Indian Association for the Cultivation of Science

(Deemed to be University under *de novo* Category)

Master's/Integrated Master's-PhD Program/ Integrated Bachelor's-Master's Program/PhD Course

Theory of Computation II: COM 5108

Lecture V

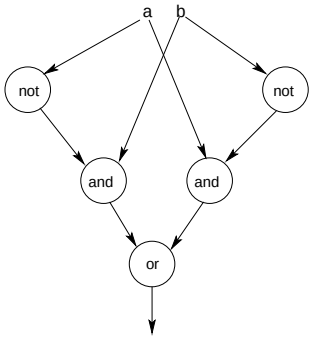
Instructor: Goutam Biswas *Autumn Semester 2023*

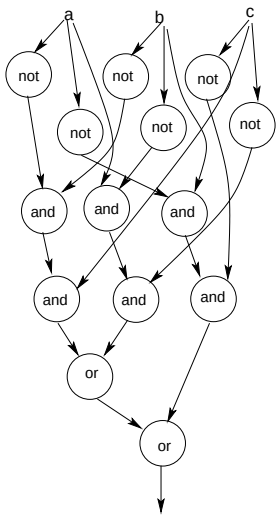
1 Circuit Complexity

1.1 Introduction

A Turing machine runs on inputs of all possible sizes. It is called a *uniform model*. A *nonuniform* model allows inputs of different lengths to be processed by different gadgets or algorithms. Boolean Circuits is a nonuniform model. Researchers in the late 1970s thought that *Boolean Circuit* may turn out to be a better model to address the **P** verses **NP** and related questions.

Definition 1. An *n-input* and 1-*output Boolean circuit* is a directed acyclic graph (DAG) with *n input vertices* (no incoming edges) and one *output vertex* (no outgoing edge). Intermediate vertices are of three kinds, $\wedge$ (and), $\vee$ (or) and $\neg$ (not), are known as *gates*. Each gate has one out degree. A gate with a label $\wedge$ or $\vee$ has two in-degree and a gate with label $\neg$ has one in-degree.





The size of a circuit C, $|C|$, is the number of gates present in it.

Input vertices of a circuit are labeled with *Boolean variables* $x_1, \cdots, x_n$. Function of each gate corresponds to its logical label.

A *Boolean circuit* may be viewed as a programming language where statements are of the form $x_i = x_j @ x_k$, where $@ \in \{\vee, \wedge\}$ and also of the form $x_i = \neg x_j$, where a variable can appear on the left-hand side of an assignment only once (acyclic).

A Boolean circuit C of n variables computes the boolean function $f_C : \{0,1\}^n \to \{0,1\}$. If the input nodes are set with the value $a_1, \cdots, a_n$, the output is $b = f_C(a_1, \cdots, a_n)$.

We wish to use circuits to test membership of a language. But an n variable circuit can only handle an input of length n. So, to test the membership of a language we define a *family of circuits* for different input lengths.

Definition 2. A *circuit family* $\mathcal{C} = \{C_n\}_{n \in \mathbb{N}_0}$ is a sequence of Boolean circuits. A string $x \in \{0,1\}^*$ is accepted by a circuit $C_{|x|} \in \mathcal{C}$ if $C_{|x|}(x) = 1$. The language $L(\mathcal{C})$ decided by a *circuit family* is the collection of all strings accepted by the elements of the family.

$$L(\mathcal{C}) = \{x \in \{0,1\}^* : \ C_{|x|}(x) = 1, \ C_{|x|} \in \mathcal{C}\}.$$

Let $f : \mathbb{N}_0 \to \mathbb{N}_0$ be a function. A *circuit family* $\mathcal{C}$ is of size $f(n)$ if each Boolean circuit $C_n \in \mathcal{C}$ with n input and one output is of size $f(n)$. This is known as the *size complexity* of a circuit family.

A language $L \in \mathbf{SIZE}(f(n))$ if there is a circuit family of size $f(n)$ such that

$$x \in L \text{ if and only if } C_n(x) = 1,$$

where $C_n \in \mathcal{C}$ and $|x| = n$.

A circuit is *size minimal* if there is no equivalent circuit with smaller size. Whether a given circuit is *minimal* is not known to be in **P** or in **NP**.

The *circuit size complexity* of a language L is is the size complexity of the minimal circuit family that decides the language.

