# Notes on **coNP** and the polynomial-time hierarchy

Vassos Hadzilacos

## The class coNP and its relation to NP and P

The class **P** is closed under complementation: If a problem $A$ can be decided by a deterministic Turing machine $M$ in polytime, then $\overline{A}$ can also be decided by a deterministic Turing machine in polytime, namely the Turing machine $\overline{M}$ that is like $M$ except that the accept and reject states are swapped.

The same trick does not work for polytime ***nondeterministic*** Turing machines. For example, consider the following polytime nondeterministic Turing machine that decides SAT (described in pseudocode):

SAT-SOLVER($\phi$):
1 "guess" a truth assignment $\tau$ on the variables of $\phi$
2 **if** $\phi$ is true under $\tau$ **then accept**
3 **else reject**

If we swap the "accept" and "reject" statements, the resulting nondeterministic TM is of course polytime, but it does not decide $\overline{\text{SAT}}$: It accepts a formula if and only if there is a truth assignment that falsifies it, that is, if and only if the formula is ***not*** a tautology. For example, it accepts any formula that has both satisfying and falsifying truth assignments, and such formulas are not unsatisfiable.

The root cause of the different effect of swapping the accept and reject states between deterministic and nondeterministic Turing machines is that for the latter to accept an input it suffices for ***one*** of its computation paths to accept, whereas for it to reject, ***all*** its computation paths must reject. If the computation tree of $M$ on input $x$ contains both accepting and rejecting computations, $M$ accepts $x$; swapping the accept and reject states in that case results in a Turing machine that still accepts $x$, and therefore does not accept the complement of $M$'s language.

We encountered a similar phenomenon in the context of recognizers that are not deciders, i.e., Turing machines that may loop on some inputs. If a Turing machine $M$ fails to accept an input $x$ because it loops, the Turing machine $\overline{M}$ obtained by swapping $M$'s accept and reject states will also fail to accept $x$, and therefore it does not recognize the complement of the language recognized by $M$. This is fundamentally the reason why the class of recognizable languages is not closed under complementation, in contrast to the class of decidable languages, which is.

So the question arises whether **NP**, like **P** and the set of decidable languages, is closed under complementation; or, like the set of recognizable languages, it is not. This is another important open question in theoretical computer science. It is generally conjectured that the answer is no, but there is no proof of this as of yet. To explore this question we define another class of languages.

**Definition** ***coNP*** *is the class of languages whose complements are in* ***NP****.*

Examples of problems in **coNP** are:

- $\overline{\text{SAT}} = \{\phi\text{: } \phi \text{ is not satisfiable}\}$.
- $\text{TAUT} = \{\phi\text{: } \phi \text{ is a tautology}\}$.
- $\overline{\text{CLIQUE}} = \{\langle G, k\rangle\text{: } G \text{ is an undirected graph that contains no clique of at least } k \text{ nodes}\}$

Recall the certificate-verifier characterization of **NP**: A decision problem $A$ is in **NP** if and only if every yes-instance $x$ of $A$ has a "short" certificate $y$ that efficiently verifies $x$'s membership in $A$; "short" here means "at most polynomial in the size of $x$"; and "efficiently" means "in polytime". More formally,

$$x \in A \;\Leftrightarrow\; \exists y \left( |y| \leq |x|^k \;\wedge\; P(x,y) \right)$$

where $k$ is some constant and $P(x,y)$ is a polytime predicate — i.e., a predicate for which there is a polytime deterministic Turing machine that, given $\langle x, y \rangle$ as input outputs 1 if $P(x,y)$ holds and outputs 0 otherwise. For example,

$$\phi \in \textsc{Sat} \;\Leftrightarrow\; \exists \tau \Big( \underbrace{\tau \text{ is a truth assignment to the variables of } \phi}_{\tau \text{ is polysize in } |\phi|} \wedge \underbrace{\tau \text{ satisfies } \phi}_{\text{polytime}} \Big)$$

$$\langle G, k \rangle \in \textsc{Clique} \;\Leftrightarrow\; \exists C \Big( \underbrace{C \text{ is a set of } k \text{ nodes of } G}_{|C| \text{ is polysize in } |G|} \wedge \underbrace{\text{any two nodes in } C \text{ are connected by an edge}}_{\text{polytime}} \Big)$$

