

Definition 6. *The semantics of a RE R are given by its language $L(R)$:*

$$L(\emptyset) = \emptyset, \quad L(\epsilon) = \{\epsilon\}, \quad L(a) = \{a\}.$$

$$L(R_1 \cup R_2) = L(R_1) \cup L(R_2), \quad L(R_1 R_2) = L(R_1)L(R_2), \quad L(R^*) = (L(R))^*.$$

Theorem 4. *A language A is regular $\iff$ there exists a DFA, NFA, or regular expression R such that $A = L(R)$.*

4 September 26

4.1 Generalized NFA

Definition 7. *A Generalized NFA (GNFA) has transitions labeled with regular expressions. Can follow a transition by reading a matching block of input symbols.*

We can convert any GNFA into a "simple form":

1. Accept state is unique. No transitions leaving accept state. Add new accept state, ε-transitions from old accept states to new one.

2. No transitions entering the start state. Add new start state, ε-transitions from new start states to old one.

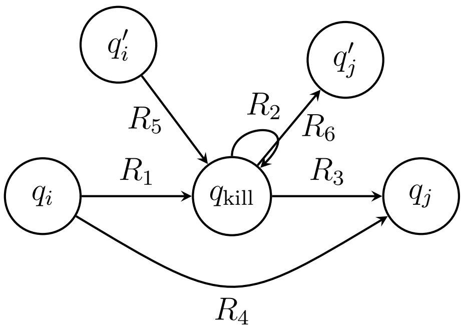
3. At most 1 transition between each pair of states. If multiple transitions between two nodes labeled $R_1, R_2, ...$, replace with transition $R_1 \cup R_2 \cup ...$

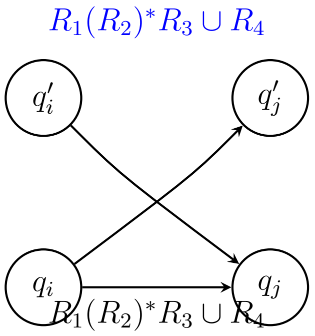
Theorem 5. *A is Regular $\Rightarrow$ $\exists$ reg. ex. R : $A = L(R)$.*

- Start with DFA D that recognizes A.

- Convert it to GNFA G that has "simple form".

- **Remove intermediate states of G one at a time without changing which language it recognizes.**
  - **Remove state q_{kill}**
  - **For each pair of states q_i, q_j connected through q_{kill} as shown, add a transition from q_i to q_j labeled with RE .**
  - **Continue to combine edges with $\cup$ until the GNFA is just a single RE**





Regular Expressions In Practice:

- Used widely for pattern-matching, searching.
  - Specify the format of a string such as a phone number, address, credit card number, license plate . . .
  - Search a document to see if it contains some credit card number.
- What algorithm is used:
  regex → NFA → DFA

4.2 Non-Regular Languages (Pumping Lemma)

Not all languages are regular. Consider: $L = \{0^n 1^n : n \geq 0\}$(e.g. $0011 \in L$ but $001 \notin L$)

Theorem 6. *Intuition: A DFA for L would need to remember how many 0's it has seen. Not enough memory*

Our Plan:

- Direct proof that $L = \{0^n 1^n : n \geq 0\}$ is not regular
- Generalize the above ideas to "pumping lemma"
- Use "Pumping Lemma" to prove several other languages are not regular.

Proof. 6 Assume by contradiction there is a DFA M that recognized L. Let number of states in M $= p$.
Let $r_k \in Q$ be the state M reaches after reading 0^k.
Then for some $0 \leq i \leq j \leq q$ we have $r_i = r_j$.
M must accept $0^i 1^i \in L$.
$\Rightarrow$ M accepts $0^j 1^i \notin L$. □

