CSIE 5046: Topics in Complexity Theory

Tony Tan
Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Table of contents

Lesson 0. Preliminaries: Review of some material such as the Big Oh and Omega notations, Turing machines, the notion of algorithms, basic complexity classes and the efficient universal Turing machines (with only logarithmic blow-up in run-time).

Lesson 1. The class NP: Cook-Levin reductions, parsimonious reductions, Ladner's theorem on NP-intermediate languages and Gill-Baker-Solovay's theorem.

Lesson 2. The class NL: Log-space reductions, **NL**-completeness, Savitch's theorem and Immerman-Szelepcsényi's theorem ($\mathbf{NL} = \mathbf{coNL}$).

Lesson 3. The class PSPACE: **PSPACE**-complete languages, the language TQBF and the generalization of Savitch and Immerman-Szelepcsényi's theorem.

Lesson 4. Alternating Turing machines: Complexity classes based on alternating Turing machines and their relations with the deterministic counterpart.

Lesson 5. The polynomial hierarchy and the complexity classes for counting: The polynomial time hierarchy and the complexity classes four counting: **FP** and $\sharp\mathbf{P}$.

Lesson 6. Computing permanent: Computing the permanent of matrices is $\sharp\mathbf{P}$-complete.

Lesson 7. Boolean circuits, part 1: The class $\mathbf{P}_{/\mathsf{poly}}$, Karp-Lipton's theorem, Meyer's theorem and the classes **NC** and **AC**.

Lesson 8. Boolean circuits, part 2: Switching lemma and its application to prove circuit lower bound.

Lesson 9. Probabilistic Turing machines: Probabilistic Turing machines, one-/two-sided error, zero error and classical results such as Adleman and Sipser-Gács-Lautemann's theorem.

Lesson 10. The probabilistic method: The basic counting argument, the expectation argument, sample and modify and Lovász local lemma.

Lesson 11. Probabilistic reductions: Valiant-Vazirani's lemma, the language $\oplus\mathsf{SAT}$, the class $\oplus\mathbf{P}$ and probabilistic reductions from SAT and $\overline{\mathsf{SAT}}$ to $\oplus\mathsf{SAT}$ and the generalization to every language in **PH** (preliminary to Toda's theorem).

Lesson 12. Toda's theorem: Reduction from $\oplus\mathsf{SAT}$ to $\sharp\mathsf{SAT}$ and the proof of Toda's theorem.

Lesson 0: Preliminaries

Theme: Review of some introductory material.

Let $\mathbb{N}$ denote the set of natural numbers $\{0, 1, 2, \ldots\}$. Let f and g be functions from $\mathbb{N}$ to $\mathbb{N}$.

- $f = O(g)$ means that there is c and n_0 such that for every $n \geqslant n_0$, $f(n) \leqslant c \cdot g(n)$.
  It is usually phrased as "there is c such that for (all) sufficiently large n," $f(n) \leqslant c \cdot g(n)$.
- $f = \Omega(g)$ means $g = O(f)$.
- $f = \Theta(g)$ means $g = O(f)$ and $f = O(g)$.
- $f = o(g)$ means for every $c > 0$, $f(n) \leqslant c \cdot g(n)$ for sufficiently large n.
  Equivalently, $f = o(g)$ means $f = O(g)$ and $g \neq O(f)$.
  Another equivalent definition is $f = o(g)$ means $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.
- $f = \omega(g)$ means $g = o(f)$.

To emphasize the input parameter, we will write $f(n) = O(g(n))$. The same for the Ω, o, ω notations. We also write $f(n) = \mathsf{poly}(n)$ to denote that $f(n) = c \cdot n^k$ for some c and $k \geqslant 1$.

Throughout the course, for an integer $n \geqslant 0$, we will denote by $\lfloor n \rfloor$ the binary representation of n. Likewise, $\lfloor G \rfloor$ the binary encoding of a graph G. In general, we write $\lfloor X \rfloor$ to denote the encoding/representation of an object X as a binary string, i.e., a 0-1 string. To avoid clutter, in most cases we simply write X instead of $\lfloor X \rfloor$.

We usually write Σ to denote a finite input alphabet. Often $\Sigma = \{0, 1\}$. Recall also that for a word $w \in \Sigma^*$, $|w|$ denotes the length of w. For a DTM/NTM $\mathcal{M}$, we write $L(\mathcal{M})$ to denote the language $\{w : \mathcal{M} \text{ accepts } w\}$.

We often view a language $L \subseteq \Sigma^*$ as a boolean function, i.e., $L : \Sigma^* \to \{\mathsf{true}, \mathsf{false}\}$, where $L(x) = \mathsf{true}$ if and only if $x \in L$, for every $x \in \Sigma^*$.

1 Time complexity

Definition 0.1 Let $\mathcal{M}$ be a DTM/NTM, $w \in \Sigma^*$, $t \in \mathbb{N}$ and let $f : \mathbb{N} \to \mathbb{N}$ be a function.

- $\mathcal{M}$ *decides w in time t (or, in t steps)*, if every run of $\mathcal{M}$ on w has length at most t. That is, for every run of $\mathcal{M}$ on w:
  $$C_0 \vdash C_1 \vdash \cdots \vdash C_m \qquad \text{where } C_m \text{ is a halting configuration,}$$
  we have $m \leqslant t$.
- $\mathcal{M}$ *runs in time $O(f(n))$*, if there is $c > 0$ such that for sufficiently long word w, $\mathcal{M}$ decides w in time $c \cdot f(|w|)$.
- $\mathcal{M}$ *decides/accepts a language L in time $O(f(n))$*, if $L(\mathcal{M}) = L$ and $\mathcal{M}$ runs in time $O(f(n))$.
- $\textsc{Dtime}[f(n)] \overset{\mathsf{def}}{=} \{L : \text{there is a DTM } \mathcal{M} \text{ that decides } L \text{ in time } O(f(n))\}$.
- $\textsc{Ntime}[f(n)] \overset{\mathsf{def}}{=} \{L : \text{there is an NTM } \mathcal{M} \text{ that decides } L \text{ in time } O(f(n))\}$.

We say that $\mathcal{M}$ runs in *polynomial* and *exponential time*, if there is $f(n) = \mathsf{poly}(n)$ such that $\mathcal{M}$ runs in time $O(f(n))$ and $O(2^{f(n)})$, respectively. In this case we also say that $\mathcal{M}$ is a polynomial/exponential time TM.

The following are some of the important classes in complexity theory.

$$\mathbf{P} \stackrel{\mathsf{def}}{=} \bigcup_{f(n)=\mathsf{poly}(n)} \textsc{Dtime}[f(n)] \qquad \mathbf{EXP} \stackrel{\mathsf{def}}{=} \bigcup_{f(n)=\mathsf{poly}(n)} \textsc{Dtime}[2^{f(n)}]$$

$$\mathbf{NP} \stackrel{\mathsf{def}}{=} \bigcup_{f(n)=\mathsf{poly}(n)} \textsc{Ntime}[f(n)] \qquad \mathbf{NEXP} \stackrel{\mathsf{def}}{=} \bigcup_{f(n)=\mathsf{poly}(n)} \textsc{Ntime}[2^{f(n)}]$$

$$\mathbf{coNP} \stackrel{\mathsf{def}}{=} \{L : \Sigma^* - L \in \mathbf{NP}\} \qquad \mathbf{coNEXP} \stackrel{\mathsf{def}}{=} \{L : \Sigma^* - L \in \mathbf{NEXP}\}$$

Theorem 0.2 (Padding theorem) *If* $\mathbf{NP} = \mathbf{P}$*, then* $\mathbf{NEXP} = \mathbf{EXP}$.

Likewise, if $\mathbf{NP} = \mathbf{coNP}$*, then* $\mathbf{NEXP} = \mathbf{coNEXP}$.

2 Space complexity

Definition 0.3 Let $\mathcal{M}$ be a DTM/NTM, $w \in \Sigma^*$, $s \in \mathbb{N}$ and $f : \mathbb{N} \to \mathbb{N}$ be a function.

- $\mathcal{M}$ *decides* w *using* s *space* (or, in space s), if $\mathcal{M}$ halts on w and for every run of $\mathcal{M}$ on w:

  $$C_0 \vdash C_1 \vdash \cdots \vdash C_m \qquad \text{where } C_m \text{ is a halting configuration,}$$

  we have $|C_i| \leqslant s$ for every $1 \leqslant i \leqslant m$.
- $\mathcal{M}$ *uses space* $O(f(n))$ (or, *runs in space* $O(f(n))$), if there is $c \geqslant 0$ such that for sufficiently long word w, $\mathcal{M}$ decides w in space $c \cdot f(|w|)$.
- $\mathcal{M}$ *decides/accepts a language* L *in space* $O(f(n))$, if $L(\mathcal{M}) = L$ and $\mathcal{M}$ uses space $O(f(n))$.
- $\textsc{Dspace}[f(n)] \stackrel{\mathsf{def}}{=} \{L :$ there is a DTM $\mathcal{M}$ that decides L in space $O(f(n))\}$.
- $\textsc{Nspace}[f(n)] \stackrel{\mathsf{def}}{=} \{L :$ there is an NTM $\mathcal{M}$ that decides L in space $O(f(n))\}$.

We say that $\mathcal{M}$ uses *polynomial* space, if there is $f(n) = \mathsf{poly}(n)$ such that $\mathcal{M}$ uses space $O(f(n))$. In this case, we also say that $\mathcal{M}$ is a polynomial space TM.

The following are some of the important classes in the complexity theory.

$$\mathbf{PSPACE} \stackrel{\mathsf{def}}{=} \bigcup_{f(n)=\mathsf{poly}(n)} \textsc{Dspace}[f(n)]$$

$$\mathbf{NPSPACE} \stackrel{\mathsf{def}}{=} \bigcup_{f(n)=\mathsf{poly}(n)} \textsc{Nspace}[f(n)]$$

$$\mathbf{coNPSPACE} \stackrel{\mathsf{def}}{=} \{L : \Sigma^* - L \in \mathbf{NPSPACE}\}$$

In a few weeks we will show that $\mathbf{PSPACE} = \mathbf{NPSPACE} = \mathbf{coNPSPACE}$.

Definition 0.4 (logarithmic space) Let $\mathcal{M}$ be a k-tape DTM/NTM, where $k \geqslant 2$.

- *DTM/NTM* $\mathcal{M}$ *uses space* $O(\log(n))$, if there is $c > 0$ such that for sufficiently long word w, the following holds.

  - The first tape always contains only the input word w, i.e., $\mathcal{M}$ never changes the content of the first tape.
  - For all the other tapes, the number of cells used by $\mathcal{M}$ is $\leqslant c \cdot \log(|w|)$.

- $\mathcal{M}$ *decides/accepts a language* L *in space* $O(\log(n))$, if $L(\mathcal{M}) = L$ and $\mathcal{M}$ uses space $O(\log(n))$.

  In this case, we say that $\mathcal{M}$ uses *logarithmic* space, or that $\mathcal{M}$ is a *logarithmic* space TM.

Similar to above, we can define the following clasess.

$$\begin{aligned}
\mathbf{L} &\overset{\mathsf{def}}{=} \{L : \text{there is a DTM } \mathcal{M} \text{ that decides } L \text{ in space } O(\log(n))\} \\
\mathbf{NL} &\overset{\mathsf{def}}{=} \{L : \text{there is an NTM } \mathcal{M} \text{ that decides } L \text{ in space } O(\log(n))\} \\
\mathbf{coNL} &\overset{\mathsf{def}}{=} \{L : \Sigma^* - L \in \mathbf{NL}\}
\end{aligned}$$

In a few weeks we will also show that $\mathbf{NL} = \mathbf{coNL}$. It is still an open question if $\mathbf{L} \overset{?}{=} \mathbf{NL}$.

3 Universal Turing machines

Remark 0.5 For every k-tape TM $\mathcal{M}$ over input alphabet $\Sigma = \{0, 1\}$, there is a k-tape TM $\mathcal{M}'$ over the same input alphabet $\Sigma = \{0, 1\}$ and tape alphabet $\Gamma = \{0, 1, \sqcup\}$ such that $L(\mathcal{M}) = L(\mathcal{M}')$. Moreover, if $\mathcal{M}$ runs in time/space $O(f(n))$, so does $\mathcal{M}'$.

Due to this, we always assume that the input and tape alphabet of Turing machines are $\Sigma = \{0, 1\}$ and $\Gamma = \{0, 1, \sqcup\}$, respectively.

Recall that $\lfloor \mathcal{M} \rfloor$ denotes the encoding of a TM $\mathcal{M}$.

Definition 0.6 A *Universal Turing machine* (UTM) is a k-tape DTM $\mathcal{U}$, for some $k \geqslant 1$, such that $L(\mathcal{U}) = \{\lfloor \mathcal{M} \rfloor \$ w \mid \mathcal{M} \text{ accepts } w \text{ and } w \in \{0, 1\}^*\}$.

Theorem 0.7 *There is a UTM* $\mathcal{U}$ *such that for every DTM* $\mathcal{M}$ *and every word* w*, if* $\mathcal{M}$ *decides* w *in time* t*, then* $\mathcal{U}$ *decides* $\lfloor \mathcal{M} \rfloor \$ w$ *in time* $(\alpha \cdot t \cdot \log t)$*, where* α *does not depends* $|w|$*, but on size of the tape alphabet of* $\mathcal{M}$ *as well as the number of tapes and states of* $\mathcal{M}$*.*

Appendix

A Turing machines

We reserve a special symbol $\sqcup$, called the *blank* symbol.

A 1-tape *Turing machine* (TM) is a system $\mathcal{M} = \langle \Sigma, \Gamma, Q, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}, \delta \rangle$, where each component is as follows.

- Σ is a finite alphabet, called the *input* alphabet, where $\sqcup \notin \Sigma$.
- Γ is a finite alphabet, called the *tape* alphabet, where $\Sigma \subseteq \Gamma$ and $\sqcup \in \Gamma$.
- Q is a finite set of states.
- $q_0 \in Q$ is the initial state.
- $q_{\mathsf{acc}}, q_{\mathsf{rej}} \in Q$ are two special states called the *accept* and *reject* states, respectively.

- $\delta : Q - \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\} \times \Gamma \to Q \times \Gamma \times \{\texttt{Left}, \texttt{Right}\}$ is the transition function.

Intuitively, the intuitive meaning of $\delta(p, a) = (q, b, \texttt{Move})$ is as follows. When the head reads a symbol a, if $\mathcal{M}$ is in state p, it "writes" symbol b on top of a, enters state q, and the head moves left, if $\texttt{Move} = \texttt{Left}$, or moves right, if $\texttt{Move} = \texttt{Right}$.

To describe how a TM computes, we need a few terminologies. A configuration of $\mathcal{M}$ is a string C from $(Q \cup \Gamma)^*$ which contains <u>*exactly one symbol*</u> from Q. We call such symbol the state of C. Intuitively, a configuration $C = a_1 \cdots a_{i-1}\ pa_i \cdots a_m$ means the content of the tape is:

$$\cdots \sqcup \sqcup \sqcup \ a_1 \cdots a_{i-1} a_i \cdots a_m \ \sqcup \sqcup \sqcup \cdots$$

with the head reading a_i.

On input word $w \in \Sigma^*$, the *initial* configuration of $\mathcal{M}$ on w is the string $q_0 w$. A configuration is called *accepting*, if it contains q_{acc}, and it is called *rejecting*, if it contains q_{rej}. A *halting* configuration is either an accepting or a rejecting configuration.

Let $C = a_1 \cdots a_{i-1}\ pa_i \cdots a_m$ be a configuration, where $a_1, \ldots, a_m \in \Gamma$ and $p \in Q$ such that $p \neq q_{\mathsf{acc}}, q_{\mathsf{rej}}$. The transition δ yields the subsequent configuration C', denoted by $C \vdash C'$, as follows.

- If $\delta(p, a_i) = (q, b, \texttt{Left})$ and $i \geqslant 2$, then $C' = a_1 \cdots a_{i-2}\ qa_{i-1} b a_{i+1} \cdots a_m$.
- If $\delta(p, a_i) = (q, b, \texttt{Left})$ and $i = 1$, then $C' = q \sqcup ba_2 \cdots a_m$.
- If $\delta(p, a_i) = (q, b, \texttt{Right})$ and $i \leqslant m - 1$, then $C' = a_1 \cdots a_{i-1} b\ qa_{i+1} \cdots a_m$.
- If $\delta(p, a_i) = (q, b, \texttt{Right})$ and $i = m$, then $C' = a_1 \cdots a_{m-1} b\ q\sqcup$.

The *run* of $\mathcal{M}$ on w is the (possibly infinite) sequence:

$$C_0 \ \vdash \ C_1 \ \vdash \ C_2 \ \vdash \ \cdots, \tag{1}$$

where C_0 is the initial configuration of $\mathcal{M}$ on w.

$\mathcal{M}$ stops when it reaches a halting configuration, i.e., when it reaches either q_{acc} or q_{rej}. If $\mathcal{M}$ halts in an accepting configuration, then we say that $\mathcal{M}$ *accepts* w. If it halts in a rejecting configuration, then we say that $\mathcal{M}$ *rejects* w. We denote by $L(\mathcal{M}) \overset{\mathsf{def}}{=} \{w : \mathcal{M} \text{ accepts } w\}$.

Remark 0.8 Our definition of Turing machine above is usually called *two-way infinite tape*, in the sense that the tape is unbounded on both the left and the right side. In most textbooks, Turing machine is defined as only "one-way" in the sense that the left side is bounded, but the right side is unbounded. Both definitions are equivalent. Neither one is computationally stronger than the other.

Multi-tape Turing machines. A multi-tape Turing machine is a Turing machine that has a few tapes. On each tape, the Turing machine has one head. Formally, it is defined as follows. Let $k \geqslant 1$. A k-tape Turing machine is $\mathcal{M} = \langle \Sigma, \Gamma, Q, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}, \delta \rangle$, where δ is a function

$$\delta \quad : \quad (Q - \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}) \times \Gamma^k \to Q \times (\Gamma - \{\sqcup\})^k \times \{\texttt{Left}, \texttt{Right}\}^k$$

As before, an element of δ is written in the form:

$$(q, a_1, \ldots, a_k) \to (p, b_1, \ldots, b_k, \texttt{Move}_1, \ldots, \texttt{Move}_k).$$

Intuitively, it means that if the TM is in state q, and on each $i = 1, \ldots, k$, the head on tape i is reading a_i, then it enters state p, and for $i = 1, \ldots, k$, the head on tape i writes the symbol b_i and moves according to $\texttt{Move}_i$.

A *configuration* of $\mathcal{M}$ is of the form $(q, u_1, \ldots, u_k)$, where $q \in Q$ and each u_i is a string over $\Gamma \cup \{\bullet\}$ and the symbol $\bullet$ appears exactly once in each u_i. The symbol $\bullet$ is to denote the position of the head.

The input is always written in the first tape. All the other tapes are initially blank. Formally, the initial configuration on input w is $(q_0, \bullet w, \bullet, \ldots, \bullet)$.

The notion of "one step computation" $C \vdash C'$ is defined similarly as in the standard Turing machine. Likewise, the conditions of acceptance and rejection are defined as when the Turing machines enters the accepting and rejecting states, respectively.

Theorem 0.9 *For k-tape TM $\mathcal{M}$, there is a single tape TM $\mathcal{M}'$ such that for every word w, the following holds.*

- *If $\mathcal{M}$ accepts w, then $\mathcal{M}'$ accepts w.*
- *If $\mathcal{M}$ rejects w, then $\mathcal{M}'$ rejects w.*
- *If $\mathcal{M}$ does not halt on w, then $\mathcal{M}'$ does not halt on w.*

Non-deterministic Turing machines. A non-deterministic Turing machine (NTM) is a system $\mathcal{M} = \langle \Sigma, \Gamma, Q, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}, \delta \rangle$ defined as the standard Turing machine, with the exception that δ is now a relation:

$$\delta \quad \subseteq \quad (Q - \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}) \times \Gamma \ \times Q \times \Gamma \times \{\texttt{Left}, \texttt{Right}\}$$

As before, we write an element of δ is in the form:

$$(q, a) \to (p, b, \texttt{Move}).$$

The initial configuration of $\mathcal{M}$ on input word w is $q_0 w$. For two configurations C, C', the notion of "one step computation" $C \vdash C'$ is defined similarly as in the standard Turing machine. A *run* of $\mathcal{M}$ on input w is a sequence:

$$C_0 \vdash C_1 \vdash \cdots ,$$

where C_0 is the initial configuration on w. A run is accepting/rejecting, if it ends up in an accepting/rejecting configuration, respectively. However, due to non-determinism, for each C there can be a few configurations C' such that $C \vdash C'$, thus, there can be many runs. Some are accepting, some are rejecting, and some other do not halt.

Encoding of Turing machines. We always assume that the alphabet and the tape alphabet of our TM are $\Sigma = \{0, 1\}$ and $\Gamma = \{0, 1, \sqcup\}$, respectively. Without loss of generality, we can also assume that $Q = \{0, 1, \ldots, n\}$ for some positive integer n with 0 being the initial state.

We note the following.

- Each state $i \in Q$ is written as a string in its binary form.
- Each transition $(i, a) \to (j, b, \alpha) \in \delta$ can be written as string over the symbols 0, 1, (,), **,** , $\tilde{\sqcup}$, L, R, where the symbol $\tilde{\sqcup}$ represents $\sqcup$, and L, R represent `Left`, `Right`, respectively.

