CSI 5140

Computational Complexity

Iluju Kiringa

kiringa@eecs.uottawa.ca

Outline of the Course

- Models of computation : Turing machine, RAM ...
- Computability : Undecidability, Reducibility, examples
- Complexity :
  - Time Complexity :
    * Measuring complexity
    * Complexity classes P and NP
    * NP - Completeness : The Cook theorem, polynomial time reducibility
    * NP-complete problems
  - Space complexity :
    * Savitch 's theorem
    * Complexity classes PSPACE , L , and NL
    * PSPACE - Completeness
  - Hierarchy Theorems and classes of higher complexity
  - Logical characterizations of complexity classes
- Miscellaneous advanced topics

Basic Definitions

Consider these problems:

problem 1 : Is a given number n prime?

problem 2 : What are the prime factors of n?

problem 3 : Do given numbers n and m have a common prime factor?

Observations:

- Problem 1 was solved by Eratosthenes' algorithm:
  Consider integers $0 \dots n$;
  Then cross out all even numbers, all multiples of 3, 5, 7, ..., up to $\sqrt{n}$
  If n wasn't crossed out, then n is prime
- Problem 3 was solved by Euclid's algorithm (GCD):
- Problem 2 has no efficient algorithm yet
- Eratosthenes' algorithm is inefficient; Agrawal-Kayal-Saxena's alg. is efficient
- Euclid's alg. is efficient

Question: How to classify problems?

Basic Definitions (and Notations)

- Σ denotes a finite alphabet of symbols;
  Σ^* denotes the set of all finite strings over elements of Σ;
  $L \subseteq \Sigma^*$ denotes a <u>language</u> over Σ;
  Given $x \in \Sigma^*$, $|x|$ denotes the length of x.

- Now, we need to formalize the notions of "problem", "algorithm" and so on ...

- A <u>problem</u> is a general question (usually parameterized) to be answered:
  - description of parameters
  - statement of the properties of the answer (= solution)

- An <u>instance</u> of a problem = problem + values for all parameters

Basic Definitions

- Sample problem: TRAVELING SALESMAN

  parameters: $C = \{c_1, \dots, c_m\}$ set of cities

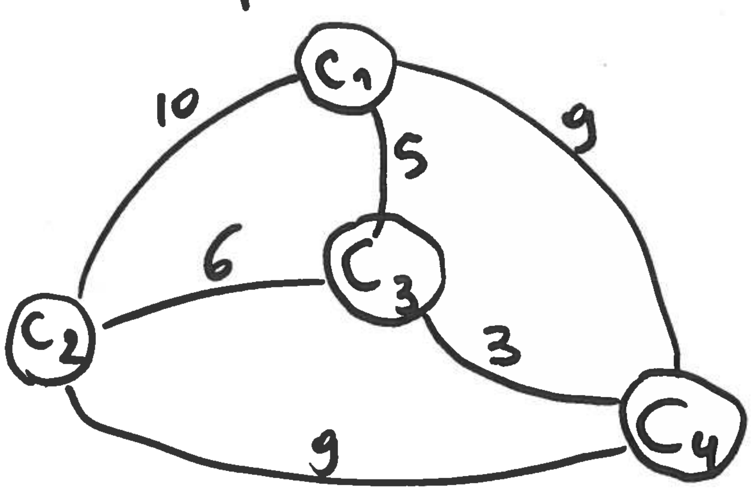
  for each $c_i, c_j \in C$, give $d(c_i, c_j)$

  solution: ordering $\langle c_{\pi(1)}, \dots, c_{\pi(m)} \rangle$ of cities s.t.

  $$\sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(m)}, c_{\pi(1)})$$

  is minimal.

- Sample instance of TRAVELING SALESMAN



$C = \{c_1, c_2, c_3, c_4\}$

$d(c_1, c_2) = 10$, $d(c_2, c_3) = 6$,

etc...

A solution for this instance:

$\langle c_1, c_2, c_4, c_3 \rangle$

corresponding tour has lenght 27

Basic Definitions

- An algorithm is a step-by-step procedure for solving problems;
  Concretely, it is a program to run on M.
- Instances of the problem are provided as input to the algorithm to solve that problem.
- Instances can be described by <u>encoding</u> them; i.e., each problem is associated with a fixed <u>encoding scheme</u> that maps instances to strings describing them.
- Given instance I of problem Π, the <u>input length</u> of I is the number of symbols in the encoding of I.
- Example:
  Alphabet of TRAVELING SALESMAN:
  {c, [,], /, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
  Sample instance:
  c[1] c[2] c[3] c[4] //10/5/9//6/9//3

Basic Definitions

- The <u>time complexity function</u> for an algorithm A gives, for each possible input length, the largest amount of time needed by A to solve an instance of that length.
- Much of computability theory and complexity theory is designed for <u>decision problems</u>
- Intuitively, a decision problem is one whose solution is either "yes" or "no".
- Formally, a decision problem consists of
  - a set D_Π of instances s.t. $D_\Pi \subseteq \Sigma^*$ for some alphabet Σ
  - a subset $Y_\Pi \subseteq D_\Pi$ of yes-instances
- Standard format used very often has 2 parts
  - generic instance.
  - yes-no question

Basic Definitions

- Alternatively, a <u>decision problem</u> is a function

$$f : \Sigma^* \longrightarrow \{yes, no\}$$

from strings to Boolean values.

- A <u>language</u> associated with a given decision problem f is the set

$$\{x \in \Sigma^* \mid f(x) = yes\}.$$

This is the subset $Y_\Pi \subseteq D_\Pi$ seen earlier.
Example: SAT = set of satisfiable Boolean formulas.

- A <u>function problem</u> is a function

$$f : \Sigma^* \longrightarrow \Sigma^*$$

from strings to strings.

Basic Definitions

- Example of a decision problem

  TRAVELING SALESMAN

  INSTANCE : set $C = \{c_1, \ldots, c_m\}$ of cities; distance $d(c_i, c_j) \in Z^+$ for each pair (c_i, c_j); bound $B \in Z^+$

  QUESTION : Is there a tour of all cities in C with total length $\leq B$, i.e., an ordering $\langle c_{\pi(1)}, \ldots, c_{\pi(m)} \rangle$ of C such that

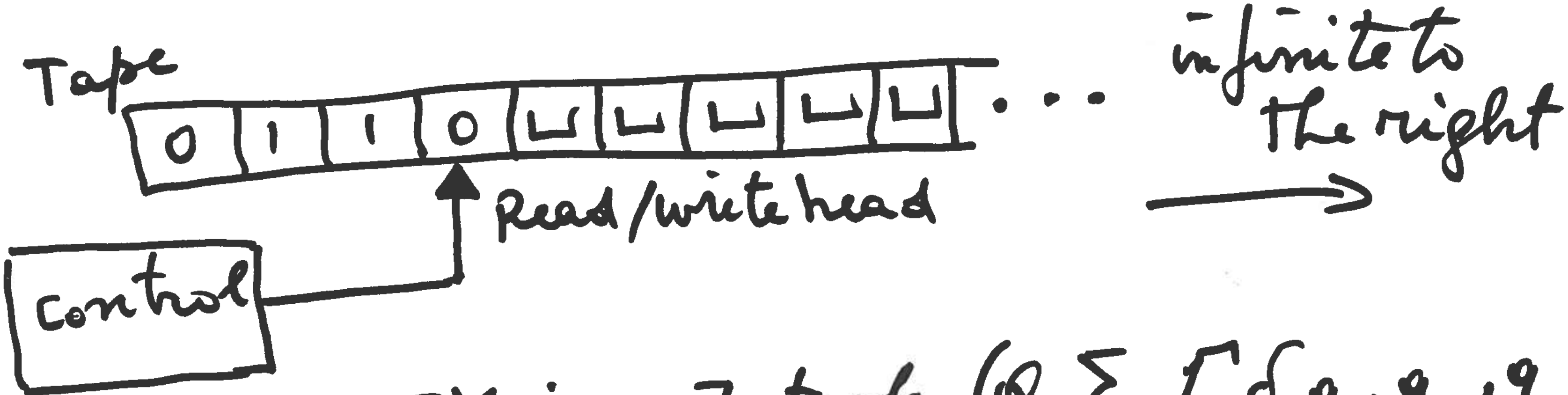
$$\sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(m)}, c_{\pi(1)}) \leq B \ ?$$

- Example of a function problem: See p. 5.

Turing Machines

- Powerful computation model proposed by A. Turing in 1936; accurate model of a general purpose computer.
- Informally, a TM consists of
  - an infinite tape ("unlimited memory")
  - a tape head (finite control) that can read and write symbols and move around the tape
- The tape is divided up into squares, each of which holds a symbol from a <u>finite</u> tape alphabet; the read/write head scans squares on the tape, and depending on the state of the control, it can
  - print a symbol on the square
  - move left or right by one square
  - assume a new state

Turing Machines



- Formally: A TM is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ where Q, Σ, Γ are all finite sets and
  - Q is the set of states.
  - Σ is the input alphabet (without blank symbol $\sqcup$)
  - Γ is the tape alphabet, with $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
  - $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is the transition function
  - $q_0 \in Q$ is the start state
  - $q_a \in Q$ is the accept state
  - $q_r \in Q$ is the reject state with $q_a \neq q_r$

Turing Machines

- The critical part in a TM is the function δ. Intuitively $\delta(q, s) = (q', s', h)$ means that, if q is the current state of the TM and the r/w head scans symbol s, then q' is the new state, the r/w head prints s'; and h is either L or R; L means "move to the left by one square" and similarly for R.
- Intuitively, a TM works as follows: initially, it receives an input string $x \in \Sigma^*$ on the leftmost n squares ($n = |x|$), with the head pointing to the leftmost symbol of x. The control is initially in q_0. Then, the TM moves according to δ. If the TM halts, it does so either in state q_a or in state q_r.

Turing Machines

Definition 1 (The language accepted by a TM) A TM M accepts a string $x \in \Sigma^*$ if M with input x eventually halts in state q_a; $\mathcal{L}(M) = \{x \in \Sigma^* \mid M \text{ accepts } x\}$ is the language accepted by M (or recognized by M).

Definition 2 A language is Turing-recognizable if there is a TM that recognizes it (These languages have been called "recursively enumerable" in classic books).

Definition 3 (Decidable language) TMs that halt on all inputs are called deciders; they are said to decide a language. A language is Turing-decidable (or recursive) if there is a TM that decides it.

Turing Machines

- Example

  $PARITY = \{x \mid x \in \{0,1\}^* \text{ and } x \text{ has even number of occurrences of } 1\}$

  A TM M to recognize PARITY works as follow

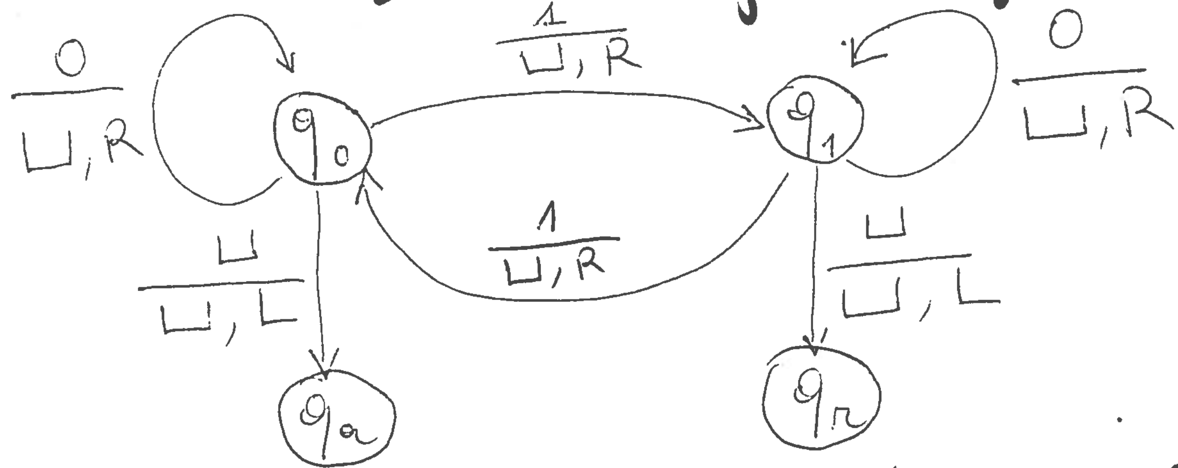
  - Scan x once from left to right.
  - At each point remember the number of 1s seen so far (whether it is even or odd).
  - If M reaches a blank while the count is even then accept, else reject.

State transition function of M:

q	s	$\delta(q,s)$
q_0	0	$(q_0, \sqcup, R)$
q_0	1	$(q_1, \sqcup, R)$
q_0	$\sqcup$	$(q_a, \sqcup, L)$
q_1	0	$(q_1, \sqcup, R)$
q_1	1	$(q_0, \sqcup, R)$
q_1	$\sqcup$	$(q_r, \sqcup, L)$

Turing Machines

- Example (cont'd):
  state diagram for M_1 of PARITY



Tracing input $x = 0101$

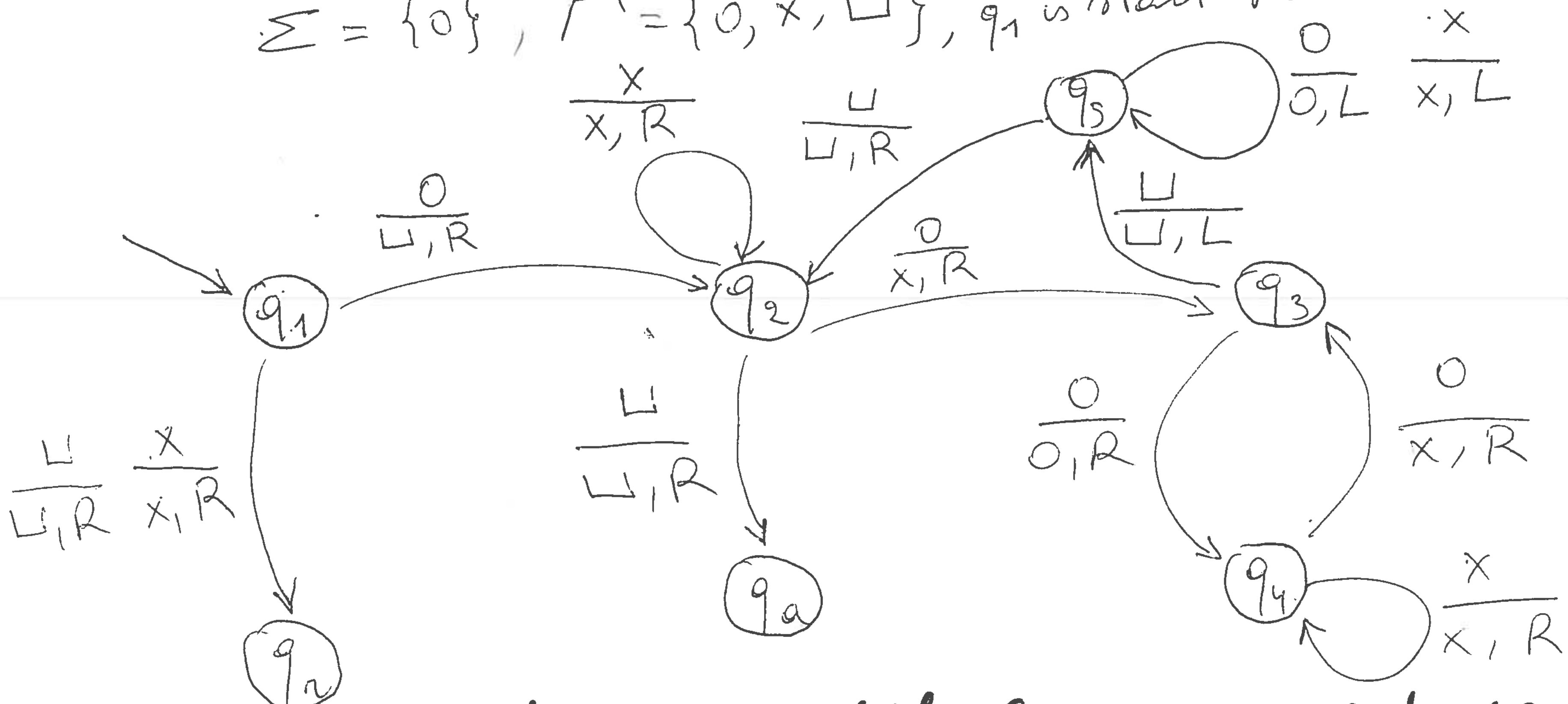
head

0	1	0	1	$\sqcup$		…	q_0
$\sqcup$	1	0	1	$\sqcup$		…	q_0
$\sqcup$	$\sqcup$	0	1	$\sqcup$		…	q_1
$\sqcup$	$\sqcup$	$\sqcup$	1	$\sqcup$		…	q_1
$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$		…	q_0
$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	…	q_a
$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$				

Turing Machines

- The following TM M_2 decides
  $L = \{0^{2^n} \mid n \geq 0\}$ (i.e., all strings of 0s whose length is a power of 2)

$Q = \{q_1, q_2, q_3, q_4, q_5, q_a, q_r\}$, δ is shown below,
$\Sigma = \{0\}$, $\Gamma = \{0, x, \sqcup\}$, q_1 is start state



Note: x is used to cross off the 0s;
$\sqcup$ is used to find the leftmost end of the tape.

Turing Machines

Informally, M_2 works as follows:

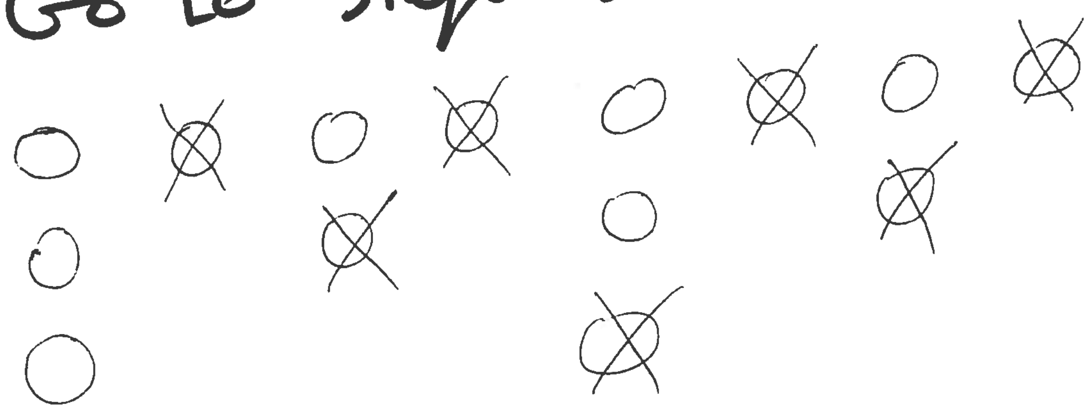
(1) Scan tape from left to right by crossing off every other 0.

(2) If in Step (1) the tape has a single 0, accept.

(3) If in Step (1) the tape has more than a single 0 and that number is odd, reject

(4) Return the head to the left-hand end of the tape.

(5) Go to Step (1).

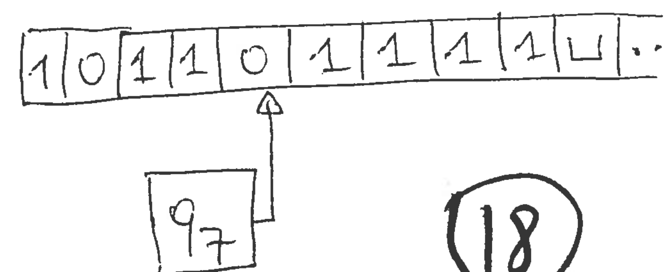


Turing Machines

- As a TM runs, changes happen in the current state, the current tape content, and the current head location. A particular setting of these three items is called a <u>configuration</u> of the TM.

- Representing configurations:
  for a state q and strings u and v over Γ, we write
  $$u\,q\,v$$
  to denote the configuration where
  - current state is q
  - current tape content is uv
  - current head location is the first symbol of v

- Example:



$$1011\,q_7\,01111$$

Turing Machines

- We use the concept of configuration to describe the moves of a TM M.

- We say that configuration C_1 yields C_2 (i.e., $C_1 \vdash_M C_2$) if M can legally go from C_1 to C_2 in a single step.

- Formally we define moves as follows:
  Suppose that $u\,a\,q_i\,b\,v$ and $u\,q_j\,a\,c\,v$ are configurations, with $a, b, c \in \Gamma$ and $u, v \in \Gamma^*$.
  Then
  $$u\,a\,q_i\,b\,v \vdash_M u\,q_j\,a\,c\,v$$
  if $\delta(q_i, b) = (q_j, c, L)$;
  $$u\,a\,q_i\,b\,v \vdash_M u\,a\,c\,q_j\,v$$
  if $\delta(q_i, b) = (q_j, c, R)$.

Turing Machines

- Special cases when the head is at the end of the configuration:

  1) Left-hand end:

  $q_i b v \vdash_M q_j c v$ if $\delta(q_i, b) = (q_j, c, L)$

  (this prevents M from going off the left end of tape)

  $q_i b v \vdash_M c q_j v$ if $\delta(q_i, b) = (q_j, c, R)$

  2) Right-hand end:

  $u a q_i$ is equivalent to $u a q_i \sqcup$

- $q_0 w$ is the <u>start configuration</u>.

  An <u>accepting</u> configuration is one whose state is q_a.

  A <u>rejecting</u> configuration is one whose state is q_r.

  Accepting and rejecting configurations are <u>halting configurations</u>.

Turing Machines

Definition 4 (Acceptance)

A TM M accepts input w if there is a sequence of configurations $C_1, C_2, \ldots, C_k$ such that

1. C_1 is the start configuration
2. for each C_i, C_{i+1}: $C_i \vdash_M C_{i+1}$
3. C_k is an accepting configuration

Definition 5 (Computation)

The sequence $C_1 \vdash C_2 \vdash \cdots C_{k-1} \vdash C_k$ is called a computation of M, with $C_1 = q_0 w$ and C_k a halting configuration.