Since languages in **coNP** are complements of those in **NP**, we have a similar characterization of **coNP**. A language $A \in$ **coNP** (in other words, $\overline{A} \in$ **NP**) if and only if

$$x \in \overline{A} \;\Leftrightarrow\; \forall y \left( |y| \leq |x|^k \;\rightarrow\; P(x,y) \right)$$

where $k$ is some constant and $P(x,y)$ is a polytime predicate. For example,

$$\phi \in \overline{\textsc{Sat}} \;\Leftrightarrow\; \forall \tau \Big( \underbrace{\tau \text{ is a truth assignment to the variables of } \phi}_{\tau \text{ is polysize in } |\phi|} \rightarrow \underbrace{\tau \text{ falsifies } \phi}_{\text{polytime}} \Big)$$

$$\langle G, k \rangle \in \overline{\textsc{Clique}} \;\Leftrightarrow\; \forall C \Big( \underbrace{C \text{ is a set of } k \text{ nodes of } G}_{|C| \text{ is polysize in } |G|} \rightarrow \underbrace{\text{some two nodes in } C \text{ are not connected by an edge}}_{\text{polytime}} \Big)$$

These characterizations of **NP** and **coNP** highlight the difference between these two classes, and support the intuition that they are, in fact, distinct: Yes-instances of problems in **NP** have short, efficiently checkable certificates: I can convince you that a formula is satisfiable by supplying a truth assignment (which is certainly of polysize compared to the formula) and you can quickly (in polytime) verify that this truth assignment indeed satisfies the formula. ***Finding*** a satisfying truth assignment, if one exists, is apparently difficult — but verifying one is not. Similarly with all **NP** problems.

With the complements of **NP** problems, i.e., with **coNP** problems, it is not at all clear that such short and efficiently verifiable certificates exist for the yes-instances. For example, it is not at all clear what short proof I can provide to convince you that a formula is unsatisfiable, apart from a listing of all truth assignments for you to verify that none of them satisfies the formula — but that is not a short proof! So, it feels intuitively that **coNP** is a different class of problems than **NP**. Unfortunately, we don't know if this intuition is sound, as we do not have a proof that **NP** $\neq$ **coNP**. In fact, as we will see, a proof of this would immediately imply that **P** $\neq$ **NP** (see Theorem 11.3).

We now prove some simple facts about **coNP** and its relation to **P** and **NP**.

**Theorem 11.1** ***P*** $\subseteq$ ***NP*** $\cap$ ***coNP***.

Proof. We already know that **P** $\subseteq$ **NP**, so it remains to show that **P** $\subseteq$ **coNP**. Let $A$ be any problem in **P**. Since **P** is closed under complementation, $\overline{A} \in$ **P**, so $\overline{A} \in$ **NP**; and by definition of **coNP**, $A \in$ **coNP**. So, **P** $\subseteq$ **coNP**. □

It is not known if the containment in Theorem 11.1 is proper, or if equality actually holds. Note that in the analogous result in computability theory (with **NP** playing the role of recognizable sets and **P** playing the role of decidable sets) equality holds: **Decidable** = **Recognizable** $\cap$ **coRecognizable**.

The next theorem states that the question **NP** $\stackrel{?}{=}$ **coNP** is exactly equivalent to asking whether **NP** is closed under complementation.

**Theorem 11.2** ***NP* = *coNP*** *if and only if* ***NP*** *is closed under complementation.*

PROOF. We have:

$\mathbf{NP} = \mathbf{coNP}$

$\Leftrightarrow$ for all $A$, $A \in \mathbf{NP} \Leftrightarrow A \in \mathbf{coNP}$

$\Leftrightarrow$ for all $A$, $A \in \mathbf{NP} \Leftrightarrow \overline{A} \in \mathbf{NP}$

$\Leftrightarrow$ **NP** is closed under complementation. □

**Theorem 11.3** *If* $\boldsymbol{NP \neq coNP}$ *then* $\boldsymbol{P \neq NP}$.

PROOF. Suppose that $\mathbf{NP} \neq \mathbf{coNP}$. By Theorem 11.2, **NP** is not closed under complementation. Thus, there is some $A \in \mathbf{NP}$ such that $\overline{A} \notin \mathbf{NP}$. Since **P** is closed under complementation $A \notin \mathbf{P}$. So $A \in \mathbf{NP} - \mathbf{P}$, and therefore $\mathbf{P} \neq \mathbf{NP}$. □

It is not known whether the converse of Theorem 11.3 holds: Based on what we know, it is possible that $\mathbf{P} \neq \mathbf{NP}$, yet $\mathbf{NP} = \mathbf{coNP}$. Thus, the conjecture that $\mathbf{NP} \neq \mathbf{coNP}$ is at least as strong as, and possibly stronger than, the conjecture that $\mathbf{P} \neq \mathbf{NP}$.