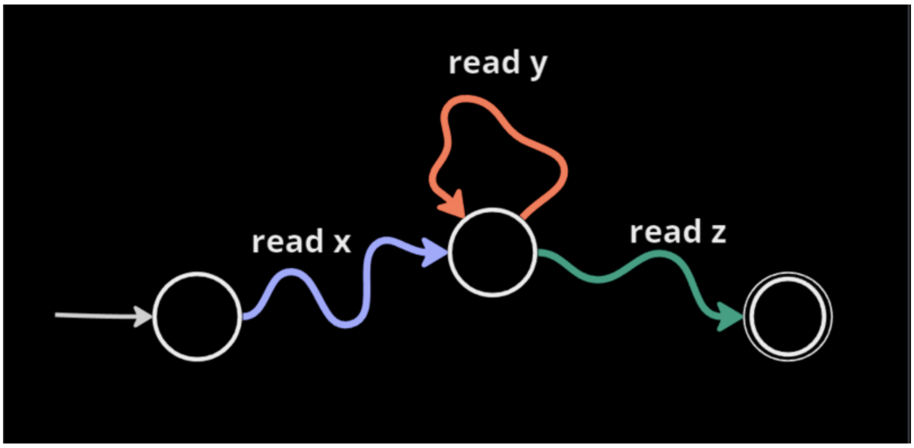
Lemma 7. *Pumping Lemma:*
If L is a regular language, then:
$\exists p \in \mathbb{Z}, p \geq 0$ *(pumping length)*
$\forall$*strings* $w \in L$ *of length* $|w| \geq P$
$\exists$ *strings* $x,y,z : w = xyz, |y| > 0, \ |xy| \leq p$
$\forall i \in \mathbb{N}, xy^i z \in L.$

To prove L is not regular, use contrapositive:

Definition 8. *Contrapositive:* $A \Rightarrow B = \neg B \Rightarrow \neg A$

Recall De Morgan's law:

Definition 9. $\neg \forall x \phi(x)$ *is the same as*



Recall that the pumping lemma requires you to give a strategy for winning the following game:

- The adversary chooses some integer p.
- You choose a string $w \in L$ such that $|w| \geq p$.
- The adversary chooses strings x, y, z such that $w = xyz$ and $|xy| \leq p, |y| \geq 1$.
- You choose an integer i and you win if $xy^i z$ is not in L.

Therefore, for each language you ONLY need to answer the following questions (be concise):

1. What's your strategy for choosing the string w in the above game?
2. What's your strategy for choosing the integer i in the above game?
3. How do you know that $xy^i z$ is not in L?

To show L is not regular, show:

1. $\forall p \in \mathbb{N}$
2. $\exists$ strings $w \in L$ of length $|w| \geq P$
3. $\forall$ strings $x, y, z \ : \ w = xyz, |y| > 0, |xy| \leq p$
4. $\exists i \in \mathbb{N}, xy^i z \notin L$

5 October 3

5.1 Turing Machines

"If you want to learn anything about automata you can just ask chatGPT"
A Turing machine is a General Model of Computation

- **Algorithms have been around since dawn of time.**
  - Long addition, multiplication, division.
  - Compass and straightedge constructions
  - Euclid's greatest common divisor algorithm
  - Quadratic formula: finding roots of polynomials
- Traditionally, algorithms were understood as a human construct. No precise mathematical definition.

Already saw a limited notion of algorithms (DFA). Using the pumping lemma, we proved that there are some problems that are not computable in this model.

5.1.1 David Hilbert's Descision Problem

In 1928, David Hilbert asked for an "algorithm" that takes as input a mathemattical statement and decides whether the statement is true or false.
During the years 1931-1936, a series of works showed there is no algorithm for the decision problem.
Each of these works included a different definition of a "general algorithm".

- Kurt Godel relied on recursive functions.
- Alonzo Church developed λ-calculus.
- Alan Turing developed the Turing Machine.

All of these definitions turn out to be equivalent.
Turing Machines are perhaps the most intuitive. They provided inspiration for a general computer, the Von Neumann Architecture

Turing Machines cont.

Our Plan

- Define Turing Machines (TM). See how they work.
- Convince ourselves that TMs are powerful enough to implement any "reasonable algorithm".

A TM is like a DFA with infinite memory tape. Information can be saved and accessed using the tape instead of the DFA's state space.