Example: The following is a computation for M_2 on input 0000:

$q_1 0000 \vdash \sqcup q_2 000 \vdash \sqcup x q_3 00 \vdash \sqcup x 0 q_4 0 \vdash \sqcup x 0 x q_3 \sqcup$

$\vdash \cdots \quad \sqcup x x x q_2 x \sqcup \vdash \sqcup x x x \sqcup q_a$

Turing Machines : Variants

Definition 6 (Multi-tape TMs)

A k-tape TM has k tapes and k heads (though one single control unit). The transition function δ is now:

$$\delta : Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{L, R, S\}^k.$$

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, D_1, \ldots, D_k)$$

means: If M is in state q_i and heads read symbols $a_1, \ldots, a_k$, then M goes to state q_j, writes symbols $b_1, \ldots, b_k$ and moves the heads $1 \ldots k$ as prescribed by the directions $D_1, \ldots, D_k$.

Theorem 1

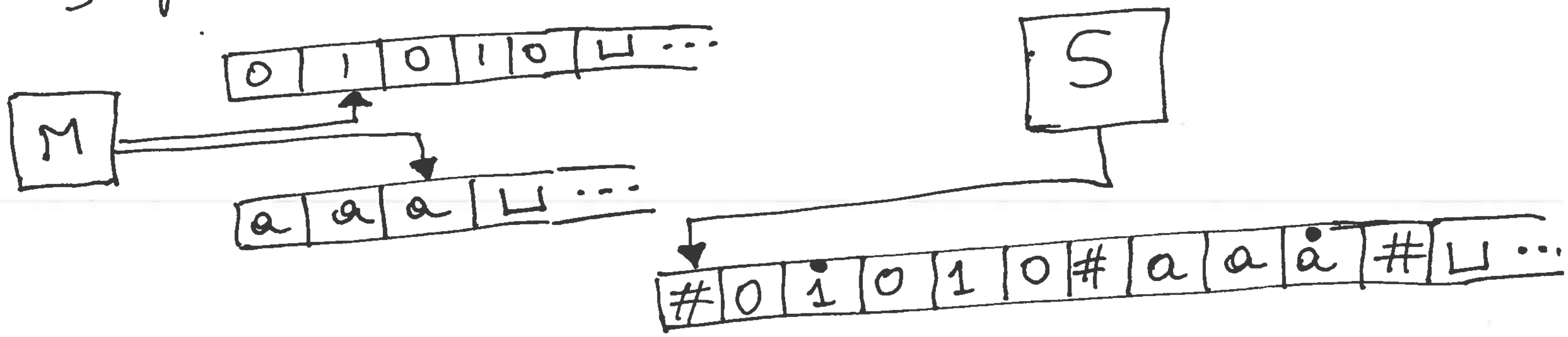
Every multi-tape TM M has an equivalent single-tape TM S.

Proof Idea Simulate M with S.

Turing Machines : Variants

Proof of Theorem

- Suppose M has k tapes.
- S simulates effect of the k tapes of M by storing their info on its single tape, using $\#$ as a delimiter.
- **S keeps** tracks of locations of the heads by writing a dot on top of the current tape symbol; i.e. $\dot{s} \in \Gamma_S$ for each $s \in \Gamma$:



- Suppose input is $w = w_1 \cdots w_n$:
  - Tape of S contains: $\#\dot{w}_1 \cdots w_n \# \dot{\sqcup} \# \cdots \# \dot{\sqcup} \# \sqcup\sqcup \cdots$
  - Make 2 scans from left to right to
    - * determine the dotted symbols
    - * update the S-tape according to M's δ.
  - If S encounters a $\#$, S writes a $\sqcup$ instead of $\#$ and shifts the tape content from the current cell to the rightmost $\#$ by one unit.

Turing Machines: Variants

Definition 7 (Non-Deterministic TMs)

A NDTM proceeds according to several possibilities at any point in the computation. The definition differs from the one for DTMs in the function δ which now is

$$\delta : Q \times \Gamma \longrightarrow 2^{(Q \times \Gamma \times \{L, R\})}$$

$$\delta(q, s) = \{(q_1, s_1, D_1), (q_2, s_2, D_2), \ldots, (q_k, s_k, D_k)\}$$

where $k \in \mathbb{N}$.

The computation of an NDTM is a tree. If some branch of the tree leads to the accept state, then the NDTM accepts the input.

Theorem 2 (Equivalence)

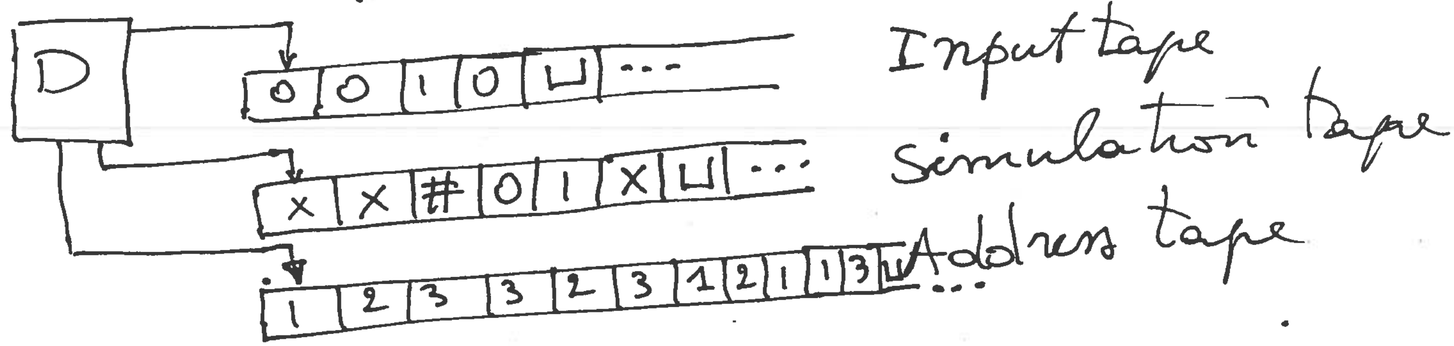
Every NDTM N has an equivalent DTM D. I.e., if N is an NDTM, then there's a DTM D such that $\mathcal{L}(N) = \mathcal{L}(D)$.

Turing Machines

Proof of Theorem 2

Idea: View N's computation as a tree and do a breadth-first search on that tree until we find an accepting state.
This is done by simulating the NDTM N with a DTM D by trying all the possible branches of N's computation.

Proof: Build D as follows:

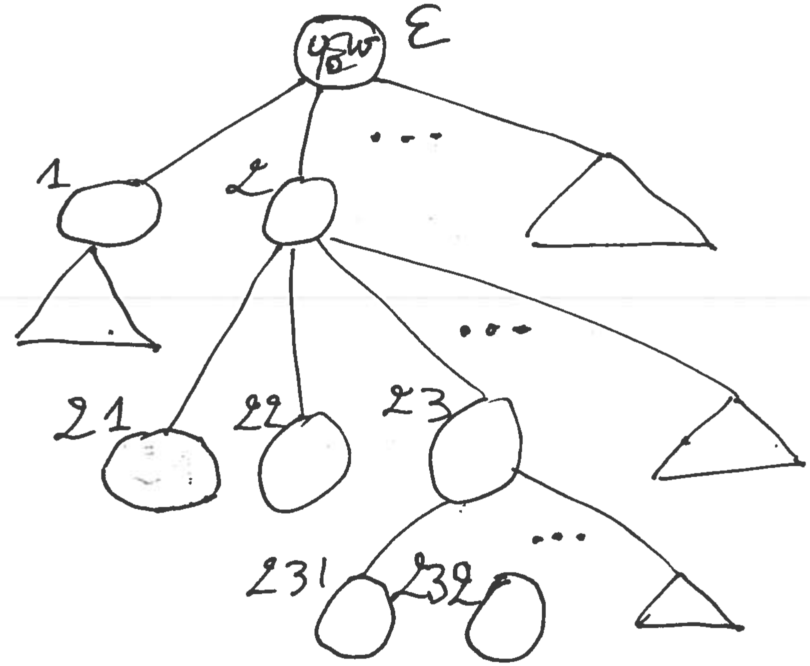


- Input tape: never altered
- Simulation tape: holds copy of N's tape
- Address tape: tracks D's location in N's computation tree

Turing Machines

Proof of Theorem 2 (Cont'd)

- Data representation on Address tape:
  - Every node in computation tree is assigned a string over $\Sigma_b = \{1, \dots, b\}$, with b = largest set of possible transition choices:



  - Address tape contains a string over Σ_b

Turing Machines

Execution of D:

(1) Initially: IT contains input w, ST and AT are empty.

(2) Copy content of IT to ST.

(3) Use ST to simulate N as follows:
   - Before each step of N, check AT to determine N's next choice as allowed by N's δ function.
   - If AT has no more content, go to Step (4).
   - If ST encounters a rejecting configuration, goto (4).
   - If ST encounters an accepting configuration, accept.

(4) Overwrite content of AT with lexicographically next word of Σ_b. Go to Step (2). $\blacksquare$

Note that in Step (3), we might encounter invalid nondeterministic choices. If so, we just go to Step (4).

Turing Machines

- Let M be a TM over alphabet Σ.
For each $x \in \Sigma^*$, $t_M(x)$ is the number of steps required by M to halt (i.e., terminate in either q_a or q_r) on input x.
We set $t_M(x) = \infty$, if M never halts on input x.

<u>Definition 8</u> (Worst case time complexity of a TM)
The function $T_M : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ defined by

$$T_M(n) = \max\{t_M(x) \mid x \in \Sigma^* \text{ and } |x| = n\}$$

is called the worst case time complexity of M.

<u>Church-Turing thesis</u>: A TM captures exactly the intuitive notion of an algorithm. That is, anything that can be algorithmically solved (in the intuitive sense of "algorithmically") can also be solved by a TM.

Turing Machines

Recall that, per Theorem 1, a k-tape TM can be stimulated by a 1-tape TM. Now, we compare their running times:

<u>Theorem 3</u> The time taken by the 1-tape TM of Theorem 1 to simulate n moves of the k-tape TM is $O(n^2)$; alternatively, if M decides a language L in time T_M, then S decides L in $O(T_M^2)$.

<u>Proof outline</u>:

To simulate one of the n moves of M by N, we need 2 scans of N's tape. Thus the maximum time N spends on this scans is $2T_M$. With n moves, we therefore have $O(T_M^2)$. $\blacksquare$

Turing Machines

Theorem 4 (Linear Speedup)
Suppose L is a language decided by a TM M in time $T_M(n)$. Then, for any $\varepsilon > 0$, there is a TM M' that decides L in time $T_{M'} = \varepsilon T_M(n) + n + 2$.

Proof outline:
First encode k-tuples of symbols of M as single symbols of M', thus increasing the alphabet. Use the new machine (with increased alphabet), say M', to simulate the old one. (See details in Papadimitriou p. 32–33) ∎

READING:
Sipser ch. 3; Papadimitriou Ch. 2;
HMU Ch. 8.

Undecidability

- Several problems (including many that occur in practical setting) turn out to be computationally unsolvable!
- What kind of problems are unsolvable and which proof techniques can be used to show unsolvability?
- We will first establish the technical notion of unsolvability, namely the undecidability. Then we will establish the undecidability of two problems:
  - Halting problem
  - Post correspondence problem
- We will give a direct proof for the halting problem. We will use the reducibility proof technique to show the undecidability of the second problem.

Undecidability: The Halting problem

- We define HALTING as the following problem:

  $$\boxed{\begin{array}{l}\text{Given a TM } M \text{ and its input } w.\\ \text{Question: Will } M \text{ halt on } w\text{?}\end{array}}$$

HALTING can be formulated as language problem:

$$\begin{aligned} \text{HALTING} &= \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on } w\} \\ &= \{\langle M, w\rangle \mid M(w) \neq \nearrow\} \end{aligned}$$

Note that the machine used to recognize HALTING and similar languages is called a <u>Universal TM</u>

- A universal TM U is a TM which, when presented with input w, interprets w as a pair $\langle M, x\rangle$ representing a TM M with an input x to M.
- The TM U simulates the behavior of M on input x i.e., $U(M, x) = M(x)$.

Undecidability: The Halting problem

Theorem 5

HALTING is Turing-recognizable; i.e., it is recursively enumerable.

Proof:

We need a TM which accepts HALTING; i.e. that TM halts in q_a if its input is in HALTING, it halts in q_r if the input is not in HALTING, and never halts otherwise. The universal TM U does just that. Suppose U has $\langle M, w\rangle$ as input, where M is a TM and $w \in \Sigma^*$.

Then

1) Simulate M on input w
2) If M accepts w, then U accepts $\langle M, w\rangle$; else if M rejects w, then U rejects $\langle M, w\rangle$. Otherwise U loops for ever

An alternative notation for HALTING is A_{TM}, which intuitively means the set of TM that recognize (i.e., accept) their inputs.

Undecidability

- Recall Definition 2 for recursively enumerable languages. These languages are also called "semi-decidable". Theorem 5 tells us that HALTING is semi-decidable. Unfortunately, it turns out that HALTING is undecidable.
- A. Turing was the first to exhibit this problem as a paramount example that there are undecidable languages.

- Notice that there are many versions of HALTING, such as the following

$$\{M \mid M \text{ is a TM and } M \text{ halts on a blank tape}\}$$

- Notice that Theorem 5 and the fact that HALTING is undecidable (as we shall prove) means that deciders restrict the kind of languages that can be recognized. Thus recognizers are more powerful than deciders.

Undecidability

Theorem 6 HALTING is undecidable.

- The proof of this theorem goes as follows:
  - Use Cantor's diagonalization by defining
    $$\text{DIAG} = \{M \mid M \text{ does not accept input } M\}$$
  - Show that DIAG is undecidable
  - Show that HALTING is just a "version" of DIAG

- Cantor's diagonalization was an answer to the problem of telling whether one of two infinite sets is "larger" than the other without resorting to counting. The question is: How can we compare sizes of infinite sets? Cantor's answer was: two infinite sets have the same size if their respective elements can be "paired". This pairing is formally called "correspondence"

Definition 9 (Correspondence)
A function $f: A \to B$ is a correspondence iff
$\forall a, b \in A: a \neq b \implies f(a) \neq f(b)$, and
$\forall b \in B \; \exists a \in A: f(a) = b$.

Undecidability

Lemma 7 (Turing) DIAG is undecidable.

Proof (By contradiction): Suppose there is a TM D to decide DIAG. Then one of the following two cases holds:

(1) D accepts D, or

(2) D does not accept D.

Case (1): We get $D \notin$ DIAG by definition. Since D decides DIAG by assumption, D does not accept D, which is contradiction.

Case (2): We get $D \in$ DIAG by definition of DIAG. Since D decides DIAG by assumption, D accepts D, which is a contradiction.

The assumption that DIAG is decidable must be wrong. ∎

Undecidability

Proof of Theorem 6:

If HALTING were decidable by some TM H, then we could decide DIAG as follows:

$\boxed{\text{On input } \langle M \rangle \text{, run the TM } H \text{ on } (M, M) \text{, and negate the output, i.e., accept if } H \text{ rejects, and reject if } H \text{ accepts.}}$

Lemma 2, however, tells us that DIAG is undecidable.
Hence HALTING is undecidable. ∎

Where is the diagonalization in the proof of Theorem 6?

We get back to this question a bit later on, after looking into some details about the diagonalization method and some of its applications.

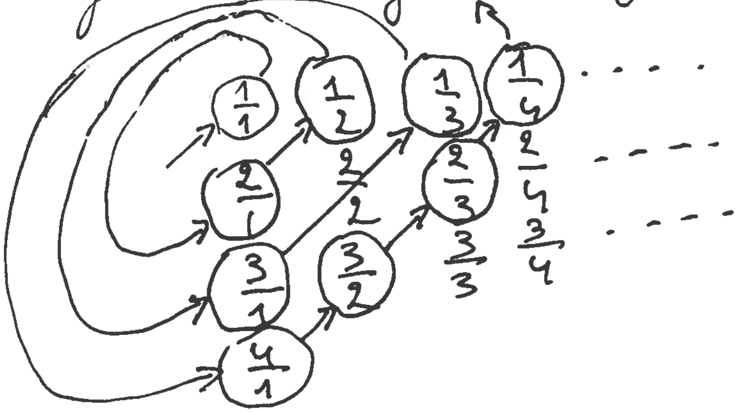
Undecidability

- Example of correspondence:
  Let $\mathbb{N}$ be the set of natural numbers, and $\mathbb{E}$ the set of even numbers. Then $f: \mathbb{N} \to \mathbb{E}$ such that $f(n) = 2n$ is a correspondence:

n	$f(n)$
1	2
2	4
3	6
$\vdots$	$\vdots$

- <u>Definition 10</u> (Countable set)
  A set S is <u>countable</u> if there is a function $f: \mathbb{N} \to S$ such that f is a correspondence.
- Sample countable set is $\mathbb{Q} = \{m/n \mid m, n \in \mathbb{N}\}$ (rational numbers)
  <u>Proof</u>: list all elements of $\mathbb{Q}$ using an infinite matrix of rational numbers:

  Establish a correspondence with $\mathbb{N}$ by avoiding repetitions



Undecidability

Theorem 7 $\mathbb{R}$ is uncountable

Proof (By contradiction):
Suppose $\mathbb{R}$ is countable. Then there is a correspondence $f: \mathbb{N} \to \mathbb{R}$. With such f, we still can construct a number $x \in \mathbb{R}$ with no $n \in \mathbb{N}$ such that $f(n) = x$ as follows: Choose each digit of x to make x different from all real numbers that have correspondents in $\mathbb{N}$. This choice is made using the diagonalization:

Objective: ensure that $\forall n \in \mathbb{N}: x \neq f(n)$.

n	$f(n)$	ensuring	setting
1	3.14159	$x \neq f(1)$	1st decimal becomes 4
2	5.55555	$x \neq f(2)$	2nd decimal becomes 6
3	0.12345 ...	$x \neq f(3)$	3rd decimal becomes 4
4	0.50000 ...		
⋮	⋮		

x differs from $f(n)$ in the n^{th} decimal digit

$x = 0.464 \ldots \neq f(n) \quad \forall n \in \mathbb{N}$. etc.

Undecidability

Corollary 8 There are non Turing-recognizable languages

proof: Idea: Show that there are uncountably many languages, but yet countably many TMs.

(1) There are countably many TMs since each TM has an encoding $\langle M \rangle$.

(2) There are uncountably many languages.

Claim: The set $\mathcal{B}$ of all infinite binary sequences (i.e., unending sequences of 0s and 1s) is uncountable.

proof of claim: done by diagonalization.

Now suppose $\mathcal{L}$ is the set of all languages over Σ.

Construct a correspondence with $\mathcal{B}$, i.e., $f: \mathcal{L} \to \mathcal{B}$.

Let $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$. We build the characteristic sequence of each $L \in \mathcal{L}$ as follows:

i-th bit of $CS(L) = 1$ if $s_i \in L$,

i-th bit of $CS(L) = 0$ if $s_i \notin L$.

One can show that $f: \mathcal{L} \to \mathcal{B}$ such that $f(L)$ is $CS(L)$ is a correspondence function. Since $\mathcal{B}$ is uncountable, $\mathcal{L}$ is also uncountable.

The Corollary now follows easily $\blacksquare$

Undecidability

Back to Theorem 6: Where is the diagonalization used?

Recall the TM D for trying to decide DIAG:

$$D(\langle M\rangle) = \begin{cases} \underline{\text{accept}} & \text{if } M \text{ does not accept } \langle M\rangle \\ \underline{\text{reject}} & \text{if } M \text{ accepts } \langle M\rangle. \end{cases}$$

Let us now try to run D on its own description $\langle D\rangle$:

$$D(\langle D\rangle) = \begin{cases} \underline{\text{accept}} & \text{if } D \text{ does not accept } \langle D\rangle \\ \underline{\text{reject}} & \text{if } D \text{ accepts } \langle D\rangle. \end{cases}$$

Now, examine behaviors of TMs H and D:

H:

	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\cdots$
M_1	A	R	A	R	
M_2	A	A	A	A	
M_3	R	R	R	R	
M_4	A	A	R	R	
$\vdots$					

H:

	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\cdots$	$\langle D\rangle$	$\cdots$
M_1	A	R	A	R		A	
M_2	A	A	A	A	$\cdots$	A	
M_3	R	R	R	R		R	
M_4	A	A	R	R		R	
$\vdots$			$\vdots$				
D	R	R	A	A		?	
$\vdots$							

D computes opposites of the diagonal entries $\rightarrow D$

Undecidability : Reducibility

- A <u>reduction</u> is a way of converting a problem Π to another problem Π' such that a solution to Π' can be used to solve Π.
- Reductions are used to prove that certain problems are computationally unsolvable. The technique that is so obtained is called <u>reducibility</u>.
- Note that reducibility says nothing about the problems Π and Π' alone, but only about the solvability of Π in presence of a solution for Π'.
- Reducibility amounts to proving that Π' is undecidable by showing that some other problem Π already known to be undecidable reduces to Π'.
- Undecidability results are nowadays almost always obtained via reducibility.

Undecidability : Reducibility

Definition 11 (Reducibility)
Suppose Π and Π' are two problems. Then, Π is reducible to Π' ($\Pi \leq \Pi'$) if there is a computable function transforming an instance of Π into an instance of Π' so that the new instance is really an instance of Π' iff the original one is an instance of Π. Formally, this means that there is a computable function $f : \Sigma^* \to \Sigma^*$ such that $f(x) \in \Pi'$ iff $x \in \Pi$.

- Note that it makes sense to restrict f by requiring that, e.g., it be polynomial time. That is, f should not be much more complex than the problem Π from which we are reducing.
- Note that the definition above can be given in terms of languages: Let $L_1, L_2 \subseteq \Sigma^*$. Then $L_1 \leq L_2$ if there is a computable function $f : \Sigma^* \to \Sigma^*$ s.t. $f(x) \in L_2$ iff $x \in L_1$.

Undecidability (Problems from Language Theory)

Theorem 9

Let $HALT^1_{TM} = \{\langle M, w\rangle \mid M \text{ halts when started on } \sqcup\}$.
$HALT^1_{TM}$ is undecidable.

