# Theory of Computation Notes

William Traub

## Contents

**1 October 3** **3**
1.1 Turing Machines . . . 3
1.1.1 David Hilbert's Descision Problem . . . 3

**2 October 7** **5**
2.1 Building a Turing Machine . . . 5

**3 October 10** **6**
3.1 Language of a TM . . . 6
3.2 Specifiying a Turing Machine . . . 6
3.3 Beyond Boolean Functions . . . 7

**4 October 14 - Turing Machine Variants** **8**

**5 October 21** **10**
5.1 Universal Turing Machine . . . 10
5.2 Non-deterministic TM . . . 10

**6 October 28** **11**
6.1 Comparing Infinities . . . 11
6.2 Uncountablility . . . 11
6.3 Undecidability . . . 11
6.4 The TM Self-Acceptance Problem . . . 12
6.5 Undecidability as Diagonalization . . . 12
6.6 Reductions in Undecidability . . . 12
6.6.1 Halting Problem . . . 12

**7 November 7** **14**
7.1 Proof Systems . . . 14
7.2 Gödel's Sentence . . . 14

**8 Exam 2 Prep November 13** **15**

**9 November 18 - Efficiency (P and NP)** **16**
9.1 What is Efficiency? . . . 16
9.2 Asymptotic Notation . . . 16
9.3 Polynomial Functions . . . 16

**10 December 2** **18**
10.1 SUBSET-SUM . . . . . . . . . . 18
10.2 Optimization Problems . . . . . . . . . . 19

# 1 October 3

## 1.1 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.

### 1.1.1 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 $\lambda$-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 1.** *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.*
- *$\Sigma$ is an input alphabet.*
- *$\Gamma$ is the tape alphabet such that $\Sigma \subseteq \Gamma$ and $\Gamma$ contains a special blank symbol '-' that is not in $\Sigma$.*
- *$q_{start} \in Q$ 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 $Q' = Q \setminus \{q_{accept}, q_{reject}\}$*

# 2 October 7

## 2.1 Building a Turing Machine

**Definition 2.** ***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 $C' = (uac, q', v)$

# 3 October 10

## 3.1 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.

## 3.2 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 1.** $L = \{a^{2n}|n \geq 0\}$ *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 2.** $L = \{w\#w : w \in (0,1)^*\}$

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

## 3.3 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 3.** *We say that f is **computable** if some TM M computes it.*

**Example 3.** $f : \{0,1\}^* \to \{0,1\}^*$*: 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.*

# 4 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 <u>underlined</u> 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 1.** *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.

# 5 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.

## 5.1 Universal Turing Machine

- There is a Turing Machine $M_{\text{UNIV}}$ 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.

## 5.2 Non-deterministic TM

- Multiple possible actions the TM can take at any point in time.
- Transition function is $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$
- TM accepts if there ecists some way to run the computation that ends in an accept state.
- TM rejects if all computations reject.

Compiler: NDTM to TMs

- THink of computation of a non-deterministic TM as a tree of TM configutations.
- Using a deterministic TM, explore this tree using a breadth-first seach. If it finds an accepting configurations, accept. If all branches reject, reject.
- If the NDTM runs for N steps, what's the run-time of the above algorithm.

# 6 October 28

Goal: Show some languages are not decidable.

## 6.1 Comparing Infinities

There are many infinite sets:

- $\mathbb{N}$ - Natural numbers
- Even numbers
- $\mathbb{Q}$ - Rationals

Some infinite sets are bigger than others. We can show that one set A is larger than another B if a one-to-one map exists between B and A.
If there is a one-to-one function in both directions, B is the same size as A.
Natural numbers are the "smallest" infinite set, if A is infinite: $|\mathbb{N}| \leq |A|$. An infinite set A is countable if $|A| = |\mathbb{N}|$. We can also show $|A| \leq |\mathbb{N}|$ as there is no set $|S| < |\mathbb{N}|$.