Definition 3. Depth of a circuit is length of the longest path from the input variable to the output node (gate). We can define *depth complexity* of a circuit family and *depth minimal* circuit in a way similar to the size complexity.

1.2 Circuit Satisfiability

Theorem 1. Let $f : \mathbb{N}_0 \to \mathbb{N}_0$ be a function such that $f(n) \geq n$. If $L \in \mathbf{TIME}(f(n))$, then $L \in \mathbf{SIZE}(O(f^2(n)))$.

The outline of the proof goes as follows: Let M be the Turing machine that decides L in time $f(n)$. For each input of length n, a Boolean circuit C_n is constructed from the computation of M on the input of length n. The computation can be captured (roughly) by an $f(n) \times f(n)$ matrix. Where the first row corresponds to the *start configuration* and the last row corresponds to the final configuration[1]. Output of the gates corresponding to the i^{th} configuration (i^{th}

[1]The Turing machine M is such that in its accepting configuration its head moves to the leftmost cell after erasing the content of the tape. Moreover, if M halts before $f(n)$ steps, the same configurations are repeated.

row of the matrix), are input to the gates of the $(i+1)^{th}$ row simulating the $(i+1)^{th}$ configuration.

Each cell of the matrix contains a tape symbol (element of Γ), or a composite symbol of state (Q) and tape (Γ) if the head is scanning that cell. So a configuration of the form $\boxed{1}\boxed{1}\boxed{0}\boxed{1}\boxed{q_5 1}\boxed{0}\boxed{0}\boxed{1}$ means that the machine is in state q_5, the head is scanning 5^{th} cell from the left containing '1' in it. The value of *cell*(i,j) is the content of the j^{th} cell of the i^{th} configuration.

We introduce Boolean variables *sym*(i,j,k), where k, a tape symbol or a composite symbol, and i,j corresponds to the *cell*(i,j). The variable *sym*(i,j,k) takes the value *true (1)* if the symbol k is present in *cell*(i,j), otherwise it is 0. We know that a cell may contain an element of $\Gamma \cup (Q \times \Gamma)$. Let $|\Gamma \cup (Q \times \Gamma)| = c$. So there are $cf(n)^2$ variables of type *sym*(i,j,k). A Boolean circuit will ensure that only one of c variables corresponding to the cell *cell*(i,j), $1 \leq i,j \leq f(n)$ will have Boolean value *true (1)*.

As we have already mentioned, a boolean variable *sym*(i,j,k) is *true (1)* if the *cell*(i,j) has the symbol k. The value of *sym*(i,j,k) depends on the the contents of *cell*$(i-1,j-1)$, *cell*$(i-1,j)$, *cell*$(i-1,j+1)$, and the transition function δ of the Turing machine M. Let the possible values of these three cells for which *cell*(i,j) has the symbol k i.e. *sym*(i,j,k) is *true (1)*, be $(a_1,b_1,c_1),\cdots,(a_m,b_m,c_m)$. Then

$$sym(i,j,k) = \bigvee_{l=1}^{m} (sym(i-1,j-1,a_l) \wedge sym(i-1,j,b_l) \wedge sym(i-1,j+1,c_l)$$

This is repeated for all the variables of all the configurations except the start configuration.

Let the input be $x = x_1 x_2 \cdots x_n$: x_1 is directly connected to *sym*$(1,1,\boxed{q_0 1})$ and connected through a not gate to *sym*$(1,1,\boxed{q_0 0})$ i.e. *sym*$(1,1,\boxed{q_0 1})$ is *true(1)* if $x_1 = 1$ and *sym*$(1,1,\boxed{q_0 0})$ is *true(1)* if $x_1 = 0$. In both the cases the machine is at its start state q_0. Similarly, *sym*$(1,2,1)$, $\cdots$, *sym*$(1,n,1)$ are directly connected to $x_2,\cdots,x_n$ respectively. $x_2,\cdots,x_n$ are connected through not-gates to *sym*$(1,2,0)$, $\cdots$, *sym*$(1,n,0)$. Boolean '1' is connected to *sym*$(1,n+1,\sqcup)$ to *sym*$(1,f(n),\sqcup)$. All other variables of first row are connected to Boolean '0'.

