Computational Complexity

Ryan O'Donnell

June 28, 2021

Contents

1 Notational conventions

Use $\llcorner x \lrcorner$ to denote some canonical binary representation of the object x.
The length of a string x is denoted by $|x|$

2 Lecture

2.1 Lecture 1

TIME($t(n)$) =all languages L decidable in $O(t(n))$ steps on input of length n
Language $L(\Sigma^*) \equiv$ decision problem -> yes/no problem

Language L	decision problem	$f : \{0,1\}^* \to \{0,1\}$
Σ^*	yes/no problem	$x \in L \Leftrightarrow f(x) = 1$
decide if a string is in language		

Model: multitape Turing machine (TM)
Time Hierarchy Theorem (More time = more power to decide a language)
$\Rightarrow$ TIME(n^2) $\subsetneq$ TIME(n^3)
P = TIME($poly(n)$), **EXP** = TIME($2^{poly(n)}$),**E** = TIME($2^{O(n)}$)
$\Rightarrow$ **P** $\subsetneq$ **E** $\subsetneq$ **EXP**
SPACE($s(n)$) = langs decidable using tape cells $\subseteq O(s(n))$
TIME($f(n)$) $\subseteq$ **SPACE**($f(n)$)
(each operation takes a cell)
PSPACE = **SPACE**($poly(n)$)
P $\subseteq$ **PSPACE**
SPACE($f(n)$) $\subseteq$ TIME($2^{O(f(n))}$) (since only $O(f(n))$ possible states if each state is different)
L = **SPACE**($\log n$)
L $\subseteq$ **P** $\subseteq$ **PSPACE** $\subseteq$ **EXP**

Theorem 2.1 (HPV77)**.** *TIME*($t(n)$) $\subseteq$ ***SPACE***($\frac{t(n)}{\log t(n)}$) $\subsetneq$ ***SPACE***($t(n)$)

Space is more valuable than time.
NP = **NTIME**($poly(n)$) computed by nondeterministic multitape turing machine
NE = **NTIME**($2^{O(n)}$)
Circuits - "Non-uniform" model
"Non-uniform" different algorithms for different input length
P /*poly*=langs decidable by $poly(n)$-size circuit familys
P $\subsetneq$ **P** /*poly*
if **NP** $\neq$ **P** then **NP** $\not\subseteq$ **P**/*poly*
which is equivalent to `SAT` not decidable by $poly(n)$-size circuits

Theorem 2.2. *there exists language with no poly-size constant-depth circuits* (*actually in* ***P***)

Theorem 2.3. *CLIQUE requires exponential size AND/OR circuits*

Theorem 2.4. *There exists a language $L \in$ **P** requiring circuit families of size $\geq 3n$*

Theorem 2.5 (Santhanam theorem)**.** *for all c there exists L s.t. L is not computable by $O(n^c)$-size circuit*

Theorem 2.6 (William's Theorem)**.** $\exists L \in$ ***NEXP*** *is not computable by $AC^0[6]$ (constant depth, poly(n) size, also get* $\mod 6$ *gates)*

Randomness
BPP = langs decidable in *poly*(n)-time using randomness
P $\subseteq$ **BPP** $\subseteq$ **EXP**
PIT = "polynomial identity testing" that are in **BPP**, but we don't know if they are in **P**
Hardness vs Randomness Paradigm

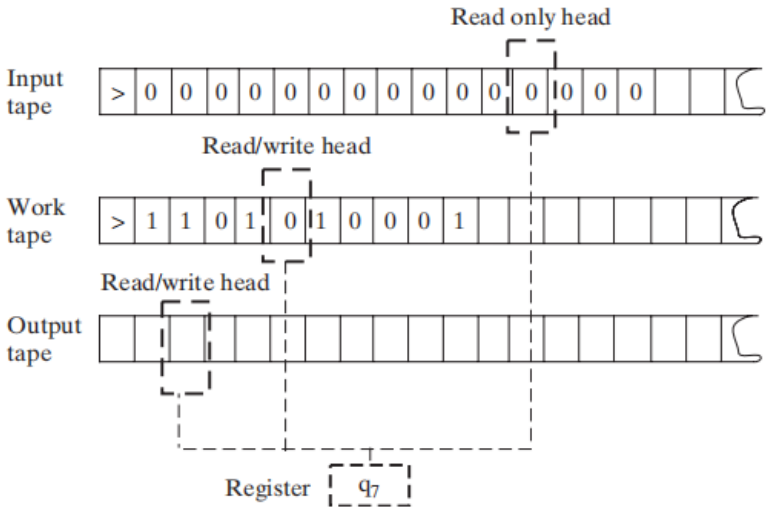
3 Book

3.1 Chapter 1: The computational model

3.1.1 Efficiency and Running Time

Definition 3.1. A TM M is described by a tuple (Γ, Q, δ) containing

- A finite set Γ of the symbols that M's tapes can contain. We assume that Γ contains a designated "blank" symbol, denoted $\square$; a designated "start" symbol, denoted $\triangleright$; and the numbers 0 and 1. We call Γ the **alphabet** of M
- A finite set Q of possible states M' register can be in. We assume that Q contains a designated start state, denoted q_{start}, and a designated halting state, denoted q_{halt}
- A function $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\text{L,S,R}\}^k$, where $k \geq 2$, describing the rules M use in performing each step. This function is called the **transition function** of M



Definition 3.2 (Computing a function and running time)**.** Let $f : \{0,1\}^* \to \{0,1\}$ and let $T : \mathbb{N} \to \mathbb{N}$ be some functions, and let M be a Turing machine. We say that M **computes** f if for every $x \in \{0,1\}^*$, whenever M is initialized to the start configuration on input x, then it halts with $f(x)$ written on its output tape. We say M **computes** f **in** $T(n)$**-time** if its computation on every input x requires at most $T(|x|)$ steps

A function $T : \mathbb{N} \to \mathbb{N}$ is **time constructible** if $T(n) \geq n$ and there is a TM M that computes the function $x \mapsto \llcorner T(|x|) \lrcorner$ in time $T(n)$. ($\llcorner T(|x|) \lrcorner$ denotes the binary representation of the number $T(|x|)$). The restriction $T(n) \geq n$ is to allow the algorithm time to read its input.

Proposition 3.3. *For every $f : \{0,1\}^* \to \{0,1\}$ and a time-constructible $bT : \mathbb{N} \to \mathbb{N}$, if f is computable in time $T(n)$ by a TM M using alphabet Γ, then it's computable in time $4\log|\Gamma|T(n)$ by a TM M using the alphabet $\{0,1,\square,\triangleright\}$.*

Proof. Let M be a TM with alphabet Γ, k tapes and state set Q that computes the function f in $T(n)$ times. We describe an equivalent TM $\tilde{M}$ computing f with alphabet $\{0,1,\square,\triangleright\}$, k tapes and a set Q' of states.