## 6.2 Uncountablility

To show A is uncountable, it is enough to show some set $|B| \leq |A|$ where B is uncountable.
**Proof:** Because of the transitivity of the $\leq$ operator,

- The real numbers $\mathbb{R}$ are uncountable: $|\mathbb{S}| \leq |\mathbb{R}|$. The one-to-one function $f : \mathbb{S} \to \mathbb{R}$ defined by: $f(s) = .a_1 a_2 .a_3...$ (in decimal) where $s = a_1, a_2, a_3...$
- The set $\mathbb{P}$ PowerSet($\mathbb{N}$) is uncountable: $|\mathbb{S}| \leq |\mathbb{P}|$. One-to-one function $f : \mathbb{S} \to \mathbb{P}$ is defined by $f(s) = \{i : a_i = 1\}$ where $s = a_1, a_2, a_3...$

## 6.3 Undecidability

We know:

- The set $\mathbb{L}$ of all languages is uncountable.
- The set $\mathbb{M}$ of all TM's are countably infinite.

Because of the previous proof, we know that $|\mathbb{M}| \leq |\mathbb{L}|$
We will show $A_{TM} = \{\langle M, w\rangle$ : M is a TM that accepts w$\}$ is undecidable.
Given a description of a TM M and a string w:

1. Decide if M accepts w. (this is $A_{TM}$)
2. Decide if M halts on the input w.
3. Decide if M halts on empty input $\varepsilon$
4. Decide if $L(M) = \emptyset$

## 6.4 The TM Self-Acceptance Problem

Take the turing machine $SA_{TM} = \{\langle M \rangle : \text{M is a TM that accepts } \langle M \rangle\}$
The complement of this: $SU_{TM} = \{\langle M \rangle : \text{M is a TM that does not accept} \langle M \rangle\}$

1. A TM is "self-accepting" if it accepts the string $\langle M \rangle$ denoting its own description

2. To decide the language $SA_{TM}$, you need to design an algorithm that gets $\langle M \rangle$ and decides M is self-accepting.

3. $SA_{TM}$, $SU_{TM}$ are complements of each other (assume every string denotes some TM). One is decidable $\iff$ the other is.

**Theorem 2.** ***Claim:*** *$SU_{TM}$ is an undecidable language.*
***Proof:*** *By contradiction, Assume we have a decider D (a TM that always halts for $SU_{TM}$). D accepts $\langle M \rangle \iff$ M does not accept $\langle M \rangle$.*
*This can be rewritten as D accepts $\langle D \rangle \iff$ D does not accept $\langle D \rangle$, hence we arrive at a contradiction.*

## 6.5 Undecidability as Diagonalization

Because the set of all TMs are countable, we can create the matrix $[M_i \times \langle M_i \rangle]$ where the diagonals of the row show when the TM accepts (resp. rejects) itself.
We can use this to directly prove undecidability of self-acceptance.
Suppose by contradiction there exists a decider D for $SA_{TM} = \{\langle M \rangle | \text{ M accepts} \langle M \rangle\}$.
We can construct $M^*(\langle M \rangle)$: Outputs $\neg D(\langle M \rangle)$
Consider $M^*(\langle M^* \rangle)$ (the diagonal position in our matrix). This would result in $M^*(\langle M^* \rangle)$accepts $\iff$ $D(\langle M^* \rangle)$rejects $\iff \neg(M^*(\langle M^* \rangle)$accepts). This results in a contradiction and therefore a counterexample.