Proof (Reduction from HALT)
Construct a function f such that $f(\langle M\rangle, w) = \langle M'\rangle$.
That is, w becomes part of the description of M'.
Recall $HALT = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$.
Define M' as follows:

(1) Write w on tape
(2) simulate M on w
(3) if M rejects w, go to an infinite loop

Step (1) done by using $|w|$ states. Step (1) makes w part of the transition table of M'.
Step (3) is easily done in 2 states: One moves left, another right, and they switch between each other independently of the tape contents. $\blacksquare$

Undecidability: Language problems (cont'd)

<u>Theorem 10</u>

Let $HALT^2_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$.

$HALT^2_{TM}$ is undecidable

<u>Proof</u> (Reduction from HALT)

Assume that we have TM R to decide $HALT^2_{TM}$. Then construct TM S to decide HALT and use S as follows:

S: Input: $\langle M, w\rangle$

1. Run R on $\langle M, w\rangle$.
2. If R rejects, then <u>reject</u>.
3. If R accepts, then simulate M on w until M halts.
4. If M accepts w, then <u>accept</u>; if M rejects, <u>reject</u>.

The operation above shows that, if R decides $HALT^2_{TM}$, then S decides HALT, which is a contradiction, since HALT is undecidable ∎

Undecidability : Language problems (Cont'd)

Theorem 11

Let $E_{TM} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \phi\}$;
i.e., the problem whether the language accepted by a TM is empty. E_{TM} is undecidable.

Proof (Reduction from HALT):

Assume E_{TM} is decidable by a TM R.
Use R to build TM S which decides HALT.
Recall $HALT = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$.
Run R on a modified version M_1 of M: M_1 rejects all $x \neq w$ and works as usual on w and accepts if M does. That is:

$S =$

Input: $\langle M, w\rangle$
1. Build M_1 as described above.
2. Run R on $\langle M_1\rangle$.
3. If R accepts, <u>reject</u>; if R rejects, <u>accept</u>

Now, if R were to decide E_{TM}, so would S also decide HALT, which is a contradiction $\blacksquare$

Undecidability: Language Problems (Cont'd)

Theorem 12

Let $EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L_1(M_1) = L_2(M_2)\}$.

EQ_{TM} is undecidable.

Proof (Reduction from $E_{TM} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$).

Idea: E_{TM} = determining whether $L(M)$ is empty.

EQ_{TM} = determining whether 2 languages L_1, L_2 are the same.

Should one of L_1 or L_2 be empty, we end up with the problem of determining whether the language of the other machine is empty. So E_{TM} is a special case of EQ_{TM} where one of M_1, M_2 is made to recognize the empty language.

Formally: Let R decide EQ_{TM}. We build S to recognize E_{TM}

S = On input $\langle M\rangle$, where M is a TM:
1. Run R on $\langle M, M_1\rangle$ where M_1 rejects all inputs.
2. If R accepts, accept; if R rejects, reject

If R decides EQ_{TM} then S decides E_{TM}. $\blacksquare$

Undecidability: Language problems (Cont'd)

- Further undecidable language problems include <u>all</u> problems about TMs that involve only the language accepted by a TM. For example, the problems of testing whether:
  - the language accepted by a TM is empty (Th. 11).
  - the language accepted by a TM is finite.
  - the language accepted by a TM is context-free.
  - the language accepted by a TM is regular.

  etc.
- All the above follow from a more general result — Rice's theorem — which states that any non-trivial property of the languages accepted by TMs is undecidable.
- Beware of the following: Rice's theorem does not imply that all properties about TMs are undecidable! For example, many properties about the states of TMs are decidable; e.g., it is decidable whether a TM has n states.

Undecidability: Rice's Theorem

Theorem 13 (Rice's Theorem):

Let P be a language consisting of descriptions of TMs such that

1) P is <u>nontrivial</u>; i.e. P does not contain all TMs and P is not empty;
2) P is a property of languages of TMs; i.e., If $L(M_1) = L(M_2)$, we have $\langle M_1 \rangle \in P$ iff $\langle M_2 \rangle \in P$.

Then P is an undecidable language.

That is, every nontrivial property of the recursively enumerable languages is undecidable.

Note: 1) the empty set $\emptyset$ is not the same property as $\{\emptyset\}$, i.e., the property of being an empty language.

2) P is the set of codes of TMs M_i such that $L(M_i)$ satisfies a property of the RE languages.

Undecidability: Rice's Theorem (Cont'd)

<u>Proof of Rice's Theorem</u> (By contradiction):

Assume P is decidable using a TM R_P. We build a TM S to decide HALT using R_P as follows:

- Let T_ϕ be a TM such that $L(T_\phi) = \phi$ (i.e., T_ϕ always rejects.) Assume that $\langle T_\phi \rangle \notin P$ (Should $\langle T_\phi \rangle \in P$, reason with $\overline{P}$). Since P is non trivial, there is a TM T such that $\langle T \rangle \in P$.
- S uses R_P's ability to distinguish between T_ϕ and T:

$S =$

> Input: $\langle M, w \rangle$
>
> 1. Use M and w to build TM M_w:
>
> $M_w =$
>
>   > Input: x
>   >
>   > (a) Simulate M on x. If M halts and rejects, <u>reject</u>. If M accepts, go to (b).
>   >
>   > (b) Simulate T on x. If accepts, <u>accept</u>.
>
> 2. Use TM R_P to determine whether $\langle M_w \rangle \in P$. If $\langle M_w \rangle \in P$, <u>accept</u>. If $\langle M_w \rangle \notin P$, <u>reject</u>

In the above, if M accepts w, then M_w simulates T. So, if M accepts w, then $L(M_w) = L(T)$, else $L(M_w) = \phi$. Hence $\langle M, w \rangle \in P$ iff M accepts w. Also S decides

Undecidability: Applications of Rice's Theorem

- Many undecidable problems are proved using Rice's theorem, e.g.:
  - Is $L(M)$ infinite?
  - Is $L(M)$ finite?
  - Is $L(M)$ empty?
  - Is $L(M)$ a regular language? Is $L(M)$ context-free?
  - Is $L(M)$ containing at least $k \geq 0$ strings?
- Both conditions in Rice's theorem are necessary for proving that P is undecidable.
- Proving that a given language P is undecidable is done by showing that P satisfies both conditions of the Rice's Theorem.

Undecidability: Post Correspondence Problem

- Undecidability goes beyond the abstraction of TMs! For example it goes to problems concerning manipulation of strings, logic, number theory, etc.

- Definition 12:
  Suppose M is a TM. An accepting computation history for M on input w is a sequence $C_1, C_2, \ldots, C_\ell$ of configurations such that the 3 conditions of Definitions 4 hold; i.e.: C_1 is the start configuration, C_ℓ is an accepting configuration, and, for each C_i, $C_{\ell-1}$, $C_{i-1} \vdash C_i$ holds. A rejecting computation history differs from an accepting one by having C_ℓ be a rejecting configuration.

- Note that computation histories are by definition finite sequences. So, if a TM does not halt on w, no computation history exists for M on input w.

- Informally, PCP is the following problem:

  <u>Given</u>: a collection of <u>dominos</u> (i.e., pairs of strings)

  e.g., $\left\{ \left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right] \right\}$

  <u>Question</u>: Is there a list of (possibly repeating) dominos such that the string obtained by concatenating the top strings is the same as the string obtained by concatenating the bottom strings? The list is called a <u>match</u>.

  e.g., $\left[\frac{a}{ab}\right]\left[\frac{b}{ca}\right]\left[\frac{ca}{a}\right]\left[\frac{a}{ab}\right]\left[\frac{abc}{c}\right]$

- The following collection has no match:

  $\left\{ \left[\frac{abc}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{acc}{ba}\right] \right\}$

  Reason: lengths of top strings are greater than lengths of bottom strings [illegible] DCD (Cont'd)

Undecidability: PCP (Cont'd)

Definition 13 (PCP):

Given: A collection of dominoes:

$$P = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \dots, \left[\frac{t_k}{b_k}\right] \right\}$$

Question: Does P have a match, i.e., a sequence $i_1, \dots, i_\ell$ such that $t_{i_1} \dots t_{i_\ell} = b_{i_1} \dots b_{i_\ell}$?

$PCP = \{\langle P \rangle \mid P$ is an instance of the PCP with a match$\}$.

Theorem 14: PCP is undecidable.

Proof outline (Reduction from HALT):
Suppose a TM M and an input w. We build an instance P of PCP for which a match corresponds to an accepting computation history for M on w. Therefore, if we can determine whether P has a match, we can also determine whether M accepts w, i.e., we could decide HALT. ∎

Undecidability : PCP (Cont'd)

Let us take a look at an alternative formulation of PCP.

<u>Definition 14</u> (PCP - version Hopcroft):
An instance of PCP consists of two lists of equal length
$$A = w_1, w_2, \dots, w_k \text{ , and } B = x_1, x_2, \dots, x_k$$
over some alphabet Σ, for some $k \in \mathbb{N}$.
Each pair (w_i, x_i) is called a <u>corresponding pair</u>.
A PCP instance <u>has a solution</u>, if there is a sequence of one or more integers $i_1, i_2, \dots, i_m$ that, when interpreted as indexes for strings in the A and B lists, yield the same string, i.e., $w_{i_1}, w_{i_2}, \dots, w_{i_m} = x_{i_1} x_{i_2} \dots x_{i_m}$.
The sequence $i_1, i_2, \dots, i_m$ is called a <u>solution</u> to this PCP instance.

<u>Given</u> : a PCP instance P
<u>Question</u> : Does P have a solution?

Undecidability : PCP (Cont'd)

Example 1

Instance of PCP

	A	B
i	w_i	x_i
1	1	111
2	10111	10
3	10	0

$\Sigma = \{0, 1\}$

A Solution : $i_1 = 2$, $i_2 = 1$, $i_3 = 1$, $i_4 = 3$; $m = 4$.

$i_1, i_2, i_3, i_4 = 2, 1, 1, 3$

$w_2 w_1 w_1 w_3 = x_2 x_1 x_1 x_3 = 101111110$.

Undecidability : PCP (Cont'd)

Proof of Theorem 14 :

Technicalities used in the proof:

1) Assume TM M running on w never attempts to go off the left-hand end.
2) If $w = \varepsilon$, use $\sqcup$ instead of w.
3) Use the modified PCP:

$$MPCP = \{\langle P\rangle \mid P \text{ is instance of PCP with a match that starts with the } 1^{st} \text{ domino } \left[\frac{t_1}{b_1}\right]\}$$

Suppose TM R decides PCP. Construct S to decide HALT.
Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ with the usual semantics.
Then S builds an instance P of PCP that has a match iff M has an accepting configuration history on w.
Now the proof proceeds as follows:

<u>Step 1</u>: Build TM S to construct instance P' of MPCP.

<u>Step 2</u>: Convert P' into instance P of PCP.

The symbol # will be used to separate configurations.

Undecidability: PCP (cont'd)

Step 1: Construct P' such that M accepts w with accepting configuration history $C_1, C_2, \cdots, C_\ell$.

Part 1: Put $\left[\dfrac{\#}{\# q_0 w_1 w_2 \cdots w_n \#}\right]$ into P' as domino $\left[\dfrac{t_1}{b_1}\right]$. Note that $t_1 = \#$.

Set $C_1 = q_0 w_1 w_2 \cdots w_n \,(= b_1)$. Note that P' to extend t_1 to get a match.
We must add more dominos in P' to extend t_1 to get a match.
The new dominos make C_2 to appear as extension of b_1 and force a single-step simulation of M.
Note that Part 1 handles the start configuration of M.

Part 2 (handling head motion to the right):

$$\delta(q, a) = (r, b, R) \implies \text{put } \left[\frac{qa}{br}\right] \text{ into } P',$$

where $a, b \in \Gamma$; $q, r \in Q$; $q \neq q_r$.

Undecidability: PCP (Cont'd)

Part 3 (handling head motion to the left):

$$\delta(q,a) = (r, b, L) \Longrightarrow \text{put } \left[\frac{c\,q\,a}{r\,c\,b}\right] \text{ into } P',$$

where $a, b, c \in \Gamma$; $q, r \in Q$; $q \neq q_r$.

Part 4 (handling cells not adjacent to the head):

Put $\left[\frac{a}{a}\right]$ into P', for every $a \in \Gamma$.

Part 5 (handling separation of configurations and the blank portion of tape):

Put $\left[\frac{\#}{\#}\right]$ and $\left[\frac{\#}{\sqcup\#}\right]$ into P'.

This handles separation of configurations.

This adds $\sqcup$ at the end of the configuration to simulate the infinite sequence of $\sqcup$ at the right end of the tape.

Undecidability: PCP (Cont'd)

- Applications of the steps (2)–(5) above to construct a match force the simulation of M on input w until M reaches either q_a or q_r.
- Part 6 handles the case where the bottom reaches q_a by arranging for the top to catch up with the bottom to reach a complete match:

Part 6 (handling the "catch up"):

Put $\left[\dfrac{a\, q_a}{q_a}\right]$ and $\left[\dfrac{q_a\, a}{q_a}\right]$ into P', for every $a \in \Gamma$.

The effect of this part is: the head erases adjacent symbols until none is left.

Part 7 (Completing the match):

Put $\left[\dfrac{q_a \# \#}{\#}\right]$ into P'.

The effect is: a pseudo-step to erase $\#$.

Undecidability : PCP (Cont'd)

Step 2 : Convert P' into an instance P of PCP.

suppose $u = u_1 u_2 \cdots u_n$. Then define

$$* u = * u_1 * u_2 * \cdots * u_n \quad \text{(add '*' before every } u_i\text{)}$$
$$u * = u_1 * u_2 * \cdots * u_n * \quad \text{(add '*' after every } u_i\text{)}$$
$$* u * = * u_1 * u_2 * \cdots * u_n * \quad \text{(add '*' before and after every } u_i\text{)}$$

suppose $P' = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \ldots \left[\frac{t_k}{b_k}\right] \right\}$. Then, build P as follows:

$$P = \left\{ \left[\frac{*t_1}{*b_1*}\right], \left[\frac{*t_1}{b_1*}\right], \left[\frac{*t_2}{b_2*}\right], \ldots, \left[\frac{*t_k}{b_k*}\right], \left[\frac{*\diamond}{\diamond}\right] \right\}.$$

Observe that the role or inserting $*$ is as follows:

- Only the first domino $\left[\frac{*t_1}{*b_1*}\right]$ could possibly start a match since it is the only one where the top and the bottom start with $*$.
- The $*$ symbol has no impact on possible matches: it simply interleaves with the original symbols.
- The role of $\left[\frac{*\diamond}{\diamond}\right]$ is to allow the top to add the extra $*$ at the end of the match.

Undecidability: Application of the PCP

We will prove a few result by using a reduction for PCP. To start, let us recall a few notions from the language theory.

<u>Definition 15</u> (Context-free Grammar)

A Context-free grammar G is a tuple (V, T, P, S), where

- V is a finite set of <u>variables</u> (or <u>non terminal</u> symbols), each representing a set of strings.
- T is a finite set of <u>terminal</u> symbols that form strings when sequenced.
- S is the start symbol; $S \in V$
- P is a finite set of <u>production rules</u> which recursively define a language. Each rule is of the form $A \longrightarrow \alpha$, where
  - A is the head of the rule; $A \in V$.
  - $\longrightarrow$ is the production symbol.
  - α is a string involving zero or more terminal symbols or variables.

Undecidability: Application of the PCP (Cont'd)

Example 2

The grammar G_{pal} for palindromes, is given by

$$G_{pal} = (\{P\}, \{0,1\}, P, S)$$

where P is as follows:

$S \longrightarrow \varepsilon$ (ε is the empty word)

$S \longrightarrow 0$

$S \longrightarrow 1$

$S \longrightarrow 0S0$

$S \longrightarrow 1S0$

Compact notation for grammar above:

$$S \longrightarrow \varepsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1.$$

Definition 16 (Derivation using a grammar)
The use of production rules from head to body by repeatedly expanding the head (starting with the start symbol) until we reach a string of terminals is called the derivation of that string.

Undecidability: Application of the PCP (Cont'd)

More formally, given a CFG $G = (V, T, P, S)$, and let u, v, and w be strings of variables or terminals (i.e., $u, v, w \in (V \cup T)^*$). Let $A \to w \in P$. Then uAv <u>yields</u> uwv (denoted by $uAv \Rightarrow uwv$); and u <u>derives</u> v (denoted by $u \overset{*}{\Rightarrow} v$), if $u = v$ or there is a sequence $u_1, u_2, \ldots, u_k$ with $k \geq 0$ such that

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v.$$

The language of the grammar G, denoted $L(G)$, is the set $\{ w \in T^* \mid S \overset{*}{\Rightarrow} w \}$. $\blacksquare$

<u>Definition 17</u> (Parse Tree)
Suppose $G = (V, T, P, S)$. The <u>parse tree</u> of G is a tree where
(1) Each inner node is labelled with a $A \in V$
(2) Each leaf is labeled with either a variable, a terminal, or ε. A leaf ε is the only child of its parent.
(3) For each inner node labeled with A whose children are $X_1, X_2, \ldots, X_k$, there is $A \to X_1 X_2 \cdots X_k \in P$. $\blacksquare$

Undecidability: Application of the PCP (Cont'd)

Theorem 15

It is undecidable whether a CFG G is ambiguous.
That is, let $AMBIG_{CFG} = \{\langle G\rangle \mid G \text{ is an ambiguous CFG}\}$.
$AMBIG_{CFG}$ is undecidable.

Proof (Reduction from PCP) — DO THIS AS ASSIGNMENT —

Given an instance

$$P = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \ldots, \left[\frac{t_k}{b_k}\right] \right\}$$

of PCP. Construct a CFG G such that

$G = (\{T, B, S\}, \{t_1, \ldots, t_k, b_1, \ldots, b_k, a_1, \ldots, a_k\}, P, S)$, where P is

$$S \to T \mid B$$
$$T \to t_1 T a_1 \mid \cdots \mid t_k T a_k \mid t_1 a_1 \mid \cdots \mid t_k a_k$$
$$B \to b_1 T a_1 \mid \cdots \mid b_k B a_k \mid b_1 a_1 \mid \cdots \mid b_k a_k.$$

Show that G generates a word (or string) iff the PCP instance has a match

Undecidability: Application of the PCP (Cont'd)

Recall the grammar of the proof of Theorem 15. We introduce a special <u>language for the list A</u> as follows:

Given a list $A = w_1, w_2, \ldots, w_k$. Construct a CFG with A as the only variable. Terminals are the set $\{w_1, \ldots, w_k\}$, plus a distinct set of index symbols $a_1, \ldots, a_k$ that represent the choices of ~~dom~~ dominos in a match of the PCP instance. Production rules are:

$$A \to w_1 A a_1 \mid w_2 A a_2 \mid \cdots \mid w_k A a_k \mid$$
$$w_1 a_1 \mid w_2 a_2 \mid \cdots \mid w_k a_k .$$

The language obtained is $L_A(G_A)$ over the above grammar that we call G_A.

$$\overline{L_A} = \{ w \in (T \cup \{a_1, \ldots, a_k\})^* \mid w \notin L_A \}.$$

These complement languages serve the purpose of proving further undecidability results.

Undecidability: Application of the PCP (Cont'd)

Theorem 16: If L_A is the language for list A, then $\overline{L_A}$ is context-free.

Proof Idea: Give a Push Down Automaton for $\overline{L_A}$. ∎

Theorem 17: If L is a recursive language, so is $\overline{L}$.

Theorem 18

Suppose G_1, G_2 are CFGs. Then the following are undecidable

(a) $L(G_1) \cap L(G_2) = \phi$?

(b) $L(G_1) = L(G_2)$?

(c) $L(G_1) = T^*$ for some alphabet T?

(d) $L(G_1) \subseteq L(G_2)$?

Proof Idea (Reduction from PCP): Take an instance of PCP. Convert it to a question about CFG whose answer is "yes" iff that PCP instance has a match.
We might either directly use the problem itself or use the complement of the problem, thus Theorem 17.

Undecidability: Application of the PCP (Cont'd)

Proof of Theorem 18 (See remaining ones in Hopcroft *et al.*):

(a) Set $L(G_1) = L_T$ and $L(G_2) = L_B$.

$\Longrightarrow$ $L(G_1) \cap L(G_2)$ is the set of solutions to the involved instance of PCP.

$L(G_1) \cap L(G_2) = \phi$ iff no match exists for the PCP instance.

$\Longrightarrow$ I.e., we have shown that $L(G_1) \cap L(G_2) \neq \phi$? is undecidable. Henceforth, by Theorem 17, $L(G_1) \cap L(G_2) = \phi$ is undecidable.

(d) Let G_1 be a CFG for $(\Sigma \cup I)^*$, where

$$\Sigma = \{t_1, \ldots, t_k, b_1, \ldots, b_k\}$$
$$I = \{a_1, \ldots, a_k\}.$$

Let G_2 be a CFG for $\overline{L_T} \cup \overline{L_B}$.

$\Longrightarrow$ $L(G_1) \subseteq L(G_2)$ iff $\overline{L_T} \cup \overline{L_B} = (\Sigma \cup I)^*$

($\overline{L_T} \cup \overline{L_B}$ means the PCP instance has no solution)

Undecidability: The Recursion Theorem

- The recursion Theorem provides the ability to implement <u>self-reference</u> in TMs; i.e., the ability for a TM to refer to its own description.
- The technique used is as follows: a TM M obtains its own description and then proceeds by computing with it.
- An example of such a machine is one that obtains its own description and then simply prints out a copy of that description; Call such a machine SELF

<u>Lemma 19</u>

There is a computable function $q : \Sigma^* \longrightarrow \Sigma^*$ such that, if $w \in \Sigma^*$, $q(w)$ describes a TM P_w that prints out w.

<u>Proof</u>: Build TM Q to compute $q(w)$ as follows:

Q = | Input: w
1. Build TM P_w = \| Input: x
Erase x; write x on tape; halt \|
2. Output $\langle P_w \rangle$ |

Undecidability : The Recursion Theorem (cont'd)

- Construction of SELF: Two parts, A and B.