One can encode any member of Γ using $\log|\Gamma|$ bits. Thus each of $\tilde{M}$'s work tapes will simply encode one of M's tapes: For every cell in M's tape we will have $\log|\Gamma|$ cells in the corresponding tape of $\tilde{M}$

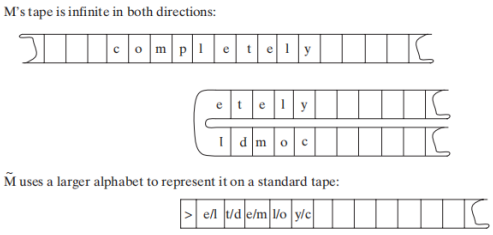
To simulate one step of M, the machine $\tilde{M}$ will 1. use $\log|\Gamma|$ steps to read from each tape the $\log|\Gamma|$ bits encoding of a symbol of Γ 2. use its state register to store the symbols read 3. use M's transition function to compute the symbols M writes and M's new state given this information 4. store this information in its state register 5. use $\log|\Gamma|$ steps to write the encodings of these symbols on its tapes $\square$

Proposition 3.4. *Define a single-tape Turing machine to be a TM that has only one read-write tape. For every $f : \{0,1\}^* \to \{0,1\}$ and time-constructible $T : \mathbb{N} \to \mathbb{N}$ if f is computable in time $T(n)$ by a TM M using k tapes, then it is computable in time $5kT(n)^2$ by a single-tape TM M*

Proof. The TM $\tilde{M}$ encodes k tapes of M on a single tape by using locations $1, k+1, 2k+1, \ldots$ to encode the first tape, locations $2, k+2, 2k+2, \ldots$ to encode the second tape etc. For every symbol a in M's alphabet, $\tilde{M}$ will contain both the symbol a and the symbol $\hat{a}$. In the encoding of each tape, exactly one symbol will be of the "^ type", indicating that the corresponding head of M is positioned in that location. $\tilde{M}$ will not touch the first $n+1$ locations of its tape (where the input is located) but rather start by taking $O(n^2)$ steps to copy the input bit by bit into the rest of the tape, while encoding it in the above way. □

Remark (Oblivious Turing machines). One can ensure that the proof of Proposition 3.4 yields a TM $\tilde{M}$ with the following property: its head movements do not depend on the input but only depend on the input length. That is, every input $x \in \{0,1\}^*$ and $i \in \mathbb{N}$, the location of each of M's at the ith step of execution on input x is only a function of $|x|$ and i. A machine with this property is called **oblivious**.

Proposition 3.5. *Define a bidirectional TM to be a TM whose tapes are infinite in both directions. For every $f : \{0,1\}^* \to \{0,1\}^*$ and time-constructible $T : \mathbb{N} \to \mathbb{N}$ if f is computable in time $T(n)$ by a directional TM M, then it is computable in time $4T(n)$ by a standard (undirectional) TM $\tilde{M}$*



Proof.

If M uses alphabet Γ, then $\tilde{M}$ will use the alphabet Γ^2 □

3.1.2 Machines as Strings and the Universal Turing Machine

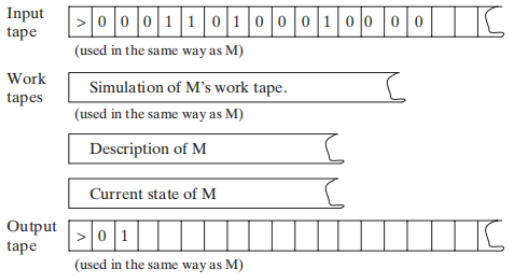
We will also find it convenient to assume that our representation scheme satisfies the following properties:

1. We will also find it convenient to assume that our representation scheme satisfies the following properties:
2. Every TM is represented by infinitely many strings

We denote by $\llcorner M \lrcorner$ the TM M's representation as a binary string. If α is a string then M_α denotes the TM that α represents.

Theorem 3.6 (Efficient universal Turing machine)**.** *There exists a TM $\mathcal{U}$ s.t. for every $x, \alpha \in \{0,1\}^*$, $\mathcal{U}(x,\alpha) = M_\alpha(x)$. Moreover, if M_α halts on input x within T steps then $\mathcal{U}(x,\alpha)$ halts within $CT \log T$ steps, where C is a number independent of $|x|$ and depending only on M_α's alphabet size, number of tapes and number of states.*

Proof of relaxed version of theorem 3.6. We assume M has a single work tape (in addition to the input and output tape) and uses he alphabet $\{\triangleright, \square, 0, 1\}$. The reason is that $\mathcal{U}$ can transform a representation of every TM M into a representation of an equivalent TM $\tilde{M}$ that satisfies these properties. (which my takes $C'T^2$ time)



□

3.1.3 Uncomputablity: An Introduction

Theorem 3.7. *There exists a function $UC : \{0,1\}^* \to \{0,1\}$ that is not computable by any TM*

Proof. For every $\alpha \in \{0,1\}^*$, if $M_\alpha(\alpha) = 1$ then $\mathrm{UC}(\alpha) = 0$; otherwise $\mathrm{UC}(\alpha) = 1$.

If its computable, then there exists a TM M s.t. $M(\alpha) = \mathrm{UC}(\alpha)$, then $M(\llcorner M \lrcorner) = \mathrm{UC}(\llcorner M \lrcorner)$ □

Theorem 3.8. *HALT is not computable by any TM*

3.1.4 The Class P

A **complexity class** is a set of function that can be computed within given resource bounds.

We say that a machine **decides** a language $L \subseteq \{0,1\}^*$ if it computes the function $f_L : \{0,1\}^* \to \{0,1\}$ where $f_L(x) = 1 \Leftrightarrow x \in L$

Definition 3.9. Let $T : \mathbb{N} \to \mathbb{N}$ be some function. A language L is in **DTIME**$(T(n))$ iff there is a Turing machine that runs in $c\dot{T}(n)$ for some constant $c > 0$ and decides L.

The D in **DTIME** refers to "deterministic".

Definition 3.10. $\mathbf{P} = \bigcup_{c \geq 1} \mathbf{DTIME}(n^c)$

3.2 2: NP and NP completeness

3.2.1 The Class NP

Definition 3.11. A language $L \subseteq \{0,1\}^*$ is in **NP** if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time TM M (called the **verifier** for L) s.t. for every $x \in \{0,1\}^*$

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 1$$

If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $M(x,u) = 1$ then we call u a **certificate** for x

Example 3.1 (`INDSET` $\in$ **NP**)**.** By representing the possible invitees to a dinner party with the vertices of a graph having an edge between any two people who don't get along. The dinner party computational problem becomes the problem of finding a maximum sized **independent set** (set of vertices without any common edges) in a given graph. The corresponding language is

$$\texttt{INDSET} = \{\langle G,k\rangle : \exists S \subseteq V(G) \text{ s.t. } |S| \geq k \text{ and } \forall u,v \in S, \overline{uv} \notin E(G)\}$$

Consider the following polynomial-time algorithm M: Given a pair $\langle G,k\rangle$ and a string $u \in \{0,1\}^*$, output 1 iff u encodes a list of k vertices of G s.t. there is no edge between any two members of the list. Note that if n is the number of vertices in G, then a list of k vertices can be encoded using $O(k \log n)$ bits, where n is the number of vertices in G. Thus u is a string of at most $O(n \log n)$ bits, which is polynomial in the size of the representation of G.

