Computational Complexity Theory

Lecture 13: Polynomial Hierarchy (contd.); Boolean circuits; Class P/poly

Department of Computer Science,
Indian Institute of Science

Recap: Class $\sum_i$

- Definition. A language L is in $\sum_i$ if there's a polynomial function $q(.)$ and a poly-time TM M (the "verifier") s.t.

$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} \ \forall u_2 \in \{0,1\}^{q(|x|)} \ Q_i u_i \in \{0,1\}^{q(|x|)}$$

$$\text{s.t. } M(x,u_1,\ldots, u_i) = 1,$$

where Q_i is $\exists$ or $\forall$ if i is odd or even, respectively.

- Obs. $\sum_i \subseteq \sum_{i+1}$ for every i.

Recap: Polynomial Hierarchy

- **Definition.** A language L is in Σ_i if there's a polynomial function $q(.)$ and a poly-time TM M (the "verifier") s.t.

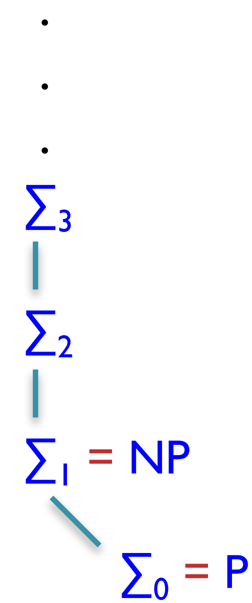
$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} \ \forall u_2 \in \{0,1\}^{q(|x|)} \ Q_i u_i \in \{0,1\}^{q(|x|)}$$

$$\text{s.t. } M(x, u_1, \ldots, u_i) = 1,$$

where Q_i is $\exists$ or $\forall$ if i is odd or even, respectively.

- **Definition.** *(Meyer & Stockmeyer 1972)*

$$PH = \bigcup_{i \in \mathbf{N}} \Sigma_i \, .$$



Recap: Class Π_i

- Definition. Π_i = co-Σ_i = $\{ L : \overline{L} \in \Sigma_i \}$.

- Obs. A language L is in Π_i if there's a polynomial function $q(.)$ and a poly-time TM M (the "verifier") s.t.

$$x \in L \iff \forall u_1 \in \{0,1\}^{q(|x|)}\ \exists u_2 \in \{0,1\}^{q(|x|)}\ Q_i u_i \in \{0,1\}^{q(|x|)}$$
$$\text{s.t. } M(x,u_1,\ldots,u_i) = 1,$$

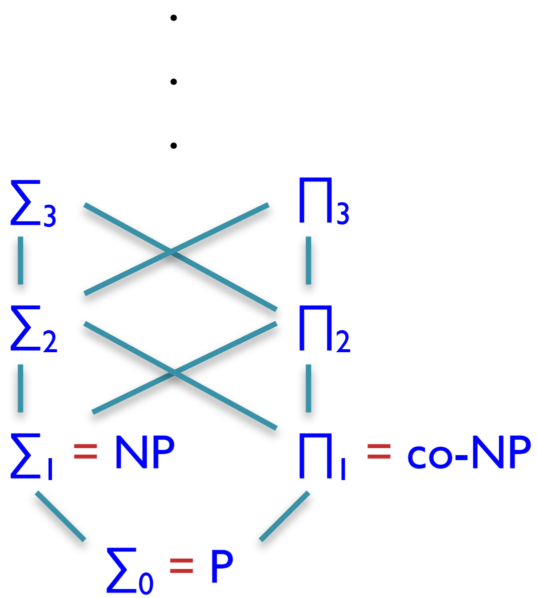
  where Q_i is $\forall$ or $\exists$ if i is odd or even, respectively.

- Obs. $\Sigma_i \subseteq \Pi_{i+1} \subseteq \Sigma_{i+2}$.

Recap: Polynomial Hierarchy

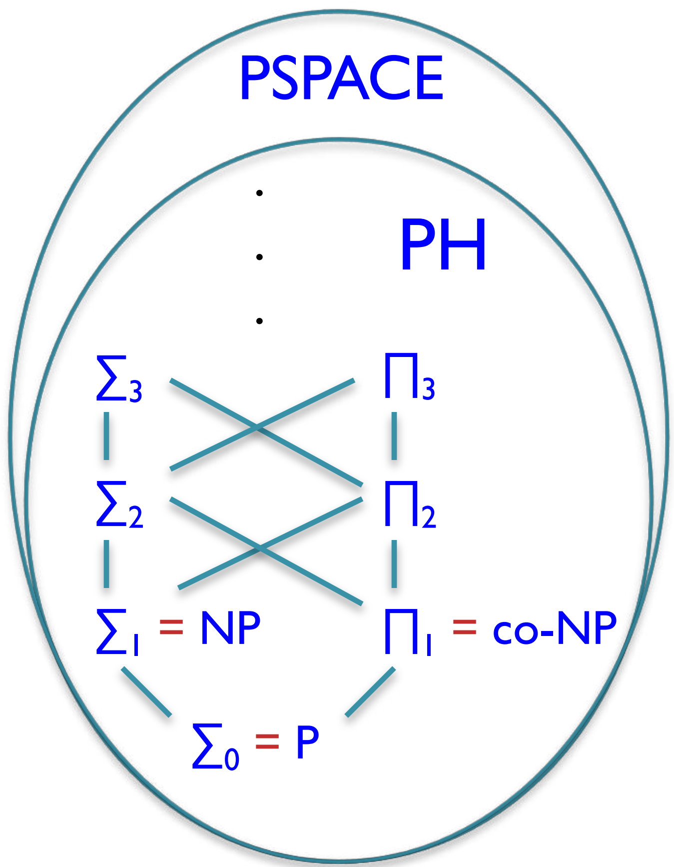
- Obs. $PH = \bigcup_{i \in \mathbf{N}} \Sigma_i = \bigcup_{i \in \mathbf{N}} \Pi_i$.