The output gate corresponds to *sym*$(f(n),1,\boxed{q_A \sqcup})$. Every cell has fixed number of symbols *sym*(i,j,k), and every symbol is connected to the symbols of three neighboring cells of the previous row (except the first row) through fixed number of gates. So the size of this circuit is $O(f(n)^2)$.
Also note that the construction of C_n takes $O(f(n)^2)$ time.

Definition 4. CKT-SAT = $\{< C >: C$ is a satisfiable Boolean Circuit $\}$.

Proposition 1. CKT-SAT is **NP**-*complete*.

Proof: It is clear that CKT-SAT is in **NP**. The certificate is an input that produces the output 1. This can be verified in linear time as the output of each gate can be computed in constant time.
To show that it is **NP**-*complete* we have to design a polynomial time reduction function $f : \Sigma^* \to \Sigma^*$ so that for all $x \in \Sigma^*$

$$x \in L \text{ if and only if } f(x) =< C > \text{ is satisfiable,}$$

where $L \in \mathbf{NP}$. For L there is a polynomial time verifier V and a polynomial $p : \mathbb{N}_0 \to \mathbb{N}_0$ so that for each $x \in \{0,1\}^*$, $x \in L$ if and only if $\exists w \in \{0,1\}^{p(|x|)}$, s.t. V accepts $< x, w >$.

The reduction function f constructs a circuit C corresponding to the verifier V with input size $|x| + p(|x|)$ where the input x is plugged in. The remaining inputs correspond to the certificate.

If C is satisfiable, then there is a certificate. If $x \in L$, then there is a certificate, and C is satisfiable.

The verifier V runs for n^k time, so the size of the circuit is $O(n^{2k})$. Connection to every gate is a constant amount of work. So the reduction is polynomial time bounded. QED.

Proposition 2. CKT-SAT $\leq_P$ 3SAT.

Proof: We can give another proof of **NP**-*hardness* of 3SAT by reducing CKT-SAT to 3SAT.

Let C be an n-input circuit. A 3SAT formula ϕ will be constructed so that $C(x_1, \cdots, x_n) = 1$ if and only if $\phi(x_1, \cdots, x_n)$ is *true (1).*

Let w be a node of C. There are three possibilities,

1. Label of w is AND: the input of w are u and v. We introduce a variable y_w for w. We assume that we already have variables y_u and y_v corresponding to u and v. We encode $y_w = y_u \wedge y_v$.
$$(y_w \vee \bar{y_u} \vee \bar{y_v}) \wedge (\bar{y_w} \vee y_u) \wedge (\bar{y_w} \vee y_v).$$
2. Label of w is OR: We encode $y_w = y_u \vee y_v$.
$$(y_w \vee \bar{y_u}) \wedge (y_w \vee \bar{y_v}) \wedge (\bar{y_w} \vee y_u \vee y_v).$$
3. Label of w is NOT: We encode $y_w = \bar{y_u}$.
$$(y_w \vee y_u) \wedge (\bar{y_w} \vee \bar{y_u}).$$
4. Finally corresponding to the output gate v_o of C we add y_o i.e. the clause ϕ is true if and only if the output of the circuit is true.

QED.

1.3 P/poly

Definition 5. **P/poly** is the class of languages decidable by polynomial-sized circuits. $\mathbf{P/poly} = \bigcup_{c \geq 1} \mathbf{SIZE}(n^c)$.

Proposition 3. $\mathbf{P} \subseteq \mathbf{P/poly}$.

Proof: We know that if a language $L \in TIME(f(n))$, then $L \in SIZE((f(n))^2)$ for $f(n) \geq n$. If $f(n) = n^c$, where c is a constant, then $L \in \mathbf{P}$ so it is in $SIZE(n^{2c}) \subseteq \mathbf{P/poly}$.

QED.

Any language $L \subseteq \{1\}^*$ is in **P/poly**. If $1^n \in L$, then the circuit C_n with $(n-1)$ *AND-gate* is in the family. If it is not there, C_n is a circuit that always gives 0. So any unary language is in **P/poly**. But then there are uncountably many languages in **P/poly** and the inclusion of **P/poly** in **P** is not possible. Consider the following language. e.g.

$$UA_{TM} = \{1^n : < M, x > \text{is the binary encoding of } n \text{ and } M \text{ accepts } x\}.$$

UA_{TM} is undecidable but in **P/poly**.

1.4 P - uniform

The class **P/poly** is too large. There are languages in **P/poly** that are undecidable, where the circuit family exists but cannot be effectively constructed. Can we restrict the size of the class where the circuit family can be constructed in reasonable time or space?