Proposition 3.12. *Let* $\boldsymbol{EXP} = \bigcup_{c>1} \boldsymbol{DTIME}(2^{n^c})$*. Then* $\boldsymbol{P} \subseteq \boldsymbol{NP} \subseteq \boldsymbol{EXP}$

Proof. **P** $\subseteq$ **NP**. Suppose $L \in$ **P** is decided in polynomial-time by a TM N. Then we take N as the machine M and make $p(x)$ the zero polynomial

NP $\subseteq$ **EXP**. We can decide L in time $2^{O(p(n))}$ by enumerating all possible n and using M to check whether u is a valid certificate for the input x. Note that $p(n) = O(n^c)$ for some $c > 1$, the number of choices for u is $2^{O(n^c)}$. □

NP stands for **nondeterministic polynomial time**.

NDTM has **two** transition function δ_0 and δ_1, and a special state denoted by q_{accept}. When an NDTM M computes a function, we envision that at each computational step M makes an arbitrary choice at to which of its two transition functions to apply. For every input x, we say that $M(x) = 1$ if there **exists** some sequence of this choices that would make M reach q_{accept} on input x. We say that M runs in $T(n)$ time if for every input $x \in \{0,1\}^*$ and every sequence of nondeterministic choices, M reaches the halting state or q_{accept} within $T(|x|)$ steps

Definition 3.13. For every function $f : \mathbb{N} \to \mathbb{N}$ and $L \subseteq \{0,1\}^*$ we say that $L \in$ **NTIME**$(T(n))$ if there is a constant $c > 0$ and a $c\dot{T}(n)$-time NDTM M s.t. for every $x \in \{0,1\}^*$, $x \in L \Leftrightarrow M(x) = 1$

Theorem 3.14. $\boldsymbol{NP} = \bigcup_{c \in \mathbb{N}} \boldsymbol{NTIME}(n^c)$

Proof. The main idea is that the sequence of nondeterministic choices made by an accepting computation of an NDTM can be viewedas a certificate that the input is in the language, and vice versa

Suppose $p : \mathbb{N} \to \mathbb{N}$ is a polynomial and L is decidable by a NDTM N that runs in time $p(n)$. For every $x \in L$, there is a sequence of nondeterministic choices that makes N reach q_{accept} on input x. We can use this sequence as a certificate for x. This certificate has length $p(|x|)$ and can be verified in polynomial time by a deterministic machine.

Conversely, if $L \in$ **NP**, then we describe a polynomial time NDTM N that decides L. On input x, it uses the ability to make nondeterministic choices to write down a string u of length $p(|x|)$. (Having transition δ_0 correspond to writing a 0 and δ_1). Then it runs the deterministic verifier □

3.2.2 Reducibility and NP-Completeness

Definition 3.15. A language $L \subseteq \{0,1\}^*$ is **polynomial-time Karp reducible to a language** $L' \subseteq \{0,1\}^*$ (sometimes shortened to just "polynomial-time reducible"), denoted by $L \leq_p L'$ if there is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ s.t. for every $x \in \{0,1\}^*$, $x \in L$ iff $f(x) \in L'$

We say that L' is **NP-hard** if $L \leq_p L'$ for every $L \in$ **NP**. We say that L' is **NP-complete** if L' is **NP**-hard and $L' \in$ **NP**

Theorem 3.16.
1. *(Transitivity) If $L \leq_p L'$ and $L' \leq_p L''$ then $L \leq_p L''$*
2. *If language L is **NP**-hard and $L \in \boldsymbol{P}$ then $\boldsymbol{P} = \boldsymbol{NP}$*
3. *If language L is **NP**-complete, then $L \in \boldsymbol{P}$ iff $\boldsymbol{P} = \boldsymbol{NP}$*

Theorem 3.17. *The following language is **NP**-complete*

$$\texttt{TMSAT} = \{\langle \alpha, x, 1^n, 1^t \rangle : \exists u \in \{0,1\}^n \text{ s.t. } M_\alpha \text{ outputs } 1 \text{ on input } \langle x, u \rangle \text{ within } t \text{ steps}\}$$

Proof. There is a polynomial p and a verifier TM M s.t. $x \in L$ iff there is a string $u \in \{0,1\}^{p(|x|)}$ satisfying $M(x,u) = 1$ and M runs in time $q(n)$ for some polynomial q.

Map every string $x \in \{0,1\}^*$ to the tuple $\langle \ulcorner M \urcorner, x, 1^{p(|x|)}, 1^{q(m)}$ where $m = |x| + p(|x|)$ and $\ulcorner M \urcorner$ denotes the representation of M as string.

$$\begin{aligned}
&\langle \ulcorner M \urcorner, x, 1^{p(|x|)}, 1^{q(m)} \rangle \in \texttt{TMSAT} \\
&\Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) \text{ outputs 1 within } q(m) \text{ steps} \\
&\Leftrightarrow x \in L
\end{aligned}$$

□

3.2.3 The Cook-Levin Theorem: Computation is Local

We denote by SAT the language of all satisfiable CNF formulae and by 3SAT the language of all satisfiable 3CNF formulae

Theorem 3.18 (Cook-Levin Theorem)**.** *1. SAT is **NP**-complete*

*2. 3SAT is **NP**-complete*

Lemma 3.19 (Universality of AND, OR, NOT)**.** *For every Boolean function $f : \{0,1\}^l \to \{0,1\}$, there is an l-variable CNF formula φ of size $l2^l$ s.t. $\varphi(u) = f(u)$ for every $u \in \{0,1\}^l$, where the size of a CNF formula is defined to be the number of $\wedge/\vee$ symbols it contains*

Proof. For every $v \in \{0,1\}^l$, there exists a clause $C_v(z_1, \ldots, z_l)$ s.t. $C_v(v) = 0$ and $C_v(u) = 1$ for every $u \neq v$.