PH =



Recap: Polynomial Hierarchy

- Claim. PH $\subseteq$ PSPACE .
- Proof. Similar to the proof of TQBF $\in$ PSPACE.



Recap: Does PH collapse?

- **General belief.** Just as many of us believe $P \neq NP$ (i.e. $\Sigma_0 \neq \Sigma_1$) and $NP \neq \text{co-}NP$ (i.e. $\Sigma_1 \neq \Pi_1$), we also believe that for every i, $\Sigma_i \neq \Sigma_{i+1}$ and $\Sigma_i \neq \Pi_i$.

- **Definition.** We say PH <u>collapses to the i-th level</u> if $\Sigma_i = \Sigma_{i+1}$. (justified in the next theorem)

- **Conjecture.** There is no i such that PH collapses to the i-th level.

This is stronger than the $P \neq NP$ conjecture.

Recap: PH collapse theorems

- Theorem. If $\Sigma_i = \Sigma_{i+1}$ then $PH = \Sigma_i$.
- Theorem. If $\Sigma_i = \Pi_i$ then $PH = \Sigma_i$.

Recap: Complete problems in PH ?

- Recall, to define completeness of a complexity class, we need an appropriate notion of a *reduction*.
- What kind of reductions will be suitable is guided by *a complexity question*, like a comparison between the complexity class under consideration & another class.
- Is P = PH ? …use poly-time Karp reduction!

- Definition. A language L' is *PH-hard* if for every L in PH, $L \leq_p L'$. Further, if L' is in PH then L' is *PH-complete*.

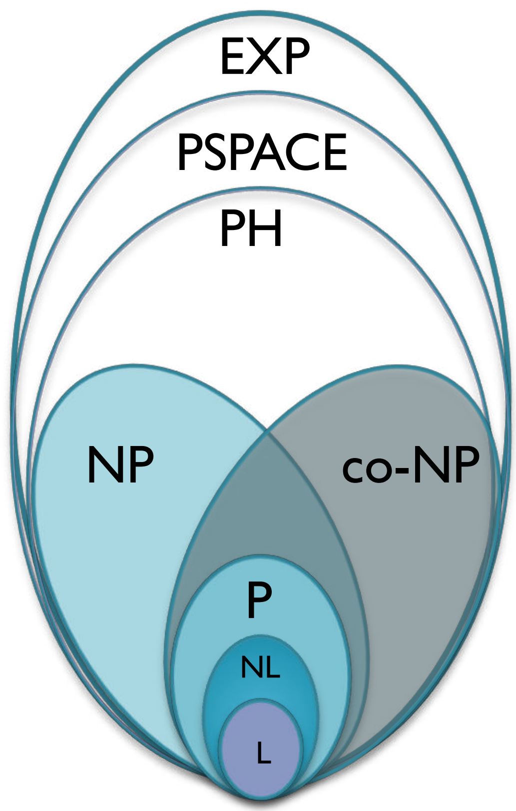
Recap: Complete problems in PH ?

- Fact. If L is poly-time reducible to a language in Σ_i then L is in Σ_i . (we've seen a similar fact for NP)

- Observation. If PH has a complete problem then PH collapses.
- Proof. If L is *PH-complete* then L is in Σ_i for some i. Now use the above fact to infer that PH = Σ_i .

Recap: Complete problems in PH ?

- Fact. If L is poly-time reducible to a language in Σ_i then L is in Σ_i . (we've seen a similar fact for NP)
- Corollary. PH $\subsetneq$ PSPACE unless PH collapses.



Recap: Complete problems in Σ_i

- Recall, to define completeness of a complexity class, we need an appropriate notion of a *reduction*.
- What kind of reductions will be suitable is guided by *a complexity question*, like a comparison between the complexity class under consideration & another class.
- Is $P = \Sigma_i$? …use poly-time Karp reduction!

- Definition. A language L' is Σ_i *-hard* if for every L in Σ_i, $L \leq_p L'$. Further, if L' is in Σ_i then L' is Σ_i *-complete*.

Recap: Complete problems in Σ_i

- Definition. The language Σ_i-SAT contains all *true* QBF with i alternating quantifiers starting with $\exists$.

- Theorem. Σ_i-SAT is Σ_i *-complete*. (Σ_1-SAT is just SAT)

- Observation. Owing to the proof of the Cook-Levin theorem, we can assume that the formula in a Σ_i-SAT instance is a CNF (if i is odd) or a DNF (if i is even).