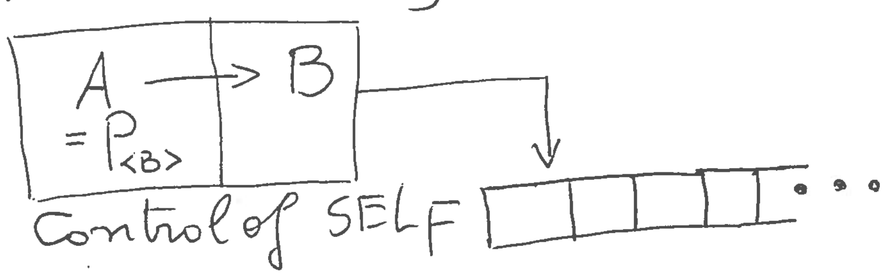
SELF =

A = $\boxed{P_{\langle B \rangle}}$

B =

Input: $\langle M \rangle$
1. Compute $q(\langle M \rangle) = M'$
2. Combine result M' with $\langle M \rangle$ to make M''
3. print $\langle M'' \rangle$ and halt.

Schematically:



A run of SELF:

1. Run A which outputs $\langle B \rangle$
2. Start B which finds $\langle B \rangle$ on tape
3. B computes $q(\langle B \rangle) = \langle A \rangle$ and combines $\langle A \rangle$ with $\langle B \rangle$ into $\langle SELF \rangle$
4. B prints $\langle SELF \rangle$ and halts.

- Note that Lemma 19 was used in construction of SELF.

Undecidability: The Recursion Theorem (Cont'd)

<u>Theorem 20</u> (Recursion Theorem):
Suppose T is a TM that computes a function $t: \Sigma^* \times \Sigma^* \to \Sigma^*$.
Then there is a TM R that computes a function $r: \Sigma^* \to \Sigma^*$ such that, for every $w \in \Sigma^*$,

$$r(w) = t(\langle R \rangle, w).$$

<u>Intuition behind the theorem</u>:

- Recall that we want to build a TM that obtains its own description and then go on to compute with it.
- That machine that obtains its own <u>description</u> is R, which uses another TM T that obtains its own description as extra input.
- To build a TM that receives its own description as input, and then computes with it, just build a machine T that takes $\langle T \rangle$ as extra input. Then the Recursion Theorem produces a new TM R which operates as T does, but with $\langle R \rangle$ filled in automatically.

Undecidability: The Recursion Theorem (Cont'd)

<u>Proof of Theorem 20</u> (similar to the construction of SELF):
Build TM R in 3 parts, A, B, and T such that

- T is a TM that computes $t : \Sigma^* \times \Sigma^* \longrightarrow \Sigma^*$.
- A is the TM $P_{\langle BT \rangle}$ described by $q(\langle BT \rangle)$.
  $P_{\langle BT \rangle}$ writes its output $\langle BT \rangle$ following any preexisting string on tape; that is, if w is preexisting on tape, $P_{\langle BT \rangle}$ prints $w\langle BT \rangle$.
- <u>B</u> applies q to the content of its tape and outputs $\langle A \rangle$.
  Then B combines $\langle A \rangle$, $\langle B \rangle$, and $\langle T \rangle$ into $\langle ABT \rangle = \langle R \rangle$.
  Finally, B processes $\langle R, w \rangle$ and passes control to T.

One obvious application of the construction above is a "<u>virus</u>". That construction correspond to the primary task of self-replication of computer programs designed to spread themselves among computers.

Undecidability: Application of the Recursion Theorem

Recall Theorem 6, i.e., HALTING is undecidable.

Alternative proof of Theorem 6 (By contradiction, using Recursion Th.):
Suppose a TM H that decides HALT. Then build TM B:

B =

> Input: w
> 1. Apply Recursion Theorem to obtain $\langle B \rangle$.
> 2. Run H on input $\langle B, w \rangle$.
> 3. If H rejects then accept; if H accepts, then rejects.

B offers an immediate contradiction when run on w:
B does exactly the opposite of what H is supposed to do.
Hence H cannot decide HALT, which contradicts our assumption.

Undecidability: Logical Theories

Boolean Logic

Boolean variables $X = \{x_1, x_2, \cdots, y_1, y_2, \cdots, z_1, z_2, \cdots, x, y, z\}$.
X is countably infinite.
Boolean connectives: $\{\vee, \wedge, \neg\}$ (also called Boolean operators)
Quantifiers: $\exists, \forall$
Alphabet used: $\{\wedge, \vee, \neg, (,), x, y, z, x_1, \cdots, y_1, \cdots, z_1, \cdots\}$

__Definition 18__ (Boolean Expressions):
A string ϕ is a Boolean expression if it is one of the following:

1. a variable,
2. an expression of the form $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, or $\neg \phi$, where ϕ, ϕ_1, and ϕ_2 are Boolean expressions,
3. a constant 0, or 1

A string ϕ is a quantified Boolean expression if, in addition to (1) – (3) above, we have

4. a quantified expression $\exists x \phi$, or $\forall x \phi$, where $\exists, \forall$ are quantifiers, and ϕ is a quantified Boolean expression

Undecidability: Logical Theories (Cont'd)

<u>Definition 19</u> (Truth assignment):
A truth assignment is a mapping from a finite set $X' \subset X$ of Boolean variables to the set $\{\underline{true}, \underline{false}\}$ of truth values.
A truth assignment T satisfies a Boolean expression ϕ (i.e., $T \models \phi$) if the following holds:

If $\phi = x_i$, then $T \models \phi$ if $T(x_i) = \underline{true}$;
If $\phi = \neg \phi_1$, then $T \models \phi$ if $T \not\models \phi_1$;
If $\phi = \phi_1 \vee \phi_2$, then $T \models \phi$ if either $T \models \phi_1$ or $T \models \phi_2$;
If $\phi = \phi_1 \wedge \phi_2$, then $T \models \phi$ if $T \models \phi_1$ and $T \models \phi_2$.

A Boolean expression ϕ is <u>satisfiable</u> if there is a truth assignment T such that $T \models \phi$; ϕ is <u>valid</u> or a <u>tautology</u> if $T \models \phi$ for all truth assignments.

Undecidability: Logical Theories (Cont'd)

First-order Logic

Vocabulary: set of relation symbols: $\{R, S, R_1, \ldots, S_1, \ldots\}$
set of function symbols: $\{f, g, f_1, \ldots, g_1, \ldots\}$
countable set of variables: $\{x, y, z, \ldots\}$
Note: equality ("$=$") is one of the relations;
0-ary function symbols are constants

terms: any variable is a term;
if f is a k-ary function symbol and $t_1, \ldots, t_k$ are terms, then $f(t_1, \ldots, t_k)$ is a term.

Definition 20 (Well-formed formulas of FO Logic):

- If R is k-ary relation symbol, and $t_1, \ldots, t_k$ are terms, then $R(t_1, \ldots, t_k)$ is an atomic formula.
- If ϕ, ϕ_1 and ϕ_2 are wffs, then so are $\neg\phi$, $\phi_1 \vee \phi_2$, and $\phi_1 \wedge \phi_2$
- If ϕ is a wff and x is any variable, then $(\forall x\, \phi)$, and $(\exists x)\, \phi$ are wffs.

Undecidability: Logical Theories (Cont'd)

- We used <u>truth assignment</u> to ascribe a meaning to Boolean expressions; we will use <u>models</u> to give a meaning to wffs of FO Logic.

<u>Definition 21 (Model):</u>

Suppose a FO logic with vocabulary Σ. Note that Σ is a collection of constant symbols $c_1, \dots, c_n, \dots$ relation (predicate) symbols $P_1, \dots, P_n, \dots$, and function symbols $f_1, \dots, f_n, \dots$ of some arity.

A Σ-structure (also called a <u>model</u>)

$$M = \langle U, \{c_i^M\}, \{P_i^M\}, \{f_i^M\} \rangle$$

consists of a universe U together with an interpretation I of

- each constant c_i from Σ as an element $c_i^M \in U$;
- each k-ary relation symbol P_i from Σ as a k-ary relation from U; i.e., a set $P_i^M \subseteq U^k$;
- each k-ary function symbol f_i from Σ as a function $f_i^M : U^k \to U$.

Undecidability: Logical Theories (Con'd)

A sample structure or model is as follow

$$\Sigma = \{0, 1, <, \cdot, +\}$$

$$R = \langle \mathbb{R}, 0^R, 1^R, <^R, +^R, \cdot^R \rangle$$, where all symbols have the expected meaning.

<u>Definition 22</u> (Semantics of wf FO formulas)
Given a Σ-structure M, we define for each term t with free variables $x_1, \dots, x_n$ the value $t^M(\bar{a})$, with $\bar{a} \in U^n$, and for each formula $\phi(x_1, \dots, x_n)$, the notion $M \models \phi(\bar{a})$ (i.e., $\phi(\bar{a})$ is true in M):

- t is constant $c \implies I^M(t) = c^M$.
- t is variable $x_i \implies I^M(x_i) = a_i$.
- $t = f(t_1, \dots, t_k) \implies I^M(t) = f^M(I^M(t_1), \dots, I^M(t_k))$.
- $\phi \equiv (t_1 = t_2) \implies M \models \phi(\bar{a})$ iff $t_1^M(\bar{a}) = t_2^M(\bar{a})$.
-

Undecidability: Logical Theories (Cont'd)

<u>Definition 2.2</u> (Cont'd):

- $\varphi \equiv P(t_1, \ldots, t_k) \implies M \models \varphi$ iff $(t_1^M(\bar{a}), \ldots, t_k^M(\bar{a})) \in P^M$.
- $M \models \neg \varphi(\bar{a})$ iff $M \models \varphi(\bar{a})$ does not hold.
- $M \models \varphi_1(\bar{a}) \wedge \varphi_2(\bar{a})$ iff $M \models \varphi_1(\bar{a})$ and $M \models \varphi_2(\bar{a})$.
- $M \models \varphi_1(\bar{a}) \quad \varphi_2(a)$ iff $M \models \varphi_1(\bar{a})$ or $M \models \varphi_2(\bar{a})$.
- $\psi(\bar{x}) \equiv \exists y\, \varphi(y, \bar{x}) \implies$
  $M \models \psi(\bar{a})$ iff $M \models \varphi(a', \bar{a})$ for some $a' \in U$.
- $\psi(\bar{x}) \equiv \forall y\, \varphi(y, \bar{x}) \implies$
  $M \models \psi(\bar{a})$ iff $M \models \varphi(a', a)$ for all $a' \in U$.

A <u>theory</u> over Σ is a set of sentence. A Σ-structure M is a <u>model</u> of a theory T iff for every sentence Φ of T, the structure M is such that $M \models \Phi$.
A theory is consistent if it has a model.

Undecidability: Logical Theories (Cont'd)

Let $Th(M)$ denote the collection of all sentences Φ in Σ^* such that $M \models \Phi$.

We want to show that $Th(\mathbb{N}, +, \cdot)$ is undecidable.

Lemma 21

Suppose a TM M, together with $w \in \Sigma^*$. There is a formula $\phi_{M,w} \in Th(\mathbb{N}, +, \cdot)$ obtainable from M and w such that

- $\phi_{M,w}$ contains a single free variable x,
- the sentence $\exists x\, \phi_{M,w}$ is true iff M accepts w.

Proof (sketch). Informally, the formula $\phi_{M,w}$ states the following: "x is a suitable encoding of an accepting computation history of M on w."

$\phi_{M,w}$ shows that computations of M can be expressed by formulas of $Th(\mathbb{N}, +, \cdot)$.

See details in Section 6.2 in Papadimitriou ∎

Undecidability: Logical Theories (Cont'd)

Theorem 22 (Undecidability of the Arithmetic):
$Th(\mathbb{N}, +, \cdot)$ is undecidable

proof (sketch):
Build a reduction from HALT to $Th(\mathbb{N}, +, \cdot)$ as follows: Use Lemma 21 to construct the sentence $\exists x\, \phi_{M,w}$ from the input $\langle M, w\rangle$, where M is a TM that halts on input w. $\blacksquare$

Time Complexity

- A problem may be decidable without, however, being solvable in practice (because the time spent to find the solution is simply prohibitive).
- How can time be measured?
- How can we classify problems w.r.t. the time needed to solve them?
- How can we determine to which class a particular problem belongs to?
- Time is an important computational ressource, along with space.

Time Complexity: Measuring Complexity

<u>Definition 23</u> (Time Complexity function)
For a given DTM M that halts for all inputs $w \in \Sigma^*$, the <u>time complexity function</u> $T_M : \mathbb{N} \longrightarrow \mathbb{N}$ of M is given by

$$T_M(n) = \max \{ t_M(w) \mid w \in \Sigma^* \text{ and } |w| = n \},$$

where $t_M(w)$ is the number of steps required by M to halt on input w, for any w.

Definition 23 is essentially a repetition of Definition 8, which gives the <u>worst case time complexity</u> of TM M.

<u>Definition 24</u> (Big-O notation for asymptotic upper bound):
Suppose f and g are functions $f, g : \mathbb{N} \longrightarrow \mathbb{R}^+$. Then $f(n) = O(g(n))$ if there are $c, n_0 \in \mathbb{Z}^+$ s.t., for any $n \geq n_0$

$$f(n) \leq c\, g(n).$$

When $f(n) = O(g(n))$, $g(n)$ is an <u>upper bound</u> for $f(n)$.
Bounds of the form n^c are <u>polynomial</u>; those of the ~~form~~ c^d are <u>exponential</u>.

Time Complexity: Measuring (Cont'd)

Definition 25 (*Time Complexity Class*):
Suppose $t : \mathbb{N} \to \mathbb{R}^+$ is a function. Then the *time complexity class* $\mathbf{TIME}(t(n))$ is the set of all languages that are decidable by a DTM in time $O(t(n))$.

Example 3:
There is a TM to decide $L = \{0^k 1^k \mid k \geq 0\}$ in $O(n^2)$; hence $L \in \mathbf{TIME}(n^2)$. There is also a TM to decide L in $O(n \log n)$, hence $L \in \mathbf{TIME}(n \log n)$. This is all we can do with a single tape TM. A different choice of the TM can affect the time complexity of a language. For example, $L \in \mathbf{TIME}(n)$, if we choose a 2-tape TM. Fortunately, the choice of a DTM does not lead to great differences as the following theorem shows.

<u>Time Complexity: Measuring (Cont'd)</u>

<u>Theorem 3:</u>

Suppose $t : \mathbb{N} \to \mathbb{R}^+$ is a function with $t(n) \geq n$. Then the time taken by the 1-tape TM of Theorem 1 to simulate a k-tape TM that runs in time $t(n)$ is $O(t^2(n))$.

<u>Theorem 23:</u>

Suppose $t : \mathbb{N} \to \mathbb{R}^+$ is a function with $t(n) \geq n$. Then the time taken by the DTM of Theorem 2 to simulate a NDTM that runs in time $t(n)$ is $2^{O(t(n))}$.

<u>Proof</u> idea: Reasoning about the <u>height</u> and the total <u>number of leaves</u> in the computation tree of the NDTM which are $t(n)$ and $b^{t(n)}$, respectively, where b is the largest number of possible transition choices of the NDTM. A breadth first search of that tree by the DTM is done in time $2^{O(t(n))}$. ■

Time Complexity: Classes P and NP

- Lessons learnt from theorems 3 and 23:
  - The difference between the times complexity measured on deterministic 1-tape and deterministic k-tape TMs is <u>at most polynomial</u>;
  - The difference between the times measured on deterministic and non-deterministic TMs is <u>at most</u> <u>exponential</u>.

- In computational complexity, polynomial differences are considered to be neglectable; exponential ones are not. Usually exponential times are obtained when a mere <u>brute-force</u> search of space solution is done (by exhaustion).
- All "reasonable" deterministic models are <u>polynomially equivalent</u> (e.g. Theorem 3). Since polynomial differences are considered to be "small", we "disregard" them.

Time Complexity: The class P

Definition 26 (the class P):
We set
$$P = \mathrm{TIME}(n^k) = \bigcup_{j>0} \mathrm{TIME}(n^j).$$
P is the class of languages that are decidable by a DTM in polynomial time.

P is very important in complexity theory:

- P is invariant for all models of computation that are polynomially equivalent to the 1-tape DTM,
- P corresponds to the class of problems that are realistically solvable on computers.

Time Complexity: The class NP

Definition 27 (Nondeterministic Time Complexity Class):
Suppose $t : \mathbb{N} \to \mathbb{R}^+$ is a function. Then the nondeterministic time complexity class $\mathrm{NTIME}(t(n))$ is the set of all languages decidable by an NDTM in time $\mathcal{O}(t(n))$.

Definition 28 (The class NP):
We set
$$NP = \mathrm{NTIME}(n^k) = \bigcup_{j>0} \mathrm{NTIME}(n^j).$$

NP is the class of languages that are decidable by a NDTM in polynomial time.

NP is very important in complexity theory:

- NP is invariant for all nondeterministic computation models that are polynomially equivalent to the NDTM,
- NP corresponds to the class of problems that are known to have no poly-time algorithms.

Time Complexity: The class NP (Cont'd)

Let us give an alternative definition for NP.

<u>Definition 29</u> (Verifier)

Suppose L is a language. A NDTM V is a verifier for L if $L = \{ w \mid V \text{ accepts } \langle w, c \rangle \text{ for some } c \in \Sigma^* \}$.

The verifier V is a <u>polynomial time verifier</u> if V runs in time polynomial in $|w|$.

L is said to be <u>polynomially verifiable</u> if L has a verifier.

<u>Definition 30</u> (Alternative characterization of NP):

NP is the class of languages that are polynomially verifiable.

The string c in Definition 29 is called a <u>certificate</u> of membership in L.

Time Complexity: The class NP (Cont'd)

Now we give an example of a language in NP:

HAMILTONIAC CIRCUIT (HC)

INSTANCE: A graph $G = (V, E)$

QUESTION: Does G contain a HC, that is, an ordering $\langle v_1, \ldots, v_n \rangle$ of the $v_i \in V$, where $n = |V|$, s.t. $(v_n, v_1) \in E$ and $(v_i, v_{i+1}) \in E$, for all i, $1 \le i \le n$?

HC = $\{\langle G, s, t \rangle \mid$ G is a graph and the path from s to t is a HC $\}$

Theorem 24: HC is in NP.

Proof: Build a NDTM for HC:

M = Input: $\langle G, s, t \rangle$
1. Non deterministically select an ordering $\langle v_1, \ldots, v_n \rangle$.
2. Check for repetitions. If any, reject.
3. Check whether $v_1 = s$ and $v_n = t$. If not, reject.
4. For each i, $1 \le i \le n$, Check whether $(v_i, v_{i+1}) \in E$. If any are not, reject. Else, accept.

Time Complexity: The class NP (Cont'd)

Further example

CLIQUE
INSTANCE: A graph $G=(V,E)$ and $j \in \mathbb{Z}^+$, $j \leq |V|$
QUESTION: Does G contain a <u>clique</u> of size j or more, that is, a subset $V' \subseteq V$ s.t. $|V'| \geq j$ and every two nodes of V' are joined by an edge in E?

$CLIQUE = \{ \langle G, k \rangle \mid G$ is an undirected graph with a k-clique $\}$

Theorem 25: CLIQUE is in NP.

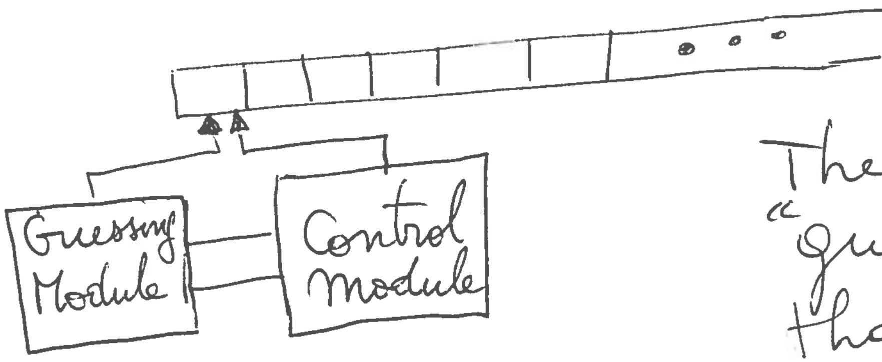
<u>Proof</u>: Give a verifier V:

$V =$
> Input: $\langle \langle G, k \rangle, c \rangle$ with $G=(V,E)$
> 1. Check whether c has k nodes from E
> 2. Check whether G has all edges connecting nodes in c
> 3. If both tests pass, <u>accept</u>; If not <u>reject</u>

Note that the <u>clique</u> is the certificate.

Time Complexity: The Class NP (Cont'd)

Notice that a NDTM N is in fact as a DTM, except that D now has a "guessing" module for non-deterministic choices:



The guessing module "guesses" the input w that D will have to start with and writes it on tape.

With this view in mind, we have an alternative proof for Theorem 25 in terms of NDTMs:

$N =$ Input: $\langle G, k \rangle$
1. Nondeterministically "guess" $c \subseteq E$ s.t. c has k nodes
2. Test whether G contains all edges connecting nodes in c
3. If yes, accept; otherwise, reject

Time Complexity: P versus NP

It is clear that $P \subseteq NP$, since DTMs are a special case of NDTMs.

Note that P is associated with <u>deciders</u> and NP with <u>verifiers</u>.

Polynomial verifiability looks stronger and more powerful than polynomial decidability.

No one has yet been able to prove the existence of a single language L such that $L \in NP$ and $L \notin P$.

$$P \stackrel{?}{=} NP$$

This is one of the greatest unsolved problems in computer science today. It's a hard problem!

Time Complexity: NP-Completeness

- <u>NP-Complete</u> problems are certain problem from NP whose complexity is related to the entire class NP in a way that, if a solution exists for any one of them, then <u>all</u> problems in NP would be polynomial time solvable.
- Practically, if we believe that $P \subsetneq NP$, then NP-completeness prevents us to waste our time looking for a solution to a problem.
- Historically the first problem that was shown to be NP-Complete is the following:

$$SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}.$$

Time Complexity; NP-Completeness (Cont'd)

<u>Definition 31</u>

A function $f : \Sigma^* \longrightarrow \Sigma^*$ is <u>polynomial time computable</u> if there is a polytime TM M that halts with $f(w)$ on its tape, when started on input w.