Definition 6. A circuit family $\{C_n\}$ is called **P**-*uniform* if there is a polynomial time Turing machine that on input 1^n produces the description of C_n. Similarly we can talk about *log-space-uniform* circuit family.

This restriction reduces the collection of languages of the circuit family to **P**.

Proposition 4. A language L is decidable by a **P**-*uniform* circuit family if and only if $L \in \mathbf{P}$.

Proof: Let L be decided by a **P**-*uniform* circuit family $\{C_n\}$. So there is a polynomial time Turing machine that M_c that given 1^n generates C_n so that for all $x \in \{0,1\}^n$, $x \in L$ if and only if $C_n(x) = 1$. We design the polynomial time decider for L to prove that $L \in \mathbf{P}$.

M: input x

1. Run M_c on $1^{|x|}$ and generate C_n.
2. If $C_n(x) = 1$, *accept*; else *reject*.

M is a polynomial time Turing machine that decides L.

Let $L \in \mathbf{P}$. There is a n^c time bounded TM M that decides L. We know that given n and a function $f(n) \geq n$; if there is a language $L \in DTIME(f(n))$, then a circuit of size $O(f(n)^2)$ can be constructed by an $O(f(n)^2)$ time bounded TM M_L^c. We design M_c as follows.

M_c: input 1^n

1. Use M_L^c to build a circuit C_n for strings of length n in L.
2. Output C_n.

So L is in **P**-uniform. QED.

1.5 TMs take Advice and P/poly

We can define a Turing machine that takes '*advice*' - a string α_n for each input of length $n \in \mathbb{N}_0$. The machine uses the string in its computation.

Definition 7. Let $a, f : \mathbb{N}_0 \to \mathbb{N}_0$. We define $\mathbf{DTIME}(f(n))/a(n)$, the class of languages decidable by a $f(n)$ time bounded TM taking $a(n)$ bits of advice.

A language $L \subseteq \{0,1\}^*$ is in $\mathbf{DTIME}(f(n))/a(n)$, if there is a $f(n)$-time bounded Turing machine M and a sequence $\{\alpha_n\}$, $\alpha_n \in \{0,1\}^{a(n)}$, such that $x \in L$ if and only if M accepts $< x, \alpha_n >$, where $|x| = n$.

Example 1. Any unary language is decided by a polynomial time Turing machine with 1-bit advice. For each $n \in \mathbb{N}_0$, α_n of the sequence $\{\alpha_n\}_{n \in \mathbb{N}_0}$ is 1, if $1^n \in L$, otherwise it is 0. In this case advice length $a(n) = 1$ for all n.

We have the following characterization of **P/poly**.

Theorem 2. $\mathbf{P/poly} = \bigcup_{c,d \geq 1} \mathbf{DTIME}(n^c)/n^d$.

Proof: Let $L \in \mathbf{P/poly}$. We have a polynomial-sized family of circuits $\{C_n\}_{n \in \mathbb{N}_0}$ that decides L. If the input x is of length n, then the description of C_n is

bounded by some polynomial over n and can be taken as the advice. The Turing machine takes $< x, C_n >$ and accepts if and only if $C_n(x) = 1$.

If $L \in \mathbf{DTIME}(p(n))/q(n)$, where p, q are polynomials, then there is a $p(n)$ time bounded Turing machine M that on input x of length $|x| = n$, takes an advice α_n of length $q(n)$ and decides whether $x \in L$.

From the computation of M on $< x, \alpha_n >$ we can construct a polynomial size circuit D_n such that M accepts $< x, \alpha_n >$ if and only if $D_n(x, \alpha_n) = 1$. We construct C_n by directly connecting the input α_n to D_n. QED.

Can SAT be solved using a polynomial circuit? The answer is most likely negative and was supplied in [**?**] using the following theorem.

Theorem 3. (Karp, Lipton & Sipser) If $\mathbf{NP} \subseteq \mathbf{P/poly}$, then $\mathbf{PH} = \Sigma_2^P$.

If every **NP** problem has a polynomial size circuit family, then the polynomial hierarchy collapses to Σ_2^P.

Proof: We have the following argument:

(a) We prove that under the assumption $\mathbf{NP} \subseteq \mathbf{P/poly} \cdots (a)$, $\Pi_2^P = \Sigma_2^P$, so that $\mathbf{PH} = \Sigma_2^P$.