Recap: Other complete problems in Σ_2

- Ref. *"Completeness in the Polynomial-Time Hierarchy: A Compendium"* by *Schaefer and Umans (2008)*.

- Theorem. Eq-DNF and Succinct-SetCover are Σ_2 *-complete*.

An alternate characterization of PH

Oracle definition of Σ_i

- **Definition.** A language L is in $NP^{\Sigma_i\text{-}SAT}$ if there is a poly-time NTM with oracle access to Σ_i-SAT that decides L.

- **Theorem.** $\Sigma_{i+1} = NP^{\Sigma_i\text{-}SAT}$.

Oracle definition of Σ_i

- Definition. A language L is in $NP^{\Sigma_i\text{-SAT}}$ if there is a poly-time NTM with oracle access to Σ_i-SAT that decides L.

- Theorem. $\Sigma_{i+1} = NP^{\Sigma_i\text{-SAT}}$.

- Observe that Σ_1-SAT = SAT. We'll prove the special case $\Sigma_2 = NP^{SAT}$. The proof of the theorem is similar.

Oracle definition of Σ_i

- Theorem. $\Sigma_2 = NP^{SAT}$.
- Proof. Let L be a language in Σ_2. There's a polynomial function q(.) and a poly-time TM M s.t.

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

Oracle definition of Σ_i

- Theorem. $\Sigma_2 = NP^{SAT}$.
- Proof. Let L be a language in Σ_2. There's a polynomial function q(.) and a poly-time TM M s.t.

$$x \in L \iff \exists u \in \{0,1\}^{q(|x|)} \ \forall v \in \{0,1\}^{q(|x|)} \text{ s.t. } \underbrace{\phi(x,u,v)}_{\substack{\text{Boolean circuit} \\ (\textit{by Cook-Levin})}} = 1.$$

- In fact, owing to the proof of the Cook-Levin theorem, we can assume that ϕ is a DNF.

Oracle definition of $\sum_i$

- Theorem. $\sum_2 = NP^{SAT}$.
- Proof. Let L be a language in $\sum_2$. There's a polynomial function $q(.)$ and a poly-time TM M s.t.

$x \in L \Leftrightarrow \exists u \in \{0,1\}^{q(|x|)}\ \forall v \in \{0,1\}^{q(|x|)}$ s.t. $\neg\phi(x,u,v) = 0$.

- Think of a NTM N that has the knowledge of M. On input x, it guesses $u \in \{0,1\}^{q(|x|)}$ non-deterministically and computes the circuit $\phi(x,u,v)$. Then, it queries the SAT oracle with $\neg\phi(x,u,v)$.

Oracle definition of $\sum_i$

- Theorem. $\sum_2 = NP^{SAT}$.
- Proof. Let L be a language in $\sum_2$. There's a polynomial function $q(.)$ and a poly-time TM M s.t.

  $x \in L \Leftrightarrow \exists u \in \{0,1\}^{q(|x|)} \ \forall v \in \{0,1\}^{q(|x|)}$ s.t. $\neg\phi(x,u,v) = 0$.

- Think of a NTM N that has the knowledge of M. On input x, it guesses $u \in \{0,1\}^{q(|x|)}$ non-deterministically and computes the circuit $\phi(x,u,v)$. Then, it queries the SAT oracle with $\neg\phi(x,u,v)$.
- Note that $\neg\phi(x,u,v)$ is a CNF.

Oracle definition of Σ_i

- Theorem. $\Sigma_2 = NP^{SAT}$.
- Proof. Let L be a language in NP^{SAT}. There's a NTM N that decides L with oracle access to SAT.
- Special case: N asks at most **one** query to the SAT oracle on every computation path on input x.

Oracle definition of Σ_i

- Theorem. $\Sigma_2 = NP^{SAT}$.
- Proof. Let L be a language in NP^{SAT}. There's a NTM N that decides L with oracle access to SAT.
- Special case: N asks at most **one** query to the SAT oracle on every computation path on input x.
- We need to construct a Σ_2-statement that captures N's computation on input x.

Oracle definition of Σ_i

- Theorem. $\Sigma_2 = NP^{SAT}$.
- Proof. Let L be a language in NP^{SAT}. There's a NTM N that decides L with oracle access to SAT.
- Special case: N asks at most one query to the SAT oracle on every computation path on input x.
- Think of a TM M that takes input x and $w \in \{0,1\}^{q(|x|)}$, $a_1 \in \{0,1\}$ and u_1, $v_1 \in \{0,1\}^{q(|x|)}$, where $q(|x|)$ is the runtime of N on input x, and does the following:

Oracle definition of Σ_i

- Theorem. $\Sigma_2 = NP^{SAT}$.
- Proof. Let L be a language in NP^{SAT}. There's a NTM N that decides L with oracle access to SAT.
- Special case: N asks at most one query to the SAT oracle on every computation path on input x.
- Think of a TM M that takes input x and $w \in \{0,1\}^{q(|x|)}$, $a_1 \in \{0,1\}$ and u_1, $v_1 \in \{0,1\}^{q(|x|)}$, where $q(|x|)$ is the runtime of N on input x, and does the following:
- M simulates N on input x with w as the non-deterministic choices.