We let φ be the AND of all the clauses C_v for v s.t. $f(v) = 0$

$$\varphi = \bigwedge_{v : f(v)=0} C_v(z_1, \ldots, z_l)$$

Note that φ has size at most $l2^l$. □

Lemma 3.20. *SAT is **NP**-hard*

Proof. Let L be an **NP** language. By definition, there is a polynomial time TM M s.t. for every $x \in \{0,1\}^*$, $x \in L \Leftrightarrow M(x,u) = 1$ for some $u \in \{0,1\}^{p(|x|)}$, where $p : \mathbb{N} \to \mathbb{N}$ is some polynomial. We show L is polynomial-time Karp reducible to SAT by describing a polynomial-time transformation $x \to \varphi_x$ from strings to CNF formulae s.t. $x \in L$ iff φ_x is satisfiable. Equivalently

$$\varphi_x \in \texttt{SAT} \quad \text{iff} \quad \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x \circ u) = 1$$

where $\circ$ denotes concatenation

Assume M

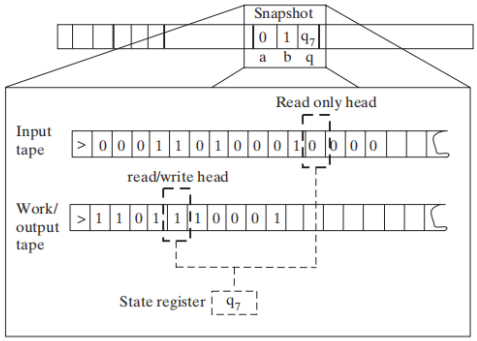
1. M only has two tapes - an input tape and a work/output tape
2. M is an oblivious TM in the sense that its head movement does not depend on the contents of its tapes. That is, M's computation takes the same time for all inputs of size n, and for every i the location of M's head at the ith step depends only on i and the length of the input

We can make these assumptions without loss of generality because for every $T(n)$-time TM M there exists a two-tape oblivious TM $\tilde{M}$ computing the same function in $O(T(n)^2)$. Thus in particular, if $L \in \mathbf{NP}$, then there exists a two-tape oblivious polynomial-time TM M and a polynomial p s.t.

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x \circ u) = 1 \tag{1}$$

Note that because M is oblivious, we can run it on the trivial input $(x, 0^{p(|x|)})$ to determine the precise head position of M during its computation on every other input of the same length.

Denote by Q the set of M's possible states and by Γ its alphabet. The **snapshot** of M's execution on some input y at a particular step i is the triple $\langle a, b, q \rangle \in \Gamma \times \Gamma \times Q$ s.t. a, b are the symbols read by M's heads from the two tapes and q is the state M is in at the ith step. Clearly the snapshot can be encoded as a binary string. Let c denote the length of this string, which is some constant depending upon $|Q|$ and $|\Gamma|$



For every $y \in \{0,1\}^*$, the snapshot of M's execution on input y at the ith step depends on its state in the $(i-1)$st step and the contents of the current cells of its input and work tapes.

Suppose somebody were to claim the existence of some u satisfying $M(x \circ u) = 1$ and as evidence, present you with the sequence of snapshots that arise from M's execution on $x \circ u$. How can you tell that the snapshots present a valid computation that was actually performed by M.

Clearly, it suffices to check that for each $i \leq T(n)$, the snapshot z_i is correct given the snapshot for the previous $i-1$ steps. However, since the TM can only read/modify one bit at a time, to check the correctness of z_i it suffices to look at only *two* of the previous snapshots. Specifically, to check z_i we need to only look at the following: z_{i-1}, $y_{\text{inputpos}(i)}$, $z_{\text{prev}(i)}$.

Here y is a shorthand for $x \circ u$. inputpos(i) denotes the location of M's input tape head at the ith step. prev(i) is the last step before i when M's head was in the same cell on its work tape that it is during step i. The reason this small amount of information suffices to check the correctness of z_i is that the contents of the current cell have not been affected between step prev(i) and step i.

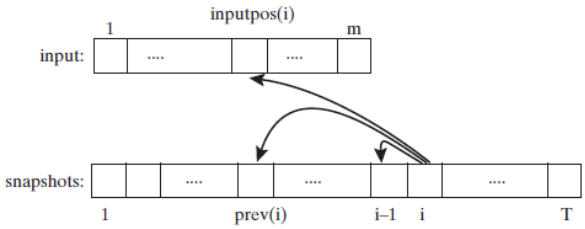


Figure 2.3. The snapshot of M at the ith step depends on its previous state (contained in the snapshot at the $(i-1)$st step), and the symbols read from the input tape, which is in position `inputpos(i)`, and from the work tape, which was last written to in step `prev(i)`.

Since M is a deterministic TM, for every triple of values to $z_{i-1}, y_{\text{inputpos}(i)}, z_{\text{prev}(i)}$, there is at most one value of z_i that is correct. Thus there is some function F that maps $\{0,1\}^{2c+1}$ to $\{0,1\}^c$ s.t. a correct z_i satisfies

$$z_i = F(z_{i-1}, z_{\text{prev}(i)}, y_{\text{inputpos}(i)})$$

Because M is oblivious, the values $\text{inputpos}(i)$ and $\text{prev}(i)$ do not depend on the particular input i. These indices can be computed in polynomial-time by simulating M on a trivial input.

By (1) , $x \in \{0,1\}^n \in L$ iff $M(x \circ u) = 1$ for some $u \in \{0,1\}^{p(n)}$. The previous discussion shows this latter condition occurs iff there exists a string $y \in \{0,1\}^{n+p(n)}$ and a sequence of strings $z_1, \ldots, z_{T(n)} \in \{0,1\}^c$ (where $T(n)$ is the number of steps M takes on inputs of length $n + p(n)$) satisfying the following four conditions

1. The first n bits of y are equal to x
2. The string z_1 encodes the initial snapshot of M. That is, z_1 encodes the triple $\langle \triangleright, \square, q_{\text{start}} \rangle$.
3. For every $i \in \{2, \ldots, T(n)\}$, $z_i = F(z_{i-1}, z_{\text{prev}(i)}, y_{\text{inputpos}(i)})$.
4. The last string $z_{T(n)}$ encodes a snapshot where the machine halts and outputs 1

The formula φ_x will take variables $y \in \{0,1\}^{n+p(n)}$ and $z \in \{0,1\}^{cT(n)}$ and will verify that y, z satisfy the AND of these four conditions. Thus $x \in L \Leftrightarrow \varphi_x \in \texttt{SAT}$.

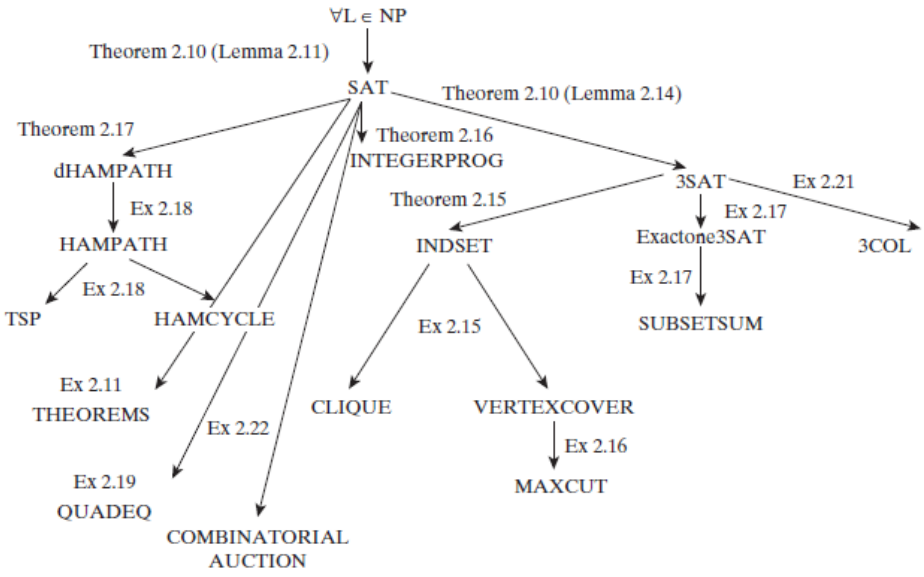
Condition 1 can be expressed as a CNF formula of size $4n$. Conditions 2 and 4 each depend on c variables and hence by Proposition 3.19 can be expressed by CNF formulae of size $c2^c$. Condition 3, which is an AND of $T(n)$ conditions each depending on at most $3c + 1$ variables, can be expressedas a CNF formula of size at most $T(n)(3c + 1)2^{3c+1}$. Hence the AND of all these conditions can be expressed as a CNF formula of size d(n + T(n)) where d is some constant depending only on M. Moreover, this CNF formula can be computedin time polynomial in the running time of M. □