<u>Definition 32</u>

Suppose two languages L_1, L_2.
L_1 is <u>polynomial time reducable</u> to L_2 (i.e., $L_1 \leq_p L_2$), if a polytime computable function $f : \Sigma^* \longrightarrow \Sigma^*$ exists s.t. for every $w \in \Sigma^*$

$$w \in L_1 \iff f(w) \in L_2$$

The function f is called a <u>polynomial time reduction</u> of L_1 to L_2.

Time Complexity: NP-Completeness (cont'd)

Definition 33 (NP-Completeness)
Suppose L is a language. Then L is NP-complete if it satisfies two conditions:

1. L is in NP, and
2. Every $L' \in NP$ is polytime reducible to L.

Lemma 26: Suppose $L_1 \leq L_2$. Then, if $L_2 \in P$, we have $L_1 \in P$.

Proof: Suppose Σ_1 and Σ_2 are alphabets of L_1 and L_2; Let $f: \Sigma_1^* \to \Sigma_2^*$ be a polytime reduction of L_1 to L_2; and let M_2 be a polytime DTM deciding L_2. The M_1 for deciding L_1 is:

$M_1 =$
Input: w
1. Compute $f(w)$
2. Run M_2 on $f(w)$ and output M_2's output

$f(w) \in L_2 \Rightarrow w \in L_1$, since f is a polytime reduction.
Thus M accepts $f(w)$ whenever $w \in L_1$; and step (1) and (2) in M_1 are polytime

Time Complexity : NP – Completeness (Cont'd)

Lemma 27:

If $L_1 \leq L_2$ and $L_2 \leq L_3$, then $L_1 \leq L_3$.

Proof: Suppose Σ_1, Σ_2, and Σ_3 are the alphabets of L_1, L_3, and L_3. Suppose

$$f_1 : \Sigma_1^* \longrightarrow \Sigma_2^*$$
$$f_2 : \Sigma_2^* \longrightarrow \Sigma_3^*$$

are polytime tranformations from L_1 to L_2, and L_2 to L_3.

Then $f : \Sigma_1^* \longrightarrow \Sigma_3^*$ defined as $f(w) = f_2(f_1(w))$, for all $w \in \Sigma_1^*$ is the desired reduction of L_1 to L_3.

We have: $f(w) \in L_3$ iff $w \in L_1$ and f can be computed by a polytime DTM. $\blacksquare$

Lemma 28:

If $L_1, L_2 \in NP$, L_1 is NP-complete, and $L_1 \leq L_2$, then L_2 is NP-complete.

Proof: Show that for all $L' \in NP$, $L' \leq L_2$. this follows from Lemma 27 $\blacksquare$

Time Complexity: NP-Completeness (Cont'd)

Theorem 29 (Cook's Theorem): SAT is NP-Complete.

Proof: We must show that SAT $\in$ NP, and that every L $\in$ NP is reducible to SAT.

SAT $\in$ NP since we can build a polytime NDTM that "guesses" a truth assignment T to a given formula ϕ and accepts ϕ if $T \models \phi$.

Now show that every L $\in$ NP is reducible to SAT.

Idea: Suppose L $\in$ NP, and M is a NDTM for L. Then we build a formula ϕ that simulates M on a given input w.

If M accepts w $\Longrightarrow$ ϕ has a satisfying assignment that simulates an accepting history.

If M rejects w $\Longrightarrow$ ϕ has no such assignment.

Henceforth, w $\in$ L iff ϕ is satisfiable

Time Complexity: Cook's Theorem (Cont'd)

Suppose $M = (\Gamma, \Sigma, Q, q_0, q_a, q_r, \delta)$ decides L.
Let $T_M(n) \le p(n)$. We give a generic reduction f_L of L to instances of SAT with the property

$$\forall w \in \Sigma^* : w \in L \text{ iff } f_L(w) \text{ is a satisfiable Boolean formula.}$$

We build f_L by constructing a set of clauses which can be used to check whether w is accepted by M_L.

$$Q = \{q_0, q_1 = q_a, q_2 = q_r, q_3, \dots, q_\ell\}, \quad \ell = |Q| - 1$$

$$\Gamma = \{s_0 = \sqcup, s_1, s_2, \dots, s_v\}, \quad v = |\Gamma| - 1$$

The clauses of ϕ are drawn from a set U of variable types:

$$U = \{Q[i,k], H[i,j], S[i,j,k]$$

Time Complexity: Cook's Theorem (Cont'd)

Var	Range	Semantics
$Q[i,k]$	$0 \le i \le p(n)$ $0 \le k \le l$	At time i, M is in state q_k
$H[i,j]$	$0 \le i \le p(n)$ $-p(n) \le j \le p(n)+1$	At time i, the RW-head scans tape square j.
$S[i,j,k]$	$0 \le i \le p(n)$ $-p(n) \le j \le p(n)+1$ $0 \le k \le v$	At time i, the content of tape square j is s_k

At time 0, the tape contents are as follows:

...		....			w_1	....	w_n	...
	$-\lvert x\rvert$		-1	0	1	...	n	

$\underbrace{-|x| \quad \cdots \quad -1}_{\text{guess } x,\ |x| \le n}$ $\quad 0 \quad$ $\underbrace{1 \quad \cdots \quad n}_{\text{Input } w,\ |w| = n}$

Note: We assume for this proof that the tape is infinite on both ends.

Time Complexity : Cook's Theorem (Cont'd)

We note that an arbitrary truth assignment <u>needs not</u> correspond to a Computation, nor to an accepting history. However, we want a truth assignment such that

$w \in L \iff$ there is an accepting history of M on w.

$\iff$ there is an accepting history of M on w with $\leq p(n)$ steps in its checking stage and a guessed x s.t. $|x| \leq p(n)$.

$\iff$ there is a satisfying truth assignment for $\phi = f_L(w)$

Clauses in $\phi = f_L(w)$ are divided in six groups.

Each group of clauses imposes restrictions on any satisfying truth assignment.

Time Complexity: Cook's Theorem (Cont'd)

Group of clauses	Restriction imposed
G_1	At each time i, M is exactly in one state
G_2	At each time i, WR-head scans exactly 1 square
G_3	At each time i, each square has exactly 1 symbol
G_4	A time 0, computation is in the initial configuration for checking w
G_5	By time $p(n)$, M has entered state q_a
G_6	For each time i, $0 \leq i < p(n)$, the configuration at time $i+1$ follows by a single application of δ for configuration at time i

Time Complexity: Cook's Theorem (Cont'd)

G_1

(1) $\{Q[i,0], Q[i,1], \cdots, Q[i,l]\}$
$$0 \leq i \leq p(n)$$

(2) $\{\overline{Q[i,j]}, \overline{Q[i,j']}\}$
$$0 \leq i \leq p(n), \quad 0 \leq j < j' \leq l$$

Conjunction of the clauses in $G_1(1)$ is satisfied iff for each time i, M is in at least one state.

Conjunction of the clauses in $G_1(2)$ is satisfied iff at no time i, M is in more than one state.

Time Complexity: Cook's Theorem (Cont'd)

G_2 (1) $\{H[i, -p(n)], H[i, -p(n)+1], \ldots, H[i, p(n)+1]\}$,

$$0 \le i \le p(n)$$

(2) $\{\overline{H[i,j]}, \overline{H[i,j']}\}$,

$$0 \le i \le p(n)$$
$$-p(n) \le j < j' \le p(n)+1$$

Conjunction of the clauses in $G_2(1)$ is satisfied iff for each time i, the RW-head is scanning <u>at least one</u> square.

Conjunction of the clauses in $G_2(2)$ is satisfied iff at no time i, the RW-head is scanning <u>more than one</u> square.

Time Complexity : Cook's Theorem (Cont'd)

G_3

(1) $\{S[i,j,0], S[i,j,1], \ldots, S[i,j,v]\}$

$$0 \le i \le p(n), \quad -p(n) \le j \le p(n)+1$$

(2) $\{\overline{S[i,j,k]}, \overline{S[i,j,k']}\}$

$$0 \le i \le p(n)$$
$$-p(n) \le j \le p(n)+1$$
$$0 \le k < k' \le v$$

Conjunction of the clauses in G_3(1) is satisfied iff for each time i, the content of each square j is <u>at least one</u> symbol of Γ.

Conjunction of the clauses in G_3(2) is satisfied iff for no time i and no square j, the content of j is <u>more than one</u> symbol of Γ.

Time Complexity: Cook's Theorem (Cont'd)

G_4 (1) $\{Q[0,0]\}$, $\{H[0,1]\}$, $\{S[0,0,0]\}$,

(2) $\{S[0,1,k_1]\}$, $\{S[0,2,k_2]\}$, $\ldots$, $\{S[0,n,k_n]\}$,

(3) $\{S[0,n+1,0]\}$, $\{S[0,n+2,0]\}$, $\cdots$, $\{S[0,p(n)+1,0]\}$,

where $w = s_{k_1} s_{k_2} \cdots s_{k_n}$; $s_0 = \sqcup$.

$G_4(1)$ clauses are satisfied iff, at time 0, M is in state q_0, and the RW-head is at square 1 and the content of square 0 is s_0.

$G_4(2)$ clauses are satisfied iff, at time 0, the content of squares $1, \ldots, n$ is the input word w.

$G_4(3)$ clauses are satisfied iff, at time 0, the content of squares $n+1, \ldots, p(n)+1$ is s_0.

G_5 $\{Q[p(n), 1]\}$

This is satisfied iff, at time $p(n)$, M has reached state $q_1 = q_a$.

Time Complexity: Cook's Theorem (cont'd)

G_6

(1) $\{\overline{S[i,j,m]},\ H[i,j],\ S[i+1,j,m]\}$

$0 \le i < p(n),\ -p(n) \le j \le p(n)+1,\ 0 \le m \le v$

(2) $\{\overline{H[i,j]},\ \overline{Q[i,k]},\ \overline{S[i,j,m]},\ H[i+1,j+\Delta]\}$

$\{\overline{H[i,j]},\ \overline{Q[i,k]},\ \overline{S[i,j,m]},\ Q[i+1,k']\}$

$\{\overline{H[i,j]},\ \overline{Q[i,k]},\ \overline{S[i,j,m]},\ S[i+1,j,m']\}$

$0 \le i < p(n),\ -p(n) \le j \le p(n)+1,\ 0 \le k \le \ell,\ 0 \le m \le v.$

G_6 (1) Clause is satisfied iff, whenever the RW-head is not scanning square j at time i, symbol in j does not change between times i and $i+1$.

Note: if $q_k \in Q - \{q_a, q_r\}$, then Δ, k', and m' are such that

$$\delta(q_k, s_m) = (q_{k'}, s_{m'}, \Delta);$$

if $q_k \in \{q_a, q_r\}$, then $\Delta = 0$, $k' = k$, and $m = m'$.

$\Delta \in \{-1, 0, +1\}$ describes the direction of the NDTM M.

Time Complexity: Cook's Theorem (Cont'd)

Recall subgroup $G_6(2)$ of clauses:

$G_6(2)$

$$\{\overline{H[i,j]}, \overline{Q[i,k]}, \overline{S[i,j,m]}, H[i+1, j+\Delta]\}$$

$$\{\overline{H[i,j]}, \overline{Q[i,k]}, \overline{S[i,j,m]}, Q[i+1, k']\}$$

$$\{\overline{H[i,j]}, \overline{Q[i,k]}, \overline{S[i,j,m]}, S[i+1, j, m']\}$$

$G_6(2)$ Clauses are satisfied iff
for each time i, square j, state q_k, and symbol s_m, whenever it is <u>not</u> the case that the RW-head scans j and the current state is q_k and s_m is read, then, at time $i+1$, the RW-head is scanning $j+\Delta$, the state entered by M is $q_{k'}$, and the symbol scanned is $s_{m'}$.

Time Complexity: Cook's Theorem (Cont'd)

Let $C = \bigcup_{1 \le i \le 6} G_i$. The construction shows that:

(1) If $w \in L$, then M accepts w in time $p(n)$, and the accepting history, together with the interpretation of the variables, imposes a truth assignment that satisfies C.

(2) If there is a satisfying truth assignment for C, then this corresponds to an accepting history of M on x.

Henceforth, $f_L(w)$ is satisfiable iff $w \in L$.

Given w, the length $|f_L(w)|$ is bounded by a polynom:

$$|f_L(w)| \le |U| \cdot |C| = O(p(n)^2) \cdot O(p(n)^2) = O(p(n)^4).$$

$\blacksquare$

Time Complexity: Proving NP-Completeness Results

- The proof of the NP-Completeness of SAT is lengthy.
- Fortunately, proving NP-completeness results does not require such convoluted proofs
- The process for proving NP-completeness for a decision problem Π goes as follows:

  (1) Show that Π is in NP,
  (2) Select a known NP-complete problem Π',
  (3) construct a reduction f from Π' to Π,
  (4) Prove that f is polytime, and
  (5) Prove that for all $I \in D_{\Pi'}$,
  $I \in Y_{\Pi'}$ iff $f(I) \in Y_{\Pi}$.

Time Complexity: Proving NP-Completeness Results (Cont'd)

- There is a core of NP-complete problems that have proved to be useful in obtaining NP-completeness results. These problems seem to be more suitable than others in practice for proving NP-completeness:

3-SATISFIABILITY (3-SAT)

INSTANCE: Collection $C = \{c_1, \ldots, c_m\}$ of clauses on finite set U of variables s.t. $|c_i| = 3$, $1 \leq i \leq m$.

QUESTION: Is there a truth assignment for U that satisfies all the clauses in C?

3-DIMENSIONAL MATCHING (3DM)

INSTANCE: Set $M \subseteq W \times X \times Y$, were W, X, and Y are disjoint sets s.t. $|W| = |X| = |Y| = q$.

QUESTION: Does M contain a <u>matching</u>, that is, a subset $M' \subseteq M$ s.t. $|M'| = q$ and no two elements of M' agree in any coordinate?

Time Complexity: Proving NP-Complete Results (Cont'd)

VERTEX COVER (VC)
INSTANCE: Graph $G = (V, E)$ and positive integer $K \leq |V|$
QUESTION: Is there a vertex cover of size K or less for G, that is, a subset $V' \subseteq V$ such that $|V'| \leq K$, and, for each edge $\{u, v\} \in E$, at least one of u or $v \in V'$.

CLIQUE
INSTANCE: Graph $G = (V, E)$ and positive integer $J \leq |V|$
QUESTION: Does G contain a clique of size J that is, a subset $V' \subseteq V$ s.t. $|V'| \geq J$ and every two vertices in V' are joined by an edge in E?

HAMILTONIAN CIRCUIT (HC)
INSTANCE: Graph $G = (V, E)$
QUESTION: Does G contain a HC, that is, an ordering $\langle v_1, \ldots, v_n \rangle$ of the vertices of G, with $n = |V|$, s.t. $(v_n, v_1) \in E$ and $(v_i, v_{i+1}) \in E$, for all i, $1 \leq i < n$?

Time Complexity: Proving NP-Completeness Results (cont'd)

PARTITION

INSTANCE: Finite set A and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$

QUESTION: Is there a subset $A' \subseteq A$ s.t.

$$\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a) \;?$$

The following diagram shows the reductions used to prove the NP-completeness of the 6 basic problems:

$$\begin{tikzcd}
 & \text{SAT} \arrow[d] & & \\
 & \text{3SAT} \arrow[dl] \arrow[dr] & & \\
\text{3DM} \arrow[d, dash] & & \text{VC} \arrow[dl] \arrow[dr] & \\
\text{PARTITION} & \text{HC} & & \text{CLIQUE}
\end{tikzcd}$$

Time Complexity: Proving NP-Completeness Results (Cont'd)

Theorem 30: 3SAT is NP-Complete.

Proof:

3SAT $\in NP$ for the same reasons as for SAT.

Reduce SAT to 3SAT. Let

$U = \{u_1, \dots, u_n\}$ set of variables

$C = \{c_1, \dots, c_m\}$ set of clauses of an instance of SAT.

We build collection C' of 3-literal clauses on a set U' of variables s.t. C' is satisfiable iff C is satisfiable

Idea: Replace each individual clause $c_j \in C$ by a logically equivalent collection C'_j of 3-litteral clauses based on the original vars in U and some additional vars U'_j used only in clauses of C'_j.

Then we will set:

$$U' = U \cup \bigcup_{j=1}^{m} U'_j \quad \text{and} \quad C' = \bigcup_{j=1}^{m} C'_j .$$

Time Complexity; Proving NP-Completeness Results (Cont'd)

Construction of C_j' and U_j' from c_j:

Suppose $c_j = \{z_1, z_2, \ldots, z_k\}$. Then, depending on value of k:

Case 1. $k = 1$. $U_j' = \{y_j^1, y_j^2\}$

$$C_j' = \{\{z_1, y_j^1, y_j^2\}, \{z_1, y_j^1, \bar{y}_j^2\}, \{z_1, \bar{y}_j^1, y_j^2\}, \{z_1, \bar{y}_j^1, \bar{y}_j^2\}\}$$

Case 2. $k = 2$. $U_j' = \{y_j^1\}$,

$$C_j' = \{\{z_1, z_2, y_j^1\}, \{z_1, z_2, \bar{y}_j^1\}\}$$

Case 3. $k = 3$. $U_j' = \phi$, $C_j' = \{c_j\}$

Time Complexity: Proving NP-Completeness Results (Cont'd)

Case 4. $k > 3$. $U_j' = \{y_j^i : 1 \le i \le k-3\}$

$$C_j' = \{\{z_1, z_2, y_j^1\}\} \cup \{\{\bar{y}_j^i, z_{i+2}, y_j^{i+1}\} : 1 \le i \le k-4\} \cup \{\{\bar{y}_j^{k-3}, z_{k-1}, z_k\}\}$$

It can be shown that C' is satisfiable iff C is. Since the number of 3-literal clauses is bounded by a polynomial in $m \cdot n$, the size of resulting 3SAT instance is bounded by a polynomial function of the size of the given SAT instance. ∎

Time Complexity: Proving NP-Completeness (Cont'd)

Examples of collections C_j' of 3-literal clauses:

<u>Case 1</u>: $k = 1$

single literal formula: x

2 new variables: u, v

new formula:

$$(x \vee u \vee v) \wedge (x \vee u \vee \neg v) \wedge (x \vee \neg u \vee v) \wedge (x \vee \neg u \vee \neg v)$$

<u>Case 2</u>: $k = 2$

2 literals formula: $(x \vee y)$

1 new variable: z

new formula:

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

Time Complexity: Proving NP-Completeness (Cont'd)

Case 3 : $k = 3$

3 literal formula : $(X \vee Y \vee Z)$. Remains as is.

Case 4 : $k \geq 4$

k-literal formula: $X_1 \vee X_2 \vee \cdots \vee X_k$

k−3 new variables: $y_1, y_2, \ldots, y_{k-3}$

new formula :

clauses = k − 2

$$\left\{ \begin{array}{l} (X_1 \vee X_2 \vee Y_1) \wedge (X_3 \vee \neg Y_1 \vee Y_2) \wedge (X_4 \vee \neg Y_2 \vee Y_3) \wedge \\ \ldots (X_{k-2} \vee \neg Y_{k-4} \vee Y_{k-3}) \wedge \\ \qquad (X_{k-1} \vee X_k \vee X \neg Y_{k-3}) \end{array} \right.$$

$$(X_1 \vee X_2 \vee X_3 \vee X_4) \Longrightarrow$$

$$(X_1 \vee X_2 \vee Y) \wedge (X_3 \vee X_4 \vee \neg Y)$$

with 1 new variable: Y

Time Complexity: Proving NP-Completeness (Contd)

$$(X_1 \vee X_2 \vee X_3 \vee X_4 \vee X_5) \Longrightarrow$$

$$(X_1 \vee X_2 \vee Y_1) \wedge (X_3 \vee \neg Y_1 \vee Y_2) \wedge (X_4 \vee X_5 \vee \neg Y_2)$$

with 2 new variables: Y_1, Y_2

$$(X_1 \vee X_2 \vee X_3 \vee X_4 \vee X_5 \vee X_6 \vee X_7) \Longrightarrow$$

$$(X_1 \vee X_2 \vee Y_1) \wedge (X_3 \vee \neg Y_1 \vee Y_2) \wedge$$
$$(X_4 \vee \neg Y_2 \vee Y_3) \wedge$$
$$(X_5 \vee \neg Y_3 \vee Y_4) \wedge$$
$$(X_6 \vee X_7 \vee \neg Y_4)$$

Time Complexity: Proving NP-Completeness (Cont'd)

Suppose we have

$$c_j = X_1 \vee X_2 \vee X_3 \vee X_4 \vee X_5 \vee X_6 \vee X_7$$

We show that the constructed formula

$$\begin{aligned} C_j' = (X_1 \vee X_2 \vee Y_1) \wedge\ & (X_3 \vee \neg Y_1 \vee Y_2) \wedge \\ & (X_4 \vee \neg Y_2 \vee Y_3) \wedge \\ & (X_5 \vee \neg Y_3 \vee Y_4) \wedge \\ & (X_6 \vee X_7 \vee \neg Y_4) \end{aligned}$$

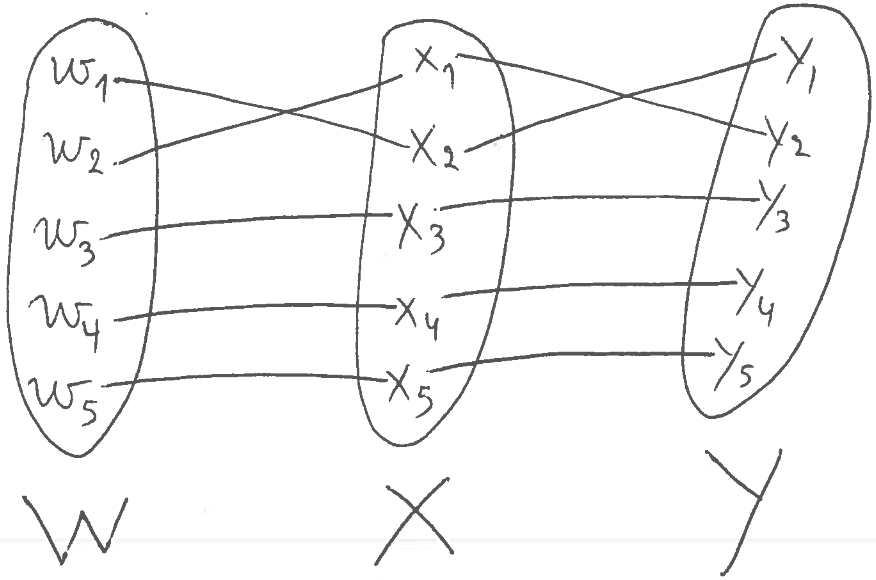
is satisfiable iff c_j is:

Suppose assignment T satisfies c_j and makes X_j true.
Then if we make $y_1, \ldots, y_{j-2}$ true and $y_{j-1}, y_j, \ldots, y_3$ false, we satisfy all clauses in C_j'.