(b) We know that if $\Pi_2^P \subseteq \Sigma_2^P$ then $\Pi_2^P = \Sigma_2^P$.

$$L \in \Sigma_2^P \Rightarrow \overline{L} \in \Pi_2^P \Rightarrow \overline{L} \in \Sigma_2^P \Rightarrow L \in \Pi_2^P \Rightarrow \Sigma_2^P \subseteq \Pi_2^P.$$

(c) We prove that $\Pi_2^P \subseteq \Sigma_2^P$ under the assumption (a).

(d) If a Π_2^P-*complete* problem is in Σ_2^P, then $\Pi_2^P \subseteq \Sigma_2^P$.

Let L be Π_2^P-complete and also $L \in \Sigma_2^P$. Then for all $L' \in \Pi_2^P$, $L' \leq_p L$. Ans through the reduction $L' \in \Sigma_2^P$.

(e) $\Pi_2 SAT$ is a complete problem of Π_2^P. We claim that it is in Σ_2^P.

The set $\Pi_2 SAT$ is a collection of Boolean formulae of the following form.

$$\{\forall u \exists v \phi(u, v) = 1,$$

where u and v are two vectors of boolean variables and ϕ is a quantifier free boolean formula. In other words there is a polynomial time bounded TM M and a polynomial p such that

$$\forall u \in \{0,1\}^n \exists v \in \{0,1\}^n M(\phi, u, v) = 1.$$

Given a ϕ and a boolean vector u (the first u variables are initialized), we get the boolean formula ϕ_u so that

$$\exists v \phi_u(v) = 1.$$

$\phi_u(v)$ is an element of SAT. In other words there is a polynomial time bounded TM M_u and a polynomial p such that

$$\exists v \in \{0,1\}^{p(n)} M_u(\phi_u, v) = M(\phi, u, v).$$

$\phi_u(v)$ is an element of SAT and M_u is its verifier.

According to our assumption (a), $\phi_u(v) \in \mathbf{P/poly}$. So there is a $r(n)$ size circuit family $\{C_n\}$ (r is a polynomial) such that

$$\forall \phi \forall u \in \{0,1\}^n C_n(\phi, u) = 1 \text{ if and only if } \exists v \phi_u(v) = 1.$$

So a polynomial size circuit solves the decision problem of SAT.

We know that if there is a decision algorithm for SAT, then we can have a search algorithm to generate the satisfying assignment (if there is one) for a given formula ϕ.

The assignment generation algorithm may also be viewed as a circuit and from $\{C_n\}$, we obtain a $q(n)$ size circuit family $\{C'_n\}$ such that for any ϕ and $u \in \{0,1\}^n$, if there exists $v \in \{0,1\}^n$ such that $\phi(u,v) = 1$, then $\{C'_n\}$ produces v (multiple bit output).

But then a $q(n)$ size circuit can be described using polynomial $r(n)$ many bits and can be guessed! So we have

$$\forall u \in \{0,1\}^n \exists v \in \{0,1\}^n \phi(u,v) = 1,$$

if and only if

$$\exists w \in \{0,1\}^{r(n)} \forall u \in \{0,1\}^n \text{ s.t. } w =< C'_n > \text{ and } \phi_u(C'(\phi,u)) = 1$$

The second formula is in Σ_2^P as verification can be done in polynomial time. So $\Pi_2 SAT$ is in Σ_2^P and the hierarchy collapses to Σ_2^P. QED.

References

[MS] *Theory of Computation* by Michael Sipser, (3rd. ed.), Pub. Cengage Learning, 2007, ISBN 978-81-315-2529-6.

[SABB] *Computational Complexity, A Modern Approach* by Sanjeev Arora & Boaz Barak, Pub. Cambridge University Press, 2009, ISBN 978-0-521-42426-4.

[CHP] Computational Complexity by *Christos H Papadimitriou*, Pub. Addision-Wesley, 1994, ISBN 0-201-53082-1.

[JES] Models of Computation by *John E Savage*, Brown University, `http://cs.brown.edu/~jes/book/pdfs/ModelsOfComputation_Chapter9.pdf`.

[KL] R Karp and R Lipton, *Turing machine that take advice*, L' Ensignement Mathématique, 28:191-210, 1982.