1 The Polynomial Hierarchy

1.1 Defining the Polynomial Hierarchy via Oracle Machines

Here we show a third definition of the levels of the polynomial hierarchy in terms of oracle machines.

**Definition 1** Define $\Sigma_i, \Pi_i$ inductively as follows:

- $\Sigma_0 \overset{\text{def}}{=} P$.
- $\Sigma_{i+1} \overset{\text{def}}{=} \mathcal{N}P^{\Sigma_i}$ and $\Pi_{i+1} = \text{co}\mathcal{N}P^{\Sigma_i}$.

(Note that even though we believe $\Sigma_i \neq \Pi_i$, oracle access to $\Sigma_i$ gives the same power as oracle access to $\Pi_i$. Do you see why?)

We show that this leads to an equivalent definition. For this section only, let $\Sigma_i^O$ refer to the definition in terms of oracles. We prove by induction that $\Sigma_i = \Sigma_i^O$. (Since $\Pi_i^O = \text{co}\Sigma_i^O$, this proves it for $\Pi_i$ as well.) For $i = 1$ this is immediate, as $\Sigma_1 = \mathcal{N}P = \mathcal{N}P^{\Sigma_0}$.

Assuming $\Sigma_i = \Sigma_i^O$, we prove that $\Sigma_{i+1} = \Sigma_{i+1}^O$. Let us first show that $\Sigma_{i+1} \subseteq \Sigma_{i+1}^O$. Let $L \in \Sigma_{i+1}$. Then there exists a polynomial-time Turing machine $M$ such that

$$x \in L \iff \exists w_1 \forall w_2 \cdots Q_{i+1} w_{i+1} M(x, w_1, \ldots, w_{i+1}) = 1.$$  

In other words, there exists a language $L' \in \Pi_i$ such that

$$x \in L \iff \exists w_1 \ (x, w_1) \in L'.$$

By our inductive assumption, $\Pi_i = \Pi_i^O$; thus, $L \in \mathcal{N}P^{\Pi_i^O} = \mathcal{N}P^{\Sigma_i^O} = \Sigma_{i+1}^O$ and so $\Sigma_{i+1} \subseteq \Sigma_{i+1}^O$.

It remains to show that $\Sigma_{i+1}^O \subseteq \Sigma_{i+1}$. Let $L \in \Sigma_{i+1}^O$. This means there exists a non-deterministic polynomial-time machine $M$ and a language $L' \in \Sigma_i^O$ such that $M$, given oracle access to $L_i$, decides $L$. In other words, $x \in L$ iff $\exists y, q_1, a_1, \ldots, q_n, a_n$ (here, $y$ represents the non-deterministic choices of $M$, while $q_j, a_j$ represent the queries/answers of $M$ to/from its oracle) such that:

1. $M$, on input $x$, non-deterministic choices $y$, and oracle answers $a_1, \ldots, a_n$, makes queries $q_1, \ldots, q_n$ and accepts.

2. For all $j$, we have $a_j = 1$ iff $q_j \in L'$.

Since $L' \in \Sigma_i^O = \Sigma_i$ (by our inductive assumption) we can express the second condition, above, as:

- $a_j = 1 \iff \exists y_{i,j}^1 \forall y_{i,j}^2 \cdots Q_i y_{i,j}^j M'(q_j, y_{i,j}^1, \ldots, y_{i,j}^j) = 1$
- $a_j = 0 \iff \forall y_{i,j}^1 \exists y_{i,j}^2 \cdots Q_i y_{i,j}^j M'(q_j, y_{i,j}^1, \ldots, y_{i,j}^j) = 0$
for some (deterministic) polynomial-time machine $M'$. The above leads to the following specification of $L$ as a $\Sigma_{i+1}$ language:

$$x \in L \text{ iff } \exists (y, q_1, a_1, \ldots, q_n, a_n, \{y_j^n\}_{j=1}^n) \forall (\{y_j^n\}_{j=1}^n) \cdots Q_{i+1} (\{y_{i+1}^n\}_{j=1}^n):$$

- $M$, on input $x$, non-deterministic choices $y$, and oracle answers $a_1, \ldots, a_n$, makes queries $q_1, \ldots, q_n$ and accepts, and
- Let $Y$ be the set of $j$'s such that $a_j = 1$, and let $N$ be the set of $j$'s such that $a_j = 0$.
  - For all $j \in Y$, we have $M'(q_j, y_1^j, \ldots, y_i^j) = 1$
  - For all $j \in N$, we have $M'(q_j, y_2^j, \ldots, y_{i+1}^j) = 0$.