If T makes all the X_j false, we can't extend T to make C_j' true. <u>Reason</u>: there are 5 clauses in C_j', and each of the 4 Y's can only make one clause true regardless of whether that Y is true or false.

Time Complexity: Proving NP-Completeness (Cont'd)

Theorem 31: 3DM is NP-Complete.



Sample matching:

$\{\langle w_1, x_2, y_1\rangle, \langle w_2, x_1, y_2\rangle, \langle w_3, x_3, y_3\rangle, \langle w_4, x_4, y_4\rangle, \langle w_5, x_5, y_5\rangle\}$

3DM generalizes the "marriage problem":

Given: n men, n women

Question: Arrange n marriages that avoid polygamy and polyandry.

Note: 2DM is in P (Karp[1973]).

Time Complexity: Proving NP-Completeness (Cont'd)

3DM $\in$ NP:

Guess a subset of $q = |W| = |X| = |Y|$ triples from M.
Check in polytime whether the guessed triples agree in any coordinate.

NP-Completeness (Reduction from 3SAT):

$U = \{u_1, u_2, \ldots, u_n\}$: Variables.

$C = \{c_1, c_2, \ldots, c_m\}$: Clauses of an arbitrary 3SAT instance.

$\boxed{\text{Build disjoint sets } W, X, Y \text{ s.t. } |W|=|X|=|Y|, \text{ and set } M \subset W \times X \times Y \text{ s.t. } M \text{ contains a matching } M' \text{ iff } C \text{ is satisfiable.}}$

M has 3 classes of triples with specific functions:

- Truth-setting & fan-out
- Satisfaction ~~testing~~.
- Garbage collection

Time Complexity: Proving NP-Completeness (Cont'd)

Truth-setting & fan-out: 2 sets for each $u_i \in U$:

$$T_i^t = \{(\bar{u}_i[j],\ a_i[j],\ b_i[j]) : 1 \le j \le m\}$$

$$T_i^f = \{(u_i[j],\ a_i[j+1],\ b_i[j]) : 1 \le j < m\} \cup \{(u_i[m],\ a_i[1],\ b_i[m])\}.$$

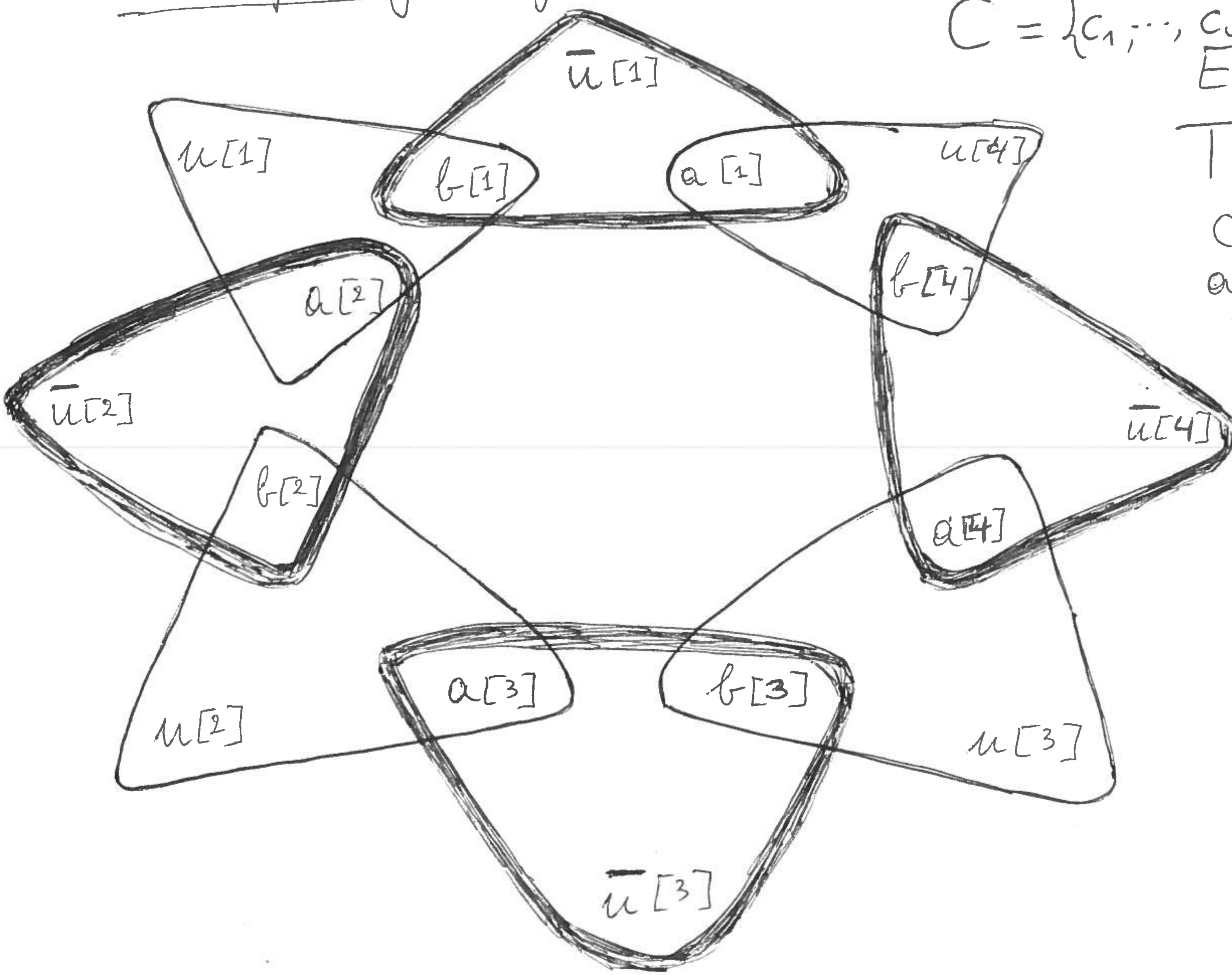
$a_i[j] \in X$, $b_i[j] \in Y$, $1 \le j \le m$: "internal elements".

$u_i[j], u[j] \in W$, $1 \le j \le m$: "external elements".

$T_i = T_i^t \cup T_i^f$. T_i forces a matching M' to either set u_i <u>true</u> or set u_i <u>false</u> (since internal nodes will not appear in triples outside of T_i).

Time Complexity: NP-Completeness (Cont'd)

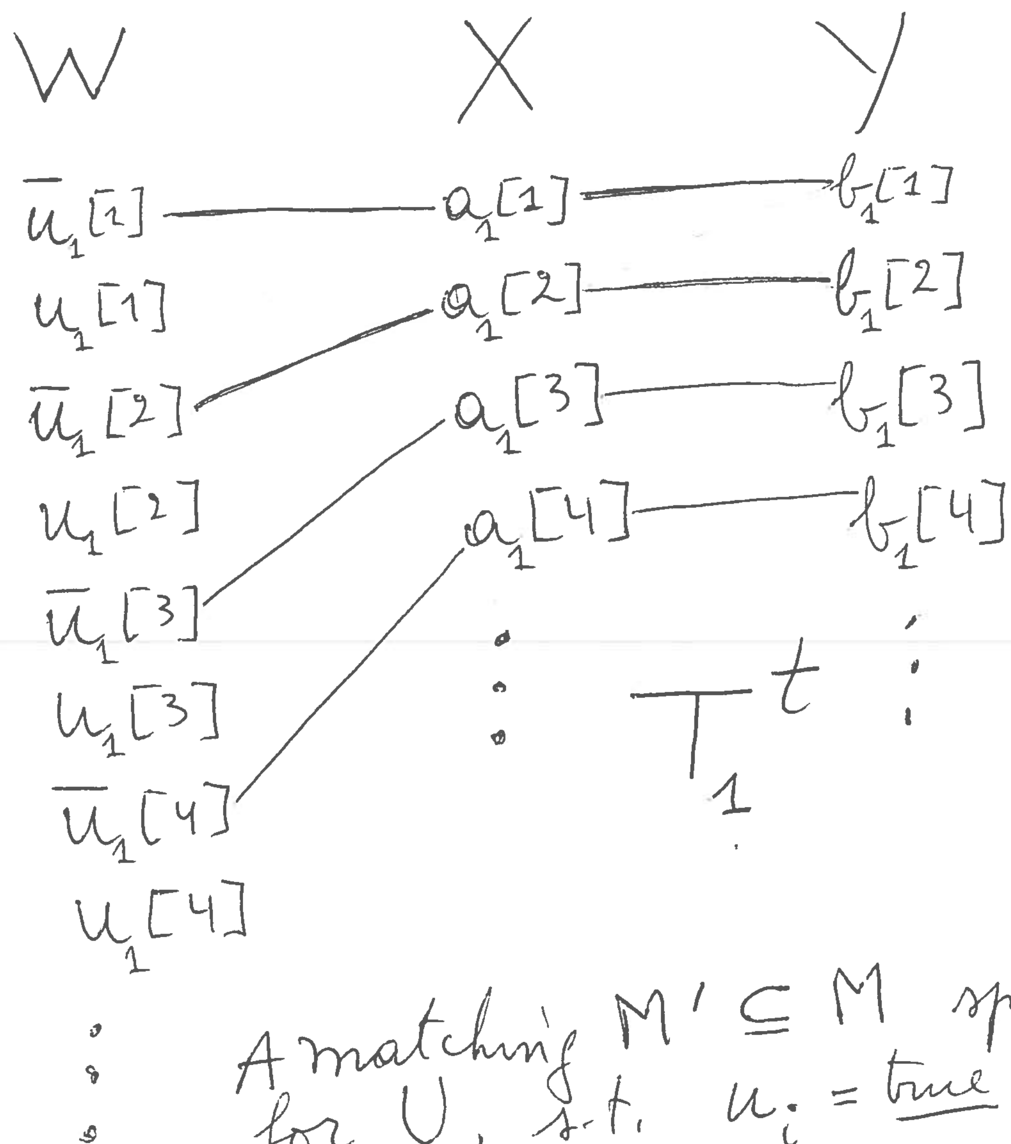
Example of T_i for $m = 4$: $U = \{u_1, u_2, \ldots, u_n\}$, $C = \{c_1, \ldots, c_4\}$.



Either T_i^t or T_i^f must be chosen by a matching.

Build and T_i for $1 \le i \le n$.

Time Complexity: NP-Completeness (Cont'd)



T_1 forces a matching to choose between making u_1 <u>true</u> or making u_1 <u>false</u>.

A matching $M' \subseteq M$ specifies a truth assignment for U, s.t. $u_i =$ <u>true</u> iff $M' \cap T_i = T_i^t$.

Time Complexity: NP-Completeness (Cont'd)

<u>Satisfaction testing</u>: for each $c_j \in C$:

$$C_j = \{(u_i[j], s_1[j], s_2[j]) : u_i \in c_j\} \cup \{(\bar{u}_i[j], s_1[j], s_2[j]) : \bar{u}_i \in c_j\}$$

$s_1[j] \in X$, $s_2[j] \in Y$: internal elements.

$u_i[j]$ or $\bar{u}_i[j] \in W$, $1 \le i \le n$: external elements.

Any matching $M' \subseteq M$ will have <u>exactly</u> one triple from C_j.

A matching $M' \subseteq M$ will have exactly one triple from C_j iff the truth setting determined by M' satisfies clause c_j.

Time Complexity: NP-Completeness (Cont'd)

Garbage collection:

$$G = \{(u_i[j], g_1[k], g_2[k]), (\bar{u}_i[j], g_1[k], g_2[k]) : 1 \le k \le m(n-1), 1 \le i \le n, 1 \le j \le m\}.$$

$g_1[k] \in X$, $g_2[k] \in Y$, $1 \le k \le m(n-1)$: internal elements

$u_i[j]$ or $\bar{u}_i[j] \in W$: external elements.

That is each pair $(g_1[k], g_2[k])$ must be matched with a unique $u_i[j]$ or $\bar{u}_i[j]$ not occurring in any triples of $M' - G$.

There are $m(n-1)$ such unique "uncovered" external elements.

Thus G merely extends $M' - G$ to a matching M'

This is exactly the idea of garbage collection.

Time Complexity : NP- Completeness (Cont'd)

In summary :

$$W = \{u_i[j], \bar{u}_i[j] : 1 \le i \le n, 1 \le j \le m\} \cup$$

$$X = \{a_i[j] : 1 \le i \le n, 1 \le j \le m\} \cup \{g_1[j] : 1 \le j \le m(n-1)\}$$

$$\{s_1[j] : 1 \le j \le m\} \cup$$

$$Y = \{b_1[j] : 1 \le i \le n, 1 \le j \le m\} \cup \{g_2[j] : 1 \le j \le m(n-1)\}$$

$$\{s_2[j] : 1 \le j \le m\}$$

$$M = \bigcup_{i=1}^{n} T_i \cup \bigcup_{j=1}^{m} C_j \cup G.$$

M has $2mn + 3m + 2m^2 n(n-1)$ triples and can be obtained in polytime.

<u>Claim</u> : M contains a matching iff C is satisfiable

Time Complexity: NP-Complexity (Cont'd)

INDEPENDENT SET
INSTANCE Graph $G=(V,E)$, positive integer $J \leq |V|$
QUESTION Does G contain an <u>independent set</u> V' s.t. $|V'| \geq J$; i.e., a subset $V' \subseteq V$ s.t., for all $u,v \in V'$, $(u,v) \notin E$?

<u>Lemma 32</u>: Given any graph $G=(V,E)$, and $V' \subseteq V$, the following are equivalent statements:

(a) V' is a <u>vertex cover</u> for G.

(b) $V-V'$ is an <u>independent set</u> for G.

(c) $V-V'$ is a <u>clique</u> for $G^c=(V,E^c)$, with $E^c = \{(u,v) : u,v \in V \text{ and } (u,v) \notin E\}$.

As a consequence, NP-Completeness of all 3 problems follows from NP-Completeness of any of Them.

Time Complexity: NP-Completeness (Cont'd)

Theorem 33: VC is NP-complete.

Proof (Reduction from 3SAT):

Let $U = \{u_1, \dots, u_n\}$, $C = \{c_1, \dots, c_m\}$ 3SAT instance.

Build $G = (V, E)$ and $K \le |V| \in \mathbb{Z}^+$ s.t.
G has a VC of size K iff C is satisfiable.

G has 3 classes of components:

- Truth-setting components
- Satisfaction testing components
- Garbage collection components

Time Complexity: NP-Completeness (Cont'd)

Truth-setting components: for each $u_i \in U$

$$T_i = (V_i, E_i) \quad \text{with } V_i = \{u_i, \bar{u}_i\};\ E_i = \{(u_i, \bar{u}_i)\}.$$

Any VC will have to contain at least u_i or $\bar{u}_i$ to cover the edge $(u_i, \bar{u}_i)$.

Satisfaction testing: for each $c_j \in C$

$$S_j = (V_j', E_j') \quad \text{with}$$

$$V_j' = \{a_1[j], a_2[j], a_3[j]\}$$

$$E_j' = \{(a_1[j], a_2[j]), (a_1[j], a_3[j]), (a_2[j], a_3[j])\}$$

Any VC will have to contain at least 2 vertices from V_j' to cover edges in E_j'.

Time Complexity: NP-Completeness (Cont'd)

Garbage collection (to ensure communication between components):

These depend on which literals occur in which clauses:

For each $c_j = \{x_j, y_j, z_j\}$, we add the following edges:

$$E''_j = \{(a_1[j], x_j), (a_2[j], y_j), (a_3[j], z_j)\}$$

Putting all together: The VC instance is:

$K = n + 2m$, $G = (V, E)$ with

$$V = \bigcup_{i=1}^{n} V_i \cup \bigcup_{j=1}^{m} V'_j$$

$$E = \bigcup_{i=1}^{n} E_i \cup \bigcup_{j=1}^{m} E'_j \cup \bigcup_{j=1}^{m} E''_j .$$

The construction is made in polytime and C is satisfiable iff G has a VC of size K. $\blacksquare$

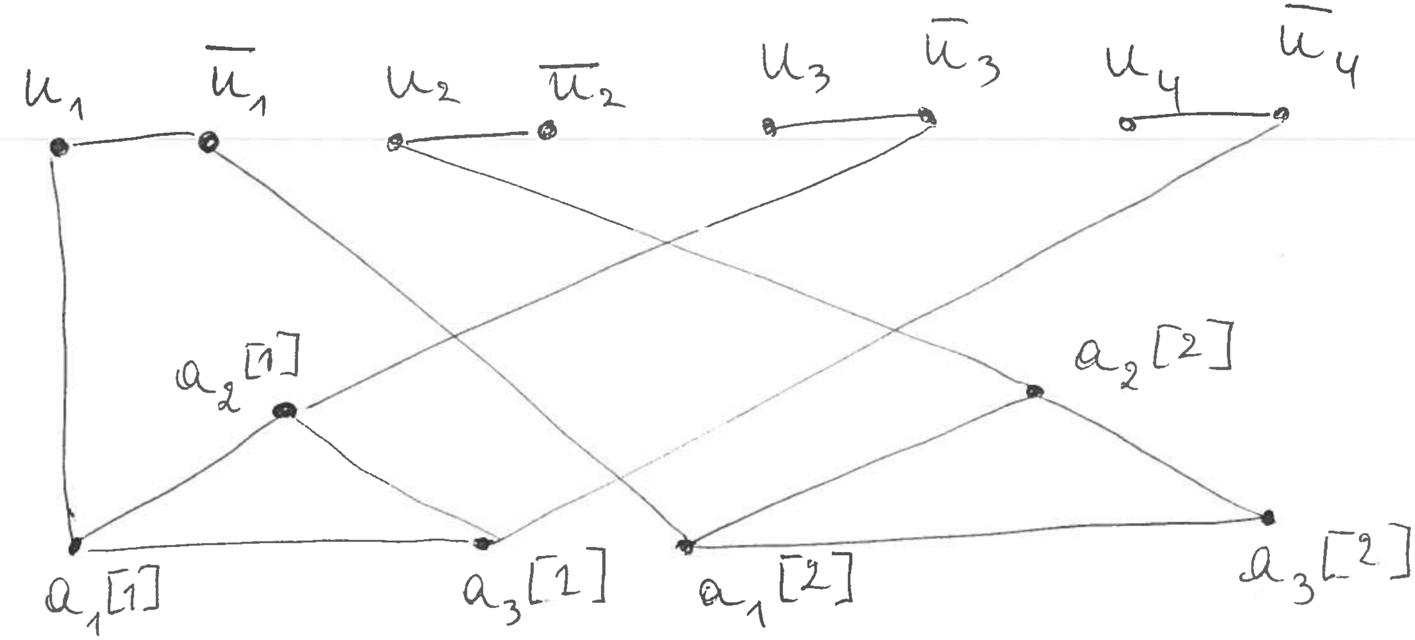
Time Complexity: NP-Completeness (Cont'd)

Sample construction for

$U = \{u_1, \dots u_4\}$, $C = \{\{u_1, \bar{u}_3, \bar{u}_4\}, \{\bar{u}_1, u_2, \bar{u}_4\}\}$

$K = n + 2m = 8$

G is as follows:



Time Complexity: NP-Completeness Techniques

There are several proof techniques for NP-Completeness that have proved to work well in practice.

Restriction

- Most frequently applicable.
- Consists in showing that the problem Π at hand contains a known NP-Complete problem Π' as a special case.
- Crux: specify additional restrictions on the instances of Π s.t. the resulting problem will be identical to Π', up to a one-to-one correspondence.
- Focus on the target problem by eliminating its inessential aspects until Π' appears.

Time Complexity: NP-Completeness Techniques

Theorem 34: HITTING SET is NP-Complete.

Proof:

HITTING SET
INSTANCE: Collection C of subsets of a set S, $K \in \mathbb{Z}^+$
QUESTION: Does S contain a hitting set for C of size K or less; that is, a subset $S' \subseteq S$ with $|S'| \leq K$ s.t. S' contains at least one element from each subset in C?

We restrict to VC by allowing only instances with $|c| = 2$ for all $c \in C$. $\blacksquare$

Time Complexity: NP-Completeness Techniques

Theorem 35: SUBGRAPH ISOMORPHISM is NP complete.

Proof:

SUBGRAPH ISOMORPHISM

INSTANCE: Two graphs $G = (V_1, E_1)$, $H = (V_2, E_2)$.

QUESTION: Does G contain a subgraph isomorphic to H, that is, subsets $V \subseteq V_1$ and $E \subseteq E_1$ s.t. $|V| = |V_2|$, $|E| = |E_2|$, and there is a one-to-one function $f: V_2 \to V$ s.t. $(u,v) \in E_2$ iff $(f(u), f(v)) \in E$?