So, the whole system $\mathcal{M} = \langle \Sigma, \Gamma, Q, 0, q_{\mathsf{acc}}, q_{\mathsf{rej}}, \delta \rangle$ can be written as a string:

$$\lfloor \Sigma \rfloor \ \# \ \lfloor \Gamma \rfloor \ \# \ \lfloor Q \rfloor \ \# \ \lfloor 0 \rfloor \ \# \ \lfloor q_{\mathsf{acc}} \rfloor \ \# \ \lfloor q_{\mathsf{rej}} \rfloor \ \# \ \lfloor \delta \rfloor$$

where $\lfloor \cdot \rfloor$ denotes the string representing the component $\cdot$ and # the symbol separating two consecutive components.

This shows that every Turing machine (whose tape alphabet is $\Gamma = \{0, 1, \sqcup\}$) can be described as a string over a fixed set of the symbols, i.e., 0, 1, (,), **,** , $\tilde{\sqcup}$, `L`, `R`, #. All these symbols can be further encoded into strings over 0 and 1 to obtain a binary string, which we denote by $\lfloor \mathcal{M} \rfloor$. That is, $\lfloor \mathcal{M} \rfloor$ is the binary string representing the Turing machine $\mathcal{M}$. Sometimes, we will also say $\lfloor \mathcal{M} \rfloor$ is the string description/encoding of $\mathcal{M}$, or the description/encoding of $\mathcal{M}$, for short.

B An informal definition of (deterministic) algorithms

Designing a TM is often a very tedious process. So we often resort to describing an "algorithm" which is defined (informally) as consisting of one "main" Boolean function of the form:

```
Boolean main (String w)
{       statement;
          ⋮
        statement;
}
```

and some (finite number of) functions of the form:

```
⟨value-type⟩ function ⟨function-name⟩ (⟨variable-name⟩,...,⟨variable-name⟩)
{       statement;
          ⋮
        statement;
}
```

Statements in the algorithm are of the following form:

- ⟨`variable-name`⟩ := ⟨`expression`⟩;
- ⟨`variable-name`⟩ := ⟨`function-name`⟩(⟨`variable-name`⟩,...,⟨`variable-name`⟩);
- **return** ⟨`variable-name`⟩/⟨`some-value`⟩;
- **if** ⟨`condition`⟩

```
if ⟨condition⟩
{       statement;
          ⋮
        statement;
} else
{       statement;
          ⋮
        statement;
}
```

```
• while ⟨condition⟩ do
  {       statement;
              ⋮
          statement;
  }
```

Note that we define our algorithm to mimic closely the C/C++/Java language. We may assume that the variables used in each function have different names. Moreover, each variable can only contain "string" value. Values of other types such as Integer and Real are represented in binary forms as strings.

In measuring the space complexity of an algorithm, we count the maximum total length of the strings stored in all variables at any time during its execution process.

C Non-deterministic algorithms

One can define a "non-deterministic" algorithm as a deterministic algorithm extended with an additional special variable z and an instruction of the following form:

$$z \quad := \quad 0 \parallel 1; \tag{2}$$

This instruction means "randomly assign variable z with either 0 or 1."

A non-deterministic algorithm A "accepts" an input word w, if on every instruction of the form (2), variable z can be assigned with 0 or 1 such that A will "return true." Note that the instruction (2) can be encountered more than once during the execution of algorithm A. For example, it may appear inside a `while`-loop.

Lesson 1: The class NP

Theme: Some classical results on the class **NP**.

1 Definitions

We recall the following definition of **NP**.

Definition 1.1 A language L is in **NP** if there is $f(n) = \mathsf{poly}(n)$ and an NTM $\mathcal{M}$ such that $L(\mathcal{M}) = L$ and $\mathcal{M}$ runs in time $O(f(n))$.

There is an alternative definition of **NP**.

Definition 1.2 A language $L \subseteq \Sigma^*$ is in **NP** if there is a language $K \subseteq \Sigma^* \times \Sigma^*$ such that the following holds.

- For every $w \in \Sigma^*$, $w \in L$ if and only if there is $v \in \Sigma^*$ such that $(w, v) \in K$.
- There is $f(n) = \mathsf{poly}(n)$ such that for every $(w, v) \in K$, $|v| \leqslant f(|w|)$.
- The language K is accepted by a polynomial time DTM.

For $(w, v) \in K$, the string v is called the *certificate/witness* for w. We call the language K the *certificate/witness language* for L.

Indeed Def. 1.1 and 1.2 are equivalent. That is, for every language L, L is in **NP** in the sense of Def. 1.1 if and only if L is in **NP** in the sense of Def. 1.2.

2 NP-complete languages

Recall that a DTM $\mathcal{M}$ computes a function $F : \Sigma^* \to \Sigma^*$ in time $O(g(n))$, if there is a constant $c > 0$ such that on every word w, $\mathcal{M}$ computes $F(w)$ in time $\leqslant cg(|w|)$. If $g(n) = \mathsf{poly}(n)$, such fucntion F is called *polynomial time computable* function. Moreover, if $\mathcal{M}$ uses only logarithmic space, it is called *logarithmic space computable* function.

Definition 1.3 A language L_1 is *polynomial time reducible* to another language L_2, denoted by $L_1 \leqslant_p L_2$, if there is a polynomial time computable function F such that for every $w \in \Sigma^*$:

$$w \in L_1 \quad \text{if and only if} \quad F(w) \in L_2$$

Such function F is called polynomial time reduction, also known as *Karp reduction.*

If F is logarithmic space computable function, we say that L_1 is *log-space reducible* to L_2, denoted by $L_1 \leqslant_{\log} L_2$.

If L_1 and L_2 are in **NP** with certificate languages K_1 and K_2, respectively, we say that F is *parsimonious*, if for every $w \in \Sigma^*$, w has the same number of certificates in K_1 as $F(w)$ in K_2.

Definition 1.4 Let L be a language.

- L is **NP**-*hard*, if for every $L' \in \mathbf{NP}$, $L' \leqslant_p L$.
- L is **NP**-*complete*, if $L \in \mathbf{NP}$ and L is **NP**-hard.

Recall that a propositional formula (Boolean expression) with variables $x_1, \ldots, x_n$ is in Conjunctive Normal Form (CNF), if it is of the form: $\bigwedge_i \bigvee_j \ell_{i,j}$ where each $\ell_{i,j}$ is a literal, i.e., a variable x_k or its negation $\neg x_k$. It is in 3-CNF, if it is of the form $\bigwedge_i \big(\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}\big)$. A formula φ is satisfiable, if there is an assignment of Boolean values true or false to each variable in φ that evaluates to true.

SAT	
Input:	A propositional formula φ in CNF.
Task:	Output true, if φ is satisfiable. Otherwise, output false.

3-SAT	
Input:	A propositional formula φ in 3-CNF.
Task:	Output true, if φ is satisfiable. Otherwise, output false.

Obviously, SAT can be viewed as a language, i.e., $\mathsf{SAT} \overset{\mathsf{def}}{=} \{\varphi : \varphi \text{ is satisfiable CNF formula}\}$. Likewise, for $\mathsf{3\text{-}SAT}$.

Theorem 1.5 (Cook 1971, Levin 1973) SAT *and* $\mathsf{3\text{-}SAT}$ *are* $\mathbf{NP}$*-complete.*

3 Ladner's theorem: NP-intermediate language

Theorem 1.6 (Ladner 1975) *If* $\mathbf{P} \neq \mathbf{NP}$*, then there is* $L \in \mathbf{NP}$ *such that* $L \notin \mathbf{P}$ *and* L *is not* $\mathbf{NP}$*-complete.*

For a function $H : \mathbb{N} \to \mathbb{N}$, define SAT_H as follows.

$$\mathsf{SAT}_H \;\overset{\mathsf{def}}{=}\; \{\varphi 0 \underbrace{1 \cdots 1}_{n^{H(n)}} : \varphi \in \mathsf{SAT} \text{ and } |\varphi| = n\}$$

We define $H : \mathbb{N} \to \mathbb{N}$ such that SAT_H is the language L required in Theorem 1.6. For every $n \geqslant 1$, the value $H(n)$ is defined by **Algorithm 1** below. Here $\mathcal{M}_i$ is the DTM whose encoding is the binary representation of i.

Algorithm 1

Input: 1^n, where $n \geqslant 1$.

Task: Compute $1^{H(n)}$.

```
1: for $i = 1, \ldots, \log\log(n) - 1$ do
2:   Let $\mathcal{M}_i$ be the $i^{\text{th}}$ (1-tape) DTM.
3:   for all $x \in \{0,1\}^*$ where $|x| \leqslant \log n$ do
4:     Compute $\mathsf{SAT}_H(x)$.
5:     Simulate $\mathcal{M}_i$ on $x$ in $i|x|^i$ steps (using the UTM in Theorem 0.7).
6:   if the results in lines 4 and 5 agree on all $x \in \{0,1\}^*$ where $|x| \leqslant \log n$ then
7:     return $i$.
8: return $\log\log n$.
```

Lemma 1.7

- **Algorithm 1** *runs in polynomial time and* $\mathsf{SAT}_H \in \mathbf{NP}$.
- $\mathsf{SAT}_H \in \mathbf{P}$ *if and only if* $H(n) = O(1)$.
- $\mathsf{SAT}_H \notin \mathbf{P}$ *and* SAT_H *is not* $\mathbf{NP}$*-complete.*

4 TM with oracles

A TM $\mathcal{M}$ with oracle access to a language K, denoted by $\mathcal{M}^K$, is a TM with a special tape called *oracle tape* and three special states $q_{\mathsf{query}}, q_{\mathsf{yes}}, q_{\mathsf{no}}$. Each time it is in q_{query}, it moves to q_{yes}, if $w \in K$ and to q_{no}, if $w \notin K$, where w is the string found in the oracle tape. In other words, when it is in q_{query}, the machine can "query" the membership of the language K. Regardless of the choice of K, such query counts only as one step. We denote by $L(\mathcal{M}^K)$ the language accepted by $\mathcal{M}^K$.

For a language K, we define the classes **P** and **NP** relativized to K as follows.

$$\begin{aligned}\mathbf{P}^K &\overset{\mathsf{def}}{=} \{L : \text{there is a polynomial time DTM } \mathcal{M}^K \text{ such that } L(\mathcal{M}^K) = L\}\\ \mathbf{NP}^K &\overset{\mathsf{def}}{=} \{L : \text{there is a polynomial time NTM } \mathcal{M}^K \text{ such that } L(\mathcal{M}^K) = L\}\end{aligned}$$

Theorem 1.8 (Baker, Gill, Solovay 1975) *There is language A and B such that $\mathbf{P}^A = \mathbf{NP}^A$ and $\mathbf{P}^B \neq \mathbf{NP}^B$.*

Proof. For a **PSPACE**-complete language A, we can show that $\mathbf{P}^A = \mathbf{NP}^A$.

To show the existence of B, we need the following notation. For a language $C \in \{0,1\}^*$, define $\mathsf{unary}(C) \overset{\mathsf{def}}{=} \{1^n : \text{there is } w \in C \text{ with length } n\}$. Obviously, for every $C \in \{0,1\}^*$, $\mathsf{unary}(C) \in \mathbf{NP}^C$.

The language B will be defined as $B \overset{\mathsf{def}}{=} \bigcup_{i\in\mathbb{N}} B_i$ where each B_i is a finite set defined inductively as follows. Each B_i is associated with an integer k_i such that $B_i = B \cap \{0,1\}^{\leqslant k_i}$. Here $\{0,1\}^{\leqslant k_1} \overset{\mathsf{def}}{=} \{w \in \{0,1\}^* : |w| \leqslant k_i\}$.

The base case is $B_0 = \emptyset$ and $k_0 = 0$. For the induction step, B_{i+1} is defined as follows, where we assume an enumeration of all oracle DTM $\mathcal{M}_0, \mathcal{M}_1, \ldots$.

- Let $n = k_i + 1$.
- Simulate oracle TM $\mathcal{M}_{i+1}$ on 1^n within $2^n/10$ steps.

  During the simulation $\mathcal{M}_{i+1}$ may query the oracle. If the query strings are of length $\leqslant k_i$, then answer according to B_i. For all the other query strings, answer "no."
- Let k_{i+1} be as follows.

$$k_{i+1} \overset{\mathsf{def}}{=} \begin{cases} n, & \text{if all the query strings has length } \leqslant k_i \\ m, & m \text{ is the maximal length of the query strings with length } \geqslant k_i + 1 \end{cases}$$

- If $\mathcal{M}_{i+1}$ accepts 1^n within $2^n/10$ steps, we set $B_{i+1} \overset{\mathsf{def}}{=} B_i$.
- If $\mathcal{M}_{i+1}$ does not accept 1^n within $2^n/10$ steps, we set $B_{i+1} \overset{\mathsf{def}}{=} B_i \cup \{w\}$, where $w \in \{0,1\}^n$ and w is not one of the query strings.

From the definition of B, we can show that $\mathsf{unary}(B) \notin \mathbf{P}^B$. ■

Appendix

A coNP-complete problems

Analogous to **NP**-complete, we can also define **coNP**-complete problems.

Definition 1.9 Let K be a language.

- K is **coNP**-*hard*, if for every $L \in \mathbf{coNP}$, $L \leqslant_p K$.
- K is **coNP**-*complete*, if $K \in \mathbf{coNP}$ and K is **coNP**-hard.

Note that for every language K, K is **NP**-complete if and only if its complement $\overline{K}$ is **coNP**-complete, where $\overline{K} \overset{\mathsf{def}}{=} \Sigma^* - K$. Thus, $\overline{\mathsf{SAT}} \overset{\mathsf{def}}{=} \{\varphi : \varphi \text{ is not satisfiable}\}$ is **coNP**-complete.

Lesson 2: The class NL

Theme: Some classical results on the class **NL**.

We recall the notion of *log-space reduction.* Let $F : \Sigma^* \to \Sigma^*$ be a function. We say that F is computable in logarithmic space, if there is a 3-tape DTM $\mathcal{M}$ such that on input word w, it works as follows.

- Tape 1 contains the input word w and its content never changes.
- There is a constant c such that $\mathcal{M}$ uses only $c \log |w|$ space in tape 2.
- The head in tape 3 can only "write" and move right, i.e., once it writes a symbol to a cell, the content of that cell will not change.

Tape 1 is called the *input tape*, tape 2 the *work tape* and tape 3 the *output tape.*

Definition 2.1 A language L is log-space reducible to another language K, denoted by $L \leqslant_{\log} K$, if there is a function $F : \Sigma^* \to \Sigma^*$ computable in logarithmic space such that for every $w \in \Sigma^*$, $w \in L$ if and only if $F(w) \in K$.

Remark 2.2 The relation $\leqslant_{\log}$ is transitive in the sense that if $L_1 \leqslant_{\log} L_2$ and $L_2 \leqslant_{\log} L_3$, then $L_1 \leqslant_{\log} L_3$.

Definition 2.3 Let K be a language.

- K is **NL**-*hard*, if for every language $L \in \mathbf{NL}$, $L \leqslant_{\log} K$.
- K is **NL**-*complete*, if $K \in \mathbf{NL}$ and K is **NL**-hard.

Define the following language PATH.

$$\mathsf{PATH} \stackrel{\mathsf{def}}{=} \{(G, s, t) : G \text{ is } \textit{directed} \text{ graph and there is a path in } G \text{ from vertex } s \text{ to vertex } t\}$$

Theorem 2.4 PATH *is* **NL**-*complete.*

Theorem 2.5 (Savitch 1970) NL $\subseteq$ $\textsc{Dspace}[\log^2 n]$.

To prove Theorem 2.5, it suffices to show that $\mathsf{PATH} \in \textsc{Dspace}[\log^2 n]$. See Appendix A.

Theorem 2.6 (Immerman 1988 and Szelepcsényi 1987) NL = coNL.

To prove Theorem 2.6, we consider the complement language of PATH:

$$\overline{\mathsf{PATH}} \stackrel{\mathsf{def}}{=} \{(G, s, t) : G \text{ is } \textit{directed} \text{ graph and there is } \textit{no} \text{ path in } G \text{ from vertex } s \text{ to vertex } t\}$$

Note that $\overline{\mathsf{PATH}}$ is **coNL**-complete. To prove Theorem 2.6, it suffices to show that $\overline{\mathsf{PATH}} \in \mathbf{NL}$. See Appendix B.

Appendix

A Proof of Theorem 2.5

Algorithm 1 below decides the language PATH.

Algorithm 1

```
Input: $(G, s, t)$, where $G$ is a directed graph and $s$ and $t$ are two vertices in $G$.
Task: ACCEPT iff there is a path in $G$ from $s$ to $t$.
Let $n$ be the number of vertices in $G$.
ACCEPT iff $\textsc{Check}_G(s, t, \lceil \log n \rceil) = \mathsf{true}$.
```

It uses Procedure $\textsc{Check}_G$ defined below.

Procedure $\textsc{Check}_G$

```
Input: $(u, v, k)$ where $u$ and $v$ are two vertices in $G$, and $k$ is an integer $\geqslant 0$.
Task: Return true, if there is a path in $G$ of length $\leqslant 2^k$ from $u$ to $v$. Otherwise, return false.
if $k = 0$ then
    return true iff ($u = v$ or $(u, v)$ is an edge in $G$).
for all vertex $x$ in $G$ do
    $b := \textsc{Check}_G(u, x, k - 1)$.
    if $b = \mathsf{true}$ then
        $b := \textsc{Check}_G(x, v, k - 1)$.
        if $b = \mathsf{true}$ then
            return true.
return false.
```

Note that when computing $\textsc{Check}_G(u, x, k-1)$ and $\textsc{Check}_G(x, v, k-1)$, Procedure $\textsc{Check}_G$ can use the same space. Thus, it uses only $O(k \log n)$ space. Since k is initialized with $\lceil \log n \rceil$, **Algorithm 1** uses $O(\log^2 n)$ space in total.

B Proof of Theorem 2.6

Consider the following algorithm.

Algorithm No-path

```
Input: $(G, s, t)$ where $G$ is directed graph and $s$ and $t$ are two vertices in $G$.
Task: ACCEPT iff there is *no* path in $G$ from $s$ to $t$.
$m :=$ the number of vertices in $G$ reachable from $s$.
{Note: This value $m$ is computed with Procedure $\textsc{Count-Vertex}_G$ below.}
for all vertex $x$ in $G$ do
    Guess if $x$ is reachable from $s$.
    if the guess is "yes" then
        $m := m - 1$.
        Guess a path from $s$ to $x$.
        if it is not possible to guess such a path then REJECT.
        if there is such a path and $x = t$ then REJECT.
ACCEPT iff $m = 0$.
```

The number of vertices reachable from s can be computed with Procedure COUNT-VERTEX$_G$ defined below.

```
Procedure COUNT-VERTEX$_G$
Input: $u$ where $u$ is a vertex in $G$.
Task: Return the number of vertices in $G$ reachable from vertex $u$, where the number is written
    in binary form.
 1: Let $n$ be the number of vertices in $G$.
 2: $m := 1 +$ the outdegree of $u$.
 3: {Note: $m$ is initialized with the number of vertices reachable from $u$ in $\leqslant 1$ steps.}
 4: for $i = 2, \ldots, n$ do
 5:     $m' := 0$.
 6:     for all vertex $x$ in $G$ do
 7:         Guess if there is a path from $u$ to $x$ with length $\leqslant i$.
 8:         if the guess is "yes" then
 9:             Verify it by guessing such a path (of length $\leqslant i$).
10:             $m' := m' + 1$.
11:         if the guess is "no" then
12:             Verify that indeed there is no such a path (of length $\leqslant i$).
13:     $m := m'$.
14:     {Note: On each iteration, $m$ is the number of vertices reachable from $u$ in $\leqslant i$ steps.}
15: return $m$
```

The verification in Line 12 above is done with the following procedure.

```
Procedure VERIFY$_G$
Input: $(u, x, m, i)$ where $u$ and $x$ are vertices in $G$, $i \geqslant 2$ is an integer and $m$ is the number of
    vertices in $G$ reachable from $u$ in $\leqslant i - 1$ steps.
Task: Verify that $x$ is not reachable from $u$ in $\leqslant i$ steps.
 1: $\ell := m$.
 2: for all vertex $y$ in $G$ do
 3:     Guess if there is a path from $u$ to $y$ with length $\leqslant i - 1$.
 4:     if the guess is "yes" then
 5:         $\ell := \ell - 1$.
 6:         Guess a path (of length $\leqslant i - 1$) from $u$ to $y$.
 7:         Verify that the edge $(y, x)$ does not exist in $G$.
 8: Verification is complete iff $\ell = 0$.
```

Note that if any of the verification in Lines 9 and 12 in Procedure COUNT-VERTEX$_G$ and Line 7 in Procedure VERIFY$_G$ fails, the whole algorithm rejects immediately.

The correctness of Procedure COUNT-VERTEX$_G$ can be established by induction on i. The correctness of Algorithm NO-PATH follows immediately from COUNT-VERTEX$_G$.

Lesson 3: The class PSPACE

Theme: Some classical results on the class **PSPACE**.

Definition 3.1 Let K be a language.

- K is **PSPACE**-*hard*, if for every language $L \in \mathbf{PSPACE}$, $L \leqslant_p K$.
- K is **PSPACE**-*complete*, if $K \in \mathbf{PSPACE}$ and K is **PSPACE**-hard.