**Definition** *A decision problem A is* ***coNP-complete*** *if*

*(a)* $A \in$ ***coNP****, and*

*(b) for all* $B \in$ ***coNP****,* $B \leq_m^p A$.

That is, to within a polynomial amount of time, **coNP**-complete problems are the hardest in **coNP**.

For the proof of the following theorem, recall that if $B \leq_m^p A$ then $\overline{B} \leq_m^p \overline{A}$. The very same mapping that shows $B \leq_m^p A$ also shows that $\overline{B} \leq_m^p \overline{A}$.

**Theorem 11.4** *A is* ***NP****-complete if and only if* $\overline{A}$ *is* ***coNP****-complete.*

PROOF. We prove the only-if direction; the converse is similar. Suppose $A$ is **NP**-complete. Then,

- $A \in \mathbf{NP}$, and therefore (a) $\overline{A} \in \mathbf{coNP}$.
- For every $B \in \mathbf{NP}$, $B \leq_m^p A$; therefore, (b) for every $\overline{B} \in \mathbf{coNP}$, $\overline{B} \leq_m^p \overline{A}$.

(a) and (b) imply that $\overline{A}$ is **coNP**-complete. □

From this theorem and our wealth of knowledge about **NP**-complete problems, we get an equally extensive knowledge about **coNP**-complete problems.

For the proof of the next theorem, recall Corollary 8.3, one of the simple but important properties of Karp reductions: If $X \leq_m^p Y$ and $Y \in \mathbf{P}$ then $X \in \mathbf{P}$. We proved this by noting that we can combine the polytime reduction of $X$ to $Y$ with the polytime DTM that solves $Y$ to obtain a DTM that solves $X$ in polytime. Using exactly the same idea, but now with NTMs instead of DTMs, we have:

$$\text{If } X \leq_m^p Y \text{ and } Y \in \mathbf{NP} \text{ then } X \in \mathbf{NP}. \qquad (*)$$

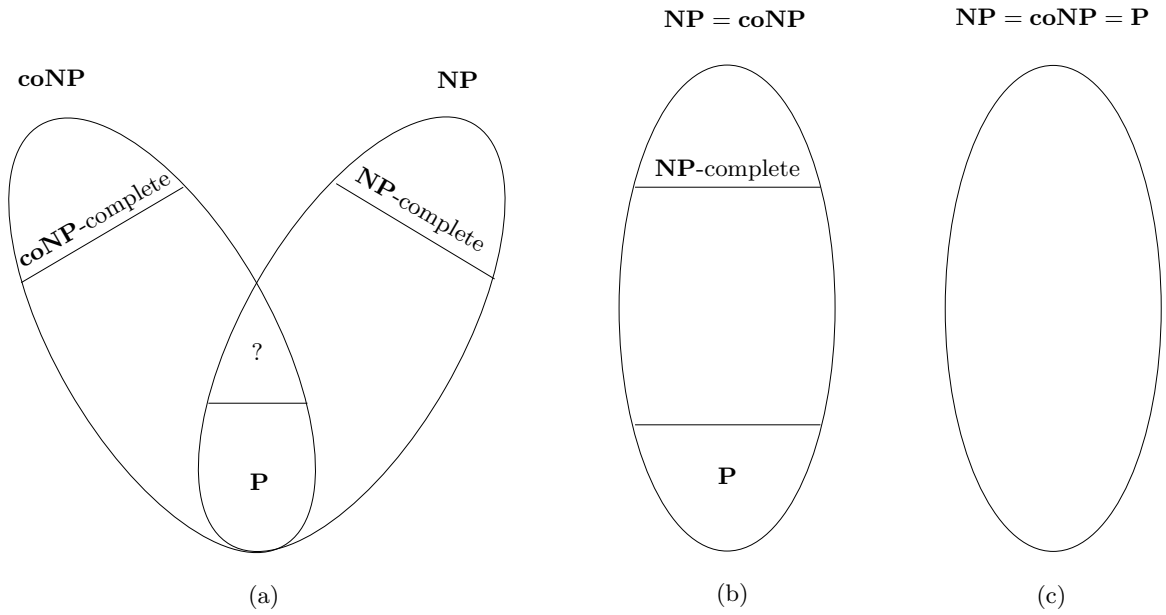
**Theorem 11.5** ***NP = coNP*** *if and only if there is some* ***NP****-complete problem A whose complement is also in* ***NP****.*