We restrict to CLIQUE by allowing only instances for which H is a complete graph; i.e., E_2 contains all possible edges joining 2 members of V_2

Time Complexity: NP- Completeness Techniques

Local replacement

- Pick some aspects of the known NP-complete problem instance and make up a collection of <u>basic units</u>.
- Obtain the corresponding instance of the target problem by replacing each basic unit by a different structure (in a uniform way)

Example: SAT ⟶ 3SAT:

- Basic units of SAT are clauses.
- Each clause is replaced by a collection of clauses according to the same general rule.
- The replacement constitutes only a <u>local</u> modification of the structure of SAT.

Time Complexity : NP- Completeness Techniques

Component design

- Use constituents of the target problem to design certain <u>components</u> that will ultimately be combined to produce the known NP- complete problem.
- Two broad classes of components:
  - those for "making choices" (e.g., selecting vertices, truth values, ...)
  - those for "testing properties" (e.g., checking that each edge is covered, that each clause is satisfied, ...)
- The 2 broad classes are joined together through direct connections (e.g., links between truth-setting and satisfaction testing in 3SAT → VC) and global constraints (e.g., bound K in 3SAT → VC).
- Proof of Cook's theorem is a generic example of component design.

Time Complexity: Using NP-Completeness

- How can NP-Completeness be used for analyzing problems? Given a problem Π,
  - Can Π be solved with a polytime algorithm
  - If $\Pi \notin P$, consider NP-Completeness
- Usually, Π will neither obviously have a polytime algorithm nor will it obviously be NP-complete.
- So some effort is needed to determine whether Π is polytime solvable or whether it is NP-complete.
- Notice that, if $P \neq NP$, some problems in NP won't be neither polytime solvable, nor NP-complete.

Time Complexity: Using NP-Completeness (Cont'd)

- Beware: our intuition can be misleading, since many polytime problems differ only slightly from others that are NP-Complete:

EDGE COVER INSTANCE: $G=(V,E)$, $K \in \mathbb{Z}^+$ QUESTION: does G have an edge cover; i.e., an $E' \subseteq E$ s.t. $\lvert E' \rvert \le K$ and for each $v \in V$, there is an $e \in E'$ s.t. $v \in e$?	VERTEX COVER INSTANCE: $G=(V,E)$, $K \in \mathbb{Z}^+$ QUESTION: does G have a vertex cover; i.e., a $V' \subseteq V$ s.t. $\lvert V' \rvert \le K$ and for each $e \in E$ there is a $v \in V'$ s.t. $v \in e$?
SHORTEST PATH INSTANCE: $G=(V,E)$; $l(e) \in \mathbb{Z}^+$ for each e; $a, b \in V$; $B \in \mathbb{Z}^+$ QUESTION: Is there a simple path from a to b with length $\le B$?	LONGEST PATH INSTANCE: $G=(V,E)$; $l(e) \in \mathbb{Z}^+$ for each e; $a, b \in V$; $B \in \mathbb{Z}^+$ QUESTION: Is there a simple path from a to b with length $\ge B$.
P	NP-Complete

Time Complexity : Using NP-Completeness (Cont'd)

- Use 2-sided approach: try to construct an NP-Completeness while trying to discover a polytime algorithm, and move back and forth between both approaches.

- Suppose we succeed in proving NP-Completeness, we can still analyze subproblems of the original ones, and special cases that are polytime solvable.

<u>Definition 34</u> (Subproblem): Suppose a problem $\Pi = (D, Y)$, with domain D and set of yes-instances Y. A problem $\Pi' = (D', Y')$ is a <u>subproblem</u> (or subcase) of Π if $D' \subseteq D$ and $Y' = Y \cap D'$. That is, Π' asks the same question as Π, but only over a subset of the domain of Π.

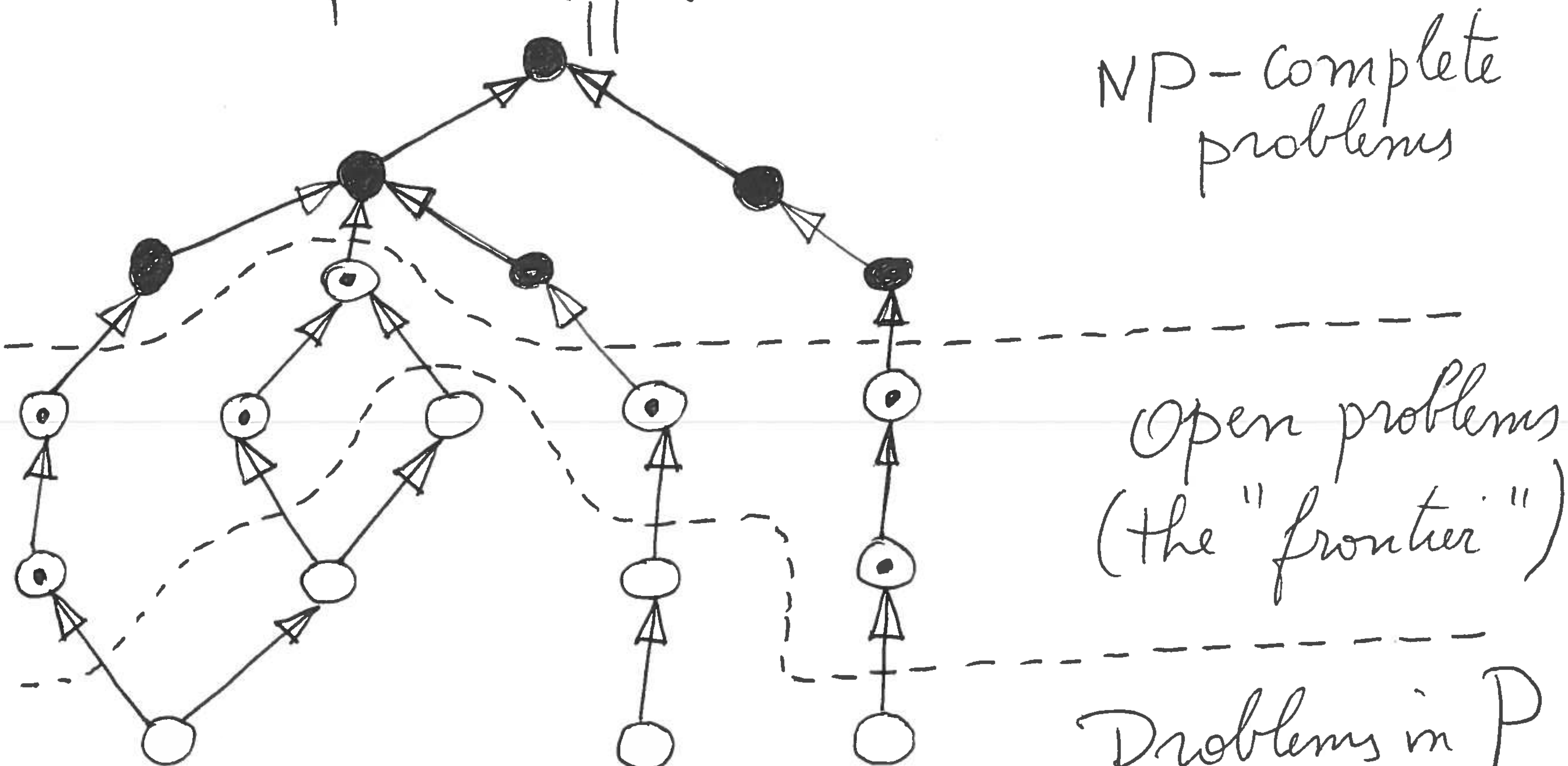
Time Complexity: Using NP-Completeness (Cont'd)

- Subproblems are obtained by imposing additional restrictions on the instances; e.g., require that graphs be
  - planar
  - bipartite
  - acyclic

  etc
- From possible subproblems, do analyze those that occur in practice in applications we have in mind.
- Π might be NP-complete, but each of its subproblems might independently be either NP-complete or polytime solvable.
- There is a "frontier" between problems known to be NP-complete and those known to be polytime solvable.

Time Complexity: Using NP-Completeness (Cont'd)

The state of knowledge about subproblems of Π can be depicted as follows:



Legend: ● — NP-complete
⊙ — open problem
○ — polytime problem

$\Pi' \longrightarrow \Pi$: Π' is a subproblem of Π

Time Complexity: Using NP-Completeness (Cont'd)

Illustration of the use of NP-Completeness on the PRECEDENCE CONSTRAINED SCHEDULING problem:

INSTANCE: Set T of tasks; partial order $\lessdot$ on T; m processors; deadline $D \in \mathbb{Z}^+$.

QUESTION: Is there a schedule

$$\sigma: T \longrightarrow \{0, 1, \cdots, D\}$$

such that for each $i \in \{0, 1, \ldots, D\}$, $|\{t \in T : \sigma(t) = i\}| \leq m$, and whenever $t \lessdot t'$, then $\sigma(t) < \sigma(t')$?

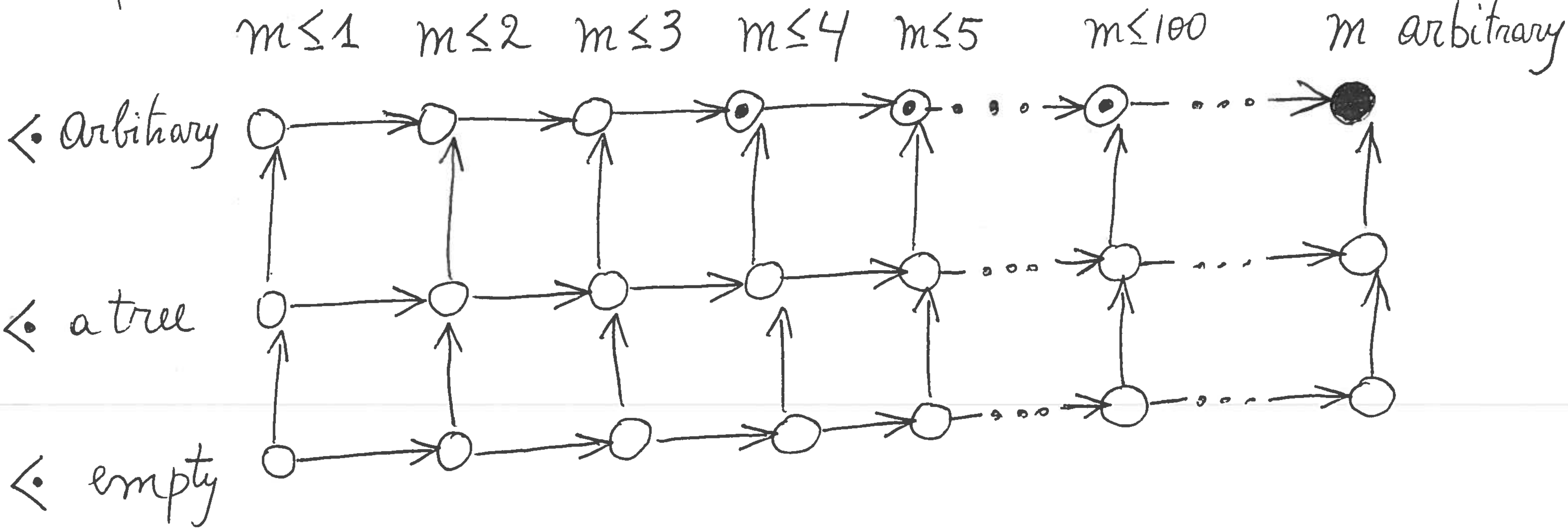
Sample scheduling problem of this sort is the problem of constructing schedules for students:

T: courses; $\lessdot$: prereq-relationship

m: max # of courses taken at one time; D: # of semester expected to graduate.

Time Complexity: Using NP-Completeness (Cont'd)

Current state of knowledge about subproblems of PRECEDENCE CONSTRAINED SCHEDULING:



<u>Theorem 37</u> (Ullman [1975]):
PCS is NP-complete.

Time Complexity: Using NP-Completeness (Cont'd)

Techniques for proving NP-completeness of a subproblem:

- Essentially the same as seen so far.
- Only important difference:
  * Use (or try to use) an existing NP-Completeness proof for some generalized version of the subproblem and modify it, or
  * Use the generalized version as a candidate for reduction.
- Restrictions are often used in proofs, as well as local replacements.

Time Complexity: Using NP-Completeness (Cont'd)

Further illustration of use of NP-Completeness in analyzing graph problems:

GRAPH K-COLORABILITY

INSTANCE: $G = (V, E)$

QUESTION: Is G K-colorable, that is, does there exist a function $f: V \to \{1, 2, 3, \ldots, K\}$ such that $f(u) \neq f(v)$ whenever $\{u, v\} \in E$?

<u>Theorem 38</u>: GRAPH 3-COLORABILITY is NP-complete.

Several restrictions may be considered; e.g.,

- Bounding the maximum vertex <u>degree</u> (the degree of $v \in V$ is the number of edges e s.t. $v \in e$)
- Restriction to <u>planar</u> graphs (G is planar if it can be embedded in the plane s.t. no 2 edges or lines cross over)

Time Complexity: Using NP-Completeness (Cont'd)

	In P for $D \leq$	NP-Complete for $D \geq$
VERTEX COVER	2	3
HAMILTONIAN CIRCUIT	2	3
GRAPH 3-COLORABILITY	3	4

- Above graph problems are degree-limited NP-complete.
- Local replacements are used in the NP-completeness proofs of the above.

Time Complexity; Using NP-Completeness (Cont'd)

Theorem 39: GRAPH 3-COLORABILITY with no vertex degree exceeding 4 is NP-complete.

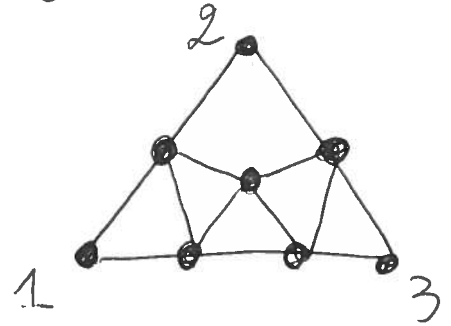
Proof: Suppose $G = (V, E)$ is an arbitrary instance of the general 3-COLORABILITY problem.

> Build a corresponding graph $G' = (V', E')$ s.t. G' has no vertex degree exceeding 4 and is 3-colorable iff G' is 3-colorable.

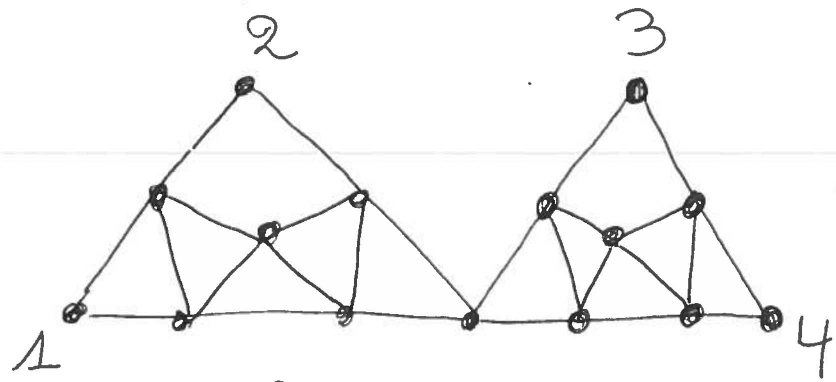
The proof uses local replacement in the form of vertex substitutes (similar to "clause substitutes" used in 3SAT).

Time Complexity: Using NP-Completeness (Cont'd)

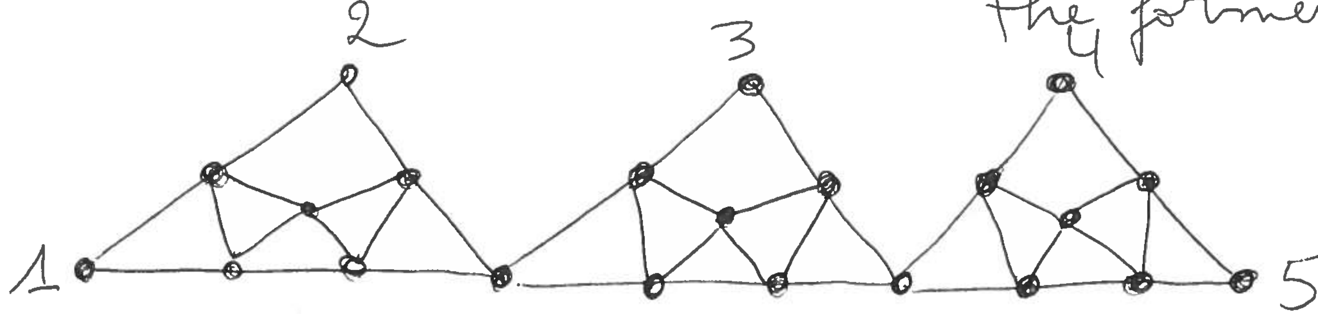
The following 8-vertex graph H_3 is used to construct vertex substitutes:



H_3 has 3 outlets, labeled 1, 2, 3. H_3 is used as building block to construct H_k for $k \geq 4$.



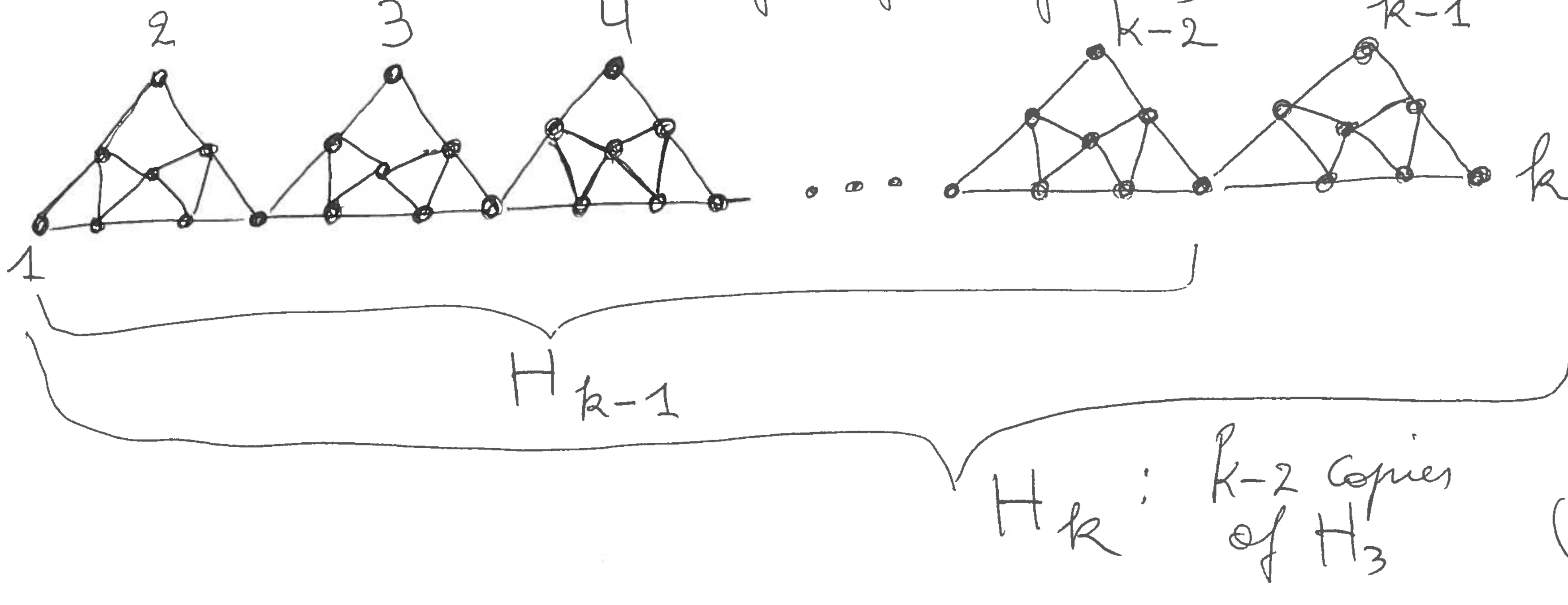
H_4 is formed by adjoining to H_3 a copy of H_3 where outlet 1 of the later coincides with outlet 3 of the former.



H_5 is formed from 3 copies of H_3

Time Complexity: Using NP-Completeness (Cont'd)

- For $k \geq 4$, form the k-outlet vertex substitute by adjoining to H_{k-1} a copy of H_3 where outlet 1 of H_3 coincides with outlet $k-1$ of H_{k-1}.
- Outlet vertices of H_k have degree two.
- New labeling of H_k as follows:
  - Outlets that originally belonged to H_{k-1} retain their labels
  - Second outlet of adjoining H_3 becomes outlet $k-1$
  - Third outlet of adjoining H_3 becomes outlet k



Time Complexity: Using NP-Completeness (Cont'd)

The vertex substitute H_k has the following properties:

(1) H_k has $7(k-2)+1$ vertices, including k outlets

(2) No vertex of H_k has degree exceeding 4

(3) Each outlet of H_k has degree 2

(4) H_k is 3-colorable, but not 2-colorable; and every 3-coloring assigns the same color to all k outlets.

Suppose $v_1, v_2, \ldots, v_r$ are the r vertices with degree over 4 in G. Then construct a sequence

$$G = G_0, G_1, G_2, \ldots, G_r = G'$$

as follows: (See next page)

Time Complexity: Using NP-Completeness (Cont'd)

- Each G_i, $1 \leq i \leq n$, is build from G_{i-1}
- Suppose d is the degree of v_i in G_{i-1} and $(u_1, v_i), (u_2, v_i), \ldots, (u_d, v_i)$ are the edges adjacent to v_i. Then form G_i as follows:
  - delete vertex v_i from G_{i-1}
  - replace v_i with a copy of H_d
  - replace each edge (u_j, v_i), $1 \leq j \leq d$, by an edge joining u_j to outlet j

From the construction and properties (1) – (4), it follows that, for $0 \leq k \leq n$, the following holds:
G_k has $n-k$ vertices of degree over 4 and
G_k is 3-colorable iff G is 3-colorable.
Hence $G' = G_n$ is exactly fulfiling our goal.

Time Complexity: Using NP-Completeness (Contd)

Lessons learnt from the proof of Theorem 39:

- Different vertex substitutes are required for different problems
- Finding a vertex substitute for a given problem requires lots of <u>ingenuity</u>
- Each vertex substitute has its own set of the necessary properties it must satisfy.