Lemma 3.21. *SAT* $\leq_p$ *3SAT*

Proof. Suppose φ is a 4CNF. Let C be a clause of φ, say $C = u_1 \vee \bar{u}_2 \vee \bar{u}_3 \vee u_4$. We add a new variable z to the φ and replace C with the pair $C_1 = u_1 \vee \bar{u}_2 \vee z$ and $C_2 = \bar{u}_3 \vee u_4 \vee \bar{z}$. If C is true, then there is an assignment to z that satisfies both C_1 and C_2. If C is false, then no matter what value we assign to z either C_1 or C_2 will be false.

For every clause C of size $k > 3$, we change it into an equivalent pair of clauses C_1 of size $k-1$ and C_2 of size 3. □

3.2.4 The Web of Reductions



Theorem 3.22. *INDSET is* ***NP****-complete*

Proof. Transform in polynomial time every m-clause 3CNF formula φ into a $7m$-vertex graph G

We associate a cluster of 7 vertices in G with each clause of φ. The vertices in a cluster associated with a clause C correspond to the seven possible satisfying partial assignments to the three variables on which C depends. For example, if C is $\bar{u}_2 \vee \bar{u}_5 \vee u_7$, then the seven vertices in the cluster associated with C correspond to all partial assignments of the form $u_1 = a, u_2 = b, u_3 = c$ for a binary vector $\langle a, b, c \rangle \neq \langle 1, 1, 0 \rangle$. We put an edge between two vertices of G if they correspond to inconsistent partial assignments. In addition, we put edges between every two vertices that are in the same cluster

φ is satisfiable iff G has an independent set of size m □

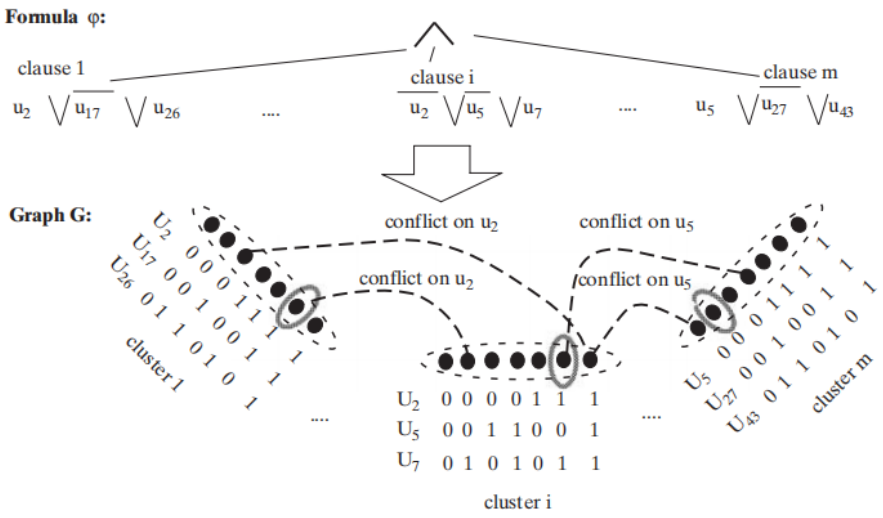


Figure 2.5. We transform a 3*CNF* formula φ with m clauses into a graph G with $7m$ vertices as follows: each clause C is associated with a cluster of 7 vertices corresponding to the 7 possible satisfying assignments to the variables C depends on. We put edges between any two vertices in the same cluster and any two vertices corresponding to *inconsistent* partial assignments. The graph G will have an independent set of size m if and only if φ was satisfiable. The figure above contains only a sample of the edges. The three circled vertices form an independent set.

We let `1/0 IPROG` be the set of satisfiable 0/1 integer programs. That is, a set of linear inequalities with rational coefficients over variables $u_1, \ldots, u_n$ is in `1/0 IPROG` if there is an assignment of numbers in $\{0,1\}$ to $u_1, \ldots, u_n$ that satisfies it

Theorem 3.23. *`1/0 IPROG` is **NP**-complete*

Every CNF formula can be expressed as an integer program by expressing every clause as inequality. For example, the clause $u_1 \vee \bar{u}_2 \vee \bar{u}_3$ can be expressed by $u_1 + (1-u_2) + (1-u_3) \geq 1$.

A **Hamilton path** in a directed graph is a path that visits all vertices exactly once. Let `dHAMPATH` denote the set of all directed graphs that contain such a path

Theorem 3.24. *`dHAMPATH` is **NP**-complete*

Proof. The graph G has

1. m vertices for each of φ's clause $c_1, \ldots, c_m$
2. a special starting vertex v_{start} and ending vertex v_{end}
3. n "chains" of $4m$ vertices corresponding to the n variables of φ . A chain is a set of vertices $v_1, \ldots, v_{4m}$ s.t. for every $i \in [1, 4m-1]$, v_i and v_{i+1} are connected by two edges in both directions

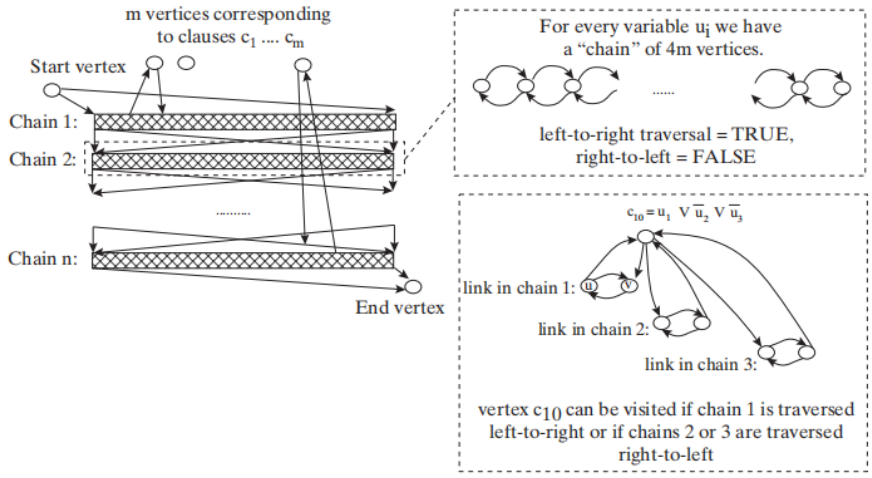


Figure 2.6. Reducing SAT to dHAMPATH. A formula φ with n variables and m clauses is mapped to a graph G that has m vertices corresponding to the clauses and n doubly linked chains, each of length $4m$, corresponding to the variables. Traversing a chain left to right corresponds to setting the variable to True, while traversing it right to left corresponds to setting it to False. Note that in the figure every Hamiltonian path that takes the edge from u to c_{10} must immediately take the edge from c_{10} to v, as otherwise it would get "stuck" the next time it visits v.