PROOF. ONLY IF: Suppose **NP** = **coNP**. By Theorem 11.2, **NP** is closed under complementation. Thus, the complement of every problem in **NP**, and therefore of every **NP**-complete problem, is in **NP**. So, the desired conclusion holds not just for some **NP**-complete problem $A$, but for all of them.

IF: Suppose $A$ is an **NP**-complete problem whose complement is also in **NP**, i.e., $A \in$ **coNP**. We want to show that **NP** = **coNP**; by Theorem 11.2, it suffices to show that **NP** is closed under complementation. To that end, let $B$ be any problem in **NP**; we will show that $\overline{B} \in$ **NP** as well. Since $A$ is **NP**-complete $B \leq_m^p A$, and therefore $\overline{B} \leq_m^p \overline{A}$. But, by assumption, $\overline{A} \in$ **NP** so, by $(*)$, $\overline{B} \in$ **NP**, as wanted. □

Theorem 11.5 suggests a possible line of attack for the **NP** $\stackrel{?}{=}$ **coNP** question: To settle it in the (less expected) equality direction, identify an **NP**-complete problem whose complement is also in **NP**. To settle it in the (more expected) inequality direction, prove that no such problem can exist.

The figure below illustrates the three possibilities for the relationship between **P**, **NP**, and **coNP**, based on our present knowledge about these classes.




Possibility (a) is what most complexity theorists believe to be the case: **P** ≠ **NP**, and **NP** ≠ **coNP**. Even in this case we do not know whether **P** is exactly the intersection of **NP** and **coNP** or only a proper subset of it. Possibility (b) represents the situation where **P** ≠ **NP** but **NP** = **coNP**. Possibility (c) represents the situation where **P** = **NP** = **coNP**. In cases (a) and (b), i.e., if **P** ≠ **NP**, it is known that the line between **P** and the **NP**-complete problems is not a sharp dividing line: there are intermediate problems that are not solvable in polytime but are not **NP**-complete. (This result is known as Ladner's theorem and its proof is beyond the scope of this course.) Following are two candidates for such problems:

- GRAPH ISOMORPHISM: Given two graphs, can we match-up their nodes one-to-one so that their edges also match-up one-to-one? More precisely:

  **Instance:** $\langle G, G' \rangle$, where $G = (V, E)$ and $G' = (V', E')$ are undirected graphs. (A similar problem can be defined for directed graphs.)

  **Question:** Is there a bijection $f : V \to V'$ such that $(u, v) \in E$ if and only if $\big(f(u), f(v)\big) \in E'$?

- FACTORING: Given positive integers $n$ and $k$, is there a non-trivial factor of $n$ that is no greater than $k$? More precisely:

  **Instance:** $\langle n, k \rangle$, where $n, k > 1$ are integers.

  **Question:** Is there some integer $q$ such that $1 < q \leq k$ and $n \bmod q = 0$?

Both of these problems have been studied extensively. We do not know of any polynomial time algorithm to solve either of them, but we also don't know if either of them is **NP**-complete.

## The polynomial-time hierarchy

There is a classification of certain decision problems into degrees of (apparently) increasing time complexity that mirrors the arithmetic hierarchy, which as you will recall classifies decision problems into degrees of increasing undecidability. This is called the ***polynomial-time hierarchy***, which I will abbreviate as ***polytime hierarchy***. Just as we had two equivalent ways of characterizing the arithmetic hierarchy — one based on the types of formulas that define the languages and one based on the kind of oracles needed to recognize the languages — there are also two equivalent ways of characterizing the polytime hierarchy.