Oracle definition of Σ_i

- Theorem. $\Sigma_2 = NP^{SAT}$.
- Proof. Let L be a language in NP^{SAT}. There's a NTM N that decides L with oracle access to SAT.
- Special case: N asks at most one query to the SAT oracle on every computation path on input x.
- Think of a TM M that takes input x and $w \in \{0,1\}^{q(|x|)}$, $a_1 \in \{0,1\}$ and $u_1, v_1 \in \{0,1\}^{q(|x|)}$, where $q(|x|)$ is the runtime of N on input x, and does the following:
- M simulates N on input x with w as the <u>computation path</u>. Suppose ϕ is the query asked by N on the path of computation defined by w.

Oracle definition of Σ_i

- Theorem. $\Sigma_2 = NP^{SAT}$.
- Proof. Let L be a language in NP^{SAT}. There's a NTM N that decides L with oracle access to SAT.
- Special case: N asks at most one query to the SAT oracle on every computation path on input x.
- Think of a TM M that takes input x and $w \in \{0,1\}^{q(|x|)}$, $a_1 \in \{0,1\}$ and $u_1, v_1 \in \{0,1\}^{q(|x|)}$, where $q(|x|)$ is the runtime of N on input x, and does the following:

➢ If $a_1 = 1$ and $\phi(u_1) = 1$, M continues the simulation; else it stops and outputs 0. (In this case, M ignores v_1.)

Oracle definition of Σ_i

- Theorem. $\Sigma_2 = NP^{SAT}$.
- Proof. Let L be a language in NP^{SAT}. There's a NTM N that decides L with oracle access to SAT.
- Special case: N asks at most one query to the SAT oracle on every computation path on input x.
- Think of a TM M that takes input x and $w \in \{0,1\}^{q(|x|)}$, $a_1 \in \{0,1\}$ and $u_1, v_1 \in \{0,1\}^{q(|x|)}$, where $q(|x|)$ is the runtime of N on input x, and does the following:

➢ If $a_1 = 0$ and $\phi(v_1) = 0$, M continues the simulation; else it stops and outputs 0. (In this case, M ignores u_1.)

Oracle definition of Σ_i

- Theorem. $\Sigma_2 = NP^{SAT}$.
- Proof. Let L be a language in NP^{SAT}. There's a NTM N that decides L with oracle access to SAT.
- Special case: N asks at most one query to the SAT oracle on every computation path on input x.
- Think of a TM M that takes input x and $w \in \{0,1\}^{q(|x|)}$, $a_1 \in \{0,1\}$ and $u_1, v_1 \in \{0,1\}^{q(|x|)}$, where $q(|x|)$ is the runtime of N on input x, and does the following:
- At the end of the simulation, M outputs whatever N outputs. Note: M is a poly-time TM.

Oracle definition of Σ_i

- Theorem. $\Sigma_2 = NP^{SAT}$.
- Proof. Let L be a language in NP^{SAT}. There's a NTM N that decides L with oracle access to SAT.
- Special case: N asks at most one query to the SAT oracle on every computation path on input x.
- Observation. For any $w \in \{0,1\}^{q(|x|)}$ and $a_1 \in \{0,1\}$,

➢ N on computation path w gets answer a_1 from the SAT oracle and accepts x $\Leftrightarrow$

$$\exists u_1 \in \{0,1\}^{q(|x|)} \ \forall v_1 \in \{0,1\}^{q(|x|)} \text{ s.t. } M(x,w,a_1,u_1,v_1) = 1.$$

(…will prove the observation shortly. Let's finish the proof.)

Oracle definition of Σ_i

- Theorem. $\Sigma_2 = NP^{SAT}$.
- Proof. Let L be a language in NP^{SAT}. There's a NTM N that decides L with oracle access to SAT.
- Special case: N asks at most one query to the SAT oracle on every computation path on input x.
- $x \in L \iff \exists w \in \{0,1\}^{q(|x|)}, a_1 \in \{0,1\}$ s.t

➢ N on computation path w gets answer a_1 from the SAT oracle and accepts $x \iff \exists w \in \{0,1\}^{q(|x|)}, a_1 \in \{0,1\}$

$\exists u_1 \in \{0,1\}^{q(|x|)}\ \forall v_1 \in \{0,1\}^{q(|x|)}$ s.t. $M(x,w,a_1,u_1,v_1) = 1$.

Oracle definition of Σ_i

- Theorem. $\Sigma_2 = NP^{SAT}$.
- Proof. Let L be a language in NP^{SAT}. There's a NTM N that decides L with oracle access to SAT.
- Special case: N asks at most one query to the SAT oracle on every computation path on input x.
- $x \in L \iff \exists w \in \{0,1\}^{q(|x|)}, a_1 \in \{0,1\}$ s.t

➢ N on computation path w gets answer a_1 from the SAT oracle and accepts $x \iff \exists w \in \{0,1\}^{q(|x|)}, a_1 \in \{0,1\}$

$\exists u_1 \in \{0,1\}^{q(|x|)}\ \forall v_1 \in \{0,1\}^{q(|x|)}$ s.t. $M(x, w, \underbrace{a_1, u_1}_{\text{Call it } u}, v_1) = 1$.