Quantified Boolean formulas (QBF) are formulas of the form:

$$Q_1 x_1 \ Q_2 x_2 \ \cdots \ Q_n x_n \ \varphi(x_1, \ldots, x_n)$$

where each $Q_i \in \{\forall, \exists\}$ and $\varphi(x_1, \ldots, x_n)$ is a Boolean formula with variables $x_1, \ldots, x_n$.
The intuitive meaning of each Q_i is as follows.

- $\forall x \ \psi$ means that for all $x \in \{\mathsf{true}, \mathsf{false}\}$, ψ is true.
- $\exists x \ \psi$ means that there is $x \in \{\mathsf{true}, \mathsf{false}\}$ such that ψ is true.

We define the problem TQBF:

TQBF	
Input:	A QBF φ.
Task:	Return true, if φ is true. Otherwise, return false.

As usual, it can be viewed as a language $\mathsf{TQBF} \stackrel{\mathsf{def}}{=} \{\psi : \psi \text{ is a true QBF}\}$. Note also that the usual Boolean formula can be viewed as a QBF, where each Q_i is $\exists$. Thus, TQBF is a more general problem than SAT.

Theorem 3.2 (Stockmeyer and Meyer 1973) TQBF *is* **PSPACE**-*complete.*

Theorems 3.3 and 3.4 below are the polynomial space analog of Theorem 2.5 and 2.6, respectively. In fact, they can be easily generalized to the so called *time* and *space constructible functions.* See Appendix A.

Theorem 3.3 (Savitch 1970) $\textsc{Nspace}[n^k] \subseteq \textsc{Dspace}[n^{2k}]$.

Theorem 3.4 (Immerman 1988 and Szelepcsényi 1987) $\textsc{Nspace}[n^k] = \text{co}\textsc{Nspace}[n^k]$.

Note that Theorem 3.3 implies **PSPACE** = **NPSPACE** = **coNPSPACE**.

Appendix

A Time and space constructible functions

Definition 3.5 Let $T : \mathbb{N} \to \mathbb{N}$ be a function.

- We say that T is *time constructible*, if for every n, $T(n) \geqslant n$ and there is a DTM that on input 1^n computes $1^{T(n)}$ in time $O(T(n))$.
- We say that T is *space constructible*, if there is a DTM that on input 1^n computes $1^{T(n)}$ in space $O(T(n))$.

Intuitively, when we say that $\mathcal{M}$ runs in time/space $O(T(n))$, where T is time/space constructible function, we can assume that on input word w, $\mathcal{M}$ first "computes" the amount of time/space needed to decide w, before going on to process w.

Theorems 3.3 and 3.4 can be easily generalized to space constructible functions as follows.

Theorem 3.6 *Let $f : \mathbb{N} \to \mathbb{N}$ be space constructible function such that $f(n) \geqslant \log n$, for every n.*

- **(Savitch 1970)** $\textsc{Nspace}[f(n)] \subseteq \textsc{Dspace}[f(n)^2]$.
- **(Immerman 1988 and Szelepcsényi 1987)** $\textsc{Nspace}[f(n)] = \text{co}\textsc{Nspace}[f(n)]$.

B Hardness via log space reduction

In our definition of hardness for **NP**, **coNP** and **PSPACE**, we require that the reduction is polynomial time reduction. It is also common to define hardness by insisting the reduction is log-space reduction. That is, we can define K as **NP**-hard by insisting $L \leqslant_{\log} K$, for every $L \in \mathbf{NP}$, rather than $L \leqslant_p K$. Similarly, for **coNP** and **PSPACE**.

Most **NP**-, **coNP**- and **PSPACE**-complete problems are known to remain complete even under log-space reduction, including SAT, 3-SAT and TQBF.

- SAT and 3-SAT are **NP**-complete under log-space reduction.
- TQBF is **PSPACE**-complete under log-space reduction.

Lesson 4: Alternating Turing machines

Theme: The notion of alternating Turing machine and its relation with DTM.

1 Definition

A 1-tape *alternating Turing machine* (ATM) is a system $\mathcal{M} = \langle \Sigma, \Gamma, Q, U, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}, \delta \rangle$, where each component is as follows.

- $\Sigma = \{0,1\}$ and $\Gamma = \{0,1,\sqcup\}$ are the input and tape alphabets, respectively.
- Q is a finite set of states.
- $U \subseteq Q$ is a finite subset of Q.
- $q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}$ are the initial state, accepting state and rejecting state, respectively.
- $\delta \subseteq (Q - \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}) \times \Gamma \times Q \times \Gamma \times \{\mathtt{Left}, \mathtt{Right}\}$.

Note that ATM is very much like NTM, except that it has one extra component U. The states in U are called *universal* states, and the states in $Q - U$ are called *existential* states. As in DTM/NTM, for convenience, we assume that the tape is 2-way infinite.

The notions of *initial/halting/accepting/rejecting* configuration are defined similarly as in NTM/DTM. A configuration C is called *existential/universal* configuration, if the the state in C is an existential/universal state. The notion of "one step computation" $C \vdash C'$ for ATM is also similar to the one for DTM/NTM. When $C \vdash C'$, we say that C' is one of the next configuration of C (w.r.t. $\mathcal{M}$).

On input word w, *the run of* $\mathcal{M}$ *on* w is a *tree* T where each node in the tree is labelled with a configuration of $\mathcal{M}$ according to the following rules.

- The root node of T is labelled with the initial configuration of $\mathcal{M}$ on w.
- Every other node x in T is labelled as follows.

  If x is labelled with a configuration C and $C_1, \ldots, C_n$ are all the next configurations of C, then x has n children $y_1, \ldots, y_n$ labelled with $C_1, \ldots, C_n$, respectively.

Note that if x is labelled with C that does not have next configuration, then it is a leaf node, i.e., it does not have any children.

Let T be the run of $\mathcal{M}$ on w and let x be a node in T. We say that x *leads to acceptance*, if the following holds.

- x is a leaf node labelled with an accepting configuration.
- If x is labelled with an existential configuration, then one of its children leads to acceptance.
- If x is labelled with a universal configuration, then all of its children lead to acceptance.

We say that T is *accepting run*, if its root node leads to acceptance. The ATM $\mathcal{M}$ accepts w, if the run of $\mathcal{M}$ on w is accepting run. As before, $L(\mathcal{M}) \overset{\mathsf{def}}{=} \{w : \mathcal{M} \text{ accepts } w\}$.

Note that NTM is simply ATM where all the states are existential, and DTM is simply NTM where every configuration (except the accepting/rejecting configuration) has exactly one next configuration. The generalization of ATM to multiple tapes is straightforward.

2 Time and space complexity for ATM

Let $\mathcal{M}$ be a ATM, $w \in \Sigma^*$, $t \in \mathbb{N}$ and let $f : \mathbb{N} \to \mathbb{N}$ be a function.

- *$\mathcal{M}$ decides w in time t (or, in t steps)*, if the run of $\mathcal{M}$ on w has depth at most t.
- *$\mathcal{M}$ decides w in space t (or, uses t cells/space)*, if in the run of $\mathcal{M}$ on w, every node is labelled with configuration of length t.
- *$\mathcal{M}$ runs in time/space $O(f(n))$*, if there is $c > 0$ such that for sufficiently long word w, $\mathcal{M}$ decides w in time/space $c \cdot f(|w|)$.
- *$\mathcal{M}$ decides a language L in time/space $O(f(n))$*, if $\mathcal{M}$ runs in time/space $O(f(n))$ and $L(\mathcal{M}) = L$.
- $\textsc{Atime}[f(n)] \stackrel{\text{def}}{=} \{L : \text{there is ATM } \mathcal{M} \text{ that decides } L \text{ in time } O(f(n))\}$.
- $\textsc{Aspace}[f(n)] \stackrel{\text{def}}{=} \{L : \text{there is ATM } \mathcal{M} \text{ that decides } L \text{ in space } O(f(n))\}$.

Analoguous to the DTM/NTM, we can define the classes of languages accepted by ATM run in algorithmic/polynomial/exponential time/space.

$$\begin{aligned}
\mathbf{AL} &\stackrel{\text{def}}{=} \{L : \text{there is ATM } \mathcal{M} \text{ that decides } L \text{ in space } O(\log n)\} \\
\mathbf{AP} &\stackrel{\text{def}}{=} \bigcup_{f(n)=\mathsf{poly}(n)} \textsc{Atime}[f(n)] \\
\mathbf{APSPACE} &\stackrel{\text{def}}{=} \bigcup_{f(n)=\mathsf{poly}(n)} \textsc{Aspace}[f(n)] \\
\mathbf{AEXP} &\stackrel{\text{def}}{=} \bigcup_{f(n)=\mathsf{poly}(n)} \textsc{Atime}[2^{f(n)}]
\end{aligned}$$

The following lemma links time/space complexity classes for ATM with those for DTM.

Lemma 4.1 *Let $T : \mathbb{N} \to \mathbb{N}$ and $S : \mathbb{N} \to \mathbb{N}$ such that $T(n) \geqslant n$ and $S(n) \geqslant \log n$, for every n.*

(a) $\textsc{Atime}[T(n)] \subseteq \textsc{Dspace}[T(n)]$.

(b) $\textsc{Dspace}[S(n)] \subseteq \textsc{Atime}[S(n)^2]$.

(c) $\textsc{Aspace}[S(n)] \subseteq \textsc{Dtime}[2^{O(S(n))}]$.

(d) $\textsc{Dtime}[T(n)] \subseteq \textsc{Aspace}[\log T(n)]$.

Proof. (a) and (c) is by straightforward simulation of ATM with DTM. (b) is similar to the proof of Savitch's theorem. (d) is similar to the proof of Theorem 4.3 below, i.e., by viewing the computation of DTM as a boolean circuit. ■

Theorem 4.2 (Chandra, Kozen, Stockmeyer 1981)

- $\mathbf{AL} = \mathbf{P}$.
- $\mathbf{AP} = \mathbf{PSPACE}$.
- $\mathbf{APSPACE} = \mathbf{EXP}$.
- $\mathbf{AEXP} = \mathbf{EXPSPACE}$.
- $\cdots$.

Appendix

A P-complete languages

Boolean circuits. Let $n \in \mathbb{N}$, where $n \geqslant 1$. An n-input *Boolean circuit* C is a directed acyclic graph with n *source* vertices (i.e., vertices with no incoming edges) and 1 *sink* vertex (i.e., vertex with no outgoing edge).

The source vertices are labelled with $x_1, \ldots, x_n$. The non-source vertices, called *gates*, are labelled with one of $\wedge, \vee, \neg$. The vertices labelled with $\wedge$ and $\vee$ have two incoming edges, whereas the vertices labelled with $\neg$ have one incoming edge. The size of C, denoted by $|C|$ is the number of vertices in C.

On input $w = x_1 \cdots x_n$, where each $x_i \in \{0,1\}$, we write $C(w)$ is the output of C on w, defined as interpretating $\wedge, \vee, \neg$ in the natural way and 0 and 1 as false and true, respectively.

(Boolean) straight line programs. It is sometimes more convenient to view a boolean circuit a straight line program. The following is an example of straight line program, where the input is $w = x_1 \cdots x_n$.

```
1:  $p_1 := x_1 \wedge x_3$.
2:  $p_2 := \neg x_4$.
3:  $p_3 := p_1 \vee p_2$.
 ⋮
$\ell$:  $p_\ell := p_i \wedge p_j$.
```

Intuitively, straight line programs are programs without **if** branch and **while** loop, hence, the name “straight line” programs. It is assumed that such program always outputs the value in the variable in the last line. In our example above, it outputs the value of variable p_ℓ.

Define the following problem.

CIRCUIT-EVAL	
Input:	An n input boolean circuit C and $w \in \{0,1\}^n$.
Task:	Output $C(w)$.

It can also be defined as the language CIRCUIT-EVAL $\overset{\text{def}}{=} \{(C,w) : C(w) = 1\}$.

For our proof of Theorem 4.3 below, it is also convenient to assume that vertices labelled with $\wedge$ and $\vee$ can have more than 2 incoming edges.

Theorem 4.3 CIRCUIT-EVAL *is* **P***-complete via log-space reductions.*

Proof. Follows the reduction for the **NP**-completeness of SAT. ■

Lesson 5: The polynomial hierarchy and the complexity classes for counting

Theme: The polynomial time hierarchy and the complexity classes for counting problems.

1 The polynomial hierarchy

For every integer $i \geqslant 1$, the class $\mathbf{\Sigma}_i^p$ is defined as follows. A language $L \subseteq \{0,1\}^*$ is in $\mathbf{\Sigma}_i^p$, if there is a polynomial $q(n)$ and a polynomial time DTM $\mathcal{M}$ such that for every $w \in \{0,1\}^*$, $w \in L$ if and only if the following holds.

$$\exists y_1 \in \{0,1\}^{q(|w|)}\ \forall y_2 \in \{0,1\}^{q(|w|)}\ \cdots\ Q y_i \in \{0,1\}^{q(|w|)}\ \mathcal{M} \text{ accepts } (w, y_1, \ldots, y_i) \qquad (1)$$

Here $Q = \exists$, if i is odd and $Q = \forall$, if i is even.

The class $\mathbf{\Pi}_i^p$ is defined as above, but the sequence of quantifiers in (1) starts with $\forall$. Alternatively, it can also be defined as $\mathbf{\Pi}_i^p \stackrel{\mathsf{def}}{=} \{\overline{L} : L \in \mathbf{\Sigma}_i^p\}$. Note that $\mathbf{NP} = \mathbf{\Sigma}_1^p$ and $\mathbf{coNP} = \mathbf{\Pi}_1^p$.

Remark 5.1 The class $\mathbf{\Sigma}_i^p$ can also be defined as follows. A language L is in $\mathbf{\Sigma}_i^p$, if there is a polynomial time ATM $\mathcal{M}$ that decides L such that for every input word $w \in \{0,1\}^*$, the run of $\mathcal{M}$ on w can be divided into i layers. Each layer consists of nodes of the same depth in the run. (Recall that the run of an ATM is a tree.) In the first layer all nodes are labeled with existential configurations, in the second layer with universal configurations, and so on. It is not difficult to show that this definition is equivalent to the one above.

The *polynomial time hierarchy* (or, in short, *polynomial hierarchy*) is defined as the following class.

$$\mathbf{PH} \stackrel{\mathsf{def}}{=} \bigcup_{i=1}^{\infty} \mathbf{\Sigma}_i^p$$

Note that $\mathbf{PH} \subseteq \mathbf{PSPACE}$.

It is conjectured that $\mathbf{\Sigma}_1^p \subsetneq \mathbf{\Sigma}_2^p \subsetneq \mathbf{\Sigma}_3^p \subsetneq \cdots$. In this case, we say that *the polynomial hierarchy does not collapse.* We say that *the polynomial hierarchy collapses*, if there is i such that $\mathbf{PH} = \mathbf{\Sigma}_i^p$, in which case we also say that *the polynomial hierarchy collapses to level i.*

We define the notion of hardness and completeness for each $\mathbf{\Sigma}_i^p$ as follows. For $i \geqslant 1$, a language K is $\mathbf{\Sigma}_i^p$*-hard*, if for every $L \in \mathbf{\Sigma}_i^p$, $L \leqslant_p K$. It is $\mathbf{\Sigma}_i^p$*-complete*, if it is in $\mathbf{\Sigma}_i^p$ and it is $\mathbf{\Sigma}_i^p$-hard. The same notion can be defined analoguously for $\mathbf{PH}$ and each $\mathbf{\Pi}_i^p$.

Define the language Σ_i-SAT as consisting of true QBF of the form:

$$\exists \bar{x}_1\ \forall \bar{x}_2\ \cdots\ Q \bar{x}_i\ \varphi(\bar{x}_1, \ldots, \bar{x}_i)$$

where $\varphi(\bar{x}_1, \ldots, \bar{x}_i)$ is quantifier-free Boolean formula and $Q = \exists$, if i is odd, and $Q = \forall$, if i is even. Here $\bar{x}_1, \ldots, \bar{x}_i$ are all vectors of boolean variables. In other words, Σ_i-SAT is a subset of TQBF where the number of quantifier alternation is limited to $(i-1)$. The language Π_i-SAT is defined analoguously with the starting quantifiers being $\forall$.

Theorem 5.2

- *For every $i \geqslant 1$, Σ_i-SAT is $\mathbf{\Sigma}_i^p$-complete and Π_i-SAT is $\mathbf{\Pi}_i^p$-complete.*
- *If $\mathbf{\Sigma}_i^p = \mathbf{\Pi}_i^p$ for some $i \geqslant 1$, then the polynomial hierarchy collapses.*
- *If there is language that is $\mathbf{PH}$-complete, then the polynomial hierarchy collapses.*

2 Complexity classes for counting problems

2.1 The class FP

We denote by **FP** the class of functions $f : \{0,1\}^* \to \mathbb{N}$ computable by polynomial time DTM. Here the convention is that a natural number is always represented in binary form. So, when we say that a DTM $\mathcal{M}$ computes a function $f : \{0,1\}^* \to \mathbb{N}$, on input word w, the output of $\mathcal{M}$ on w is $f(w)$ in the binary representation.

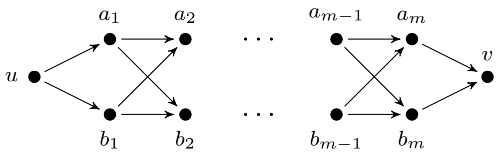
Let ♯CYCLE be the following problem.

♯CYCLE	
Input:	A directed graph G.
Task:	Output the number of cycles in G.

As before, ♯CYCLE can also be viewed as a function. Note also that the number of cycles in a graph with n vertices is at most exponential in n, thus, its binary representation only requires polynomially many bits.

Theorem 5.3 *If* ♯CYCLE *is in* **FP**, *then* $\mathbf{P} = \mathbf{NP}$.

Proof. Let G be a (directed) graph with n vertices. We construct a graph G' obtained by replacing every edge (u, v) in G with the following gadget:



Note that every simple cycle in G of length ℓ becomes $(2^m)^\ell$ cycles in G'. Now, let $m \overset{\mathsf{def}}{=} n \log n$.

It is not difficult to show that G has a hamiltonian cycle (i.e., a simple cycle of length n) if and only if G' has more than $n^{(n^2)}$ cycles. So, if ♯CYCLE $\in$ **FP**, then checking hamiltonian cycle can be done is in **P**. ■

Note that checking whether a graph has a cycle itself can be done in polynomial time. However, as Theorem 5.3 above states, it is unlikely that counting the number of cycles can be done in polynomial time.

2.2 The class ♯P

Definition 5.4 A function $f : \{0,1\}^* \to \mathbb{N}$ is in ♯**P**, if there is a polynomial $q(n)$ and a polynomial time DTM $\mathcal{M}$ such that for every word $w \in \{0,1\}^*$, the following holds.

$$f(w) \quad = \quad |\{y : \mathcal{M} \text{ accepts } (w,y) \text{ and } y \in \{0,1\}^{q(|w|)}\}|$$

Alternatively, we can say that f is in ♯**P**, if there is a polynomial time NTM $\mathcal{M}$ such that for every word $w \in \{0,1\}^*$, $f(w) =$ the number of accepting runs of $\mathcal{M}$ on w.

For a function $f : \{0,1\}^* \to \mathbb{N}$, the language associated with the function f, denoted by O_f, is defined as $O_f \overset{\mathsf{def}}{=} \{(w,i) : \text{the } i^{\text{th}} \text{ bit of } f(w) \text{ is } 1\}$. When we say that *a TM* $\mathcal{M}$ *has oracle access to a function* f, we mean that it has oracle access to the language O_f.

We define $\mathbf{FP}^f$ as the class of functions $g : \{0,1\}^* \to \mathbb{N}$ computable by a polynomial time DTM with oracle access to f.

Definition 5.5 Let $f : \{0,1\}^* \to \mathbb{N}$ be a function.

- f is $\sharp\mathbf{P}$-*hard*, if $\sharp\mathbf{P} \subseteq \mathbf{FP}^f$, i.e., every function in $\sharp\mathbf{P}$ is computable by a polynomial time DTM with oracle access to f.
- f is $\sharp\mathbf{P}$-*complete*, if $f \in \sharp\mathbf{P}$ and f is $\sharp\mathbf{P}$-hard.

Let $\sharp\mathsf{SAT}$ be the following problem.

$\sharp\mathsf{SAT}$	
Input:	A boolean formula φ.
Task:	Output the number of satisfying assignments for φ.

As before, the output numbers are to be written in binary form. We can also view $\sharp\mathsf{SAT}$ as a function $\sharp\mathsf{SAT} : \{0,1\}^* \to \mathbb{N}$, where $\sharp\mathsf{SAT}(\varphi) =$ the number of satisfying assignment for φ.

Theorem 5.6 $\sharp\mathsf{SAT}$ *is* $\sharp\mathbf{P}$-*complete.*

Proof. Cook-Levin reduction (to prove the $\mathbf{NP}$-hardness of SAT) is parsimonious. ■

There are usually two ways to prove a certain function is $\sharp\mathbf{P}$-hard, as stated in Remark 5.7 and 5.8 below.

Remark 5.7 Let f_1 and f_2 be functions from $\{0,1\}^*$ to $\mathbb{N}$. Suppose L_1 and L_2 be languages in $\mathbf{NP}$ such that f_1 and f_2 are the functions for the number of certificates for L_1 and L_2, respectively. That is, for every word $w \in \{0,1\}^*$,

$$f_i(w) \quad = \quad \text{the number of certificates of } w \text{ in } L_i, \qquad\qquad \text{for } i = 1,2.$$

If f_1 is $\sharp\mathbf{P}$-hard and there is a parsimonious (polynomial time) reduction from L_1 to L_2, then f_2 is $\sharp\mathbf{P}$-hard.