### The logic-based view

We say that a predicate $P(x, y_1, \ldots, y_i)$ is ***polynomially limited*** (***polylimited*** for short) if it can be true ***only*** for $y_1, \ldots, y_i$ whose length is at most polynomial in the length of $x$: i.e., there is some constant $k$ such that if $P(x, y_1, \ldots, y_i)$ is true then $|y_j| \leq |x|^k$, for every $j \in [1..i]$. We say that $P(x, y_1, \ldots, y_i)$ is a polynomial-time (***polytime*** for short) predicate, if there is a polynomial-time deterministic Turing machine that, on input $\langle x, y_1, \ldots, y_i \rangle$ outputs 1 if $P(x, y_1, \ldots, y_i)$ is true and outputs 0 otherwise.

**Definition** *For all positive integers $i$,*

$$\Sigma_i^p = \{A\colon x \in A \Leftrightarrow \exists y_1 \forall y_2 \ldots \boldsymbol{Q}_i y_i \quad \underbrace{P(x, y_1, y_2, \ldots, y_i)}_{\textit{polylimited, polytime predicate}} \quad \}$$

$$\Pi_i^p = \textbf{\textit{co}}\Sigma_i^p = \{A\colon x \in A \Leftrightarrow \forall y_1 \exists y_2 \ldots \overline{\boldsymbol{Q}}_i y_i \quad \underbrace{P(x, y_1, y_2, \ldots, y_i)}_{\textit{polylimited, polytime predicate}} \quad \}$$

*where $\boldsymbol{Q}_i$ is $\exists$ if $i$ is odd and is $\forall$ if $i$ is even, and $\overline{\boldsymbol{Q}}_i$ is the dual quantifier of $\boldsymbol{Q}_i$.*

From the characterization of **NP** as the set of languages (i.e., decision problems) that have short, efficiently verifiable certificates, we can see that $\Sigma_1^p = \textbf{NP}$ and thus $\Pi_1^p = \textbf{coNP}$. Following are some examples of natural and interesting problems at level 2 of the polytime hierarchy.

- UNIQUE SAT

  **Instance:** $\langle \phi \rangle$, where $\phi$ is a propositional formula.

  **Question:** Is there exactly one truth assignment to the variables of $\phi$ that satisfies $\phi$?

  We can show that UNIQUESAT is in $\Sigma_2^p$ by representing with a formula as follows:

  $$\text{UNIQUE SAT} = \left\{ \langle \phi \rangle\colon \exists \langle \tau \rangle \, \forall \langle \tau' \rangle \left( \tau \text{ satisfies } \phi \wedge (\tau' \neq \tau \rightarrow \tau' \text{ does not satisfy } \phi) \right) \right\}$$

  The quantified variables are polysize (the size of a truth assignment to the variables of $\phi$ is polynomial in the size of $\langle \phi \rangle$) and the quantifier-free predicate is polytime.

  We can generate "unique" versions of any problem in **NP** — UNIQUE CLIQUE, UNIQUE TSP, UNIQUE SET COVER, etc — in a similar way, placing all of them in $\Sigma_2^p$.

- Minimum Formula

  **Instance:** $\langle \phi \rangle$, where $\phi$ is a propositional formula.

  **Question:** Is there is no formula shorter than $\phi$ that is logically equivalent to $\phi$?

  We can show that Minimum Formula is in $\Pi_2^p$ by representing it with a formula as follows:

$$\text{Minimum Formula} = \big\{\langle \phi \rangle \colon \forall \langle \phi' \rangle \, \exists \langle \tau \rangle \, \big( |\langle \phi' \rangle| < |\langle \phi \rangle| \to \tau \text{ satisfies one of } \phi \text{ and } \phi' \text{ and falsifies the other} \big) \big\}$$

  Again note that the quantified variables are polysize and the quantifier-free predicate is polytime.

- Maximum Clique

  **Instance:** $\langle G, k \rangle$, where $G = (V, E)$ is an undirected graph and $k$ is a positive integer.

  **Question:** Does the maximum clique of $G$ have size $k$?

  We can show that Maximum Clique is in $\Sigma_2^p \cap \Pi_2^p$ by representing it with the formulas, one of which places it in $\Sigma_2^p$ and the other places it in $\Pi_2^p$:

$$\text{Maximum Clique} = \big\{\langle G, k \rangle \colon \exists C \, \forall C' \, \big( (|C| = k \wedge C \text{ is a clique of } G) \wedge (|C'| > k \to C' \text{ is not a clique of } G) \big) \big\}$$

  And also:

$$\text{Maximum Clique} = \big\{\langle G, k \rangle \colon \forall C' \, \exists C \, \big( (|C| = k \wedge C \text{ is a clique of } G) \wedge (|C'| > k \to C' \text{ is not a clique of } G) \big) \big\}$$

  The quantified variables are polysize and the quantifier-free predicate is polytime.