If C contains the literal u_j, then we take two neighboring vertices v_i, v_{i+1} in the jth chain and put an edge from v_i to C and from C to v_{i+1}. If C contains the literal $\bar{u}_j$ then we construct these edges in the opposite direction. When adding these edges, we never "reuse" a link v_i, v_{i+1} in a particular chain and always keep an unused link between every two used links.

$G \in$ dHAMPATH $\Rightarrow \varphi \in$ SAT. Suppose that G has an Hamiltonian path P. We first note that the path P must start in v_{start} and end at v_{end}. Furthermore, we claim that P needs to traverse all the chains in order and, within each each chain, traverse it either in left-to-right order or right-to-left order. □

3.2.5 Decision versus Search

Theorem 3.25. *Suppose that* $\boldsymbol{P = NP}$. *Then for every* **NP** *language* L *and a verifier TM* M *for* L, *there is a polynomial-time TM* B *that on input* $x \in L$ *outputs a certificate for* x.

Proof. We need to show that if $\mathbf{P} = \mathbf{NP}$ then for every polynomial-time TM M and polynomial $p(n)$, there is a polynomial-time TM B with the following property: for every $x \in \{0,1\}^n$ if there is $u \in \{0,1\}^{p(n)}$ s.t. $M(x,u) = 1$ then $|B(x)| = p(n)$ and $M(x, B(x)) = 1$

We start by showing the theorem for the case of SAT. In particular, we show that given an algorithm A that decides SAT, we can come up an algorithm B that on input a satisfiable CNF formula φ with n variables, finds a satisfying assignment for φ using $2n + 1$ calls to A and some additional polynomial-time computation.

We first use A to check that the input formula is satisfiable. If so, we first substitute $x_1 = 0$ and then $x_1 = 1$ in φ and then use A to decide which of the two is satisfiable. Say the first is satisfiable. Then we fix $x_1 = 0$. Continuing this way, we end up with an assignment

To solve the search problem for an arbitrary **NP**-language L, we use the fact that the reduction of Theorem 3.18 from L to SATis actually a Levin reduction. This means that we have a polynomial-time computable function f s.t. we can map a satisfying assignment of $f(x)$ into a certificate for x. □

The theorem 3.25 shows that SAT is **downward self-reducible**, which means that given an algorithm that solves SAT on inputs of length smaller than n we can solve SAT on inputs of length n.

3.2.6 CONP,EXP and NEXP

Definition 3.26. $\mathbf{coNP} = \{L : \bar{L} \in \mathbf{NP}\}$

Definition 3.27 (alternative definition)**.** For every $L \subseteq \{0,1\}^*$, we say that $L \in \mathbf{coNP}$ if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time TM M s.t. for every $x \in \{0,1\}^*$

$$x \in L \Leftrightarrow \forall u \in \{0,1\}^{p(|x|)},\ M(x,u) = 1$$

Example 3.2. The following language is **coNP**-complete

$$\texttt{TAUTOLOGY} = \{\varphi : \varphi \text{ is a tautology}\}$$

It's clearly in **coNP** by Definition 3.27 (Make u to be the all possible assignments). Modify the Cook-Levin reduction from $\bar{L}$(which is in **NP**) to SAT. For every input $x \in \{0,1\}^*$ that reduction produces a formula φ_x that is satisfiable iff $x \in \bar{L}$. Now consider the formula $\neg\varphi_x$. It is in TAUTOLOGY iff $x \in L$

3.2.7 EXP and NEXP

Theorem 3.28. *If* $\mathbf{EXP} \neq \mathbf{NEXP}$ *then* $\mathbf{P} \neq \mathbf{NP}$

Proof. We prove the contrapositive: Assuming $\mathbf{P} = \mathbf{NP}$ we show $\mathbf{EXP} = \mathbf{NEXP}$. Suppose $L \in \mathbf{NTIME}(2^{n^c})$ and NDTM M decides it. We claim that the language

$$L_{\text{pad}} = \left\{ \langle x, 1^{2^{|x|^c}} \rangle : x \in L \right\}$$

is in **NP**. Here is an NDTM for L_{pad}: Given y, first check if there is a string z s.t. $y = \langle z, 1^{2^{|z|^c}} \rangle$. If not, output 0. If y is of this form, then simulate M on z for $2^{|z|^c}$ steps and output its answer. The running time is polynomial in $|y|$, and hence $L_{\text{pad}} \in \mathbf{NP}$. Hence if $\mathbf{P} = \mathbf{NP}$ then $L_{\text{pad}} \in \mathbf{P}$. But if L_{pad} is in **P** then L is in **EXP**. To determine whether an input x is in L, we just pad the input and decide whether it is in L_{pad} using the polynomial-time machine for L_{pad} □

3.2.8 Exercise

Exercise 3.2.1. Argue at a high level that the following language is **NP**-complete

$$\{\langle \varphi, 1^n \rangle : \text{ math statement } \varphi \text{ has a proof of size at most } n \text{ in the ZF system}\}$$

Proof. Essential part is to find a reduction.
Idea: if there are n derivation rules, then we consider nSAT □

Exercise 3.2.2. Prove that $\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}$

Exercise 3.2.3. Prove that Definition 3.26 and 3.27 do indeed define the same class

Proof. Suppose $\mathbf{coNP} = \{L : \bar{L} \in \mathbf{NP}\}$.

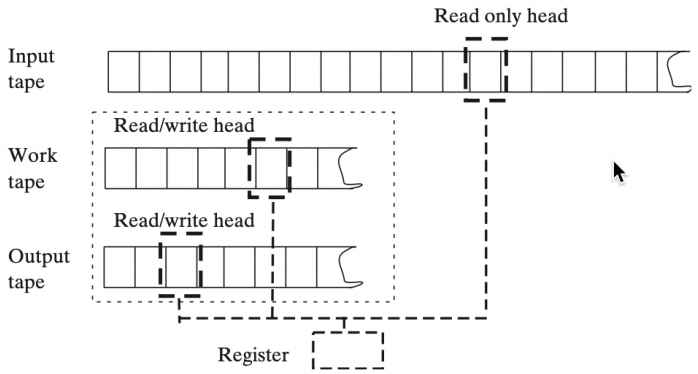
$$\begin{aligned} x \in L \in \mathbf{coNP} &\Leftrightarrow x \notin \bar{L} \in \mathbf{NP} \\ &\Leftrightarrow \neg \exists u \in \{0,1\}^{p(|x|)} M'(x,u) = 1 \\ &\Leftrightarrow \forall u \in \{0,1\}^{p(|x|)} M'(x,u) \neq 1 \\ &\Leftrightarrow \forall u \in \{0,1\}^{p(|x|)} M(x,u) = 1 \end{aligned}$$

where M' is a TM for **NP** and M is a TM for **coNP** by computing the value from M'.
Another direction is the same. □