Remark 5.8 Let f and g be two functions from $\{0,1\}^*$ to $\mathbb{N}$. If f is $\sharp\mathbf{P}$-hard and $f \in \mathbf{FP}^g$, then g is $\sharp\mathbf{P}$-hard.

Since there is a parsimonious reduction from SAT to $\mathsf{3\text{-}SAT}$, by Theorem 5.6 and Remark 5.7, we have the following corollary.

Corollary 5.9 $\sharp\mathsf{3\text{-}SAT}$ *is* $\sharp\mathbf{P}$-*complete.*

Corollary 5.9 can also be proved by showing $\sharp\mathsf{SAT} \in \mathbf{FP}^{\sharp\mathsf{3\text{-}SAT}}$.

Lesson 6: Computing permanent

Theme: The complexity of computing the permanent of a matrix.

1 Definitions

For an integer $n \geqslant 1$, let $[n] = \{1, \ldots, n\}$. The *permanent* of an $n \times n$ matrix A over integers is defined as:

$$\mathrm{per}(A) \stackrel{\mathsf{def}}{=} \sum_{\sigma} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

where σ ranges over all permutation on $[n]$. Here $A_{i,j}$ denotes the entry in row i and column j in matrix A.

Consider the following problem.

PERM	
Input:	A square matrix A over integers.
Task:	Output the permanent of A.

We denote it by 0|1-PERM, when the entries in the input matrix A are restricted to 0 or 1.

Theorem 6.1 (Valiant 1979) 0|1-PERM *is* $\sharp\mathbf{P}$*-complete.*

To show that 0|1-PERM is in $\sharp\mathbf{P}$, consider the following algorithm.

```
Input: A 0-1 matrix $A$.
1: Guess a permutation $\sigma$ on $[n]$, i.e., for each $i \in [n]$, guess a value $v_i \in [n]$.
2: If the guessed $\sigma$ is not a permutation, REJECT.
3: Compute the value $\prod_{i=1}^{n} A_{i,\sigma(i)}$.
4: ACCEPT if and only if the value is 1.
```

It is obvious that on input A, the number of accepting runs is the same as $\mathrm{per}(A)$.

2 Combinatorial view of permanent

Let $G = (V, E, w)$ be a complete directed graph, i.e., $E = V \times V$, and each edge (u, v) has a weight $w(u, v) \in \mathbb{Z}$. We write a (simple) cycle as a sequence $p = (u_1, \ldots, u_\ell)$, and its weight is defined as:

$$w(p) \stackrel{\mathsf{def}}{=} w(u_1, u_2) \cdot w(u_2, u_3) \cdot \ldots \cdot w(u_{\ell-1}, u_\ell) \cdot w(u_\ell, u_1)$$

A loop (u, u) is considered a cycle.

A *cycle cover* of G is a set $R = \{p_1, \ldots, p_k\}$ of pairwise disjoint cycles such that for every vertex $u \in V$, there is a cycle $p_j \in R$ such that u appears in p_j. The weight R is defined as:

$$w(R) \stackrel{\mathsf{def}}{=} \prod_{p_j \in R} w(C_j)$$

Note that a cycle or a cycle cover can also be viewed as a set of edges.

Assuming that the vertices in G are $\{1, \ldots, n\}$, let A be the adjacency matrix of G, i.e., A is an $(n \times n)$ matrix over $\mathbb{Z}$ such that $A_{i,j} = w(i, j)$.

A permutation $\sigma = (d_{1,1}, \ldots, d_{1,k_1}), \ldots, (d_{l,1}, \cdots, d_{l,k_l})$ on $[n]$ can be viewed as a cycle cover whose weight is exactly the value $\prod_{i\in[n]} A_{i,\sigma(i)}$. Thus, we have the equation:

$$\mathrm{per}(A) \quad = \sum_{R \text{ is a cycle cover of } G} w(R)$$

3 Reduction from 3-SAT to cycle cover

In this section we will show how to encode 3-SAT as the cycle cover problem.

3.1 Overview of the main idea

Let Ψ be a formula in 3-CNF. Let $x_1, \ldots, x_n$ be the variables and $C_1, \ldots, C_m$ be the clauses. We will construct a complete directed graph $G = (V, E, w)$, where the weight of each edge can be arbitrary integer and every boolean assignment $\phi : \{x_1, \ldots, x_n\} \to \{0, 1\}$ is associated with a set F_ϕ of cycle covers of G such that the following holds.

- For two different assignments ϕ_1, ϕ_2, the sets F_{ϕ_1} and F_{ϕ_2} are disjoint.
- If ϕ is a satisfying assignment for Ψ, the total weight of cycle covers in F_ϕ is 4^{3m}, i.e.,

$$\sum_{R\in F_\phi} w(R) \quad = \quad 4^{3m}$$

- If ϕ is not a satisfying assignment for Ψ, the total weight of cycle covers in F_ϕ is 0, i.e.,

$$\sum_{R\in F_\phi} w(R) \quad = \quad 0$$

- The total weight of cycle covers not in any F_ϕ is 0, i.e.,

$$\sum_{R\notin F_\phi \text{ for any } \phi} w(R) \quad = \quad 0$$

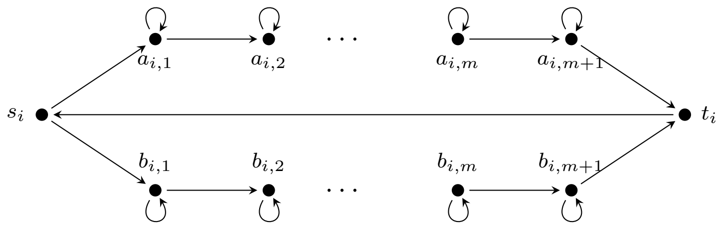
If A is the adjacency matrix of G, it is clear that:

$$\mathrm{per}(A) \quad = \quad 4^{3m} \times (\text{the number of satisfying assignment for } \Psi)$$

3.2 The construction of the graph G

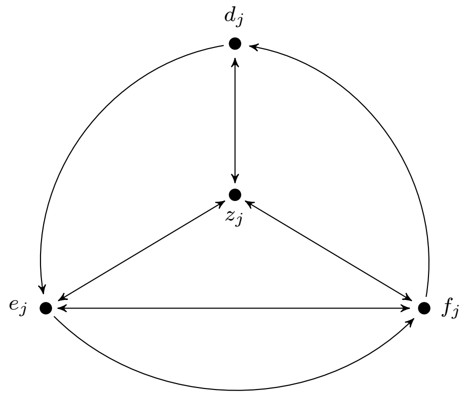
In the following we will draw an edge with a label indicating its weight. If the label is missing, it means the weight is 1. When an edge is not drawn, it means the weight is 0.

Variable gadget. For each variable x_i, we have the following "variable gadget":



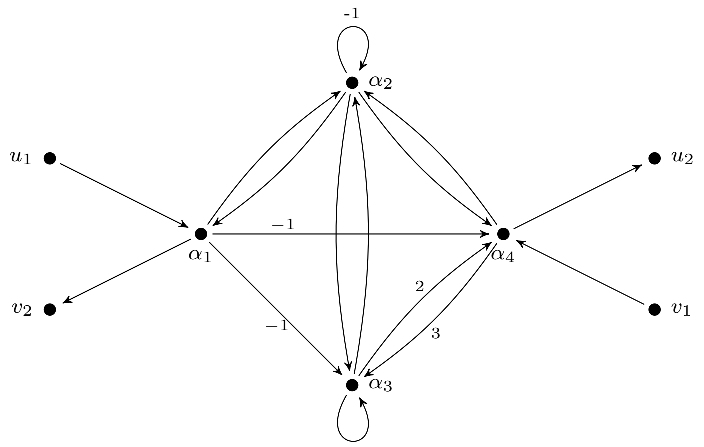
The upper edges, i.e., $(a_{i,1}, a_{i,2}), \ldots, (a_{i,m}, a_{i,m+1})$, are called the *external "true" edges* of x_i, and the lower edges, i.e., $(b_{i,1}, b_{i,2}), \ldots, (b_{i,m}, b_{i,m+1})$, the *external "false" edges* of x_i.

Clause gadget. For each clause C_j, we have the following "clause gadget":



The "outer" edges $(d_j, e_j), (e_j, f_j), (f_j, d_j)$ are intended to represent the literals in C_j. If ℓ_1, ℓ_2, ℓ_3 are the literals in C_j, then their associated edges are $(d_j, e_j), (e_j, f_j), (f_j, d_j)$, respectively. To avoid clutter, we will call those edges ℓ_1-edge, ℓ_2-edge and ℓ_3-edge, respectively.

The XOR operator. We also have the "XOR operator" between two edges (u_1, u_2) and (v_1, v_2):



Definition 6.2 Let H be a graph, and let (u_1, u_2) and (v_1, v_2) are two non-adjacent edges in H.

- For a cycle cover R of H, we say that R *respects the property* $(u_1, u_2) \oplus (v_1, v_2)$, if R contains exactly one of (u_1, u_2) or (v_1, v_2).

- Let H' denotes the graph obtained from H by replacing the edges $(u_1,u_2),(v_1,v_2)$ with the edges in the XOR operator above.

  A cycle cover R' of H' is *an associated cycle cover* of R, if it satisfies the following condiiton.

  - If R contains (u_1,u_2), then R' contains a path from u_1 to u_2.
  - If R contains (v_1,v_2), then R' contains a path from v_1 to v_2.
  - $R \setminus \{(u_1,u_2),(v_1,v_2)\} \subseteq R'$.

Lemma 6.3 *Let H, H', R and $(u_1,u_2),(v_1,v_2)$ be as in Definition 6.2. Then, the following holds.*

$$\sum_{R' \text{ is associated with } R} w(R') \;=\; \begin{cases} 4w(R), & \text{if } R \text{ respects } (u_1,u_2)\oplus(v_1,v_2) \\ 0, & \text{otherwise} \end{cases}$$

Constructing the graph G. The graph G is defined as the disjoint union of all the variable and clause gadgets and the following additional edges to connect them: For every clause C_j, for every literal ℓ in C_j, if $\ell = x_i$, we "connect" the ℓ-edge in the clause gadget of C_j with the edge $(a_{i,j}, a_{i,j+1})$ via the XOR operator; and if $\ell = \neg x_i$, we "connect" it with the edge $(b_{i,j}, b_{i,j+1})$.

For an assignment $\phi : \{x_1,\ldots,x_n\} \to \{0,1\}$, we say that *a cycle cover R is associated with* ϕ, if the following holds for every variable x_i.

- If $\phi(x_i)=1$, the cycle $(s_i, a_{i,1},\ldots,a_{i,m+1},t_i)$ is in R.
- If $\phi(x_i)=0$, the cycle $(s_i, b_{i,1},\ldots,b_{i,m+1},t_i)$ is in R.

Lemma 6.4 *For every assignment $\phi : \{x_1,\ldots,x_n\} \to \{0,1\}$, the following holds.*

$$\sum_{R \text{ is associated with } \phi} w(R) \;=\; \begin{cases} 4^{3m}, & \text{if } \phi \text{ is satisfying assignment for } \Psi \\ 0, & \text{if } \phi \text{ is not} \end{cases}$$

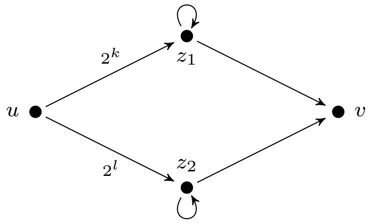
Combining Lemmas 6.3 and 6.4, it is immediate that the following holds.

$$\mathrm{per}(A) \;=\; 4^{3m} \times (\text{the number of satisfying assignments for } \Psi)$$

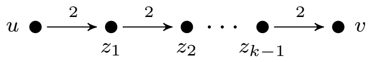
Here A is the adjacency matrix of G.

4 Reduction from matrices over $\mathbb{Z}$ to matrices over $\{0,1\}$

Reduction to matrices over integers of the form -2^k, 0 or 2^k. For each edge (u,v) with weight $2^k + 2^l$, we can replace it with 2 "parallel" edges with weights 2^k and 2^l, respectively.



Reduction to matrices over integers of the form -1, 0 or 1. For each edge (u, v) with weight 2^k, we can replace it with k "series" edges, each with weights 2.



Each weight 2 edge can be further reduced to weight 1 edge as above.

Reduction to matrices over $\{0, 1\}$, but on modular arithmetic. The permanent of an $n \times n$ matrix A over $\{-1, 0, 1\}$ can only in between $-n!$ and $n!$. Let $m = n^2$. Since $2^m + 1 > 2n!$, it is sufficient to compute $\text{per}(A)$ in $\mathbb{Z}_{2^m+1}$. Since $-1 \equiv 2^m \pmod{2^m+1}$, we can replace each -1 with 2^m, which can then be reduced to 1 as above.

5 Putting all the pieces together

Putting together all the pieces, we design a polynomial time algorithm to compute ♯3-SAT (with oracle access to language O_{per}, i.e., the language associated with permanent) . On input 3-CNF formula Ψ, do the following.

- Let n and m be the number of variables and clauses in Ψ.
- Construct a matrix A over $\{-1, 0, 1\}$ such that $\text{per}(A)$ is 4^{3m} times the number of satisfying assignments for Ψ.
- Let m be an integer for which we can compute $\text{per}(A)$ modulo $2^m + 1$.
- Let A' be the matrix obtained by replacing every -1 in A with 2^m.
- Compute $\text{per}(A')$ by querying the oracle on each bit.
- Let Z be the remainder of $\text{per}(A')$ divided by $2^m + 1$.
- Divide Z by 4^{3m}, and output it.

Lesson 7: Boolean circuits part. 1

Theme: Some classical results on boolean circuits.

Let $n \in \mathbb{N}$, where $n \geqslant 1$. An n-input *Boolean circuit* C is a directed acyclic graph with n *source* vertices (i.e., vertices with no incoming edges) and 1 *sink* vertex (i.e., vertex with no outgoing edge).

The source vertices are labelled with $x_1, \ldots, x_n$. The non-source vertices, called *gates*, are labelled with one of $\wedge, \vee, \neg$. The vertices labelled with $\wedge$ and $\vee$ have *two* incoming edges, whereas the vertices labelled with $\neg$ have one incoming edge. The size of C, denoted by $|C|$, is the number of vertices in C.

On input $w = x_1 \cdots x_n$, where each $x_i \in \{0,1\}$, we write $C(w)$ to denote the output of C on w, where $\wedge, \vee, \neg$ are interpreted in the natural way and 0 and 1 as false and true, respectively.

We refer to the in-degree and out-degree of vertices in a circuit as *fan-in* and *fan-out*, respectively. In our definition above, we require fan-in 2.

- A circuit family is a sequence $\{C_n\}_{n \in \mathbb{N}}$ such that every C_n has input n inputs and a single output.

  To avoid clutter, we write $\{C_n\}$ to denote a circuit family.

- We say that $\{C_n\}$ *decides a language* L, if for every $n \in \mathbb{N}$, for every $w \in \{0,1\}^n$, $w \in L$ if and only if $C_n(w) = 1$.

- We say that $\{C_n\}$ *is of size* $T(n)$, where $T : \mathbb{N} \to \mathbb{N}$ is a function, if $|C_n| \leqslant T(n)$, for every $n \in \mathbb{N}$.

We define the following class.

$$\mathbf{P}_{/\mathsf{poly}} \overset{\mathsf{def}}{=} \big\{L : L \text{ is decided by } \{C_n\} \text{ of size } q(n) \text{ for some polynomial } q(n)\big\}$$

That is, the class of languages decided by a circuit family of polynomial size.

Remark 7.1 It is not difficult to show that *every* unary language L is in $\mathbf{P}_{/\mathsf{poly}}$. Thus, $\mathbf{P}_{/\mathsf{poly}}$ contains some undecidable language.

Definition 7.2 A circuit family $\{C_n\}$ is $\mathbf{P}$-*uniform*, if there is a polynomial time DTM that on input 1^n, output the description of the circuit C_n.

Theorem 7.3 *A language L is in $\mathbf{P}$ if and only if it is decided by a $\mathbf{P}$-uniform circuit family.*

Theorem 7.4 (Karp and Lipton 1980) *If $\mathbf{NP} \subseteq \mathbf{P}_{/\mathsf{poly}}$, then $\mathbf{PH} = \mathbf{\Sigma}_2^p$.*

Theorem 7.5 (Meyer 1980) *If $\mathbf{EXP} \subseteq \mathbf{P}_{/\mathsf{poly}}$, then $\mathbf{EXP} = \mathbf{\Sigma}_2^p$.*

Theorem 7.6 (Shannon 1949) *For every $n > 1$, there is a function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit of size $2^n/(10n)$.*

The classes NC and AC. For a circuit C, the *depth* of C is the length of the longest directed path from an input vertex to the output vertex.[*] For a function $T : \mathbb{N} \to \mathbb{N}$, we say that a circuit family $\{C_n\}$ has depth $T(n)$, if for every n, the depth of C_n is $\leqslant T(n)$.

For every i, the classes $\mathbf{NC}^i$ and $\mathbf{AC}^i$ are defined as follows.

- A language L is in $\mathbf{NC}^i$, if there is $f(n) = \mathsf{poly}(n)$ such that L is decided by a circuit family of size $f(n)$ and depth $O(\log^i n)$.
- The class $\mathbf{AC}^i$ is defined analogously, except that gates in the circuits are allowed to have unbounded fan-in.

The classes **NC** and **AC** are defined as follows.

$$\mathbf{NC} \overset{\mathsf{def}}{=} \bigcup_{i \geqslant 0} \mathbf{NC}^i \qquad \text{and} \qquad \mathbf{AC} \overset{\mathsf{def}}{=} \bigcup_{i \geqslant 0} \mathbf{AC}^i$$

Note that $\mathbf{NC}^i \subseteq \mathbf{AC}^i \subseteq \mathbf{NC}^{i+1}$.

---

[*] Here we take the length of a path as the number of edges in it.

Lesson 8: Boolean circuits part. 2[*]

Theme: Switching lemma and that parity function is not in $\mathbf{AC}^0$.

1 Definitions

In the following we will consider circuits with unbounded fan-in. We will often use the terms "boolean formula" and "boolean function" interchangeably. Recall that a literal is either a (boolean) variable or its negation.

A *term* is a conjunction of some literals. The *length* of a term is the number of literals in it. A *k-term* is a term of length k. A formula is a DNF formula if it is a disjunction of terms. It is k-DNF, if all its terms have length at most k.

Decision tree. Let F be a boolean function with variables $x_1, \ldots, x_n$. A *decision tree* of F is a tree constructed inductively as follows.

- If F already evaluates to a constant 0 or 1, the decision tree has only one node labelled with 0 or 1, respectively.
- If F is not a constant, its decision tree has a root with two children, where the left and right children are decision trees for $F[x_1 \mapsto 0]$ and $F[x_1 \mapsto 1]$, respectively.

  Here $F[x_1 \mapsto b]$ denotes the resulting formula obtained by assigning x_1 with b.

Note that a decision tree depends on the ordering of the variables $x_1, \ldots, x_n$.

Canonical decision tree for DNF formulas. Let $F = C_1 \vee C_2 \vee \cdots \vee C_m$ be a DNF formula, i.e., each C_i is a term. The *canonical decision tree* of F, denoted by $\mathcal{T}(F)$, is the decision tree obtained with the variables being ordered as follows: All the variables in C_1 appear first, followed by all the variables in C_2 (which haven't appeared yet), and so on. Let $\mathsf{depth}(\mathcal{T}(F))$ denote the depth of the canonical decision tree of F.

Restriction. Let F be a formula with variables $x_1, \ldots, x_n$. A *restriction* (on $x_1, \ldots, x_n$) is a function $\rho : \{x_1, \ldots, x_n\} \to \{0, 1, *\}$. Intuitively, $\rho(x_i) = *$ means variable x_i is not assigned. We denote by $F|_\rho$ the resulting formula where we assign the variables in F according to ρ. Note that if the formula F is DNF, the formula $F|_\rho$ is also DNF. For $\ell \leqslant n$, $\mathcal{R}_n^\ell$ denotes the set of restrictions (on n variables) where exactly ℓ variables are unassigned.

For two restrictions ρ_1 and ρ_2 whose sets of assigned variables are disjoint, we denote by $\rho_1\rho_2$ the restriction obtained by combining both restrictions. That is, for every variable x, if x is assigned according to ρ_1 (or ρ_2), then $\rho_1\rho_2$ assigns x according to ρ_1 (or ρ_2).

2 Switching lemma: Decision tree version

Lemma 8.1 (Switching lemma – Håstad 1986) *Let F be a k-DNF formula with n variables. For every $s \geqslant 0$ and every $p \leqslant 1/7$, the following holds.*

$$\frac{|\{\rho \in \mathcal{R}_n^{pn} \ : \ \mathsf{depth}(\mathcal{T}(F|_\rho)) \geqslant s\}|}{|\mathcal{R}_n^{pn}|} \quad < \quad (7pk)^s \tag{1}$$

---

[*] Based on Sect. 13.1 in N. Immerman's textbook "Descriptive Complexity" (1998). See also P. Beame's note "A switching lemma primer" (1994).