As for planarity, many graph problems remain NP-complete when restricted to planar graphs. Two proof techniques are common here:

- Use a transformation (reduction) from another known NP-complete problem for planar graph (the reduction must preserve planarity)
- Use local replacement ("cross-over") for edge crossings.

Time Complexity; NP-Hardness

<u>Definition 35</u> (NP-Hardness):
Suppose L is a language. Then L is NP-hard if it satisfies only condition (2) of Definition 33, that is, every $L' \in NP$ is polytime reducible to L.

- Intuitively, NP-hardness means that, in a sense, the problem is at least as hard as the NP-complete problems, even though it may not belong to NP itself.

Space Complexity: Our First Step Beyond NP

- A problem may be decidable without, however, being practical, since the space spent to find the solution is prohibitive.
- The space requirement is as important as the time requirement
- In a TM, the space requirement is the number of distinct tape squares used by the RW-head before the TM halts.
- Usual questions:
  - How is space measured?
  - How to classify problem w.r.t space used?
  - How to determine class membership?

Space Complexity: Measuring Complexity

Definition 36 (Space Complexity Function):
For a given DTM M that halts on all inputs $w \in \Sigma^*$, the space complexity of M is the function $S_M : \mathbb{N} \longrightarrow \mathbb{N}$, where $S_M(n)$ is the maximum number of tape cells that M scans on any input of length n; That is, S_M is given by

$$S_M(n) = \max\{ s_M(w) \mid w \in \Sigma^* \text{ and } |w| = n \}.$$

Here $s_M(w)$ is the number of squares required by M to halt on any input w.

Definition 37 (Space Complexity Class):
Suppose $s : \mathbb{N} \longrightarrow \mathbb{R}^+$ is a function. Then the space complexity class SPACE $(s(n))$ is the set of languages decidable by a DTM in space $O(s(n))$.

Space Complexity: Savitch's Theorem

NSPACE $(s(n))$ has a definition similar to SPACE$(s(n))$.

Definition 38 (The classes PSPACE and NPSPACE):

We set

$$\text{PSPACE} = \bigcup_{k>0} \text{SPACE}(n^k)$$

$$\text{NPSPACE} = \bigcup_{k>0} \text{NPSPACE}(n^k).$$

Theorem 39 (Savitch's Theorem):

For any function $f: \mathbb{N} \longrightarrow \mathbb{R}^+$ with $f(n) \geq n$,
$\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$.
That is, any NDTM that uses $f(n)$ space can be converted to a DTM that uses only $f^2(n)$ space (Note that such a simulation requires time $2^{f(n)}$ though!).

Space Complexity : Savitch's Theorem (Cont'd)

Proof (of Savitch's Theorem):

Idea: Instead of using a naive depth-first search which takes $2^{O(f(n))}$ space, the proof use the yieldability problem:

> Given: Configurations C_1 and C_2 of NDTM N, $t \in \mathbb{N}$.
> Test : Can N get from C_1 to C_2 within t steps?

for C_1 = start configuration

C_2 = accepting configuration

t = max number of steps N can use to reach C_2.

Solve this yieldability problem deterministically as follows:

> - Find configuration C_m between C_1 and C_2
> - Recursively test whether $C_1 \vdash^* C_m$ and $C_m \vdash^* C_2$, both in $t/2$ steps.
> - Reuse space for each of the 2 recursive calls.

- Each recursion level uses $O(f(n))$ space to store a configuration.
- Recursion depth is $\log t$.
- Initially $t = 2^{O(f(n))} \Longrightarrow \log t = O(f(n)) \Longrightarrow$ Simulation uses $O(f^2(n))$ ■

Space Complexity : PSPACE-Completeness

<u>Definition 39</u> (PSPACE-Completeness):

A language L is PSPACE-Complete if it satisfies the following

(1) $L \in$ PSPACE, and

(2) any $L \in$ PSPACE is polytime reducible to L.

L is said to be PSPACE-hard if only (2) is satisfied.

<u>Definition 40</u> (Quantified Boolean Formulas - QBF):

INSTANCE: A well-formed QBF
$$\phi = (Q_1 x_1)(Q_2 x_2)\ldots(Q_n x_n)\, E$$
where E is a Boolean expression mentioning free variable $x_1, \ldots, x_m$ and each Q_i is a quantifier $\exists$ or $\forall$.

QUESTION: Is ϕ true?

As language problem
$$QBF = \{\phi \mid \phi \text{ is a true QBF}\}$$

Space Complexity: PSPACE - Completeness (Cont'd)

Theorem 40: QBF is PSPACE-Complete

Proof (Idea):

Membership in PSPACE: Check whether ϕ is true is done by going through all the 2^n truth assignments and evaluating E for each of those 2^n possibilities. Recording the current truth assignment, testing the truth value of E, bookkeeping test progress is doable in $O(n)$.

Hardness: An analogue of Cook's theorem construction is used by simulating a polyspace bounded computation. That is, a language L decided by a DTM M in space n^k is reduced to QBF as follows:

$$\boxed{\begin{array}{l}\text{Build QBF } \phi_w \text{ for } w \in L \text{ s.t.}\\ \phi_w \text{ is true iff } M \text{ accepts } w\end{array}}$$

Details in [Stockmeyer & Meyer 1973]

Space Complexity: PSPACE-Completeness (Cont'd)

- The role of "basic" PSPACE-Complete problem from which many reductions have been done is played by QUANTIFIED 3SAT [Stockmeyer & Meyer 1973]
- Combinatorial games are a rich source of PSPACE-Complete problems. That is, winning strategies for games are generally PSPACE-complete problems.
- Games correspond to QBFs: Let

$$\phi = \exists x_1 \forall x_2 \exists x_3 \cdots Q x_k \; \psi \quad \in QBF.$$

Q is either $\forall$ or $\exists$.

Game associated with ϕ:

- Players: A, E take turns in selecting values for $x_1, \ldots, x_k$.
- A selects values for $\forall$-variables;
- E selects values for $\exists$-variables.
- Order of play follows order of quantifiers.
- Use values selected by A and E to set truth value of ψ.
- E wins if ψ is true; A wins if ψ is false.

Space Complexity: DSPACE-Completeness (cont'd)

- Intuitively, read X_i as "move from position P_{i-1} to P_i":

  $\exists$ move for E from P_0 to a position P_1 s.t.,
  $\forall$ moves of A from P_1 to a position P_2,
  $\exists$ move for E from P_2 to a position P_3 s.t.,
  $\forall$ moves of A from P_3 to a position P_4
  $\vdots$
  $\exists$ move for E from P_{n-2} to a position P_{n-1} s.t.
  $\forall$ moves of A from P_{n-1} to a position P_n,
  Position P_n is a win for $\exists$.

  This formula states that $\exists$ as first player has forced a win in n even moves.

- Asking whether this formula is true corresponds to asking the following:

  $\boxed{\text{Given an initial position } P_0 \text{ of a game, does E have a forced win?}}$

Space Complexity: PSPACE-Completeness (Cont'd)

- Geography game:
  - E and A take turns in naming cities of the world.
  - Each city must begin with the same letter of the end of the previous one; no repetition allowed.
  - The start city is randomly picked.
  - Any player who can't continue looses.

- <u>Definition 41</u> (Winning Strategy):
  A player has a winning strategy for a game if that player wins when both players play optimally.

GENERALIZED GEOGRAPHY (GG)
INSTANCE an arbitrary digraph $G = (V, E)$, $s \in V$.
QUESTION Does E have a winning strategy?

$GG = \{\langle G, s\rangle \mid E$ has a winning strategy for GG starting at $s\}$

<u>Theorem 41</u>: GG is PSPACE-complete.

<u>Proof</u>: Reduction from QBF where the quantifiers are alternating $\exists$ and $\forall$ ▨

Space Complexity: Classes L and NL

<u>Definition 42</u> (The Classes L and NL):

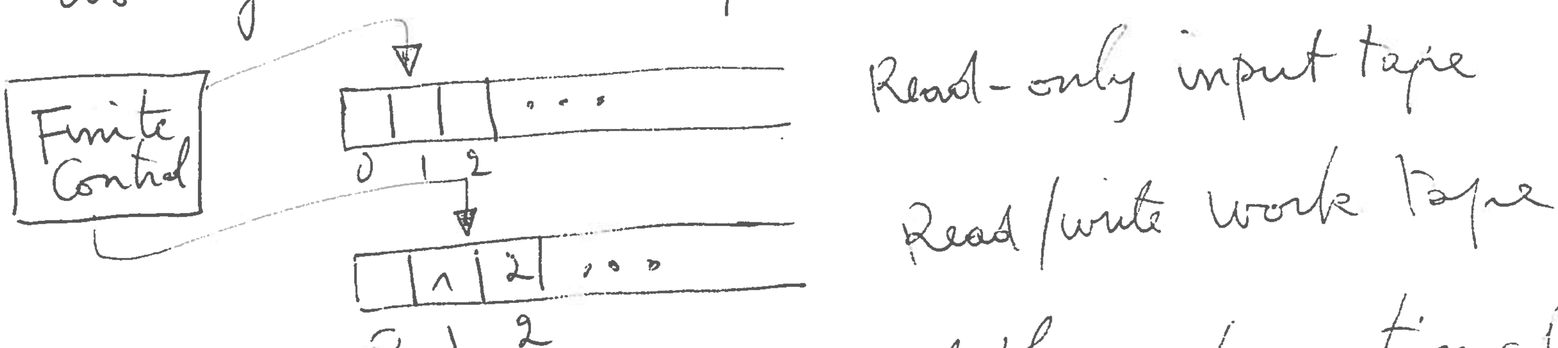
We set

$$L = \mathrm{SPACE}(\log n) \quad \text{(also called LOGSPACE)}$$
$$NL = \mathrm{NSPACE}(\log n) \quad \text{(also called NLOGSPACE)}$$

That is, L is the class of languages decidable in logarithmic space on a DTM, and NL is the class of those decidable in logspace on a NDTM.

logspace is a <u>sublinear</u> bound.

DTM used for sublinear space bounds:



Any input w is not part of the configuration of this DTM.

Space Complexity: L and NL (Cont'd)

<u>Definition 43</u> (Configuration Revisited):
Suppose M is a DTM with a separate read-only input tape, a <u>configuration</u> of M on input w is a setting of the current state, the current content of the <u>work</u> tape, and the current locations of the two heads of M.

<u>Lemma 42</u>: If M runs in $f(n)$ space and $|w| = n$, then M has $\underline{n\, 2^{O(f(n))}}$ configurations.

Savitch's Theorem still holds for $f(n) \geq \log n$.

Space Complexity: NL – Completeness

We have a notion of class completeness for NL which requires a new kind of reducibility:

<u>Definition 44</u> (Log-space Reducibility):
Suppose L_1, L_2 are languages over alphabets Σ_1 and Σ_2, respectively. Then a function $f: \Sigma_1^* \to \Sigma_2^*$ is a log-space reduction from L_1 to L_2 if

(1) f can be computed by a DTM in logarithmic space (i.e. using space bounded by $O(\log n)$) and

(2) $w \in L_1$ iff $f(w) \in L_2$.

<u>Definition 45</u> (NL-Completeness):
Suppose B is a language. Then B is NL-complete if

(1) $B \in NL$, and

(2) any $A \in NL$ is log-space reducible to B; i.e. for any $A \in NL$, $A \leq_{\log} B$.

Space Complexity: NL-Completeness (Cont'd)

<u>Theorem 45</u>: Suppose $L_1 \leq_{log} L_2$ and $L_2 \leq_{log} L_3$. Then

(1) $L_1 \leq_{log} L_3$

(2) $L_2 \in \text{LOGSPACE} \Rightarrow L_1 \in \text{LOGSPACE}$.

<u>Proof</u>: (See [Stockmeyer & Meyer 1973]).

<u>Theorem 46</u>: Let $\text{PATH} = \{\langle G, s, t\rangle : G$ is a digraph with a direct path from s to $t\}$.

PATH is NL-Complete.

<u>Proof</u>:

<u>Membership</u>: The DTM M starts at s and nondeterministically guesses nodes on the path from s to t. M only records position of current node c, not the entire path so far! M goes to the next node selected among all $v \in V_G$ s.t. $\langle c, v\rangle \in E_G$. If M reaches t, M accepts; if M has gone more than $|V_G|$ nodes, M rejects.

Space Complexity: NL - Completeness (Cont'd)

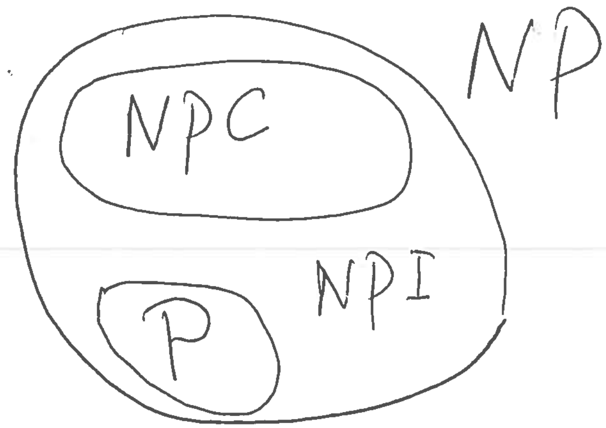
Proof of Theorem 46 (Cont'd)

NL - Completeness (outline):

Given $L \in$ LOGSPACE.
Construct a graph G_L representing the computation
of the nondeterministic log-space TM for L.
Let $w \in L$. Then nodes of G_L correspond to the
configurations of the NDTM N for L on input w.
$(v, v') \in E_G$ iff $v \vdash v'$.
M accepts w iff there is a path from the
start configuration to the accepting configuration

Beyond NP-completeness: Structure of NP

- Are there problems in NP that are not NP-Complete?
- Are there problems that are <u>harder</u> than NP-complete problems?
- Assuming $P \neq NP$, NP looks like this:



NPC $\doteq$ NP-Complete

NPI $\doteq$ NP $-$ (P $\cup$ NPC)

<u>Theorem 47</u> ([Ladner 1975]): Suppose $L \notin P$ is recursive. Then there exists a polytime recognizable language $L' \in P$ s.t. $L'' = L \cap L'$ and $L'' \notin P$, $L'' \leq L$ and $L \not\leq L''$.

Structure of NP (Cont'd)

It follows from Theorem 47 that $NPI \neq \phi$:

<u>Corollary 48</u>: The class NPI is not empty.

<u>Proof</u>:
Suppose $L \in NPC$. If $P \neq NP$, then $L \notin P$.
Therefore the hypothesis of Theorem 47 holds.
Also, because $L \in NP$ and $L' \in P$, we have: $L'' \in NP$.
By Theorem 47, it follows that $L \not\leq L''$ and,
henceforth, $L'' \notin NPC$; and, since $L'' \notin P$,
it follows that $L'' \in NPI$ ∎

Intuitively, Therom 47 and Corollary 48 means that there are classes of instances such that an NP-complete problem, when restricted to ~~those~~ classes, is neither NP-complete nor in P, provided that $P \neq NP$.

Structure of NP (Cont'd)

- Are there any natural problems that are member of NPI?
- though Theorem 47 can help in finding candidates for membership in NPI, those candidates seem "unnatural".
- Open problems believed to be candidates:

GRAPH ISOMORPHISM
INSTANCE: Graphs $G=(V,E)$, $G'=(V',E')$.
QUESTION: Are G and G' isomorphic, that is, is there a one-to-one function $f: V \to V$ s.t. $(u,v) \in E$ iff $(f(u), f(v)) \in E$?

COMPOSITE NUMBERS
INSTANCE: Positive integer K.
QUESTION: Are there $m, n \in \mathbb{Z}^+$, $m,n > 1$ s.t. $K = m \cdot n$?

Structure of NP (Cont'd)

- The argument for the candidacy of GRAPH ISOMORPHISM in NPI stems from the fact that it lacks the kind of redundancy exhibited by all NP-Complete problems, but yet has (so far) no known polytime algorithm.

<u>Definition 46</u>

$$\text{CO-NP} = \{\Pi^c ; \Pi \in NP\}$$
$$= \{\Sigma^* - L ; L \subseteq \Sigma^* \text{ and } L \in NP\}$$

where $\Pi^c = (D, Y_{\Pi^c})$, $Y_{\Pi^c} = D_\Pi - Y_\Pi$

<u>Conjecture</u>: $NP \neq \text{co-}NP$.

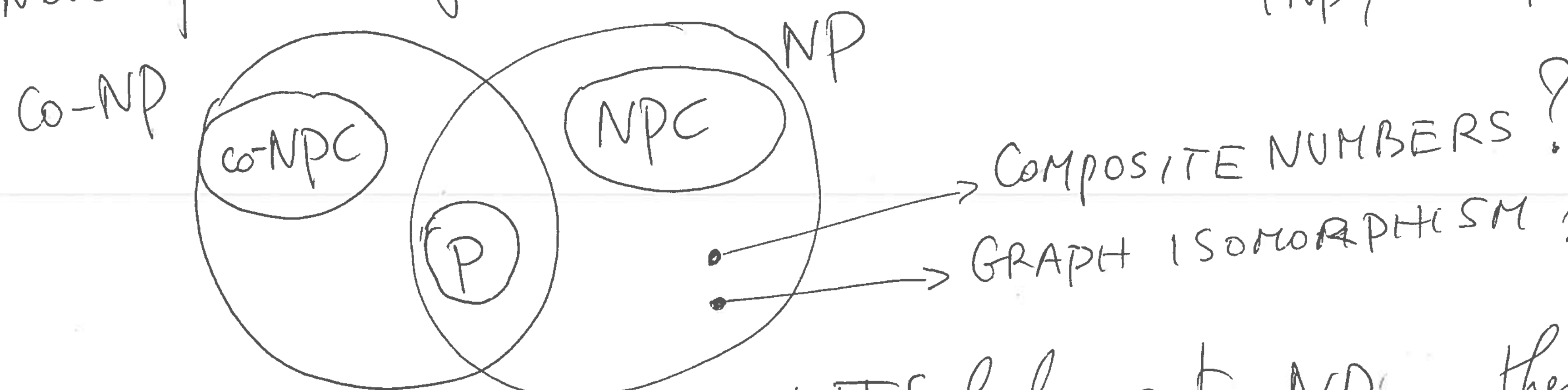
<u>Reason</u>: many problems in co-NP don't seem to be in NP.

Structure of NP (Cont'd)

<u>Theorem 49</u>: If there is a problem $\Pi \in NPC$ s.t. $\Pi^c \in NP$, then $NP = co\text{-}NP$. ■

<u>Proof</u>: By concatenation of TM programs ■

New picture of the world of NP, assuming $\begin{cases} P \neq NP \\ NP \neq co\text{-}NP \end{cases}$



Per Theorem 49, if Π and Π^c belong to NP, then $\Pi \notin NPC$, unless $NP = co\text{-}NP$.
COMPOSITE NUMBERS has this property! It belongs to NP, as does its complement, PRIME.
So this problem can't belong to NPC, unless $NP = co\text{-}NP$.

Beyond NP-Completeness: The Polynomial Hierarchy

- Problems in NPI seem to be "easier" than those in NPC. What about problems "harder" than those in NPC, without being in PSPACE?

Definition 47 (Alternating TMs)
An alternating TM is a NDTM with extra features:
- States (except q_a and q_r) are divided into universal states and existential states.
- Each node of the computation tree is labeled with $\wedge$ or $\vee$ if the corresponding configuration contains a universal or an existential state.
- A node is an accepting one if it is labeled with $\wedge$ and all its children are accepting or if it is labeled with $\vee$ and none of none of its children are accepting.

Polynomial Hierarchy (Cont'd)

Definition 48
Let $i \in \mathbb{N}$. A Σ_i-alternating TM is an alternating TM which contains at most i runs of universal or existential steps, starting with existential steps.
A Π_i-alternating TM is similar, except it starts with universal steps.

Definition 49

$\Sigma_i \mathrm{TIME}(f(n)) = \{ L \mid L$ is decided by an $O(f(n))$ time Σ_i-ATM $\}$

$\Pi_i \mathrm{TIME}(f(n)) = \{ L \mid L$ decided by an $O(f(n))$ Π_i-ATM $\}$

$\Sigma_i \mathrm{SPACE}(f(n))$ and $\Pi_i \mathrm{SPACE}$ have similar definitions.

Polynomial Hierarchy (Cont'd)

Definition 50 (Polynomial Time Hierarchy)
The polytime hierarchy is the collection of classes

$$\Sigma_i^P = \bigcup_k \Sigma_i \, \mathrm{TIME}(n^k)$$

$$\Pi_i^P = \bigcup_k \Pi_i \, \mathrm{TIME}(n^k)$$

$$PH = \bigcup_i \Sigma_i^P = \bigcup_i \Pi_i^P.$$

In fact $NP = \Sigma_1^P$ and $co\text{-}NP = \Pi_1^P$.

Polynomial Hierarchy (Cont'd)

Let C be a class of languages. Then we define:

$P^C = \{L \mid$ There is a language $L' \in C$ s.t. $L \leq_T L'\}$.

$NP^C = \{L \mid$ there is a language $L' \in C$ s.t. there is a polytime non-deterministic reduction from L to $L'\}$

Note: P^{NP} is the class of all NP-easy languages.

Note: In the notation Σ_k^P, Π_k^P, and Δ_k^P, 'p' is used just to distinguish this hierarchy from Kleene's arithmetical hierarchy.

Polynomial Hierarchy (Cont'd)

$$\Sigma_0^P = \Pi_0^P = P = \Delta_0^P$$

$$\Sigma_1^P = NP \qquad \Pi_1^P = co\text{-}NP$$

$$\Sigma_2^P = NP^{NP} \qquad \Pi_2^P = co\text{-}NP^{co\text{-}NP}$$

for all $k \geq 0$:

$$\Delta_{k+1}^P = P^{\Sigma_k^P}$$

$$\Sigma_{k+1}^P = NP^{\Sigma_k^P}$$

$$\Pi_{k+1}^P = co\text{-}\Sigma_{k+1}^P$$