Oracle definition of Σ_i

- Theorem. $\Sigma_2 = NP^{SAT}$.
- Proof. Let L be a language in NP^{SAT}. There's a NTM N that decides L with oracle access to SAT.
- Special case: N asks at most one query to the SAT oracle on every computation path on input x.
- $x \in L \iff \exists w \in \{0,1\}^{q(|x|)}, a_1 \in \{0,1\}$ s.t
  - N on computation path w gets answer a_1 from the SAT oracle and accepts x $\iff$

    $\exists u \in \{0,1\}^{2q(|x|)+1} \ \forall v_1 \in \{0,1\}^{q(|x|)}$ s.t. $M(x,u,v_1) = 1$.
- Therefore, L is in Σ_2 .

Proof of the observation

- Observation. For any $w \in \{0,1\}^{q(|x|)}$ and $a_1 \in \{0,1\}$,

➢ N on computation path w gets answer a_1 from the SAT oracle and accepts x $\Leftrightarrow$

$\exists u_1 \in \{0,1\}^{q(|x|)}$ $\forall v_1 \in \{0,1\}^{q(|x|)}$ s.t. $M(x,w,a_1,u_1,v_1) = 1$.

- Proof.($\Rightarrow$) M simulates N on computation path w. Let ϕ be the query asked by N to SAT.
- If $a_1 = 1$, $\exists u_1 \in \{0,1\}^{q(|x|)}$ $\phi(u_1) = 1$ and N accepts x.

Proof of the observation

- Observation. For any $w \in \{0,1\}^{q(|x|)}$ and $a_1 \in \{0,1\}$,

➢ N on computation path w gets answer a_1 from the SAT oracle and accepts x ⟺

$\exists u_1 \in \{0,1\}^{q(|x|)}\ \forall v_1 \in \{0,1\}^{q(|x|)}$ s.t. $M(x,w,a_1,u_1,v_1) = 1$.

- Proof.(⟹) M simulates N on computation path w. Let ϕ be the query asked by N to SAT.
- If $a_1 = 1$, $\exists u_1 \in \{0,1\}^{q(|x|)}$ s.t. $M(x,w,a_1,u_1,v_1) = 1$.

In this case, M ignores v_1.

Proof of the observation

- Observation. For any $w \in \{0,1\}^{q(|x|)}$ and $a_1 \in \{0,1\}$,
- ➢ N on computation path w gets answer a_1 from the SAT oracle and accepts x ⟺

  $\exists u_1 \in \{0,1\}^{q(|x|)}$ $\forall v_1 \in \{0,1\}^{q(|x|)}$ s.t. $M(x,w,a_1,u_1,v_1) = 1$.

- Proof.(⟹) M simulates N on computation path w. Let ϕ be the query asked by N to SAT.
- If $a_1 = 0$, $\forall v_1 \in \{0,1\}^{q(|x|)}$ $\phi(v_1) = 0$ and N accepts x.

Proof of the observation

- Observation. For any $w \in \{0,1\}^{q(|x|)}$ and $a_1 \in \{0,1\}$,

➢ N on computation path w gets answer a_1 from the SAT oracle and accepts x ⟺

$\exists u_1 \in \{0,1\}^{q(|x|)}\ \forall v_1 \in \{0,1\}^{q(|x|)}$ s.t. $M(x,w,a_1,u_1,v_1) = 1$.

- Proof.(⇒) M simulates N on computation path w. Let ϕ be the query asked by N to SAT.
- If $a_1 = 0$, $\forall v_1 \in \{0,1\}^{q(|x|)}$ s.t. $M(x,w,a_1,u_1,v_1) = 1$.

In this case, M ignores u_1.

Proof of the observation

- Observation. For any $w \in \{0,1\}^{q(|x|)}$ and $a_1 \in \{0,1\}$,

➢ N on computation path w gets answer a_1 from the SAT oracle and accepts x ⟺

$$\exists u_1 \in \{0,1\}^{q(|x|)} \ \forall v_1 \in \{0,1\}^{q(|x|)} \text{ s.t. } M(x, w, a_1, u_1, v_1) = 1.$$

- Proof.(⟹) M simulates N on computation path w. Let ϕ be the query asked by N to SAT.
- Irrespective of the value of a_1,

$$\exists u_1 \in \{0,1\}^{q(|x|)} \ \forall v_1 \in \{0,1\}^{q(|x|)} \text{ s.t. } M(x, w, a_1, u_1, v_1) = 1.$$

Proof of the observation

- Observation. For any $w \in \{0,1\}^{q(|x|)}$ and $a_1 \in \{0,1\}$,

➢ N on computation path w gets answer a_1 from the SAT oracle and accepts x ⟺

$\exists u_1 \in \{0,1\}^{q(|x|)}\ \forall v_1 \in \{0,1\}^{q(|x|)}$ s.t. $M(x,w,a_1,u_1,v_1) = 1$.