One can also write Eq. (1) as $\mathbf{Pr}_{\rho\in\mathcal{R}_n^{pn}}[\mathsf{depth}(\mathcal{T}(F|_\rho)) \geqslant s] < (7pk)^s$. Here $\mathbf{Pr}_{\rho\in\mathcal{R}_n^{pn}}[\mathcal{E}]$ denotes the probability of event $\mathcal{E}$ where ρ is randomly chosen from $\mathcal{R}_n^{pn}$.

Let $\mathsf{stars}(k,s)$ be the set that contains a sequence $\bar{Z} \stackrel{\mathsf{def}}{=} (Z_1,\ldots,Z_t)$ where $\sum_{i=1}^t |Z_i| = s$ and each Z_i is a non-empty subset of $\{1,\ldots,k\}$. When $s=0$, we define $\mathsf{stars}(k,s)$ to be $\{\varepsilon\}$, where ε denotes the "empty sequence". That is, $|\mathsf{stars}(k,0)| = 1$.

Lemma 8.2 *For every* $k,s \geqslant 1$, $|\mathsf{stars}(k,s)| \leqslant \gamma^s$, *where* γ *is such that* $(1+\frac{1}{\gamma})^k = 2$. *Hence,* $|\mathsf{stars}(k,s)| < (k/\ln 2)^s$.

Proof. The proof is by induction on s. Base case $s=0$ is trivial.

For the induction hypothesis, we assume that the lemma holds for every $s' < s$. The induction step is as follows. Observe that if Z_0 is a non-empty subset of $\{1,\ldots,k\}$ and $\bar{Z} \in \mathsf{stars}(k, s-|Z_0|)$, then $(Z_0,\bar{Z}) \in \mathsf{stars}(k,s)$. From here, we have:

$$\begin{aligned}
|\mathsf{stars}(k,s)| &= \sum_{i=1}^{\min(k,s)} \binom{k}{i} |\mathsf{stars}(k,s-i)| \leqslant \sum_{i=1}^{k} \binom{k}{i} |\mathsf{stars}(k,s-i)| \\
&\leqslant \sum_{i=1}^{k} \binom{k}{i} \gamma^{s-i} \\
&= \gamma^s \sum_{i=1}^{k} \binom{k}{i} (1/\gamma)^i \\
&= \gamma^s \big((1+1/\gamma)^k - 1\big) \\
&= \gamma^s
\end{aligned}$$

■

Proof of Switching lemma: Let F be a k-DNF formula with n variables. Let $s \geqslant 0$ and $p \leqslant 1/7$. Let $\ell = pn$. Let X be the set of restrictions ρ such that $\mathsf{depth}(\mathcal{T}(F|_\rho)) \geqslant s$. We will show that there is an injective function ξ:

$$\xi\ : X \ \to\ \mathcal{R}^{\ell-s} \times \mathsf{stars}(k,s) \times \{0,1\}^s$$

The existence of ξ implies $|X| \leqslant |\mathcal{R}^{\ell-s}| \cdot |\mathsf{stars}(k,s)| \cdot 2^s$ and Switching lemma follows immediately from Lemma 8.2 and the fact that $|\mathcal{R}_n^\ell| = \binom{n}{\ell} 2^{n-\ell}$.

Let $F \stackrel{\mathsf{def}}{=} C_1 \vee C_2 \vee \cdots$, where each C_i is a term of length at most k. Let $\rho \in X$, i.e., $\mathsf{depth}(\mathcal{T}(F|_\rho)) \geqslant s$. Consider the lexicographically first branch in $\mathcal{T}(F|_\rho)$ with length $\geqslant s$ and let b be the first s steps in this branch. To define $\xi(\rho)$, we do the following.

- Let C_{i_1} be the first term that is not set to 0 in $F|_\rho$.

  Let V_1 be the set of variables in $C_{i_1}|_\rho$. (Note that by the definition of the canonical decision tree, this means the variables in V_1 are assigned at the beginning of $\mathcal{T}(F|_\rho)$.)

  Let a_1 be the (unique) assignment that makes $C_{i_1}|_\rho$ true.

  Let b_1 be the "initial" assignment of b that assigns variables in V_1.
  (If b ends before all the variables in V_1 is used, let $b_1 = b$ and "shorten" a_1 so that both a_1 and b_1 assign the same set of variables.)

  Let $S_1 \subseteq \{1,\ldots,k\}$ be the set of index j where the j^{th} variable in C_{i_1} is assigned by a_1.
  (Note that from the term C_{i_1} and the set S_1, we can reconstruct a_1.)

- Repeat the above process but with $b \setminus b_1$, and we obtain a_2, b_2 and the set S_2,

Performing the process above, we obtain $a_1 \cdots a_t$, $b_1 \cdots b_t$ and $(S_1, \dots, S_t)$. Note that $b = b_1 \cdots b_t$. Let a denote $a_1 \cdots a_t$. Note also that the number of variables assigned by both a and b is exactly s. Thus, the sum $|S_1| + \cdots + |S_t| = s$, and hence, $(S_1, \dots, S_t) \in \mathsf{stars}(k, s)$.

Let $\delta : \{1, \dots, s\} \to \{0, 1\}$ be a function defined as follows.

$$\delta(j) \overset{\mathsf{def}}{=} \begin{cases} 1, & \text{if } a \text{ and } b \text{ assign the same value to the variable in the } j^{\text{th}} \text{ step} \\ 0, & \text{otherwise} \end{cases}$$

Note that δ can be viewed as a 0-1 string of length s.

Now we define the mapping ξ as follows.

$$\xi(\rho) \overset{\mathsf{def}}{=} (\rho a, (S_1, \dots, S_t), \delta)$$

where a, $(S_1, \dots, S_t)$ and δ are defined as above.

We need to show that ξ is injective. We will show that if $(\rho', (S_1, \dots, S_t), \delta)$ is in the range of ξ, we can construct a unique ρ such that $\xi(\rho) = \rho'$. Note that if $(\rho', (S_1, \dots, S_t), \delta)$ is in the range of ξ, there is $a_1 \cdots a_t$ such that $\rho' = \rho a$ and $(S_1, \dots, S_t)$ and δ satisfy the property imposed by the definition of ξ above. Thus, to reconstruct ρ, it suffices to reconstruct $a_1 \cdots a_t$.

We denote ρ' by $\rho a_1 \cdots a_t$ for some $a_1 \cdots a_t$ (which at this point is not known yet). We will construct $a_1, \dots, a_t$ by doing the following.

- Find out the term C_{i_1} which is the first term in F that evaluates to 1 under ρ'.

  From C_{i_1} and S_1, we reconstruct a_1.

  From a_1 and δ, we reconstruct b_1.

- Repeat the same process but replacing ρ' with $(\rho' \setminus a_1)b_1$. (Here note that $(\rho' \setminus a_1)b_1$ is the same as $\rho b_1 a_2 \cdots a_t$)

  From this step, we figure out a_2 and b_2.

We repeat the same process until we figure out all $a_1, \cdots, a_t$ and hence the restriction ρ. This completes the proof of Lemma 8.1. ■

3 Application

By the equivalence $p_1 \wedge \cdots \wedge p_m \equiv \neg(\neg p_1 \vee \cdots \vee \neg p_m)$, we can transform a circuit C into another circuit C' that uses only $\neg$ and $\vee$ gates. Moreover, $\mathsf{depth}(C') \leqslant 3 \cdot \mathsf{depth}(C)$. In this section we always assume that circuits only use $\neg$ and $\vee$ gates.

Note that every gate g in a circuit defines a boolean formula. Abusing the notation, we will often treat every gate as a formula too. For every vertex u in a circuit C, we define the height of u, denoted by $\mathsf{height}(u)$, as follows.

- The height of a source vertex (i.e., the input vertex) is 0.
- The height of a gate vertex u is the maximum of $\mathsf{height}(v) + 1$, where v ranges over all edges (u, v) in C.

So, a circuit of depth d has vertices of height from 0 to d.

In the following, log has base 2.

Lemma 8.3 *Let C be a circuit with n variables, size m and depth d. For every $1 \leqslant j \leqslant d$, let $n_j \stackrel{\mathsf{def}}{=} \frac{n}{14(14\log m)^{j-1}}$. Assume that $\log m > 1$. Then, the following holds.*

For every $1 \leqslant j \leqslant d$, there is a restriction $\rho_j \in \mathcal{R}_n^{n_j}$ such that for every gate f of height j in C, the formula $f|_{\rho_j}$ has a decision tree with height $< \log m$.

Proof. The proof is by induction on j. The base case is $j = 1$, where $n_1 \stackrel{\mathsf{def}}{=} n/14$. We randomly choose (with equal probability) a restriction ρ from $\mathcal{R}_n^{n_1}$. For a gate f of height 1, let $\mathcal{E}_f$ denote the event that "$\mathsf{depth}(\mathcal{T}(f|_\rho)) \geqslant \log m$." Let $\mathcal{E}$ denote the event that "there is a gate f of height 1 such that $\mathsf{depth}(\mathcal{T}(f|_\rho)) \geqslant \log m$."

We will first show that $\mathbf{Pr}_{\rho\in\mathcal{R}_n^{n_1}}[\mathcal{E}_f] < 1/m$, for every gate f of height 1. Let f be a gate of height 1. If f is a $\neg$-gate, then the depth of its decision tree is 1. Since $\log m > 1$, we have:

$$\mathbf{Pr}_{\rho\in\mathcal{R}_n^{n_1}}[\mathcal{E}_f] \;=\; 0 \;<\; 1/m$$

If f is an $\vee$-gate, we can view f as 1-DNF, i.e., every term has length 1. By Lemma 8.1 where $p = 1/14$, $k = 1$ and $s = \log m$, we have:

$$\mathbf{Pr}_{\rho\in\mathcal{R}_n^{n_1}}[\mathcal{E}_f] \;<\; (7\cdot(1/14)\cdot 1)^{\log m} \;=\; (1/2)^{\log m} \;=\; 1/m$$

Then,

$$\mathbf{Pr}_{\rho\in\mathcal{R}_n^{n_1}}[\mathcal{E}] \;=\; \mathbf{Pr}_{\rho\in\mathcal{R}_n^{n_1}}\Big[\bigcup_{f \text{ has height } 1} \mathcal{E}_f\Big] \;\leqslant\; \sum_{f \text{ has height } 1} \mathbf{Pr}_{\rho\in\mathcal{R}_n^{n_1}}[\mathcal{E}_f] \;<\; m\cdot(1/m) \;=\; 1$$

This means $\mathbf{Pr}_{\rho\in\mathcal{R}_n^{n_1}}[\overline{\mathcal{E}}] > 0$, which means there is a restriction $\rho \in \mathcal{R}_n^{n_1}$ such that for all gate f of height 1, $\mathsf{depth}(\mathcal{T}(f|_\rho)) < \log m$, i.e., $f|_\rho$ has a decision tree with depth $< \log m$.

For the induction hypothesis, we assume Lemma 8.3 holds for $j-1$. Let $\rho_0 \in \mathcal{R}_n^{n_{j-1}}$ be a restriction such that every gate g of height $j-1$ has decision tree with depth $< \log m$. Applying ρ_0 on all gates of height $j-1$, we can view each gate of height $j-1$ as DNF where each term has length $< \log m$.

Similar to above, we randomly choose a restriction ρ from $\mathcal{R}_{n_{j-1}}^{n_j}$. For a gate f of height j, let $\mathcal{E}'_f$ denote the event that "every decision tree of $f|_{\rho_0\rho}$ has depth $\geqslant \log m$." Let $\mathcal{E}'$ denote the event that "there is a gate f of height j such that every decision tree of $f|_{\rho_0\rho}$ has depth $\geqslant \log m$."

We will show that $\mathbf{Pr}_{\rho\in\mathcal{R}_{n_{j-1}}^{n_j}}[\mathcal{E}'_f] < 1/m$, for every gate f of height j. Let f be a gate of height j. If f is a $\neg$-gate, let $f = \neg g$, where g is of height $j-1$. Since $g|_{\rho_0}$ has decision tree with depth $< \log m$, so does $f|_{\rho_0}$. Thus,

$$\mathbf{Pr}_{\rho\in\mathcal{R}_{n_{j-1}}^{n_j}}[\mathcal{E}'_f] \;=\; 0 \;<\; 1/m$$

If f is an $\vee$-gate, we can view f as k-DNF, where $k = \log m$. By Lemma 8.1 with $p = 1/(14\log m)$, $k = \log m$ and $s = \log m$, we have:

$$\mathbf{Pr}_{\rho\in\mathcal{R}_{n_{j-1}}^{n_j}}[\mathsf{depth}(\mathcal{T}(f|_{\rho_0\rho})) \geqslant \log m] \;<\; (7\cdot\frac{1}{14\log m}\cdot\log m)^{\log m} \;=\; (1/2)^{\log m} \;=\; 1/m$$

Now, note that:

$$\mathbf{Pr}_{\rho\in\mathcal{R}_{n_{j-1}}^{n_j}}[\mathcal{E}'_f] \;\leqslant\; \mathbf{Pr}_{\rho\in\mathcal{R}_{n_{j-1}}^{n_j}}[\mathsf{depth}(\mathcal{T}(f|_{\rho_0\rho})) \geqslant \log m]$$

Thus,

$$\mathbf{Pr}_{\rho\in\mathcal{R}_{n_{j-1}}^{n_j}}[\mathcal{E}'_f] \;<\; 1/m$$

Applying similar argument as above, we obtain:

$$\mathbf{Pr}_{\rho \in \mathcal{R}_{n_{j-1}}^{n_j}}[\mathcal{E}'] \;<\; 1$$

Hence, there is a restriction $\rho \in \mathcal{R}_{n_{j-1}}^{n_j}$ such that for every gate f of height j, $f|_{\rho_0\rho}$ has a decision tree with depth $< \log m$. Now, $\rho_0\rho \in \mathcal{R}_n^{n_j}$. This completes the proof of Lemma 8.3. ■

Consider the following language $\mathsf{PARITY} \subseteq \{0,1\}^*$.

$$\mathsf{PARITY} \;\stackrel{\mathsf{def}}{=}\; \{w : \text{the number of 1's in } w \text{ is odd}\}$$

Obviously, it can be viewed as a family of boolean functions $\{f_n\}_{n\in\mathbb{N}}$, where each f_n has n variables $x_1,\ldots,x_n$ and $f_n(x_1,\ldots,x_n) \stackrel{\mathsf{def}}{=} \sum_{i=1}^n x_i \pmod 2$.

Applying Lemma 8.3, we immediately obtain that PARITY is not in $\mathbf{AC}^0$.

Theorem 8.4 (Furst, Saxe and Sipser 1981, Ajtai 1983, Yao 1985) $\mathsf{PARITY} \notin \mathbf{AC}^0$.

Lesson 9: Probabilistic Turing machines

Theme: The notion of probabilistic/randomized Turing machines and some classical results.

Probabilistic Turing machines. A *probabilistic Turing machine* (PTM) is system $\mathcal{M} = \langle \Sigma, \Gamma, Q, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}, \delta\rangle$ defined like the NTM, with the difference that $\delta \subseteq (Q - \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}) \times \Gamma \times Q \times \Gamma \times \{\texttt{Left}, \texttt{Right}\}$ is now a relation such that for every $(p,\sigma) \in (Q - \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}) \times \Gamma$, there are exactly two transitions that can be applied:

$$(p,\sigma) \to (q_1, \sigma_1, \texttt{Move}_1) \quad \text{and} \quad (p,\sigma) \to (q_2, \sigma_2, \texttt{Move}_2)$$

and the probability that each transition is applied is 1/2. Intuitively, when it is in state p reading symbol σ, $\mathcal{M}$ tosses an unbiased coin to decide whether to apply $(q_1, \sigma_1, \texttt{Move}_1)$ or $(q_2, \sigma_2, \texttt{Move}_2)$. On an input word w, the probability that $\mathcal{M}$ accepts/rejects w is defined over all possible coin tossing.

Similar to DTM/NTM, we say that $\mathcal{M}$ *runs in time* $f(n)$, if for every word w, every run of $\mathcal{M}$ on w has length $\leqslant f(|w|)$. We say that $\mathcal{M}$ *runs in polynomial time*, if there is a polynomial $p(n) = \mathsf{poly}(n)$ such that $\mathcal{M}$ runs in time $p(n)$. In this case we also say that $\mathcal{M}$ is a *polynomial time PTM.*

The class **BPP** is defined as follows. A language L is in the class **BPP**, if there a polynomial time PTM $\mathcal{M}$ such that for every input word x, the following holds.

$$\mathbf{Pr}[\ \mathcal{M}(x) = L(x)\ ] \quad \geqslant \quad 2/3$$

Here we treat a language L as a function $L : \{0,1\}^* \to \{0,1\}$, where $L(x) = 1$, if $x \in L$, and $L(x) = 0$, if $x \notin L$. Similarly, we treat TM $\mathcal{M}$ as a function $\mathcal{M} : \{0,1\}^* \to \{0,1\}$, where $\mathcal{M}(x) = 1$, if $\mathcal{M}$ accepts x, and $\mathcal{M}(x) = 0$, if $\mathcal{M}$ rejects x.

Note that **BPP** is closed under complement, union and intersection.

Remark 9.1 Alternatively, we can define the class **BPP** as follows. A language L is in the class **BPP**, if there is a polynomial $q(n)$ and a polynomial time DTM $\mathcal{M}$ such that for every $x \in \{0,1\}^*$, the following holds.

$$\mathbf{Pr}_{r \in \{0,1\}^{q(|x|)}}[\ \mathcal{M}(x,r) = L(x)\ ] \quad \geqslant \quad 2/3$$

Note that the DTM $\mathcal{M}$ takes as input (x,r). Intuitively, it can be viewed as a PTM that on input x, first randomly choose a string r of length $q(|x|)$, then run DTM $\mathcal{M}$ on (x,r).

Note the similarity with the alternative definition of **NP** (Def. 1.2), where an NTM first guesses a certificate string r, and then runs a DTM for verification.

Theorem 9.2 (Error reduction) *Let $L \in$ **BPP**. Then, for every $d \geqslant 1$, there is a polynomial time PTM $\mathcal{M}$ such that for every input word x:*

$$\mathbf{Pr}[\ \mathcal{M}(x) = L(x)\ ] \quad \geqslant \quad 1 - 2^{-\alpha |x|^d} \qquad\qquad (\textit{for some fixed } \alpha > 0)$$

Theorem 9.3 (Adleman 1978) BPP $\subseteq$ $\mathbf{P}_{/\mathsf{poly}}$.

Theorem 9.3 and Theorem 7.4 imply that if $\mathsf{SAT} \in \mathbf{BPP}$, then **PH** collapses to $\mathbf{\Sigma}_2^p$.

Theorem 9.4 (Sipser, Gács, Lautemann 1983) BPP $\subseteq \mathbf{\Sigma}_2^p \cap \mathbf{\Pi}_2^p$.

One-sided error PTM. The class **RP** is defined as follows. A language L is in the class **RP**, if there a polynomial time PTM $\mathcal{M}$ such that for every input word x, the following holds.

- If $x \in L$, then $\mathbf{Pr}[\ \mathcal{M}(x) = 1\ ] \geqslant 2/3$.
- If $x \notin L$, then $\mathbf{Pr}[\ \mathcal{M}(x) = 0\ ] = 1$.

Note that $\mathcal{M}$ is never wrong when the input $x \notin L$, hence, the name *one-sided*. The class **coRP** is defined as $\mathbf{coRP} \overset{\mathsf{def}}{=} \{L : \{0,1\}^* \setminus L \in \mathbf{RP}\}$.

Zero error PTM. A PTM $\mathcal{M}$ for a language L is a zero error PTM, if it never errs, i.e., for every input word x, $\mathbf{Pr}[\ \mathcal{M}(x) = L(x)\ ] = 1$. Now for a PTM $\mathcal{M}$ and input word x, we can define a random variable $T_{\mathcal{M},x}$ to denote the run time of $\mathcal{M}$ on x, where the probability distribution is $\mathbf{Pr}[\ T_{\mathcal{M},x} = t\ ] = p$, if with probability p over the random strings of $\mathcal{M}$ on input x, it halts in t steps .

The class **ZPP** is defined as follows. A language L is in **ZPP**, if there is a polynomial $q(n) = \mathsf{poly}(n)$ and a zero error PTM $\mathcal{M}$ for L such that for every input word x, $\mathbf{Exp}[T_{\mathcal{M},x}] \leqslant q(|x|)$.

The algorithms for languages in **BPP**/**RP**/**coRP** are also called *Monte Carlo* algorithms, and those for languages in **ZPP** are called *Las Vegas* algorithms.

Appendix

A Useful inequalities

Inclusion-exclusion principle: Let $\mathcal{E}_1, \ldots, \mathcal{E}_m$ be some m events. Then, the following holds.

$$\mathbf{Pr}\Big[\bigcup_{i=1}^{m} \mathcal{E}_i\Big] \;=\; \sum_{i=1}^{m} \mathbf{Pr}[\ \mathcal{E}_i\ ] - \sum_{1 \leqslant i_1 < i_2 \leqslant m} \mathbf{Pr}[\ \mathcal{E}_{i_1} \cap \mathcal{E}_{i_2}\ ] + \sum_{1 \leqslant i_1 < i_2 < i_3 \leqslant m} \mathbf{Pr}[\ \mathcal{E}_{i_1} \cap \mathcal{E}_{i_2} \cap \mathcal{E}_{i_3}\ ] - \cdots$$

From here, we also obtain the so called *union bound*:

$$\mathbf{Pr}\Big[\bigcup_{i=1}^{m} \mathcal{E}_i\Big] \;\leqslant\; \sum_{i=1}^{m} \mathbf{Pr}[\ \mathcal{E}_i\ ]$$

Markov inequality: Let X be a non-negative random variable with expectation μ. Then, for every real $c > 0$, the following holds.

$$\mathbf{Pr}[\ X \geqslant c\mu\ ] \;\leqslant\; 1/c$$

Markov inequality is often also called *averaging argument*.