**The oracle-based view**

Before giving the alternative definition of the polytime hierarchy in terms of oracles, it is useful to examine some analogies between phenomena we saw in computability theory and their counterparts in complexity theory.

- Recognizable languages are those whose yes-instances have decidable certificates; their no-instances might not. In general, the certificate of a yes-instance $x$ of a recognizable language $A$ is an accepting computation of a Turing machine $M$ that recognizes $A$ on input $x$. No-instances of $A$ might not have a certificate (i.e., a rejecting computation) because $M$ might loop on them.

- The complexity analogue to recognizable languages is the class **NP**: these are the languages whose yes-instances have short and efficiently decidable certificates; their no-instances have certificates but these might not be short or efficiently decidable. In general, the certificate of a yes-instance $x$ of an **NP** language $A$ is an accepting computation of a polytime nondeterministic Turing machine $M$ that decides $A$ on input $x$; this is short precisely because the Turing machine is polytime. A no-instance of $A$ might not have a short certificate: Although all computation paths of $M$ on any input are required to halt, a rejecting computation of $M$ on $x$ is not sufficient to conclude that $x \notin A$: as we noted earlier, for a nondeterministic Turing machine $M$ to accept $x$ it suffices for it to have some computation path on input $x$ that leads to the accept state. To verify that a nondeterministic $M$ does not accept $x$, we might have to supply the entire computation tree of $M$ on $x$ as certificate, and this is definitely not polynomial in the size of $x$.

- Decidable languages are the "nice" subset of recognizable laguages, whose yes- and no-instance both have decidable certificates. Since a decider is required to halt on every input $x$, accepting if $x$ is in the language and rejecting if $x$ is not in the language, we can verify both yes- and no-instances of the language using halting computations as certificates.
- The complexity analogue to decidable languages is the class **P**: This is the "nice" subset of **NP** consisting of the languages whose yes- and no-instances both have short and efficiently decidable certificates. Because a Turing machine $M$ decider for a problem in **P** by definition runs in polytime and is deterministic, the halting computation that certifies a yes- and no-instance $x$ is short; such a certificate is also efficiently decidable because we can use a Universal Turing machine to easily verify that the supplied computation is, in fact, a computation of $M$ on $x$.

Recall the oracle-based definition of the arithmetic hierarchy:

**Definition** *For each $i \geq 0$, define $\Delta_i$, $\Sigma_i$, $\Pi_i$ inductively as follows:*

- *If $i = 0$ then $\Delta_0 = \Sigma_0 = \Pi_0 =$ the set of decidable languages.*
- *If $i > 0$ then*
  - *$\Delta_i$ is the set of languages decided by oracle Turing machines using oracles in $\Sigma_{i-1}$.*
  - *$\Sigma_i$ is the set of languages recognized by oracle Turing machines using oracles in $\Sigma_{i-1}$.*
  - *$\Pi_i = \boldsymbol{co}\Sigma_i$.*

The analogous oracle-based definition of the polytime hierarchy is as follows:

**Definition** *For each $i \geq 0$, define $\Delta_i^p$, $\Sigma_i^p$, $\Pi_i^p$ inductively as follows:*

- *If $i = 0$ then $\Delta_0^p = \Sigma_0^p = \Pi_0^p = \boldsymbol{P}$.*
- *If $i > 0$ then*
  - *$\Delta_i^p$ is the set of languages decided by deterministic polytime oracle Turing machines using oracles in $\Sigma_{i-1}^p$.*
  - *$\Sigma_i^p$ is the set of languages decided by nondeterministic polytime oracle Turing machines using oracles in $\Sigma_{i-1}^p$.*
  - *$\Pi_i^p = \boldsymbol{co}\Sigma_i^p$.*

Note that, as with the logic-based definition, by this definition we also have $\Delta_1^p = \mathbf{P}$ and $\Sigma_1^p = \mathbf{NP}$. This is because an oracle in **P** is useless to a polytime oracle Turing machine: whatever information the oracle can provide in one step, the Turing machine can compute by itself with polytime overhead. The proof that the two definitions of the polytime hierarchy are, in fact, equivalent is very similar to the corresponding proof about the arithmetic hierarchy.