3.3 4: Space Complexity

3.3.1 Definition of Space-Bounded Computation

TM we use is



Definition 3.29 (Space-bounded computation)**.** Let $S : \mathbb{N} \to \mathbb{N}$ and $L \subseteq \{0,1\}^*$. We say that $L \in \mathbf{SPACE}(s(n))$ if there is a constant c and a TM M deciding L s.t. at most $c \cdot s(n)$ locations on M's work tapes (excluding the input tape) are ever visited by M's head during its computation on every input of length n

Similarly we say that $L \in \mathbf{NSPACE}(s(n))$ if there is an NDTM M deciding L that never uses more than $c \cdot s(n)$ nonblank tape locations on length n inputs

$S : \mathbb{N} \to \mathbb{N}$ is **space-constructible** if there is a TM that computes $S(|x|)$ in $O(S(|x|))$ space given x as input

Since the TM's work tapes are separated from its input tape, it makes sense to consider space-bounded machines that use space less than the input length, namely, $S(n) < n$. We will require however that $S(n) > \log n$.

$\mathbf{DTIME}(S(n)) \subseteq \mathbf{SPACE}(S(n))$ since a TM can access only one tape cell per step. But a $\mathbf{SPACE}(S(n))$ machine can run for much longer than $S(n)$ steps

Theorem 3.30. *For every space constructible $S : \mathbb{N} \to \mathbb{N}$*

$$\boldsymbol{DTIME}(S(n)) \subseteq \boldsymbol{SPACE}(S(n)) \subseteq \boldsymbol{NSPACE}(S(n)) \subseteq \boldsymbol{DTIME}(2^{O(S(n))})$$

We use the notion of a **configuration graph** of a Turing machine. Let M be a (deterministic or nondeterministic) TM. A **configuration** of a TM M consists of the contents of all nonblank entries of M's tapes, along with its state and head position, at a particular point in its execution. For every space $S(n)$ TM M and input $x \in \{0,1\}^*$, the **configuration graph of M on input x**, denoted $G_{M,x}$, is a directed graph whose nods correspond to all possible configuration of M where the input contains the value x and the work tapes have at most $S(|x|)$ nonblank cells. The graph has a directed edge from a configuration C to a configuration C' if C' can be reached from C in one step according to M's transition function. By modifying M to erase all its work tapes before halting, we can assume that there is only a single configuration C_{accept} on which M halts and outputs 1.

Claim: Let $G_{M,x}$ be the configuration graph of a space-$S(n)$ machine M on some input x of length n. Then

1. Every vertex in $G_{M,x}$ can be described using $cS(n)$ bits for some constant c (depending on M's alphabet size and number of tapes') and in particular, $G_{M,x}$ has at most $2^{cS(n)}$ nodes
2. There is an $O(S(n))$-size CNF formula $\varphi_{M,x}$ s.t. for every two strings C, C' $\varphi_{M,x}(C, C') = 1$ iff C and C' encodes two neighboring configuration in $G_{M,x}$

proof of theorem 3.30. By enumerating all possible configurations, we can construct the graph $G_{M,x}$ in $2^{O(S(n))}$-times and check whether C_{start} is connected to C_{accept} in $G_{M,x}$ using the standard breadth-first search algorithm for connectivity □

Definition 3.31.

$$\begin{aligned}\mathbf{PSPACE} &= \bigcup_{c>0} \mathbf{SPACE}(n^c)\\ \mathbf{NPSPACE} &= \bigcup_{c>0} \mathbf{NSPACE}n^c\\ \mathbf{L} &= \mathbf{SPACE}(\log n)\\ \mathbf{NL} &= \mathbf{NSPACE}(\log n)\end{aligned}$$

Example 3.3. $\texttt{3SAT} \in \mathbf{PSPACE}$. The machine just uses the linear space to cycle through all 2^k assignments to order. Once an assignment is checked, erase it on tape.

In fact, $\mathbf{NP} \subseteq \mathbf{PSPACE}$.

Let

$$\texttt{PATH} = \{\langle G, s, t\rangle : G \text{ is a directed graph in which there is a path from } s \text{ to } t\}$$

Note that $\texttt{PATH} \in \mathbf{NL}$.

Theorem 3.32 (Space Hierarchy Theorem)**.** *If f, g are space-constructible functions satisfying $f(n) = o(g(n))$, then*

$$\boldsymbol{SPACE}(f(n)) \subsetneq \boldsymbol{SPACE}(g(n))$$

3.3.2 PSPACE Completeness

Definition 3.33. A language L' is **PSPACE-hard** if for every $L \in \mathbf{PSPACE}$, $L \leq_p L'$. If in addition $L' \in \mathbf{PSPACE}$ then L' is **PSPACE-complete**

Definition 3.34 (Quantified Boolean Formula)**.** A **quantified Boolean formula** (QBF) is a formula of the form $Q_1 x_1 \ldots Q_n x_n \varphi(x_1, \ldots, x_n)$ where each $Q_i$is one of the two quantifiers $\forall$ or $\exists$, $x_1, \ldots, x_n$ ranges over $\{0, 1\}$ and φ is a quantifier-free Boolean formula.

Let $\texttt{TQBF}$ be the set of quantified Boolean formulae that are true

Theorem 3.35. *$\texttt{TQBF}$ is **PSPACE**-complete*

Proof. First we show that $\texttt{TQBF} \in \mathbf{PSPACE}$. Let

$$\psi = Q_1 x_1 \ldots Q_n x_n \varphi(x_1, \ldots, x_n)$$

be a quantified Boolean formula with n variables, where we denote the size of φ by m. We show a recursive algorithm A that can decide the truth of ψ in $O(n + m)$ space.

If $n = 0$ then φ can be evaluated in $O(m)$ time and space, and so we assume $n > 0$. For $b \in \{0,1\}$ denote by $\psi|_{x_1=b}$ the modification of ψ where the first quantifier Q_1 is dropped and all occurrences of x_1 are replaces with the constant b. Algorithm A will works as follows: if $Q_1 = \exists$ then output 1 iff at least one of $A(\psi|_{x_1=0})$ and $A(\psi|_{x_1=1})$ outputs 1. If $Q_1 = \forall$, then output 1 iff both $A(\psi|_{x_1=0})$ and $A(\psi|_{x_1=1})$ outputs 1.