2 Non-Uniform Complexity

Boolean circuits offer an alternate model of computation: a non-uniform one as opposed to the uniform model of Turing machines. (The term “uniform” is explained below.) In contrast to Turing machines, circuits are not meant to model “realistic” computations for arbitrary-length inputs. Circuits are worth studying for at least two reasons, however. First, when one is interested in inputs of some fixed size (or range of sizes), circuits make sense as a computational model. (In the real world, efficient circuit design has been a major focus of industry.) Second, from a purely theoretical point of view, the hope has been that circuits would somehow be “easier to study” than Turing machines (even though circuits are more powerful) and hence that it might be easier to prove lower bounds for the former than for the latter. The situation here is somewhat mixed: while some circuit lower bounds have been proved, those results have not really led to any significant separation of uniform complexity classes.

Circuits are directed, acyclic graphs where nodes are called gates and edges are called wires. Input gates are gates with in-degree zero, and we will take the output gate of a circuit to be the (unique) gate with out-degree zero. (For circuits having multiple outputs there may be multiple output gates.) In a boolean circuit, each input gate is identified with some bit of the input; each non-input gate is labeled with a value from a given basis of boolean functions. The standard basis is $B_0 = \{\neg, \lor, \land\}$, where each gate has bounded fan-in. Another basis is $B_1 = \{\neg, (\lor)_i\in\mathbb{N}, (\land)_i\in\mathbb{N}\}$, where $\lor_i, \land_i$ have in-degree $i$ and we say that this basis has unbounded fan-in. In any basis, gates may have unbounded fan-out.

A circuit $C$ with $n$ input gates defines a function $C : \{0, 1\}^n \rightarrow \{0, 1\}$ in the natural way: a given input $x = x_1 \cdots x_n$ immediately defines the values of the input gates; the values at any internal gate are determined inductively; $C(x)$ is then the value of the output gate. If $f : \{0, 1\}^* \rightarrow \{0, 1\}$ is a function, then a circuit family $C = \{C_i\}_{i\in\mathbb{N}}$ computes $f$ if $f(x) = C_{|x|}(x)$ for all $x$. In other words, for all $n$ the circuit $C_n$ agrees with $f$ restricted to inputs of length $n$. (A circuit family decides a language if it computes the characteristic function for that language.) This is the sense in which circuits are non-uniform: rather than having a fixed algorithm computing $f$ on all input lengths (as is required, e.g., in the case of Turing machines), in the non-uniform model there may be a completely different “algorithm” (i.e., circuit) for each input length.

Two important complexity measures for circuits are their size and their depth.¹ The size of a

¹When discussing circuit size and depth, it is important to be clear what basis for the circuit is assumed. By default, we assume basis $B_0$ unless stated otherwise.
circuit is the number of gates it has. The depth of a circuit is the length of the longest path from an input gate to an output gate. A circuit family $C = \{C_n\}_{n \in \mathbb{N}}$ has size $T(\cdot)$ if, for all sufficiently large $n$, circuit $C_n$ has size at most $T(n)$. It has depth $D(\cdot)$ if, for all sufficiently large $n$, circuit $C_n$ has depth at most $D(n)$. The usual convention is not to count “not” gates in either of the above: one can show that all the “not” gates of a circuit can be pushed to immediately follow the input gates; thus, ignoring “not” gates affects the size by at most $n$ and the depth by at most 1.

**Definition 2** $L \in \text{size}(T(n))$ if there is a circuit family $C = \{C_n\}$ of size $T(\cdot)$ that decides $L$.

We stress that the above is defined over $B_0$. Note also that we do not use big-$O$ notation, since there is no “speedup theorem” in this context.