- Proof.(⟸) Need to show that N on computation path w <u>gets answer a_1</u> from the SAT oracle. (*Homework*)

Oracle definition of Σ_i

- Theorem. $\Sigma_2 = NP^{SAT}$.
- Proof. Let L be a language in NP^{SAT}. There's a NTM N that decides L with oracle access to SAT.
- General case: N asks at most $q(|x|)$ queries to SAT oracle on every computation path on input x.
- *Homework*: Prove the general case. Define the poly-time machine M appropriately.

Oracles versus efficient algorithms

- Definition. A language L is in P^{SAT} if there is a poly-time TM with oracle access to SAT that decides L.
- $\Delta_2 := P^{SAT} \subseteq \Sigma_2 \cap \Pi_2$.
- A SAT oracle gives us the ability to solve SAT efficiently "much like" a poly-time algorithm for SAT.

Oracles versus efficient algorithms

- Definition. A language L is in P^{SAT} if there is a poly-time TM with oracle access to SAT that decides L.
- $\Delta_2 := P^{SAT} \subseteq \Sigma_2 \cap \Pi_2$.
- A SAT oracle gives us the ability to solve SAT efficiently "much like" a poly-time algorithm for SAT.
- Yet, in the first case we believe $P^{SAT} \neq NP^{SAT}$, (otherwise, PH collapses to Σ_2)

Oracles versus efficient algorithms

- Definition. A language L is in P^{SAT} if there is a poly-time TM with oracle access to SAT that decides L.
- $\Delta_2 := P^{SAT} \subseteq \Sigma_2 \cap \Pi_2$.
- A SAT oracle gives us the ability to solve SAT efficiently "much like" a <u>poly-time algorithm for SAT</u>.

- Yet, in the first case we believe $P^{SAT} \neq NP^{SAT}$, whereas in the <u>second case</u> PH collapses to P, i.e., $P^{SAT} = NP^{SAT}$.

Oracles versus efficient algorithms

- Definition. A language L is in P^{SAT} if there is a poly-time TM with oracle access to SAT that decides L.
- $\Delta_2 := P^{SAT} \subseteq \Sigma_2 \cap \Pi_2$.
- A SAT oracle gives us the ability to solve SAT efficiently "much like" a poly-time algorithm for SAT.

- Yet, in the first case we believe $P^{SAT} \neq NP^{SAT}$, whereas in the second case PH collapses to P, i.e., $P^{SAT} = NP^{SAT}$.
- Why? Think to understand the difference between oracles and poly-time algorithms for SAT (*Homework*).

Boolean Circuits

An algorithm for every input length?

- *"One might imagine that $P \neq NP$, but SAT is tractable in the following sense: for every ℓ there is a very short program that runs in time ℓ^2 and correctly treats all instances of size ℓ."* --- Karp and Lipton (1982).

An algorithm for every input length?

- *"One might imagine that P ≠ NP, but SAT is tractable in the following sense: for every ℓ there is a very short program that runs in time ℓ^2 and correctly treats all instances of size ℓ."* --- Karp and Lipton (1982).

- P ≠ NP rules out the existence of a <u>single</u> efficient algorithm for SAT that handles <u>all</u> input lengths. But, it doesn't rule out the possibility of having <u>a sequence of</u> efficient SAT algorithms – one <u>for each input length</u>.

Lesson learnt from Cook-Levin

- Locality of computation implies that an algorithm A working on inputs of some fixed length n and running in time $T(n)$ can be viewed as a Boolean circuit ϕ of size $O(T(n)^2)$ s.t. $A(x) = \phi(x)$ for every $x \in \{0,1\}^n$.
- On the other hand, a circuit on inputs of length n and of size S can be viewed as an algorithm working on length n inputs and running in time S.

Lesson learnt from Cook-Levin

- Locality of computation implies that an algorithm A working on inputs of some fixed length n and running in time $T(n)$ can be viewed as a Boolean circuit ϕ of size $O(T(n)^2)$ s.t. $A(x) = \phi(x)$ for every $x \in \{0,1\}^n$.
- On the other hand, a circuit on inputs of length n and of size S can be viewed as an algorithm working on length n inputs and running in time S.

- To rule the existence of a sequence of algorithms – one for each input length – we need to rule out the existence of a sequence of <u>(i.e., a family of) circuits</u>.

Boolean circuits

- A <u>Boolean circuit</u> is a directed acyclic graph whose nodes/gates are labelled as follows:
- ➢ A node with in-degree zero is labelled by an input variable, and it outputs the value of the variable.
- ➢ Any other node is labelled by one of the three operations $\wedge$, $\vee$, $\neg$, and it outputs the value of the operation on its input.

Nodes with out-degree zero are the output gates.

Boolean circuits

- A <u>Boolean circuit</u> is a directed acyclic graph whose nodes/gates are labelled as follows:

➢ A node with in-degree zero is labelled by an input variable, and it outputs the value of the variable.

➢ Any other node is labelled by one of the three operations $\wedge$, $\vee$, $\neg$, and it outputs the value of the operation on its input.

Nodes with out-degree zero are the output gates.

- Typically, we'll consider circuits with one output gate, and with nodes having in-degree at most two.