Chebyshev inequality: Let X be a random variable with expectation μ and variance σ^2. Then, for every real $c > 0$, the following holds.

$$\mathbf{Pr}\big[\ |X - \mu| \geqslant c\sigma\ \big] \;\leqslant\; 1/c^2$$

Chernoff inequality: Let $X_1, \ldots, X_m$ be (independent) 0,1 random variables. Suppose for every $1 \leqslant i \leqslant m$, $\mathbf{Pr}[X_i = 1] = p$, for some $p > 1/2$. Let $X \overset{\mathsf{def}}{=} \sum_{i=1}^{m} X_i$. Then, the following holds.

$$\mathbf{Pr}\Big[\ X > \lfloor m/2 \rfloor\ \Big] \;\geqslant\; 1 - 2^{-\alpha m} \qquad\qquad \text{where } \alpha = \frac{\log_2 e}{2p}\Big(p - \frac{1}{2}\Big)^2$$

Lesson 10: The probabilistic method

Theme: Some examples of the probabilistic method.

1 The basic counting argument

Let K_n be a complete (undirected) graph with n vertices without self-loop.

Proposition 10.1 *For every n and $k \leqslant n$, the following holds. If $\binom{n}{k} 2^{-\binom{k}{2}+1} < 1$, then it is possible to colour the edges of K_n (with either red or blue) so that it has no monochromatic K_k subgraph.*

Proof. Given a complete graph K_n, we randomly colour each edge independently with either red or blue (with equal probability). Note that there are exactly $\binom{n}{k}$ different k-cliques. Let $m = \binom{n}{k}$. We fix an ordering of all of these k-cliques: $C_1, \ldots, C_m$ and let $\mathcal{E}_i$ denote the event that clique C_i is monochromatic. Then, $\mathbf{Pr}[\ \mathcal{E}_i\ ] = 2^{-\binom{k}{2}+1}$.

The probability that there is a monochromatic k-clique is:

$$\mathbf{Pr}[\ \mathcal{E}_1 \cup \cdots \cup \mathcal{E}_m\ ] \ \leqslant\ \sum_{i=1}^{m} \mathbf{Pr}[\ \mathcal{E}_i\ ] \ \leqslant\ m \cdot 2^{-\binom{k}{2}+1} \ <\ 1$$

Hence, the probability that none of the cliques $C_1, \ldots, C_m$ are monochromatic is not zero, i.e., there is a colouring of the edges of K_n so that there is no monochromatic k-clique. ■

The proof above can be converted into the following Las Vegas type of algorithm.

Algorithm 1

Input: A complete graph K_n and an integer k where $\binom{n}{k} 2^{-\binom{k}{2}+1} < 1$.
Task: Output a colouring of the edges of K_n in which there is no monochromatic k-clique.

```
Let $\xi$ be a random colouring of the edges in $K_n$.
while there is a monochromatic $k$-clique with colouring $\xi$ do
    Choose another random colouring $\xi$.
Output $\xi$.
```

In principle, Algorithm 1 may not terminate, but the expected number of steps is finite. Let $p = \mathbf{Pr}[\ \mathcal{E}_1 \cup \cdots \cup \mathcal{E}_m\ ]$ and let N be the random variable for the number of iterations (of the while loop). Then, the expectation of N is $1/(1-p)$. Note that if k is fixed, $1/(1-p) = \mathsf{poly}(n)$.

2 The expectation argument

In the following example we will use the fact that if X is a random variable, and μ is its expectation, then $\mathbf{Pr}[\ X \geqslant \mu\ ] > 0$ and $\mathbf{Pr}[\ X \leqslant \mu\ ] > 0$.

Let $G = (V, E)$ be an undirected graph. A *cut* of G is a pair $C = (A, B)$ where $A \cup B$ is partition of V. Its value is the number of edges of E that cross from A to B.

Proposition 10.2 *Let G be an undirected graph with m edges. Then, it has a cut with value at least $m/2$.*

Proof. Let $G = (V, E)$ be an undirected graph with m edges. We construct a cut $C = A \cup B$ by randomly assigning each vertex $u \in V$ to either A or B (with equal probability).

Let $e_1, \ldots, e_m$ be the edges in G. For each $i = 1, \ldots, m$, let X_i denote the random variable:

$$X_i \overset{\text{def}}{=} \begin{cases} 1, & \text{if the two endpoints of } e_i \text{ are in different sets} \\ 0, & \text{otherwise} \end{cases}$$

Let $X = \sum_{i=1}^{m} X_i$, i.e., X is the random variable for the value of the cut $C = (A, B)$. Note that $\mathbf{Pr}[\ X_i = 1\ ] = 1/2$. Hence, $\mathbf{Exp}[X] = m/2$. Therefore, G has a cut with value at least $m/2$. ■

Similar to Algorithm 1 above, we can design a Las Vegas algorithm for finding a cut with value $m/2$, where m is the number of edges in the input graph. To bound its expected run time, let $p = \mathbf{Pr}[\ C \text{ has value } m/2\ ]$. Now, since $\mathbf{Exp}[X] = \mathbf{Exp}[\text{value of } C] = m/2$, we can calculate that $p \geqslant 1/(\frac{m}{2} + 1)$. Thus, the expected run time of our Las Vegas algorithm is $\leqslant 1/p = \frac{m}{2} + 1$. Below we will show how it can be derandomized.

We need a few notations. Let $G = (V, E)$ be an undirected graph and P, Q be two disjoint subsets of V. Similar to above, to get a cut $C = (A, B)$, we assign each vertex $u \in V$ to either A or B as follows.

- Every vertex $u \in P$ is assigned to A.
- Every vertex $u \in Q$ is assigned to B.
- Every vertex $u \notin P \cup Q$ is randomly assigned to either A or B (with equal probability).

Let $N(P, Q)$ denote the random variable for the value of the cut $C = (A, B)$ where $P \subseteq A$ and $Q \subseteq B$. Note that $\mathbf{Exp}[N(P, Q)]$ is exactly the value of (P, Q) plus half the number of edges in $E \setminus (P \cup Q) \times (P \cup Q)$, i.e., the number of edges whose both endpoints are not in $P \cup Q$. Consider the following deterministic algorithm.

Algorithm 2

Input: A graph $G = (V, E)$.

Task: Output a cut $C = (A, B)$ with value at least $m/2$, where m is the number of edges.

```
Let $v_1, \ldots, v_n$ be the vertices in $G$.
$P := \emptyset$ and $Q := \emptyset$.
for $i = 1, \ldots, n$ do
    if $\mathbf{Exp}[N(P \cup \{v_i\}, Q)] > \mathbf{Exp}[N(P, Q \cup \{v_i\})]$ then
        $P := P \cup \{v_i\}$ and $Q := Q$.
    else
        $P := P$ and $Q := Q \cup \{v_i\}$.
Output the cut $C = (P, Q)$.
```

That Algorithm 2 output a cut $C = (P, Q)$ with value at least $m/2$ follows from the following observations.

- $\mathbf{Exp}[N(\emptyset, \emptyset)] \geqslant m/2$ (by Proposition 10.2).
- Let $(P_0, Q_0), \ldots, (P_n, Q_n)$ denote the sets (P, Q) after the ith iteration. Then, for every $i = 0, \ldots, n - 1$:

$$\mathbf{Exp}[N(P_i, Q_i)] \quad \leqslant \quad \mathbf{Exp}[N(P_{i+1}, Q_{i+1})]$$

- $\mathbf{Exp}[N(P_n, Q_n)]$ is the value of the cut $C = (P, Q)$.

Checking whether $\mathbf{Exp}[N(P \cup \{x_i\}, Q)] > \mathbf{Exp}[N(P, Q \cup \{x_i\})]$ can be done by comparing the number of neighbours of x_i that are in P and Q.

3 Sample and modify

Proposition 10.3 *Let G be a graph with n vertices and m edges where $m = dn/2$, for some d. Then, G has an independent set with at least $n/(2d)$ vertices.*

Proof. Let G be a graph as stated. Consider the following algorithm.

- Delete every vertex (together with its incident edges) independently with probability $1-1/d$.
- For each remaining edge, remove it and one of its incident vertices.

Obviously, the remaining set of vertices is independent set. Let X denote the number of vertices that survive the first step and Y denote the number of edges that survive the first step. Note that each vertex survives with probability $1/d$ and an edge survives with probability $1/d^2$. Thus,

$$\mathbf{Exp}[X] \;=\; \frac{n}{d} \quad \text{and} \quad \mathbf{Exp}[Y] \;=\; \frac{dn}{2}\cdot\frac{1}{d^2} \;=\; \frac{n}{2d}$$

The number of vertices removed in the second step is at most Y. So the number of remaining vertices after the second step is at least $X - Y$. Since $\mathbf{Exp}[X - Y] = n/(2d)$, the expected number of vertices output the algorithm is at least $n/(2d)$. Hence, there is an independent set with at least $n/(2d)$ vertices. ■

Proposition 10.4 *For every integer $k \geqslant 3$, there is an undirected graph with n vertices, at least $\frac{1}{4}n^{1+(1/k)}$ edges and girth at least k.*[*]

Proof. Let $G_{n,p}$ be the random (undirected) graph with n vertices where between every pair of vertices the probability that there is an edge between them is p. Consider the following algorithm.

- Sample $G \in G_{n,p}$ with $p = n^{(1/k)-1}$.
- For every cycle of length $\leqslant k-1$, delete one of its edges.

Let X be the number of the edges in G after the first step and let Y be the number of the cycles with length $\leqslant k-1$. There are at most $\binom{n}{i}\frac{(i-1)!}{2}$ cycles of length i. We have:

$$\begin{aligned}
\mathbf{Exp}[X] &= p\binom{n}{2} = \frac{1}{2}\Big(1-\frac{1}{n}\Big)n^{1+(1/k)} \\
\mathbf{Exp}[Y] &= \sum_{i=3}^{k-1}\binom{n}{i}\frac{(i-1)!}{2}p^i \leqslant \sum_{i=3}^{k-1} n^i p^i = \sum_{i=3}^{k-1} n^{i/k} < kn^{(k-1)/k}
\end{aligned}$$

Thus, $\mathbf{Exp}[X-Y] \geqslant \frac{1}{4}n^{1+(1/k)}$.

Note that the number of edges after the second step is at least $X - Y$. Thus, there is a graph with n vertices, at least $\frac{1}{4}n^{1+(1/k)}$ edges and girth at least k. ■

---

[*] The girth of a graph is the length of its shortest cycle.

4 The local lemma

We say that an event $\mathcal{E}$ is *mutually independent* of events $\mathcal{E}_1,\dots,\mathcal{E}_n$, if for every subset $S \subseteq \{1,\dots,n\}$, $\mathbf{Pr}[\ \mathcal{E} \mid \bigcap_{i\in S}\mathcal{E}_i\ ] = \mathbf{Pr}[\ \mathcal{E}\ ]$. The *dependency graph* of events $\mathcal{E}_1,\dots,\mathcal{E}_n$ is a graph $G=(V,E)$ where $V=\{1,\dots,n\}$ and for every $i=1,\dots,n$, event $\mathcal{E}_i$ is mutually independent of the events in $\{\mathcal{E}_j : (i,j)\notin E\}$.

Lemma 10.5 is the symmetric version of the so called *Lovász local lemma.*

Lemma 10.5 (Symmetric local lemma) *Let $\mathcal{E}_1,\dots,\mathcal{E}_n$ be n events. Let d be the degree of its dependency graph and p be such that $4dp \leqslant 1$ and $\mathbf{Pr}[\ \mathcal{E}_i\ ] \leqslant p$, for every $i=1,\dots,n$. Then,*

$$\mathbf{Pr}[\ \bigcap_{i=1}^{n}\overline{\mathcal{E}_i}\ ] \ > \ 0, \qquad\qquad \textit{where } \overline{\mathcal{E}_i} \textit{ is the complement of } \mathcal{E}_i$$

Proof. For a subset $S \subsetneq \{1,\dots,n\}$, we denote by F_S the event $\bigcap_{i\in S}\overline{\mathcal{E}_i}$. When $S=\emptyset$, we set F_0 to be the whole sample space. We claim that for every $S\subsetneq\{1,\dots,n\}$, the following holds.

$$\mathbf{Pr}[\ F_S\ ] \ > \ 0 \qquad \text{and} \qquad \mathbf{Pr}[\ \mathcal{E}_k \mid F_S\ ] \leqslant 2p, \quad \text{for every } k\notin S$$

This claim immediately implies Lemma 10.5. To avoid clutter, for $\ell = 0,1,\dots,n$, let F_ℓ denote the event $\bigcap_{i=1}^{\ell}\overline{\mathcal{E}_i}$. Similar to above, we define $F_0 \overset{\mathsf{def}}{=} \Omega$, i.e., the whole sample space. In other words, $F_\ell = F_S$, where $S=\{1,\dots,\ell\}$. Now, we have the following.

$$\mathbf{Pr}[\ \bigcap_{i=1}^{n}\overline{\mathcal{E}_i}\ ] \ = \ \mathbf{Pr}[\ F_n\ ] \ = \ \prod_{i=1}^{n}\mathbf{Pr}[\ \overline{\mathcal{E}_i} \mid F_{i-1}\ ] \ = \ \prod_{i=1}^{n}\big(1 \ - \ \mathbf{Pr}[\ \mathcal{E}_i \mid F_{i-1}\ ]\big) \ \geqslant \ \prod_{i=1}^{n}(1-2p) \ > \ 0$$

The second last inequality is by the claim.

Now we will show that the claim holds by induction on $|S|$. The base case $S=\emptyset$ is trivial. For the induction hypothesis, we assume Claim 1 holds for every S where $|S|\leqslant \ell-1$. We will show that it holds for S where $|S|=\ell$.

Without loss of generality, we assume that $S=\{1,\dots,\ell\}$. Thus, $F_S=F_\ell$. The proof for $\mathbf{Pr}[\ F_\ell\ ]>0$ is similar to the one above:

$$\mathbf{Pr}[\ F_\ell\ ] \ = \ \prod_{i=1}^{\ell}\mathbf{Pr}[\ \overline{\mathcal{E}_i} \mid F_{i-1}\ ] \ = \ \prod_{i=1}^{\ell}\big(1 \ - \ \mathbf{Pr}[\ \mathcal{E}_i \mid F_{i-1}\ ]\big) \ \geqslant \ \prod_{i=1}^{\ell}(1-2p) \ > \ 0$$

We now show that $\mathbf{Pr}[\ \mathcal{E}_k \mid F_\ell\ ]\leqslant 2p$, for every $k\notin S$. Let $k\notin S$ and let S_1 and S_2 be as follows.

$$S_1 \ \overset{\mathsf{def}}{=} \ \{j\in S : (k,j) \text{ is an edge in } G\} \qquad \text{and} \qquad S_2 \ \overset{\mathsf{def}}{=} \ S\setminus S_1$$

Note that $\mathcal{E}_k$ is mutually independent of the events in S_2. We consider two cases: $S_2=S$ or $S_2\neq S$.

The case when $S_2=S$ is trivial since $\mathbf{Pr}[\ \mathcal{E}_k \mid F_S\ ] = \mathbf{Pr}[\ \mathcal{E}_k\ ]\leqslant p\leqslant 2p$. When $S_2\neq S$, the proof is as follows.

$$\begin{aligned}\mathbf{Pr}[\ \mathcal{E}_k \mid F_S\ ] \ = \ \frac{\mathbf{Pr}[\ \mathcal{E}_k\cap F_S\ ]}{\mathbf{Pr}[\ F_S\ ]} \ = \ \frac{\mathbf{Pr}[\ \mathcal{E}_k\cap F_{S_1}\cap F_{S_2}\ ]}{\mathbf{Pr}[\ F_{S_1}\cap F_{S_2}\ ]} \ &= \ \frac{\mathbf{Pr}[\ \mathcal{E}_k\cap F_{S_1} \mid F_{S_2}\ ]\cdot\mathbf{Pr}[\ F_{S_2}\ ]}{\mathbf{Pr}[\ F_{S_1}\mid F_{S_2}\ ]\cdot\mathbf{Pr}[\ F_{S_2}\ ]} \\ &= \ \frac{\mathbf{Pr}[\ \mathcal{E}_k\cap F_{S_1} \mid F_{S_2}\ ]}{\mathbf{Pr}[\ F_{S_1}\mid F_{S_2}\ ]}\end{aligned}$$

Note that $\mathbf{Pr}[\ \mathcal{E}_k\cap F_{S_1}\mid F_{S_2}\ ]\leqslant \mathbf{Pr}[\ \mathcal{E}_k \mid F_{S_2}\ ] = \mathbf{Pr}[\ \mathcal{E}_k\ ]\leqslant p$ with the equality comes from the fact that $\mathcal{E}_k$ is mutually independent of the events in S_2.

We can bound $\mathbf{Pr}[\ F_{S_1} \mid F_{S_2}\ ]$ as follows.

$$\begin{aligned}\mathbf{Pr}[\ F_{S_1} \mid F_{S_2}\ ] \ = \ \mathbf{Pr}[\ \bigcap_{j\in S_1} \overline{\mathcal{E}_j} \mid F_{S_2}\ ] \ = \ \mathbf{Pr}[\ \overline{\bigcup_{j\in S_1} \mathcal{E}_j} \mid F_{S_2}\ ] \ = \ &\geqslant \ 1-\sum_{j\in S_1}\mathbf{Pr}[\ \mathcal{E}_j \mid F_{S_2}\ ] \\ &\geqslant \ 1-2pd \\ &\geqslant \ 1/2\end{aligned}$$

The second last inequality comes from the induction hypothesis and $|S_1| \leqslant d$. The last equality comes from $4pd \leqslant 1$. Note that with the bound on $\mathbf{Pr}[\ \mathcal{E}_k \cap F_{S_1} \mid F_{S_2}\ ]$ and $\mathbf{Pr}[\ F_{S_1} \mid F_{S_2}\ ]$, we have $\mathbf{Pr}[\ \mathcal{E}_k \mid F_S\ ] \leqslant 2p$. ■

Proposition 10.6 *For every k-CNF formula φ, if every variable appears in at most $2^k/(4k)$ clauses, then φ is satisfiable.*

Proof. Suppose φ has m clauses. We randomly assign each variable with 0 or 1 (with equal probability). Let $\mathcal{E}_i$ be the event that the ith clause is not satisfied by the random assignment. Then, $\mathbf{Pr}[\ \mathcal{E}_i\ ] = 2^{-k}$.

Event $\mathcal{E}_i$ is mutually independent of all the events $\mathcal{E}_j$, if the jth clause does not share the same variable as the ith clause. Thus, the degree of the dependency graph is $\leqslant k\cdot 2^k/(4k) = 2^{k-2}$ and hence, $4dp \leqslant 1$. By Lemma 10.5, there is an assignment satisfying every clause. ■

Lemma 10.7 (General local lemma, Erdös and Lovász 1975) *Let $\mathcal{E}_1,\dots,\mathcal{E}_n$ be n events and $G=(V,E)$ be its dependency graph. Suppose there are real numbers $x_1,\dots,x_n$ such that $0\leqslant x_i<1$ and $\mathbf{Pr}[\ \mathcal{E}_i\ ] \leqslant x_i\prod_{(i,j)\in E}(1-x_j)$, for every $i=1,\dots,n$. Then,*

$$\mathbf{Pr}\Big[\ \bigcap_{i=1}^{n}\overline{\mathcal{E}_i}\ \Big] \ \geqslant \ \prod_{i=1}^{n}(1-x_i)$$

In particular, with positive probability no event $\mathcal{E}_i$ holds.

Proof. We use the same notation as in Lemma 10.5, where $F_S = \bigcap_{i\in S}\overline{\mathcal{E}_i}$ and $F_\ell = F_{\{1,\dots,\ell\}}$. We claim that for every $S \subsetneq \{1,\dots,n\}$ and every $k\notin S$, the following holds.

$$\mathbf{Pr}[\ \mathcal{E}_k \mid F_S\ ] \ \leqslant \ x_k \tag{1}$$

Similar to Lemma 10.5, this claim immediately implies Lemma 10.7.

$$\mathbf{Pr}[\ \bigcap_{i=1}^{n}\overline{\mathcal{E}_i}\ ] \ = \ \mathbf{Pr}[\ F_n\ ] \ = \ \prod_{i=1}^{n}\mathbf{Pr}[\ \overline{\mathcal{E}_i} \mid F_{i-1}\ ] \ = \ \prod_{i=1}^{n}\big(1 \ - \ \mathbf{Pr}[\ \mathcal{E}_i \mid F_{i-1}\ ]\big) \ \geqslant \ \prod_{i=1}^{n}(1-x_i)$$

The proof of (1) is by induction on $|S|$. The base case $S=\emptyset$ is trivial. For the induction step, the strategy is the same as in Lemma 10.5. Let S_1 and S_2 be the following sets.

$$S_1 \ \overset{\mathsf{def}}{=} \ \{j\in S : (k,j)\in E\} \qquad \text{and} \qquad S_2 \ \overset{\mathsf{def}}{=} \ S\setminus S_1$$

When $S_2 \neq S$, we have the following.

$$\mathbf{Pr}[\ \mathcal{E}_k \mid F_S\ ] \ = \ \frac{\mathbf{Pr}[\ \mathcal{E}_k\cap F_{S_1} \mid F_{S_2}\ ]}{\mathbf{Pr}[\ F_{S_1} \mid F_{S_2}\ ]} \tag{2}$$

The numerator is bounded as follows.

$$\mathbf{Pr}[\ \mathcal{E}_k \cap F_{S_1} \mid F_{S_2}\ ] \ \leqslant\ \mathbf{Pr}[\ \mathcal{E}_k \mid F_{S_2}\ ] \ =\ \mathbf{Pr}[\ \mathcal{E}_k\ ] \ \leqslant\ x_i \prod_{(i,j)\in E} (1-x_j) \tag{3}$$

The denominator is bounded as follows. Let $S_1 = \{j_1, \ldots, j_r\}$.

$$\mathbf{Pr}[\ F_{S_1} \mid F_{S_2}\ ] \ =\ \prod_{i=1}^{r} \mathbf{Pr}[\ \overline{\mathcal{E}_{j_i}} \mid F_{S_2 \cup \{j_1,\ldots,j_{i-1}\}}\ ] \ \geqslant\ \prod_{i=1}^{r} (1-x_{j_i}) \ =\ \prod_{(k,j)\in E} (1-x_j) \tag{4}$$

Combining Inequalities (2), (3) and (4), we obtain Inequality (1). ■

Lemma 10.7 implies the symmetric case with better bound.