- Initially, tape contains the input, followed by "blanks". The tape head is at the left-most position.
- In each step, the machine can overwrite the symbol under the tape-head and move the tape left or right.
  - The tape head cannot move left of the start.
  - TMs can use additional symbols to write to tape.
- At any point in time, the machine can halt the computation and accept or reject. (If there is no decision edge at the state the head is on, also reject)
- This is implemented via states and transitions like a DFA

Definition 10. *A Turing Machine consists of a tuple:*
$M = (Q, \Sigma, \Gamma, \delta, q_{start}, q_{accept}, q_{reject})$ *Where*

- Q *is a finite set called the states.*
- Σ *is an input alphabet.*
- Γ *is the tape alphabet such that* $\Sigma \subseteq \Gamma$ *and* Γ *contains a special blank symbol '-' that is not in* Σ.
- $q_{start} \in Q$ *is the start state.*
- $q_{accept} \in Q$ *is the accept state,* q_{reject} *is the reject state.*
- $\delta : Q' \times \Gamma \to Q \times \Gamma \times \{L, R\}$ *is the transition function.*
  Where $Q' = Q \setminus \{q_{accept}, q_{reject}\}$

6 October 7

6.1 Building a Turing Machine

Definition 11. ***Configuration*** *encodes all information about a particular step in the computation of a turing machine.*
All information:

- *Current state*
- *Content on the tape*
- *Tape-head position*

Let $M = (Q, \Sigma, \Gamma, \delta, q_{start}, q_{accept}, q_{reject}$ *be a Turing Machine.*

- *A* ***configuration*** *of* M *is a tuple* $C = (u, q, v)$ *such that* $u, v \in \Gamma^*$ *and* $q \in Q$. *Can write* $C = uqv$ *without commas.*
- *A configuration* C ***yields*** C' *if* M *goes from* C *to* C' *in 1 step.*
  - $C = (u, q, bw)$ *yields* $C' = (ub', q', w)$ *if* $\delta(q, b) = (q', b', R)$
  - $C = (ua, q, bw)$ *yields* $C' = (u, q', ab'w)$ *if* $\delta(q, b) = (q', b', L))$
  - $C = (q, bw)$ *yields* $C' = (q', b'w)$ *if* $(q, b) = (q', b', L)$ *(don't fall off)*
- *A start configuration of* M *on input* W *is* $q_{start}W$
- *An accepting (resp. rejecting) configuration is one where the state is* q_{accept} *(resp.* q_{reject}*).*

M *accepts (resp. rejects)* w *if there is a sequence of configurations* $C_1, C_2, ..., C_n$ *such that:*

- C_1 *is the start configuration of* M *on input* w.
- C_i *yields* C_{i+1} *for* $i = 1, ..., n - 1$.
- C_n *is an accepting (resp. rejecting) configuration.*

This is a way to save your current state for later.
If $C = (ua, q, bv)$, $\delta(a, b) = (q', c, L)$, then $C' = (u, q', acv)$
If $\delta(a, b) = (q', C, R)$ then $C' = (uac, q', v)$

7 October 10

7.1 Language of a TM

- A TM M on input w can either accept, reject, or loop
- For a TM M, we define $L(M) = \{w|M \text{ accepts } w\}$
- If TM M and a language L satisfies "for any $x \in L$, M accepts x", we cannot say "M recognizes L". We must prove that $L(M) = \{w|M \text{ accepts } w\}$
- Do this using $\subseteq$ (M accepts any string in L) and $\supseteq$ (If a string is accepted by M, the string is in L).
- We say the M **decides** L if
  - M accepts $w \in L$ and M rejects $w \notin L$.
  - equivalently: M recognizes L and M always halts.
- A language L is **recognizable** (resp. **decidable**) if there is some TM that recognizes (resp. decides) L.
- The set of all languages decided by turing machines is a subset of all languages recognized by turing machines.

7.2 Specifiying a Turing Machine

Instead of drawing a state diagram, we give a "tape-head" level description that abstracts out the states/transitions via pseudocode.

- Imagine tape-head has small local memort which is "fixed" and cannott grow with input size (states of TM).
- Describe how the tape-head should **walk across the tape** and **what it should write**.

Example 2. $L = \{a^{2n}|n \geq 0\}$ *all powers of 2.*
Walk tape-head from left to right and cross out any other a.