One could similarly define complexity classes in terms of circuit depth (i.e., $L \in \text{depth}(D(n))$ if there is a circuit family $C = \{C_n\}$ of depth $D(\cdot)$ that decides $L$); circuit depth turns out to be somewhat less interesting unless there is simultaneously a bound on the circuit size.

### 2.1 The Power of Circuits

We have seen this before (in another context) but it is worth stating again: *every* function — even an undecidable one! — is computable by a circuit family over the basis $B_0$. Let us first show how to express any $f$ as a circuit over $B_1$. Fix some input length $n$. Define $F_0 \overset{\text{def}}{=} \{x \in \{0,1\}^n \mid f(x) = 0\}$ and define $F_1$ analogously. We can express $f$ (restricted to inputs of length $n$) as:

$$f(x) = \bigvee_{x' \in F_1} [x = x'],$$

where $[x = x']$ denotes a boolean expression which is true iff $x = x'$. (Here, $x$ represents the variables, and $x'$ is a fixed string.) Letting $x_i$ denote the $i$th bit of $x$, note that $[x = x'] \iff (\bigwedge_{i : x_i = 1} x_i) \land (\bigwedge_{i : x_i = 0} \bar{x}_i)$. Putting everything together, we have:

$$f(x) = \bigvee_{x' \in F_1} ((\bigwedge_{i : x_i = 1} x_i) \land (\bigwedge_{i : x_i = 0} \bar{x}_i)).$$

But the above is just a circuit of depth $^22$ over $B_1$. (The size of the circuit is at most $\Theta(2^n)$.) The above representation is called the *disjunctive normal form* (DNF) for $f$. Another way to express $f$ is as:

$$f(x) = \bigwedge_{x' \in F_0} [x \neq x'],$$

where $[x \neq x']$ has the obvious meaning. Note, $[x \neq x'] \iff (\bigvee_{i : x_i' = 1} \bar{x}_i) \lor (\bigvee_{i : x_i' = 0} x_i)$; putting everything together gives:

$$f(x) = \bigwedge_{x' \in F_0} \left((\bigvee_{i : x_i' = 1} \bar{x}_i) \lor (\bigvee_{i : x_i' = 0} x_i)\right),$$

the *conjunctive normal form* (CNF) for $f$. This gives another circuit of depth 2 over $B_1$.

The above show how to obtain a circuit for $f$ over the basis $B_1$. But one can transform any circuit over $B_1$ to one over $B_0$. The idea is simple: each $\lor$-gate of in-degree $k$ is replaced by a “tree”

---

\(^2\)Recall that “not” gates are not counted.
of degree-2 \lor\text{-}gates, and each \land\text{-}gate of in-degree \(k\) is replaced by a “tree” of degree-2 \land\text{-}gates. In each case we transform a single gate having fan-in \(k\) to a sub-circuit with \(k - 1\) gates having depth \(\lceil \log k \rceil\). Applying this transformation to Eqs. (1) and (2), we obtain a circuit for any function \(f\) over the basis \(B_0\) with at most \(n \cdot 2^n\) gates and depth at most \(n + \lceil \log n \rceil\). We thus have:

**Theorem 1** Every function is in size \((n \cdot 2^n)\).

This can be improved to show that for every \(\varepsilon > 0\) every function is in size \((1 + \varepsilon) \cdot 2^n/n\). This is tight up to low-order terms, as we show next time.

**Bibliographic Notes**

For more on circuit complexity see the classic text by Wegener [6] and the excellent book by Vollmer [5]. (The forthcoming book by Jukna [2] also promises to be very good.) The claim that all functions can be computed by circuits of size \((1 + \varepsilon) \cdot 2^n/n\) was proven by Lupanov. A proof of a weaker claim (showing this bound over the basis \(\{\lor, \land, \lnot, \oplus\}\)) can be found in my notes from 2005 [3, Lecture 5]. A proof over the basis \(B_0\) can be found in [4, Section 2.13]. Frandsen and Miltersen [1] give another exposition, and discuss what is known about the low-order terms in both the upper and lower bounds.

**References**