Corollary 10.8 (Stronger symmetric local lemma) *Let $\mathcal{E}_1, \ldots, \mathcal{E}_n$ be n events. Let d be the degree of its dependency graph and p be such that $ep(d+1) \leqslant 1$ and $\mathbf{Pr}[\ \mathcal{E}_i\ ] \leqslant p$, for every $i = 1, \ldots, n$. Then,*

$$\mathbf{Pr}[\ \bigcap_{i=1}^{n} \overline{\mathcal{E}_i}\ ] \ >\ 0,$$

Proof. The case when $d = 0$ is trivial. So, we assume that $d \geqslant 1$. Let $G = (V, E)$ be the dependency graph. For every $i = 1, \ldots, n$, let $x_i = 1/(d+1)$. Note that since $d \geqslant 1$, $1/(d+1) < 1$ and $0 < 1 - 1/(d+1) < 1$.

For each $i = 1, \ldots, n$, we have:[†]

$$\begin{aligned} x_i \prod_{(i,j)\in E} (1-x_j) \ &=\ \frac{1}{d+1} \prod_{(i,j)\in E} \Big(1 - \frac{1}{d+1}\Big) \ \geqslant\ \frac{1}{d+1}\Big(1-\frac{1}{d+1}\Big)^{d} \\ &\geqslant\ \frac{1}{d+1}\Big(1-\frac{1}{d+1}\Big)^{d+1} \\ &\geqslant\ \frac{1}{(d+1)e} \\ &\geqslant\ p \\ &\geqslant\ \mathbf{Pr}[\ \mathcal{E}_i\ ] \end{aligned}$$

Thus, we can apply Lemma 10.7 and conclude that $\mathbf{Pr}[\ \bigcap_{i=1}^{n} \overline{\mathcal{E}_i}\ ] > 0$. ■

[†] Recall that $\left(1 - \frac{1}{x}\right)^x > 1/e$, for every $x > 1$.

Lesson 11: Probabilistic reductions

Theme: Probabilistic reductions and preliminary to Toda's theorem.

1 Probabilistic reduction from SAT to USAT

Let USAT be the following language.

$$\mathsf{USAT} \stackrel{\mathsf{def}}{=} \{\varphi : \varphi \text{ is a boolean formula with unique satisfying assignment}\}$$

Theorem 11.1 (Valiant and Vazirani, 1986) *There is a probabilistic polynomial time algorithm $\mathcal{M}$ such that on input (Boolean) formula φ, the output of $\mathcal{M}$, denoted by $\mathcal{M}(\varphi)$, satisfies the following.*

- *If $\varphi \in \mathsf{SAT}$, then $\mathbf{Pr}[\ \mathcal{M}(\varphi) \in \mathsf{USAT}\ ] \geqslant 3/(16n)$, where n is the number of variables in φ.*
- *If $\varphi \notin \mathsf{SAT}$, then $\mathbf{Pr}[\ \mathcal{M}(\varphi) \in \mathsf{SAT}\ ] = 0$.*

Proof. The algorithm $\mathcal{M}$ works as follows. On input formula φ, do the following.

- Let $x_1, \ldots, x_n$ be the variables in φ.
- Let $x \stackrel{\mathsf{def}}{=} (x_1, \ldots, x_n)$.
- Randomly choose $k \in \{2, \ldots, n+1\}$.
- Randomly choose a hash function $h \in \mathcal{H}_{n,k}$, where $\mathcal{H}_{n,k}$ is pair-wise independent.
- Output the formula $\varphi(x) \wedge (h(x) = 0)$, where 0 is a column vector of zeroes of size k.

Note that the part $h(x) = 0$ can be stated as a boolean formula. If we use the collection $\mathcal{H}_{n,k}$ as in Theorem 11.9, $h(x) = 0$ is of the form: $Ax + b = 0$, which is equivalent to $Ax = b$. This can be written into the following form:

$$\bigwedge_{i=1}^{k} \Big(\big(A_{i,1}x_1 \oplus \cdots \oplus A_{i,n}x_n\big) \ \leftrightarrow\ b_i \Big)$$

Here $\oplus$ denotes the XOR operation. Note that each $A_{i,1}x_1 \oplus \cdots \oplus A_{i,n}x_n$ can be rewritten into formulas using only $\wedge, \vee, \neg$ in quadratic time as follows. Divide it into two halves, rewrite each half (recursively) and combine them with the standard definition of XOR.

Now, we prove the correctness of our algorithm. Obviously, if the input formula φ is not satisfiable, so is the output formula. Suppose φ is satisfiable. Let S be the set of satisfying assignments of φ. With probability $1/n$, the algorithm chooses a value k such that $2^{k-2} \leqslant |S| \leqslant 2^{k-1}$. By Lemma 11.11, the probability that there is a unique $x \in S$ such that $h(x) = 0$ is $\geqslant 3/16$. Thus, the probability that $\mathcal{M}(\varphi) \in \mathsf{USAT}$ is at least $3/(16n)$. ■

2 The language $\oplus$SAT and the class $\oplus$P

The language $\oplus$SAT is defined as follows.

$$\oplus\mathsf{SAT} \overset{\mathsf{def}}{=} \{\varphi : \varphi \text{ is a Boolean formula with } \textit{odd} \text{ number of satisfying assignments}\}$$

The class $\oplus\mathbf{P}$ is defined as follows. A language $L \in \oplus\mathbf{P}$, if there is a polynomial time NTM $\mathcal{M}$ such that for every input word w, $w \in L$ if and only if the number of accepting runs of $\mathcal{M}$ on w is odd number.

We define a few terminology and notations. Let $\sharp\varphi$ denote the number of satisfying assignments of a (Boolean) formula φ. We will define operations $\sim$, $\sqcap$ and $\sqcup$ on formulas such that the following holds.

$$\sharp(\sim\varphi) = \sharp\varphi + 1 \qquad \sharp(\varphi \sqcap \phi) = \sharp\varphi \cdot \sharp\phi \qquad \sharp(\varphi \sqcup \phi) = (\sharp\varphi + 1)\cdot(\sharp\phi + 1) + 1$$

Obviously the following holds.

$$\begin{array}{rcll} \sim\varphi & \in & \oplus\mathsf{SAT} & \text{if and only if} \quad \varphi \notin \oplus\mathsf{SAT} \\ \varphi \sqcap \phi & \in & \oplus\mathsf{SAT} & \text{if and only if} \quad \text{both } \varphi,\phi \in \oplus\mathsf{SAT} \\ \varphi \sqcup \phi & \in & \oplus\mathsf{SAT} & \text{if and only if} \quad \text{at least one of } \varphi,\phi \in \oplus\mathsf{SAT} \end{array}$$

These operations are defined as follows.

- For φ with variables $x_1,\ldots,x_n$, we pick a "new" variable z and define $\sim\varphi$ as follows.

$$\sim\varphi \overset{\mathsf{def}}{=} (\neg z \wedge \varphi) \;\vee\; (z \wedge \bigwedge_{i=1}^{n} x_i)$$

- For two formulas φ and ψ, we rename the variables so that the variables in φ and ϕ are disjoint, and define $\varphi \sqcap \psi$ as follows.

$$\varphi \sqcap \phi \overset{\mathsf{def}}{=} \varphi \wedge \phi$$

- For two formulas φ and ψ, we rename the variables so that the variables in φ and ϕ are disjoint, and define $\varphi \sqcup \psi$ as follows.

$$\varphi \sqcup \phi \overset{\mathsf{def}}{=} \sim(\sim\varphi \sqcap \sim\phi)$$

3 Probabilistic reductions from SAT and $\overline{\mathsf{SAT}}$ to $\oplus$SAT

Theorem 11.1 can be easily extended to obtain reductions from SAT and $\overline{\mathsf{SAT}}$ to $\oplus$SAT.

Lemma 11.2 (Reduction from SAT to $\oplus$SAT) *There is a polynomial time PTM $\mathcal{M}$ that on input formula φ and a positive integer m (in unary), outputs a formula, denoted by $\mathcal{M}(\varphi, m)$, such that the following holds.*

- *If $\varphi \in \mathsf{SAT}$, then $\mathbf{Pr}[\ \mathcal{M}(\varphi,m) \in \oplus\mathsf{SAT}\ ] \geqslant 1 - 2^{-m}$.*
- *If $\varphi \notin \mathsf{SAT}$, then $\mathbf{Pr}[\ \mathcal{M}(\varphi,m) \in \oplus\mathsf{SAT}\ ] = 0$.*

Moreover, the output $\mathcal{M}(\varphi,m)$ uses $O(mn^2)$ variables, where n is the number of variables in φ.[*]

---

[*] Abusing the notation, $O(mn^2)$ denotes $\leqslant cmn^2$, for some constant c.

Proof. On input φ with n variables, the algorithm $\mathcal{M}$ first runs the reduction in Theorem 11.1 on φ for $8mn$ times to obtain formulas $\psi_1, \dots, \psi_{8mn}$. Then, it outputs $\sim (\sim \psi_1 \sqcap \cdots \sqcap \sim \psi_{8mn})$.[†] Obviously, $\mathcal{M}$ runs in polynomial time. Note also that the output formula uses $8mn(n+1)+1 = O(mn^2)$ variables.

Recall that on input φ with n variables, the reduction in Theorem 11.1 outputs a formula ψ such that the following holds.

- If $\varphi \in \mathsf{SAT}$, then $\mathbf{Pr}[\ \psi \in \mathsf{USAT}\ ] \geqslant 1/(8n)$.
- If $\varphi \notin \mathsf{SAT}$, then $\mathbf{Pr}[\ \psi \in \mathsf{SAT}\ ] = 0$.

Note the following.

- If $\psi \notin \oplus\mathsf{SAT}$, then $\psi \notin \mathsf{USAT}$. Thus, $\mathbf{Pr}[\ \psi \notin \oplus\mathsf{SAT}\ ] \leqslant \mathbf{Pr}[\ \psi \notin \mathsf{USAT}\ ]$.
- $\bigsqcup_{i=1}^{8mn} \psi_i \in \oplus\mathsf{SAT}$ if and only if one of $\psi_i \in \oplus\mathsf{SAT}$.

Thus, on input φ, the output $\bigsqcup_{i=1}^{8mn} \psi_i$ satisfies the following.

- If $\varphi \notin \mathsf{SAT}$, then none of the ψ_i is satisfiable. Thus, $\bigsqcup_{i=1}^{8mn} \psi_i \notin \oplus\mathsf{SAT}$. Therefore,

$$\mathbf{Pr}\Big[\ \bigsqcup_{i=1}^{8mn} \psi_i \ \in\ \oplus\mathsf{SAT}\ \Big] \ = \ 0$$

- If $\varphi \in \mathsf{SAT}$, the following holds.

$$\mathbf{Pr}\Big[\ \bigsqcup_{i=1}^{8mn} \psi_i \notin \oplus\mathsf{SAT}\ \Big] \ = \ \prod_{i=1}^{8mn} \mathbf{Pr}[\ \psi_i \notin \oplus\mathsf{SAT}\ ] \ \leqslant\ \Big(1 - \frac{1}{8n}\Big)^{8mn} \ \leqslant\ (1/e)^m \ \leqslant\ (1/2)^m$$

  Therefore, $\mathbf{Pr}[\ \bigsqcup_{i=1}^{8mn} \psi_i \in \oplus\mathsf{SAT}\ ] \geqslant 1 - (1/2)^m$.

This completes the proof of Lemma 11.2. ■

Lemma 11.3 (Reduction from $\overline{\mathsf{SAT}}$ to $\oplus\mathsf{SAT}$) *There is a polynomial time PTM $\mathcal{M}$ that on input formula φ and a positive integer m (in unary), outputs a formula, denoted by $\mathcal{M}(\varphi, m)$, such that the following holds.*

- *If $\varphi \in \overline{\mathsf{SAT}}$, then $\mathbf{Pr}[\ \mathcal{M}(\varphi, m) \in \oplus\mathsf{SAT}\ ] = 1$.*
- *If $\varphi \notin \overline{\mathsf{SAT}}$, then $\mathbf{Pr}[\ \mathcal{M}(\varphi, m) \in \oplus\mathsf{SAT}\ ] \leqslant (1/2)^m$.*

Proof. The PTM $\mathcal{M}$ works as follows. On input φ and m, it runs the reduction in Lemma 11.2 to obtain a formula ψ, and then outputs $\sim \psi$.

If $\varphi \in \overline{\mathsf{SAT}}$, then $\mathbf{Pr}[\ \psi \notin \oplus\mathsf{SAT}\ ] = 1$, and hence, $\mathbf{Pr}[\ \sim \psi \in \oplus\mathsf{SAT}\ ] = 1$.

If $\varphi \notin \overline{\mathsf{SAT}}$, then $\mathbf{Pr}[\ \sim \psi \in \oplus\mathsf{SAT}\ ] = \mathbf{Pr}[\ \psi \notin \oplus\mathsf{SAT}\ ] \leqslant (1/2)^m$. ■

Combining Lemmas 11.2 and 11.3 and Cook-Levin reduction, we have the following.

Theorem 11.4 (Reductions from languages in $\mathbf{NP} \cup \mathbf{coNP}$ to $\oplus\mathsf{SAT}$) *For every language $L \in \mathbf{NP} \cup \mathbf{coNP}$, there is a polynomial time PTM $\mathcal{M}$ that on input word w and a number m (in unary), outputs a formula $\mathcal{M}(w, m)$ such that the following holds.*

- *If $w \in L$, then $\mathbf{Pr}[\ \mathcal{M}(w, m) \in \oplus\mathsf{SAT}\ ] \geqslant 1 - (1/2)^m$.*
- *If $w \notin L$, then $\mathbf{Pr}[\ \mathcal{M}(w, m) \in \oplus\mathsf{SAT}\ ] \leqslant (1/2)^m$.*

[†] Note that $\sim (\sim \psi_1 \sqcap \cdots \sqcap \sim \psi_{8mn})$ is equivalent to $\psi_1 \sqcup \cdots \sqcup \psi_{8mn}$.

4 Probabilistic reductions from languages in PH to $\oplus$SAT

In this section we will show how to extend Theorem 11.4 to all languages in **PH**. We need some terminology and notations. We write $\bar{x}$, $\bar{y}$ or $\bar{z}$ to denote a sequence of variables, and the length is denoted by $|\bar{x}|$, $|\bar{y}|$ or $|\bar{z}|$, respectively.

Recall that a QBF is formula of the form: $Q_1\bar{z}_1 \cdots Q_k\bar{z}_k \quad \phi$ where each $Q_i \in \{\forall, \exists\}$ and $Q_i \neq Q_{i+1}$, each $\bar{z}_i$ is a vector of variables and ϕ is a formula that uses variables $\bar{z}_1, \ldots, \bar{z}_k$. Note that all variables used in ψ are "quantified."

QBF with free variables. A *QBF with free variables* is a QBF formula that has variables that are not quantified, i.e., of the form:

$$\varphi \overset{\mathsf{def}}{=} Q_1\bar{z}_1 \cdots Q_k\bar{z}_k \quad \phi$$

where ϕ uses some variables $\bar{y}$ that are "free," i.e., not quantified by any quantifiers, in addition to the variables $\bar{z}_1, \ldots, \bar{z}_k$. In this case, we write $\varphi(\bar{y})$ to indicate that $\bar{y}$ are free. For example, in the formula $\forall x \exists z (x \vee y \vee z)$, variables x, z are quantified, but variable y is free.

We usually denote an assignment that assigns variables in $\bar{y}$ as a string $\bar{a} \in \{0,1\}^n$ with the same length as $\bar{y}$. For a QBF $\varphi(\bar{y})$ with free variable $\bar{y}$ and $\bar{a}$ be an assignment on $\bar{y}$, we write $\varphi(\bar{a})$ to denote the QBF (without free variables) obtained by substituting every variable in $\bar{y}$ according to $\bar{a}$.

In the following the term "QBF" means a QBF which may or may not contain free variables. A *k-QBF* is a QBF in which there are k alternating quantifiers, i.e., $Q_1\bar{z}_1 \cdots Q_k\bar{z}_k\ \psi$, where each $Q_i \neq Q_{i+1}$.

The operations $\sim$, $\sqcap$ and $\sqcup$ with formulas with "free" variables. In the following we will deal with boolean formulas φ with "free" variables. Intuitively, free variables in a boolean formula are variables that cannot be renamed. We write $\varphi(\bar{y})$ to indicate that $\bar{y}$ are the free variables in φ.

- $\sim \varphi(\bar{y})$ is defined as before and the resulting formula $\sim (\varphi(\bar{y}))$ also have free variables $\bar{y}$.
- For $\varphi(\bar{y})$ and $\phi(\bar{y})$, we rename the variables so that $\bar{y}$ are the only common variables in φ and ϕ and define $\varphi(\bar{y}) \sqcap \phi(\bar{y}) \overset{\mathsf{def}}{=} \varphi(\bar{y}) \wedge \phi(\bar{y})$ with free variables $\bar{y}$.
- For $\varphi(\bar{y})$ and $\phi(\bar{y})$, we define $\varphi(\bar{y}) \sqcup \phi(\bar{y}) \overset{\mathsf{def}}{=} \sim (\sim \varphi(\bar{y}) \sqcap \ \sim \phi(\bar{y}))$ with free variables $\bar{y}$.

Lemma 11.5 (Reductions from Σ_k-SAT and Π_k-SAT to $\oplus$SAT) *For every $k \geqslant 1$, there is a probabilistic polynomial time algorithm $\mathcal{M}$ that on input a k-QBF $\varphi(\bar{y})$ and a positive integer m (in unary), outputs a formula $\psi(\bar{y})$ such that*

$$\mathbf{Pr}[\ \psi(\bar{y}) \text{ is "correct"}\ ] \quad \geqslant \quad 1 - (1/2)^m$$

Here we define a formula $\psi(\bar{y})$ to be "correct" when $\varphi(\bar{a})$ is a true QBF if and only if $\psi(\bar{a}) \in \oplus$SAT, for every assignment $\bar{a}$ on $\bar{y}$.

Proof. The proof is by induction on k. The base case $k = 1$ is similar to Lemmas 11.2 and 11.3. On input 1-QBF $\varphi(\bar{y})$ and integer m, the algorithm $\mathcal{M}$ works as follows.

- If $\varphi(\bar{y})$ is of the form $\exists\bar{x}\ \psi(\bar{x}, \bar{y})$, where $\bar{x}$ contains n variables, do the following.

  For each $i = 1, \ldots, 8mn$, construct formula $\alpha_i(\bar{y})$ as follows.

- Randomly choose $k \in \{2, \ldots, n+1\}$.
- Randomly choose a hash function $h \in \mathcal{H}_{n,k}$, where $\mathcal{H}_{n,k}$ is pair-wise independent.
- Let $\alpha_i(\bar{y})$ denote the formula $\psi(\bar{x}, \bar{y}) \wedge (h(\bar{x}) = 0)$.

Then, output the formula $\psi(\bar{y})$ where $\psi(\bar{y})$ is the formula $\bigsqcup_{i=1}^{8mn} \alpha_i(\bar{y})$.

- If $\varphi(\bar{y})$ is of the form $\forall \bar{x}\ \psi(\bar{x}, \bar{y})$, where $\bar{x}$ contains n variables, do the following

  For each $i = 1, \ldots, 8mn$, construct formula $\alpha_i(\bar{y})$ as follows.

  - Randomly choose $k \in \{2, \ldots, n+1\}$.
  - Randomly choose a hash function $h \in \mathcal{H}_{n,k}$, where $\mathcal{H}_{n,k}$ is pair-wise independent.
  - Let $\alpha_i(\bar{y})$ denote the formula $\neg\psi(\bar{x}, \bar{y}) \wedge (h(\bar{x}) = 0)$.

  Then, output the formula $\psi(\bar{y})$, where $\psi(\bar{y})$ is the formula $\sim \bigsqcup_{i=1}^{8mn} \alpha_i(\bar{y})$.