- *If tape contained a single 0, accept.*
- *Else if number of 0s was odd, reject.*
- *Else return to the left-hand end of tape, repeat.*

Example 3. $L = \{w\#w : w \in (0,1)^*\}$

- *Check input is of form* $\{0,1\}^*\#\{0,1\}^*$ *and reject otherwise.*
-

7.3 Beyond Boolean Functions

We can also consider TM's that output more than just "accept/reject"
Idea: define the output of a TM as the contents of its tape when it enters a halt state.
A TM M computes a function $f : \Sigma^* \to \Sigma^*$ if on every input $w \in \Sigma^*$ the TM **halts** and its tape contains f(w).

Definition 12. *We say that f is **computable** if some TM M computes it.*

Example 4. $f : \{0, 1\}^* \to \{0, 1\}^*$*: f(binary rep of n) = binary rep of n + 1. f is the binary successor function.*
Show that this is computable.
Given any input string such as '01101101111':
Move to the end of the string and continue left until the beginning. Flip all 1's until the first 0, flip the first 0.
To prevent crashing off the beginning of the tape if all ones: convert first character to a # if it is a 1 and add a zero on the end if it is reached.

8 October 14 - Turing Machine Variants

Multi-Tape TM

A TM with one input tape and multiple work tapes. Transition function is defined by $\delta = Q' \times T \to Q \times T \times \{L, R\}$. $\delta(g, w) = (g', w, L/R)$.

- You can concatenate multiple tapes to one tape and separate their contents by #.
- Remember tape-head positions by storing an underlined version of tape symbols.
- Each step of multi-tape TM is simulated by scanning entire tape of single-tape TM.
- If you run out of space on the tapes, shift all elements to the right. (halting problem)

Tape-Head Level Description:
f(a, b) = a + b. Input (binary interpretations of two integers separated by a #)

- Reverse each input and copy each one to a different tape and clear main tape.
- Return all tape heads to the left. Store 1-bit carry as 0
- Add two bits under each tape head (using 0 if one head is empty) and carry bit:
  - Write result mod 2 to the main tape.
  - Move all tape heads 1 right.
  - Repeat until both heads are empty.
- Reverse main tape.

Random Access TM

Can read/write to arbitrary locations in memory without scanning a tape. Memory modeled as infinite array R.

- In addition to the standard tape that contains the input the TM has location and value tapes.
- There is a special write transition which sets R[location] = value using the content of the tapes.
- There is a read transition which sets the contents of the values tape to R[location]

Compliling to normal TM:
We use a multi-tape machine (which can be converted to a single-tape)

- Store contents of array R on a tape as tuples (location$_n$, value$_n$).
- To simulate a read, scan R until find a location that matches content of location tape. Write the value on the value tape. Put a blank if no such value is found.
- To simulate a write, scan R until you find location that matches content of location tape. Update value. If none found, append (location, value) to end of R.

Turing Completeness

Theorem 8. *Church-Turing Thesis: Any algorithm (in an informal sense) can be computed by a TM.*

Proof Outline:
Design a compiler that converts Java program into a TM.

- All programming languages are already compiled to ”assembly code” for modern CPUs

- Assembly code instructions can be implemented on a Random-Access TM.

9 October 21

For each object O, let $< O >$ be a string that encodes O.

- If i is an integer $< i >$ can be its representation
- If O is a string, then $< O >$ is just O itself.
- If G is a graph $< G >$ can be defined in many ways; such as vertex lists, etc
- If M is a TM, we can define $< M >$ by writing down the formal definition. $M = (Q, \Sigma, \Gamma, \delta, q_{start}, q_{accept}, q_{reject})$

Now we will consider more complex languages, i.e. primes, graphs, etc.

9.1 Universal Turing Machine

- There is a Turing Machine M_{UNIV} that can run any other TM.
- $M_{\text{UNIV}}(< M >, w)$. Takes as input a description of any TM M and any string W. Runs M on w.
  - If M accepts w, then $M_{\text{UNIV}}(< M >, w)$ accepts.
  - If M rejects w, then $M_{\text{UNIV}}(< M >, w)$ rejects.
  - If M loops on w, then $M_{\text{UNIV}}(< M >, w)$ also loops.
- TMs are algorithms and a universal TM is a general purpose computer.