## 6.6 Reductions in Undecidability

The TM Acceptance problem: $A_{TM} = \{\langle M, w \rangle : \text{M is a TM that accepts w}\}$.
We previously showed that $SA_{TM} = \{\langle M \rangle | \text{ M accepts} \langle M \rangle\}$ is undecidable. If we had a decider $D_A$ for $A_{TM}$, we could construct a decider $D_S$ for $SA_{TM}$.
$D_S(\langle M \rangle)\{\text{Output } D_A(\langle M, M \rangle)\}$
Use reduction to solve problems. Reduce problem A to B - show how to solve A given a way to solve B.
If A and B are languages, than we reduce A to B by constructing decider $D_B$ as a subroutine.
By reducing A to B, we show:

- If B is decidable then A is decidable. (algorithms)
- If A is undecidable then B is undecidable. (this course)

### 6.6.1 Halting Problem

Consider the TM $H = \{\langle M, w \rangle : \text{M is a TM and M halts on w}\}$

**Theorem 3.** ***Claim:*** *H is undecidable.*
***Proof:*** *Reduce $A_{TM}$ to H.*
*Construct decider $D_{ATM}$ using decider $D_H$ as a subroutine.*
$D_{ATM}(\langle M, w\rangle)\{$
  *Run $D_H(\langle M, w\rangle)$ and if it rejects, output reject;*
  *Else run M(w) and output whatever it outputs;*
$\}$
*Because we know that $D_H$ is a decider $D_{ATM}$ always halts.*
$\forall\langle M, w\rangle, \langle M, w\rangle \in A_{TM} \iff D_{ATM}(\langle M, w\rangle) accepts \iff M\ accepts\ w \iff$

# 7 November 7

## 7.1 Proof Systems

By Russel's Paradox, sets "cannot" be non-wellfounded.

**Corollary 1.** *There is no "proof system" in Number Theory*

- *Soundness: No false statement $\phi$ has a valid proof $\pi$.*
- *Completeness: Every trye statement $\phi$ has a valid proof $\pi$.*
- *Decidability: Can decide if $\pi$ is a valid proof for $\phi$*

We generally give up soundness for completeness.

*Proof.* If there was a proof system for number theory, then we could decide number theory.

- Given $\phi$ try all valid strings $\pi$ one-by-one:
  - Check if $\pi$ is a valid proof of $\phi$. If so, accept.
  - Check if $\pi$ is a valid proof of $\neg\phi$. If so, reject.

$\square$

## 7.2 Gödel's Sentence

We want to construct a statement in a proof system that is true and unprovable.
Take proposition p(x) = "p(x) is unprovable."

1. Proving p(x) is true: $\neg p(x) \implies p(x)$ is provable $\implies p(x)$ by the property of soundedness in proof systems. This is clearly a contradiction.
2. If p(x) is true, it must also be unprovable.

Showing that a proposition can describe itself.
$\Phi^*(n)$ :

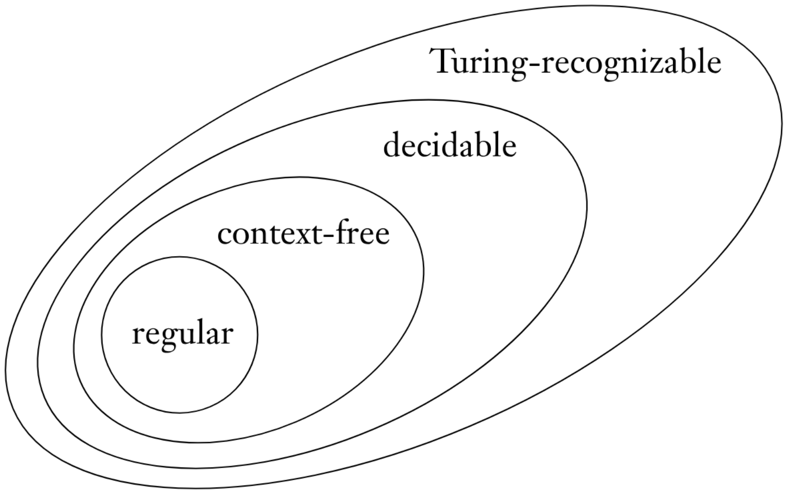
1. parse n as $\langle\phi\rangle$
2. True, if $\nexists$ proof for $\phi(n)$