The proof that $\mathbf{Pr}[\ \psi(\bar{y}) \text{ is correct }] \geqslant 1 - (1/2)^m$ is similar to Lemmas 11.2 and 11.3.

For the induction hypothesis, we assume Lemma 11.5 holds for k, i.e., there is a probabilistic algorithm $\mathcal{M}_0$ that on input a k-QBF $\varphi(\bar{y})$ and a positive integer m (in unary), outputs a formula $\psi(\bar{y})$ such that $\mathbf{Pr}[\ \psi(\bar{y}) \text{ is correct }] \geqslant 1 - (1/2)^m$.

For the induction step, on input $(k+1)$-QBF $\varphi(\bar{y})$ and m, the algorithm $\mathcal{M}$ works as follows.

- $\varphi(\bar{y})$ is of the form $\exists \bar{x}\ \phi(\bar{x}, \bar{y})$, where $\bar{x}$ contains n variables.

  For each $i = 1, \ldots, 8mn$, construct a formula $\alpha_i(\bar{y})$ as follows.

  - Let $\beta_i(\bar{x}, \bar{y})$ be the output of $\mathcal{M}_0$ on input $\phi(\bar{x}, \bar{y})$ and $(m+1)$.
  - Randomly choose $k \in \{2, \ldots, n+1\}$.
  - Randomly choose a hash function $h \in \mathcal{H}_{n,k}$, where $\mathcal{H}_{n,k}$ is pair-wise independent.
  - Let $\alpha_i(\bar{y})$ denote the formula $\beta_i(\bar{x}, \bar{y}) \wedge (h(\bar{x}) = 0)$.

  Then, output the formula $\psi(\bar{y})$ where $\psi(\bar{y}) \stackrel{\mathsf{def}}{=} \bigsqcup_{i=1}^{8mn} \alpha_i(\bar{y})$.

- $\varphi(\bar{y})$ is of the form $\forall \bar{x}\ \psi(\bar{x}, \bar{y})$, where $\bar{x}$ contains n variables.

  For each $i = 1, \ldots, 8mn$, construct a formula α_i, as follows.

  - Let $\beta_i(\bar{x}, \bar{y})$ be the output of $\mathcal{M}_0$ on input $\neg\psi(\bar{x}, \bar{y})$ and $(m+1)$.
  - Randomly choose $k \in \{2, \ldots, n+1\}$.
  - Randomly choose a hash function $h \in \mathcal{H}_{n,k}$, where $\mathcal{H}_{n,k}$ is pair-wise independent.
  - Let $\alpha_i(\bar{y})$ be the formula $\beta_i(\bar{x}, \bar{y}) \wedge (h(\bar{x}) = 0)$.

  Then, output the formula $\psi(\bar{y})$ where $\psi(\bar{y}) \stackrel{\mathsf{def}}{=} \sim \bigsqcup_{i=1}^{8mn} \alpha_i(\bar{y})$.

We now calculate the probability of the event that $\psi(\bar{y})$ is correct.

We first consider the case that $\varphi(\bar{y})$ is of the form $\exists \bar{x} \phi(\bar{x}, \bar{y})$. By the induction hypothesis, $\mathbf{Pr}[\ \beta_i(\bar{x}, \bar{y}) \text{ is correct }] \geqslant 1 - (1/2)^{m+1}$, for each $i = 1, \ldots, 8mn$. Note that $\beta_i(\bar{x}, \bar{y})$ is correct, if for every assignment $\bar{a}$ and $\bar{b}$ on $\bar{x}$ and $\bar{y}$, respectively, $\beta_i(\bar{a}, \bar{b}) \in \oplus\mathsf{SAT}$ if and only if $\phi(\bar{a}, \bar{b})$ is a true QBF.

Assume that $\beta_i(\bar{x}, \bar{y})$ is correct. Let $\bar{b} : \bar{y} \to \{0, 1\}$ be such that $\varphi(\bar{b})$ is true QBF. Thus, for every assignment $\bar{a} : \bar{x} \to \{0, 1\}$, if $\varphi(\bar{a}, \bar{b})$ is true QBF, $\beta_i(\bar{a}, \bar{b}) \in \oplus\mathsf{SAT}$. Otherwise, $\beta_i(\bar{a}, \bar{b}) \notin \oplus\mathsf{SAT}$. So, we only need to consider all those assignments $\bar{a}$ such that $\phi_i(\bar{a}, \bar{b})$ is true, which by the induction hypothesis, is equivalent to saying that $\beta_i(\bar{a},\bar{b}) \in \oplus\mathsf{SAT}$. By applying the same technique as in Lemma 11.11 on the set of $\bar{a}$ such that $\beta_i(\bar{a},\bar{b}) \in \oplus\mathsf{SAT}$, we randomly "choose" the hash function h such that there is unique assignment $\bar{a}$ such that $h(\bar{a}) = 0$, and the probability that we choose such h is $\geqslant 3/(16n)$. Thus, we have:

$$\mathbf{Pr}[\ \beta_i(\bar{x},\bar{y}) \wedge h(\bar{x}) = 0 \text{ is correct } \mid \beta_i(\bar{x},\bar{y}) \text{ is correct }] \ \geqslant\ \frac{3}{16n}$$

Thus,

$$\begin{aligned}\mathbf{Pr}[\ \psi_i(\bar{x},\bar{y}) \text{ is correct }] \ =\ \mathbf{Pr}[\ \beta_i(\bar{x},\bar{y}) \wedge h(\bar{x}) = 0 \text{ is correct }] \ &\geqslant\ \frac{3}{16n}\Big(1-(1/2)^{m+1}\Big) \\ &\geqslant\ \frac{1}{8n}\end{aligned}$$

where in the last inequality we assume that $m \geqslant 1$.

Note also that if $\bar{b} : \bar{y} \to \{0,1\}$ is an assignment such that $\varphi(\bar{b})$ is false QBF, then $\beta_i(\bar{a},\bar{b}) \notin \oplus\mathsf{SAT}$, for every assignment $\bar{a}$ (since $\beta_i(\bar{x},\bar{y})$ is a correct formula). Thus, for any choice of h, $\beta_i(\bar{x},\bar{b}) \wedge h(\bar{x}) = 0 \notin \oplus\mathsf{SAT}$.

Finally, note that $\bigsqcup_{i=1}^{8mn} \alpha_i(\bar{y})$ is correct if and only if one of $\alpha_i(\bar{y})$ is correct. Therefore,

$$\begin{aligned}\mathbf{Pr}\Big[\ \bigsqcup_{i=1}^{8mn} \alpha_i(\bar{y}) \text{ is not correct }\Big] \ &=\ \mathbf{Pr}[\ \alpha_i(\bar{y}) \text{ is not correct, for each } i = 1,\ldots,8mn\ ] \\ &\leqslant\ \Big(1-1/(8n)\Big)^{8mn} \ \leqslant\ (1/2)^m\end{aligned}$$

The proof for the case where $\varphi(\bar{y})$ is of the form $\forall\bar{x}\ \phi(\bar{x},\bar{y})$ is similar. ■

Combining Lemma 11.5 and the fact that Σ_k-SAT and Π_k-SAT are $\mathbf{\Sigma}_k^p$- and $\mathbf{\Pi}_k^p$-complete, for each $k \geqslant 1$, we have the following theorem.

Theorem 11.6 (Reductions from languages in PH to $\oplus$SAT) *For every language $L \in \mathbf{PH}$, there is a probabilistic polynomial time algorithm $\mathcal{M}$ that on input w, outputs a formula ψ such that the following holds, where $n = |w|$.*

- *If $w \in L$, then $\mathbf{Pr}[\ \psi \in \oplus\mathsf{SAT}\ ] \geqslant 1-(1/2)^n$.*
- *If $w \notin L$, then $\mathbf{Pr}[\ \psi \in \oplus\mathsf{SAT}\ ] \leqslant (1/2)^n$.*

Appendix

A Pair-wise independent collection of hash functions

Definition 11.7 For $n,k \geqslant 1$, let $\mathcal{H}_{n,k}$ be a collection of functions from $\{0,1\}^n$ to $\{0,1\}^k$. We say that $\mathcal{H}_{n,k}$ is *pair-wise independent*, if for every $x,x' \in \{0,1\}^n$ where $x \neq x'$ and for every $y,y' \in \{0,1\}^k$, the following holds.

$$\mathbf{Pr}_{h\in\mathcal{H}_{n,k}}[\ h(x) = y \ \wedge\ h(x') = y'\ ] \ =\ 2^{-2k}$$

In the following we show that $\mathcal{H}_{n,k}$ exists. First, we show that $\mathcal{H}_{n,n}$ exists. For every $n \geqslant 1$, for every $a,b \in \mathsf{GF}(2^n)$, define a function $h_{a,b}$ from $\{0,1\}^n$ to $\{0,1\}^n$ as follows.[‡]

$$h_{a,b}(x) \ \stackrel{\mathsf{def}}{=}\ xa+b$$

[‡] $\mathsf{GF}(2^n)$ denotes a finite field with 2^n elements, where each element can be encoded as a 0-1 string of length n.

Theorem 11.8 *The collection* $\mathcal{H}_{n,n} \overset{\mathsf{def}}{=} \{h_{a,b} : a,b \in \mathit{GF}(2^n)\}$ *is pair-wise independent.*

We have another candidate for pair-wise independent collection. For every $n \geqslant 1$, for every $A \in \{0,1\}^{n\times n}$ and $b \in \{0,1\}^{n\times 1}$, define a function $h_{A,b}$ from $\{0,1\}^{n\times 1}$ to $\{0,1\}^{n\times 1}$ as follows.[§]

$$h_{A,b}(x) \overset{\mathsf{def}}{=} Ax + b$$

Theorem 11.9 *The collection* $\mathcal{H}_{n,n} \overset{\mathsf{def}}{=} \{h_{A,b} : A \in \{0,1\}^{n\times n}$ *and* $b \in \{0,1\}^{n\times 1}\}$ *is pair-wise independent.*

Remark 11.10 Note that the existence of $\mathcal{H}_{n,n}$ implies the existence of $\mathcal{H}_{n,k}$. If $n < k$, then we can use $\mathcal{H}_{k,k}$ and extend n bit inputs to k by padding with zeros. If $n > k$, then we can use $\mathcal{H}_{n,n}$ and reduce n bit outputs to k by truncating the last $(n-k)$ bits.

Lemma 11.11 (Valiant and Vazirani, 1986) *Let* $\mathcal{H}_{n,k}$ *be a pair-wise independent hash function collection. Let* $S \subseteq \{0,1\}^n$ *such that* $2^{k-2} \leqslant |S| \leqslant 2^{k-1}$. *Then, the following holds.*

$$\mathbf{Pr}_{h\in\mathcal{H}_{n,k}}[\text{ there is a unique } x \in S \text{ such that } h(x) = 0^k\ ] \ \geqslant\ \frac{3}{16}$$

Proof. Let N denote the number of x's such that $h(x) = 0$, where h is randomly chosen from $\mathcal{H}_{n,k}$ (with uniform distribution). We will calculate $\mathbf{Pr}[\ N = 1\ ]$. Note that:

$$\begin{aligned}\mathbf{Pr}[\ N = 1\ ] \ &= \ \mathbf{Pr}[\ N \geqslant 1\ ] \ - \ \mathbf{Pr}[\ N \geqslant 2\ ] \\ &= \ \mathbf{Pr}\Big[\ \bigcup_{x\in S} \mathcal{E}_x\ \Big] \ - \ \mathbf{Pr}\Big[\ \bigcup_{x,x'\in S \text{ and } x\neq x'} \mathcal{E}_x \cap \mathcal{E}_{x'}\ \Big]\end{aligned}$$

where $\mathcal{E}_x$ denotes the event that $h(x) = 0$. In the following, we let $p = 2^{-k}$.

Since $\mathcal{H}_{n,k}$ is pairwise independent, $\mathbf{Pr}[\ \mathcal{E}_x\ ] = p$ and $\mathbf{Pr}[\ \mathcal{E}_x \cap \mathcal{E}_{x'}\ ] = p^2$, whenever $x \neq x'$.

By the inclusion-exclusion principle, we have:

$$\mathbf{Pr}\Big[\ \bigcup_{x\in S} \mathcal{E}_x\ \Big] \ \geqslant\ \sum_{x\in S} \mathbf{Pr}[\ \mathcal{E}_x\ ] \ - \ \sum_{x,x'\in S \text{ and } x\neq x'} \mathbf{Pr}[\ \mathcal{E}_x \cap \mathcal{E}_{x'}\ ] \ = \ |S|p \ - \ \binom{|S|}{2}\cdot p^2$$

By union bound, we have:

$$\mathbf{Pr}\Big[\ \bigcup_{x,x'\in S \text{ and } x\neq x'} \mathcal{E}_x \cap \mathcal{E}_{x'}\ \Big] \ \leqslant\ \sum_{x,x'\in S \text{ and } x\neq x'} \mathbf{Pr}[\ \mathcal{E}_x \cap \mathcal{E}_{x'}\ ] \ \leqslant\ \binom{|S|}{2}\cdot p^2$$

Combining both, we have:

$$\mathbf{Pr}[\ N = 1\ ] \ = \ \mathbf{Pr}[\ N \geqslant 1\ ] - \mathbf{Pr}[\ N \geqslant 2\ ] \ \geqslant\ |S|p \ - \ |S|^2p^2$$

Since $1/4 \leqslant |S|p \leqslant 1/2$, a straightforward calculation shows that $|S|p - |S|^2p^2 \geqslant 3/16$. ■

[§] $\{0,1\}^{n\times n}$ denotes the set of 0-1 matrices with n rows and n columns and $\{0,1\}^{n\times 1}$ denotes the set of 0-1 column vectors of n rows. Here the addition $+$ and multiplication $\cdot$ are defined over $\mathbb{Z}_2$.

Lesson 12: Toda's theorem

Theme: Toda's theorem which states that every language in the polynomial hierarchy can be decided by a polynomial time DTM with oracle access to $\sharp\mathsf{SAT}$, i.e., $\mathbf{PH} \subseteq \mathbf{P}^{\sharp\mathsf{SAT}}$.

Theorem 12.1 (Toda, 1991) $\mathbf{PH} \subseteq \mathbf{P}^{\sharp\mathbf{P}}$.

1 Reduction from $\oplus\mathsf{SAT}$ to $\sharp\mathsf{SAT}$

In the following we will use the notations from Note 11. Recall that $\sharp\varphi$ denote the number of satisfying assignments of a (Boolean) formula φ. Fwo formulas φ and ψ, the formula $\varphi \sqcap \psi$ is a formula such that $\sharp(\varphi \sqcap \psi) = \sharp\varphi \cdot \sharp\psi$.

We define an operation $+$ as follows. Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$ be the variables in φ and ψ, respectively. Let z be a new variable.

$$\varphi + \psi \quad \stackrel{\mathsf{def}}{=} \quad \Big(\varphi \wedge z \wedge \bigwedge_{i=1}^{m} y_i\Big) \quad \vee \quad \Big(\psi \wedge \neg z \wedge \bigwedge_{i=1}^{n} x_i\Big)$$

Note that $\sharp(\varphi + \psi) = \sharp\varphi + \sharp\psi$.

Lemma 12.2 *There is a deterministic polynomial time algorithm $\mathcal{T}$, that on input formula φ and positive integer m (in unary), outputs a formula ψ such that the following holds.*

- *If $\varphi \in \oplus\mathsf{SAT}$, then $\sharp\psi \equiv -1 \pmod{2^{m+1}}$.*
- *If $\varphi \notin \oplus\mathsf{SAT}$, then $\sharp\psi \equiv 0 \pmod{2^{m+1}}$.*

Proof. We will use the following identity for each $i \geqslant 0$ and n.

(a) If $n \equiv -1 \pmod{2^{2^i}}$, then $4n^3 + 3n^4 \equiv -1 \pmod{2^{2^{i+1}}}$.

(b) If $n \equiv 0 \pmod{2^{2^i}}$, then $4n^3 + 3n^4 \equiv 0 \pmod{2^{2^{i+1}}}$.

On input φ and m, the algorithm $\mathcal{T}$ does the following.

- For each $i = 0, 1, \ldots, \lceil \log(m+1) \rceil$, define a formula ψ_i as follows.

$$\psi_i \stackrel{\mathsf{def}}{=} \begin{cases} \varphi & \text{if } i = 0 \\ 4\psi_{i-1}^3 \ + \ 3\psi_{i-1}^4 & \text{if } i \geqslant 1 \end{cases}$$

  Here $4\psi_{i-1}^3 \ + \ 3\psi_{i-1}^4$ denotes the formula that has $4\sharp(\psi_{i-1})^4 + 3\sharp(\psi_{i-1})^3$ satisfying assignments. Note that such formula can be constructed easily using and using the operations $+$ and $\sqcap$.

- Output the formula $\psi_{\lceil \log(m+1) \rceil}$.

It is not difficult to show that the algorithm $\mathcal{T}$ runs in polynomial time. Its correctness follows directly from the identities (a) and (b). ■

2 Proof of Theorem 12.1

Let $L \in \mathbf{PH}$. We want to show that $L \in \mathbf{P}^{\sharp\mathsf{SAT}}$. By Theorem 11.6, there is a probabilistic polynomial time algorithm $\mathcal{M}_1$ that on input w, outputs a formula ψ such that the following holds.

- If $w \in L$, then $\mathbf{Pr}[\ \psi \in \oplus\mathsf{SAT}\ ] \geqslant 3/4$.
- If $w \notin L$, then $\mathbf{Pr}[\ \psi \in \oplus\mathsf{SAT}\ ] \leqslant 1/4$.

Using the alternative definition of PTM, we view $\mathcal{M}_1$ as a DTM with two input (w, r), where r is a random string. Let ℓ be the length of the random string.

Let $\mathcal{M}_2$ be the algorithm, that on input w and random string r, outputs the formula $\mathcal{T}(\mathcal{M}_1(w,r), \ell+2)$, where $\mathcal{T}$ is the algorithm in Lemma 12.2. That is, it first runs $\mathcal{M}_1(w,r)$ and then runs $\mathcal{T}$ on input $(\mathcal{M}_1(w,r), \ell+2)$

Combining Theorem 11.6 and Lemma 12.2, on input w and random string r, the algorithm $\mathcal{M}_2$ outputs a formula $\psi_{w,r}$ such that the following holds.

- If $w \in L$, then $\mathbf{Pr}_{r\in\{0,1\}^\ell}[\ \sharp\psi_{w,r} \equiv -1 \pmod{2^{\ell+3}}\ ] \geqslant 3/4$.
- If $w \notin L$, then $\mathbf{Pr}_{r\in\{0,1\}^\ell}[\ \sharp\psi_{w,r} \equiv -1 \pmod{2^{\ell+3}}\ ] \leqslant 1/4$.

This is equivalent to the following.

- If $w \in L$, the sum $\sum_{r\in\{0,1\}^\ell} \sharp\psi_{w,r}$ lies in between -2^ℓ and $-\frac{3}{4}2^\ell$ (modulo $2^{\ell+3}$).
- If $w \notin L$, the sum $\sum_{r\in\{0,1\}^\ell} \sharp\psi_{w,r}$ lies in between $-\frac{1}{4}2^\ell$ and 0 (modulo $2^{\ell+3}$).

The sets of values that lie in between -2^ℓ and $-\frac{3}{4}2^\ell$ and in between $-\frac{1}{4}2^\ell$ and 0 (modulo $2^{\ell+3}$) are the following sets P and Q, respectively:

$$P \stackrel{\mathsf{def}}{=} \{28\cdot 2^{\ell-2}, \ldots, 29\cdot 2^{\ell-2}\} \qquad \text{and} \qquad Q \stackrel{\mathsf{def}}{=} \{31\cdot 2^{\ell-2}, \ldots, 2^{\ell+3}-1\} \cup \{0\}$$

Note that P and Q are disjoint.

The main idea of Theorem 12.1 is that on input word w, the algorithm asks the $\sharp\mathsf{SAT}$ oracle for the value $\sum_{r\in\{0,1\}^\ell} \sharp\psi_{w,r}$ and checks whether the value is in the set P or Q. To this end, we need to construct a formula whose number of satisfying assignments is exactly $\sum_{r\in\{0,1\}^\ell} \sharp\psi_{w,r}$.

Consider the following NTM $\mathcal{M}'$. On input word w, it does the following.

- Guess a string $r \in \{0,1\}^\ell$.
- Run $\mathcal{M}_2$ on (w,r) to obtain a formula $\psi_{w,r}$.
- Guess a satisfying assignment for $\psi_{w,r}$.
- ACCEPT if and only if the guessed assignment is indeed a satisfying assignment for $\psi_{w,r}$.

Obviously, for every w, the number of accepting runs of $\mathcal{M}'$ on w is precisely $\sum_{r\in\{0,1\}^\ell} \sharp\psi_{w,r}$.

Now, to complete our proof, we present a polynomial time DTM $\mathcal{M}$ that decides L (with oracle access to $\sharp\mathsf{SAT}$). On input w, it does the following.

- Construct a formula Ψ_w such that the number of satisfying assignments of Ψ_w is exactly the number of accepting runs of $\mathcal{M}'$ on w.

  Here we use Cook-Levin construction (on w and the transitions in $\mathcal{M}'$). Recall that Cook-Levin reduction is parsimonious.
- Determine the value $\sharp\Psi_w$ (modulo $2^{\ell+3}$) by querying the $\sharp\mathsf{SAT}$ oracle.
- Determine whether $\sharp\Psi_w$ lies in P or Q, the answer of which implies whether $w \in L$.