Let $s_{n,m}$ denote the space A uses on formula with n variables and description size m. The crucial point is - here we use the fact that space can be **reused** - that both recursive computations $A(\psi|_{x_1=0})$ and $A(\psi|_{x_1=1})$ can run in the same space. Thus assuming that A uses $O(m)$ space to write $A(\psi|_{x_1=b})$ for its recursive calls, we'll get that $s_{n,m} = s_{n-1,m} + O(m)$ yielding $s_{n,m} = O(n \cdot m)$

We now show that $L \leq_p$ TQBF for every $L \in$ **PSPACE**. Let M be a machine that decides L in $S(n)$ space and let $x \in \{0,1\}^n$. We show how to construct a quantified Boolean formula of size $O(S(n)^2)$ that is true iff M accepts x. Let $m = O(S(n))$ denote the number of bits needed to encode a configuration of M on length n. By Claim 3.3.1 there is a Boolean formula $\varphi_{M,x}$ s.t. for every two strings $C, C' \in \{0,1\}^m$ $\varphi_{M,x}(C,C) = 1$ iff C and C' encode two adjacent configurations in the configuration graph $G_{M,x}$.

let $\psi_i(C,C')$ be true iff there is a path of length at most 2^i from C to C' in $G_{M,x}$. Note that $\psi = \psi_m$ and $\psi_0 = \varphi_{M,x}$. The crucial point is $\psi_i(C,C') = \exists C'' \psi_{i-1}(C,C'') \wedge \psi_{i-1}(C'',C')$

But ψ_m has size about 2^m, which is not good. Instead, we use additional quantified variables to save on description size

$$\exists C'' \forall D_1 \forall D_2((D_1 = C \wedge D_2 = C'') \vee (D_1 = C'' \wedge D_2 = C')) \rightarrow \psi_{i-1}(D_1, D_2)$$

Note that $size(\psi_i) \leq size(\psi_{i-1}) + O(m)$ and hence $size(\psi_m) \leq o(m^2)$ □

Since we don't require the graph in proof of Theorem 3.35 to have out-degree one, it actually yields a stronger statement

$$\texttt{TQBF} \in \textbf{NPSPACE}$$

Theorem 3.36 (Savitch's Theorem)**.** *For any space-constructible $S : \mathbb{N} \rightarrow \mathbb{N}$ with $S(n) \geq \log n$, **NSPACE**$(S(n)) \subseteq$ **SPACE**$(S(n)^2)$*

Proof. Let $L \in$ **NSPACE**$(S(n))$ be a language decided by a TM M s.t. for every $x \in \{0,1\}^n$, the configuration graph $G = G_{M,x}$ has at most $M = 2^{O(S(n))}$ vertices, and determining whether $x \in L$is equivalent to determining whether C_{accept} can be reached from C_{start} in this graph. We describe a recursive procedure REACH?(u,v,i) that returns "YES" if there is a path from u to v of length at most 2^i and "NO" otherwise.

$$\text{REACH?}(u,v,i) = \exists w(\text{REACH?}(u,w,i-1) \vee \text{REACH?}(w,v,i-1))$$

Let $s_{M,i}$ be the space complexity of REACH?(u, v, i), then $s_{M,i} = s_{M,i-1} + O(\log M)$ (since we need to enumerate all w in TM, the space we need is $O(\log M)$) and thus $s_{M,\log M} = O(\log^2 M) = O(S(n)^2)$ □

Example 3.4 (The QBF game)**.** The "board" for the QBF game is a Boolean formula φ whose free variables are $x_1, \ldots, x_{2n}$. player 1 will pick values for the odd-numbered variables. We say player 1 wins iff at the end $\varphi(x_1, \ldots, x_{2n})$ is true

In order for player 1 to have a **winning strategy** he must have a way to win for all possible sequences of moves by player 2, namely, if

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \forall x_{2n} \varphi(x_1, \ldots, x_{2n})$$

Thus deciding whether player 1 has a winning strategy for a given board in the QBF game is **PSPACE**-complete.

3.3.3 NL Completeness

We cannot use the polynomial-time reduction since $\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P}$ (cf. Exercise 3.3.1)

Definition 3.37 (logspace reduction and **NL**-completeness)**.** A function $f : \{0,1\}^* \to \{0,1\}^*$ is **implicitly logspace computable**, if f is polynomially bounded (i.e., there is some c s.t. $|f(x)| \leq |x|^c$ for every $x \in \{0,1\}^*$) and the language $L_f = \{\langle x, i\rangle \mid f(x)_i = 1\}$ and $L'_f = \{\langle x, i\rangle \mid i \leq |f(x)|\}$ are in **L**

A language B is **logspace reducible** to language C, denoted by $B \leq_l C$ if there is a function $f : \{0,1\}^* \to \{0,1\}^*$ that is implicitly logspace computable and $x \in B$ iff $f(x) \in C$ for every $x \in \{0,1\}^*$.

We say that C is **NL-complete** if it is in **NL** and for every $B \in \mathbf{NL}$, $B \leq_l C$

Another way (used by several texts) to think of logspace reductions is to imagine that the reduction is given a separate "write-once" output tape, on which it can either write a bit or move to the right but never move left or read the bits it wrote down previously.The two notions are easily proved to be equivalent (seeExercise 3.3.2).

Lemma 3.38. *1. If $B \leq_l C$ and $C \leq_l D$ then $B \leq_l D$*

2. if $B \leq_l C$ and $C \in \mathbf{L}$ then $B \in \mathbf{L}$

Proof. We prove that if f, g are two implicitly logspace computable functions, then so if $h(x) = g(f(x))$. Part 2 follows by letting f be the reduction from B to C and g be the characteristic function of C (i.e., $g(y) = 1$ iff $y \in C$)

Let M_f, M_g be the logspace machines that compute the mappings $x, i \mapsto f(x)_i$ and $y, j \mapsto g(y)_i$ respectively. We construct a machine M_h that given input x, j with $j \leq |g(f(x))|$ outputs $g(f(x))_j$

□

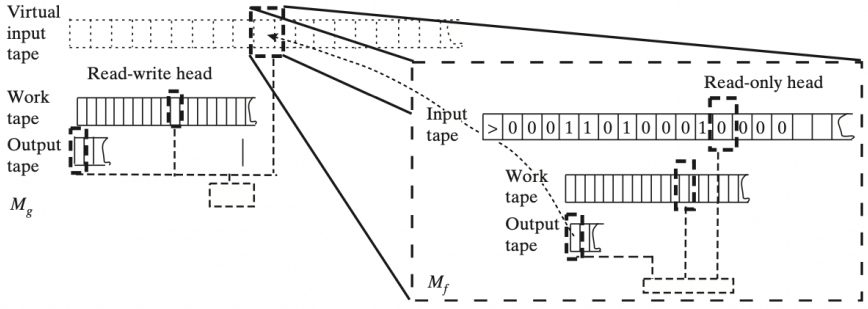


Figure 4.3. Composition of two implicitly logspace computable functions f, g. The machine M_g uses calls to f to implement a "virtual input tape." The overall space used is the space of M_f + the space of M_g + $O(\log|f(x)|)$ = $O(\log|x|)$.

3.3.4 Exericse

Exercise 3.3.1. Prove that every language L that is not the empty or $\{0,1\}^*$ is complete for **NL** under polynomial-time Karp reductions

Exercise 3.3.2.