Boolean circuits

- A <u>Boolean circuit</u> is a directed acyclic graph whose nodes/gates are labelled as follows:

  - A node with in-degree zero is labelled by an input variable, and it outputs the value of the variable.
  - Any other node is labelled by one of the three operations $\wedge$, $\vee$, $\neg$, and it outputs the value of the operation on its input.

  Nodes with out-degree zero are the output gates.

- <u>**Size**</u> of circuit is the no. of edges in it. <u>**Depth**</u> is the length of the longest path from an i/p to o/p node.

Boolean circuits

- A <u>Boolean circuit</u> is a directed acyclic graph whose nodes/gates are labelled as follows:
  - ➢ A node with in-degree zero is labelled by an input variable, and it outputs the value of the variable.
  - ➢ Any other node is labelled by one of the three operations $\wedge$, $\vee$, $\neg$, and it outputs the value of the operation on its input.

  Nodes with out-degree zero are the output gates.

θ(no. of nodes)

- **<u>Size</u>** of circuit is the <u>no. of edges</u> in it. **<u>Depth</u>** is the length of the longest path from an i/p to o/p node.

Boolean circuits

- A <u>Boolean circuit</u> is a directed acyclic graph whose nodes/gates are labelled as follows:
  - ➢ A node with in-degree zero is labelled by an input variable, and it outputs the value of the variable.
  - ➢ Any other node is labelled by one of the three operations $\wedge$, $\vee$, $\neg$, and it outputs the value of the operation on its input.

  Nodes with out-degree zero are the output gates.

- <u>**Size**</u> corresponds to "sequential time complexity". <u>**Depth**</u> corresponds to "parallel time complexity".

Boolean circuits

- A <u>Boolean circuit</u> is a directed acyclic graph whose nodes/gates are labelled as follows:
  - A node with in-degree zero is labelled by an input variable, and it outputs the value of the variable.
  - Any other node is labelled by one of the three operations $\wedge$, $\vee$, $\neg$, and it outputs the value of the operation on its input.

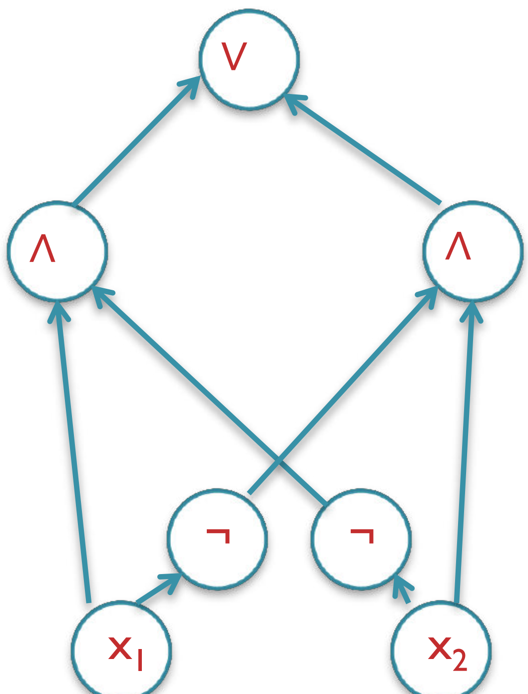
  Nodes with out-degree zero are the output gates.

- If every node in a circuit has out-degree at most one, then the circuit is called a <u>**formula**</u>.

A circuit for Parity

- $\text{PARITY}(x_1, x_2, \ldots, x_n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n$.

$x_1 \oplus x_2 = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$



ϕ

$\text{Size}(\phi) = |\phi| = 8$
$\text{Depth}(\phi) = 3$

Circuit family

- Let $T\colon \mathbb{N} \rightarrow \mathbb{N}$ be some function.
- Definition: A $T(n)$-size circuit family is a set of circuits $\{C_n\}_{n\in\mathbb{N}}$ such that C_n has n inputs and $|C_n| \leq T(n)$.

Class P/poly

- Let $T: \mathbb{N} \rightarrow \mathbb{N}$ be some function.
- Definition: A $T(n)$-size circuit family is a set of circuits $\{C_n\}_{n\in\mathbb{N}}$ such that C_n has n inputs and $|C_n| \leq T(n)$.

- Definition: A language L is in $\text{SIZE}(T(n))$ if there's a $T(n)$-size circuit family $\{C_n\}_{n\in\mathbb{N}}$ such that

$$x \in L \iff C_n(x) = 1, \text{ where } n = |x|.$$

- Defintion: Class $\text{P/poly} = \bigcup_{c \geq 1} \text{SIZE}(n^c)$.

Class P/poly

- Let $T: \mathbb{N} \rightarrow \mathbb{N}$ be some function.
- Definition: A $T(n)$-size circuit family is a set of circuits $\{C_n\}_{n\in\mathbb{N}}$ such that C_n has n inputs and $|C_n| \leq T(n)$.

- Definition: A language L is in $\mathrm{SIZE}(T(n))$ if there's a $T(n)$-size circuit family $\{C_n\}_{n\in\mathbb{N}}$ such that

$$x \in L \quad \Leftrightarrow C_n(x) = 1, \text{ where } n = |x|.$$

- Defintion: Class $\mathrm{P/poly} = \bigcup_{c\geq 1} \mathrm{SIZE}(n^c)$.