Although the analogies between the arithmetic and polytime hierarchies are striking — and clearly the former inspired the latter — the differences in our knowledge about each are just as striking.

- In the case of the arithmetic hierarchy we know that each level of the hierarchy is properly contained in the one above; for the polytime hierarchy we know that each level is contained in the one above but we do not know if the containment is proper. We know (and will prove shortly — see Theorem 11.6 below) that if proper containment breaks down at any level $i$, then the entire hierarchy above level $i$ collapses to level $i$. So, if it turns out that $\mathbf{P} = \mathbf{NP}$, the entire polytime hierarchy is just **P**; every language in any level would then be solvable in polytime by deterministic Turing machines!

- Another difference is that, in the case of the arithmetic hierarchy, we know that for every $i > 0$, $\Delta_i = \Sigma_i \cap \Pi_i$: the languages in $\Delta_i$ are precisely those that are in both $\Sigma_i$ and $\Pi_i$. We do not know if this is true in the polytime hierarchy: It is clear that $\Delta_i^p \subseteq \Sigma_i^p \cap \Pi_i^p$; but the converse is not at all clear, and is likely not true. For $i = 1$ this is precisely the open question $\mathbf{P} \stackrel{?}{=} \mathbf{NP} \cap \mathbf{coNP}$ (see the discussion after Theorem 11.1).

**Theorem 11.6** *For all $i \geq 0$, if $\Sigma_i^p = \Sigma_{i+1}^p$, then for all $j \geq i+1$, $\Sigma_j^p = \Pi_j^p = \Sigma_i^p = \Pi_i^p$.*

PROOF. Suppose that $\Sigma_i^p = \Sigma_{i+1}^p$ for some $i \geq 0$.

First we show that $\Sigma_j^p = \Sigma_i^p$, for all $j \geq i+1$; we do so by induction of $j$. The basis $j = i+1$ is true by the hypothesis that $\Sigma_i^p = \Sigma_{i+1}^p$. For the induction step, let $k \geq i+1$ and suppose that $\Sigma_k^p = \Sigma_i^p$; we will prove that $\Sigma_{k+1}^p = \Sigma_i^p$. Let $A$ be any language in $\Sigma_{k+1}^p$; therefore $A$ is decidable by a polytime nondeterministic oracle Turing machine using an oracle in $\Sigma_k^p$. But $\Sigma_k^p = \Sigma_i^p$ (by the induction hypothesis) and so $A \in \Sigma_{i+1}^p = \Sigma_i^p$. Thus we have shown that $\Sigma_{k+1}^p \subseteq \Sigma_i^p$; since $i \leq k+1$, it is obvious that $\Sigma_i^p \subseteq \Sigma_{k+1}^p$, and so $\Sigma_{k+1}^p = \Sigma_i^p$. This completes the induction step, and proves that $\Sigma_j^p = \Sigma_i^p$ for every $j \geq i+1$.

Since $\Sigma_j^p = \Sigma_i^p$, we also immediately get that $\Pi_j^p = \Pi_i^p$. Since $i \leq j$, $\Sigma_i^p \subseteq \Pi_j^p$ and $\Pi_i^p \subseteq \Sigma_j^p$. So, we have $\Sigma_j^p = \Sigma_i^p \subseteq \Pi_j^p = \Pi_i^p \subseteq \Sigma_j^p$; therefore these four sets are all equal: $\Sigma_j^p = \Pi_j^p = \Sigma_i^p = \Pi_i^p$. □

Let $\mathbf{PH} = \cup_{i=0}^{\infty} \Sigma_i^p = \cup_{i=0}^{\infty} \Pi_i^p$; that is, **PH** is the complexity class consisting of all the languages at all levels of the polytime hierarchy. By Theorem 11.6, if $\mathbf{P} = \mathbf{NP}$ (i.e., the polytime hierarchy collapses at level 0) then $\mathbf{P} = \mathbf{PH}$.