$\phi^*(n) := \nexists$ proof for $\phi(n)$ where $\langle\phi\rangle = n$.
Consider $\phi^*(\langle\phi^*\rangle)$

1. Proving $\phi^*(\langle\phi^*\rangle)$ is true: Suppose not, $\implies \exists$ proof for $\phi^*(\langle\phi^*\rangle)$. By soundedness, we then know that $\phi^*(\langle\phi^*\rangle)$ is true.
2. Proving $\phi^*(\langle\phi^*\rangle)$ is unprovable: Suppose not, $\implies$ $\phi^*(\langle\phi^*\rangle)$ is true which results in a contradiction.

We can use diagonalization to build a matrix where each row represents the mathematical equation *phi* and each column represents a natural number n. Each cell represents if there exists a proof for $\phi(n)$.

# 8 Exam 2 Prep November 13




- The set $\mathbb{S}$ of all infinite binary sequences is uncountable. We can show a set is uncountable by proving it is larger than $\mathbb{S}$
- The TM Acceptance problem: $A_{TM} = \{\langle M, w\rangle$ : M is a TM that accepts w$\}$.
- $A_{TM}$ is recognizable but not decidable. $\overline{A_{TM}}$ is not recognizable.
- A language is decidable $\iff$ it is Turing-recognizable and co-Turing-recognizable.
- If A is reducible to B and B is decidable, A also is decidable. Equivalently, if A is undecidable and reducible to B, B is undecidable.
- By reducing A to B, we show if A is undecidable then B is undecidable.
- If $A \leq_m B$ and B is decidable, then A is decidable.
- If $A \leq_m B$ and A is decidable, then B is undecidable.
- The set of all Turing machines is countable, because every Turing machine has a finite, unique description.
- Any Turing-undecidable language is also non-regular, because its contrapositive is true: any regular language is decidable by Turing machines.
- Rice's theorem. Let P be any nontrivial property of the language of a Turing machine. Prove that the problem of determining whether a given Turing machine's language has property P is undecidable.

# 9 November 18 - Efficiency (P and NP)

Are there problems that can be computed, but cannot be computed efficiently? Proving they cannot be computed efficiently using reductions.

## 9.1 What is Efficiency?

If we could compute B efficiently then we could compute A efficiently.

**Definition 4.** *The* ***run-time*** *of a TM M on input w is the number of syeps the TM takes before it halts. This is a function*

$$f(n) = max\{\text{run-time of } M \text{ on } w : |w| = n\}$$

This is the worst-case runtime of M on input of length n.
Often we don't care abot the exact runtime, but instead the asymptotic runtime.

**Example 4.** *Primality Testing:*

```
// decide whether q is prime
boolean IsPrime(q) {
  for (i = 2; i < q; i++) {
    if (q % i == 0) reject and halt
  }
  accept and halt
}
```

*The runtime of IsPrime is exponential.*

## 9.2 Asymptotic Notation

- A function $g(n) = O(f(n))$ if there is some constant $c$ such that for all large enough $n : g(n) \leq cf(n)$. Really $O(f(n))$ is a set of functions, $g(n) \in O(f(n))$
- Example: $2n^2 + 5n + 7 = O(n^2)$ O only gives an upper bound so this is also $O(n^3)$

We define a class of languages:
$TIME(t(n)) = \{L : \exists$ TM M with runtime O(t(n)), M decides L $\}$
Some observations:

- $REGULAR \subseteq TIME(n)$
- $TIME(2^n) \subseteq DECIDABLE$

This classification is reliant on the model used (i.e. a Java program might have O(n) but a TM might be different). Though the difference aren't too big.

## 9.3 Polynomial Functions

- A **Polynomial function** poly(n) $= \cup_c O(n^c) = O(n) \cup O(n^2) \cup O(n^3) \dots$
- g(n) = poly(n) if and only if there exist constants $c, c'$ such that for all large enough $n : g(n) \leq cn^{c'}$

- Composition if f(n), g(n) = poly(n) then $f(g(n)), f(n)g(n), (f(n))^p = poly(n)$
- If the runtime of a TM M is some function t(n) = poly(n) then we say that M “runs in polynomial-time”.