The circuit family $\{C_n\}_{n\in\mathbb{N}}$ <u>*decides*</u> L, i.e., C_n *decides* $L \cap \{0,1\}^n$.

Class P/poly

- Let $T\colon \mathbb{N} \rightarrow \mathbb{N}$ be some function.
- Definition: A $T(n)$-size circuit family is a set of circuits $\{C_n\}_{n\in\mathbb{N}}$ such that C_n has n inputs and $|C_n| \leq T(n)$.

- Definition: A language L is in $SIZE(T(n))$ if there's a $T(n)$-size circuit family $\{C_n\}_{n\in\mathbb{N}}$ such that

$$x \in L \Leftrightarrow C_n(x) = 1, \text{ where } n = |x|.$$

Alternatively, we say C_n *computes* the characteristic function of $L \cap \{0,1\}^n$.

- Defintion: Class $P/poly = \bigcup_{c \geq 1} SIZE(n^c)$.

Class P/poly

- Observation: $P \subseteq P/poly$.
- Proof. If $L \in P$, then there's a n^c-time TM that decides L for some constant c. By Cook-Levin, there's a $O(n^{2c})$-size circuit family $\{C_n\}_{n \in \mathbb{N}}$ such that

$$x \in L \Leftrightarrow C_n(x) = 1, \text{ where } n = |x|.$$

Class P/poly

- Observation: $P \subseteq P/poly$.
- Proof. If $L \in P$, then there's a n^c-time TM that decides L for some constant c. By Cook-Levin, there's a $O(n^{2c})$-size circuit family $\{C_n\}_{n\in N}$ such that

  $$x \in L \iff C_n(x) = 1, \text{ where } n = |x|.$$

  (Note: C_n is poly(n)-time computable from 1^n.)

- Is $P = P/poly$?

Class P/poly

- Observation: $P \subseteq P/poly$.
- Proof. If $L \in P$, then there's a n^c-time TM that decides L for some constant c. By Cook-Levin, there's a $O(n^{2c})$-size circuit family $\{C_n\}_{n\in\mathbb{N}}$ such that

  $$x \in L \Leftrightarrow C_n(x) = 1, \text{ where } n = |x|.$$

  (Note: C_n is poly(n)-time computable from 1^n.)

- Is $P = P/poly$? No! P/poly contains undecidable languages.

Class P/poly

- Let $HALT = \{(M,y) : M$ halts on input $y\}$. HALT is an undecidable language.
- Notation. $\#(M,y)$ = number corresponding to the binary string (M,y).
- Let $UHALT = \{1^{\#(M,y)} : (M,y) \in HALT\}$. Then, UHALT is also an undecidable language.

Class P/poly

- Let HALT = $\{(M,y) : M$ halts on input $y\}$. HALT is an undecidable language.
- Notation. $\#(M,y)$ = number corresponding to the binary string (M,y).
- Let UHALT = $\{1^{\#(M,y)} : (M,y) \in \text{HALT}\}$. Then, UHALT is also an undecidable language.
- Obs. Any unary language is in P/poly. (*Homework*)
  Hence, $P \subsetneq P/poly$.

Class P/poly

- What makes P/poly contain undecidable languages? *Ans:* $L \in$ P/poly implies that L is decided by a circuit family $\{C_n\}$, where $|C_n| = n^{O(1)}$. <u>We don't require that C_n is poly-time computable from 1^n</u>.

Class P/poly

- What makes P/poly contain undecidable languages? *Ans:* $L \in$ P/poly implies that L is decided by a circuit family $\{C_n\}$, where $|C_n| = n^{O(1)}$. We don't require that C_n is poly-time computable from 1^n.
- P/poly is a <u>*non-uniform class*</u> as a language in this class is allowed to have different algorithms/circuits for different input lengths.
- P is a <u>*uniform class*</u> as a language in this class has one algorithm for all inputs.

Class P/poly

- What makes P/poly contain undecidable languages? *Ans:* $L \in$ P/poly implies that L is decided by a circuit family $\{C_n\}$, where $|C_n| = n^{O(1)}$. We don't require that C_n is poly-time computable from 1^n.
- P/poly is a *non-uniform class* as a language in this class is allowed to have different algorithms/circuits for different input lengths.
- P is a *uniform class* as a language in this class has one algorithm for all inputs.

Hardware	Software
TM (uniform)	Algo/Enc. of TM
Circuits (non-uniform)	An algo per i/p length

Class P/poly

- What makes P/poly contain undecidable languages? *Ans:* $L \in$ P/poly implies that L is decided by a circuit family $\{C_n\}$, where $|C_n| = n^{O(1)}$. We don't require that C_n is poly-time computable from 1^n.
- P/poly is a *non-uniform class* as a language in this class is allowed to have different algorithms/circuits for different input lengths.
- P is a *uniform class* as a language in this class has one algorithm for all inputs.
- Is SAT $\in$ P/poly? In other words, is NP $\subsetneq$ P/poly?