The Class $P = \cup\ TIME(n^c)$ = the languages that can be decided in time poly(n).
We think of $P$ as the class of langyages that can be decided ”efficiently”.
The class $P$ is the same if we use any model.
**Extended Church-Turing thesis**: the class $P$ is the same in any ”realistic” model of computation. This is more controversial as the definition of ”realisitic” is disputed.
Problems in P

- Regular languages
- Arithmetic: addition, multiplication, division, exponentiation, etc.
- Everything in algorithms class.

# 10 December 2

## 10.1 SUBSET-SUM

**Theorem 4.** *$SUBSET\text{-}SUM \in P \implies 3SAT \in P$*

Proof outline:
We give TM $R$ that on input $\phi$:

- computes numbers $a_1, a_2, \ldots, a_n, t$ such that

$$\phi \in 3SAT \iff (a_1, a_2, \ldots, a_n, t) \in SUBSET\text{-}SUM$$

We can use binary encoding of boolean values.

**Example 5.**

$$\phi = (x \vee y \vee z) \wedge (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z)$$

*3 variables + 3 clauses ($v$ = 3, $c$ = 3) $\Rightarrow$ 6 digits for each number. For each clause in $C$, include two occurances of $a_c = 1$ in C's digit and 0 in others. Set $t$ = 1 in the first $v$ digits and 3 in the rest $k$ digits*

| | $x$ | $y$ | $z$ | 1 | 2 | 3 |
|---|---|---|---|---|---|---|
| $a_x^T =$ | 1 | 0 | 0 | 1 | 0 | 1 |
| $a_x^F =$ | 1 | 0 | 0 | 0 | 1 | 0 |
| $a_y^T =$ | 0 | 1 | 0 | 1 | 0 | 1 |
| $a_y^F =$ | 0 | 1 | 0 | 0 | 1 | 0 |
| $a_z^T =$ | 0 | 0 | 1 | 1 | 1 | 0 |
| $a_z^F =$ | 0 | 0 | 1 | 0 | 0 | 1 |
| $2 \cdot a_{c1} =$ | 0 | 0 | 0 | 1 | 0 | 0 |
| $2 \cdot a_{c2} =$ | 0 | 0 | 0 | 0 | 1 | 0 |
| $2 \cdot a_{c3} =$ | 0 | 0 | 0 | 0 | 0 | 1 |
| $t =$ | 1 | 1 | 1 | 3 | 3 | 3 |

*Proof.* ($\Rightarrow$) suppose $\phi$ has satisfying assingment, pick $a_x^T$ if x is true, else $a_x^F$. The sum of these numbers yield 1 in the first $v$ digits because $a_x^T, a_x^F$ have 1 in x's digit and 0 in others, and 1, 2, 3 in the last k digits. $\square$

## 10.2 Optimization Problems

**Example 6.** *The knapsack problem: REDUCE SUB-SUM to knapsack List = (value*$_1$*, weight*$_1$*), ... ,(value*$_n$*, weight*$_n$*)*

**Theorem 5.** *Cook-Levin Theorem:* $3SAT \in P \implies P = NP$.

**Lemma 6.** *Every function* $f : \{0,1\}^m \to \{0,1\}$ *can be represented by a CNF formula* $\phi$ *of size* $\leq m \cdot 2^m$.
***Proof:*** *For each value that makes f evaluate to 0, add a clause to ensure variables can't take that value.*

**Example 7.** $f(x_1 = 1, x_2 = 0, x_3 = 1, \ldots, x_m = 0) = 0$
*Add the clause* $(\neg x_1 \vee x_2 \vee \neg x_3 \vee \cdots \vee x_m)$