

Relativization Fallacy

Chris Calabro

June 23, 2007

Abstract

Students are often confused about the meaning of relativized complexity classes, e.g. P^{NP} . In particular, it is true that for complexity classes A, B, C ,

$$A \subseteq B \Rightarrow C^A \subseteq C^B \quad (1)$$

but

$$A \subseteq B \Rightarrow A^C \subseteq B^C, \quad (2)$$

is invalid. This document attempts to explain why by providing clear definitions and an unconditional example of where (2) fails spectacularly. We show complexity classes A, B and a language C so that

$$A \subset B \text{ and } A^C \supset B^C.$$

1 Definitions

Let \mathcal{M} be a class of machines together with a notion of what it would mean to attach an oracle to a machine in the class. By *machine*, we do not just mean a specification of the ontology of the hardware, a machine must also include what it means for it to accept or reject an input. E.g. the classes NP and coNP in some sense are defined by the same class of 'machines': polytime Turing machines with a nondeterministic transition function. But we insist that a machine include what it means to accept, and NP machines and coNP machines have different notions, so we will say they define different machine classes.

An *oracle* A is a language. If m is a machine, then m^A will mean the oracle machine m with A attached.

To be completely formal, we could define a machine class as $(M, OM, \mathcal{L} : M \cup OM \times \mathcal{P}(\Sigma^*) \rightarrow \mathcal{P}(\Sigma^*))$, where $M, OM \subseteq \Sigma^*$. The idea here is that M, OM are sets of machines, represented as strings, and that \mathcal{L} says what languages they define when you do or don't attach oracles.

No one is ever this formal in practice because it is clumsy. There is also a great deal of deference to 'you know what I mean' in practice. E.g. suppose

we went through the trouble to define the machine class for P using the above formalism, only to later desire the ability to attach 2 oracles at the same time. We'd have to go back and reformalize. Then we might get scared and think, 'what if we want k oracles?' and generalize our formalism again. But then maybe we also want random oracles, or advise strings, or interaction. It gets clumsy quickly, so we just assume that our colleagues are smart enough to figure out a natural formalism and detect when that implied formalism is robust and when it is not. It's a poor situation because occasionally it is hard to detect.

You may have noticed that even the above formalism isn't quite what we expect. E.g. suppose we want to formalize the polytime Turing machines. What does it mean for an oracle Turing machine to be polytime *independent of the oracle*? Our definition seems to force us to make the choice of which machines we allow before we know what oracle will be attached. But we might imagine that whether or not the machine takes polytime depends on the oracle. At least in this case we can get around this problem by arguing that for every polytime m^A , there is an m' such that $\mathcal{L}(m'^A) = \mathcal{L}(m^A)$ and $\forall B$ m'^B runs in time a polynomial that is independent of B . But we can't always make this kind of argument because e.g. the machine class might be defined by restricting the behavior of the machine in a way that the machine itself can't detect.

Now that we have at least a provisional formalism, we can define A^B , where A is defined by a machine class $\mathcal{M} = (M, OM, \mathcal{L})$ and B is a set of languages, as $\{\mathcal{L}(m^b) \mid m \in OM, b \in B\}$.

2 The confusion begins

(1) is an immediate consequence of this definition. But (2) is not because A^B is defined by the machine class used to define A , not the set A itself. This is confusing because unlike in most of mathematics, this notation is *context sensitive*. But this only says that (2) doesn't hold obviously, maybe it can be shown by some theorem? We do have the following weak form of (2) which follows from the definitions:

Claim 1. *Let A, B be defined by machine classes $\mathcal{M} = (M, OM, \mathcal{L}), \mathcal{M}' = (M', OM', \mathcal{L}')$ where $M \subseteq M', OM \subseteq OM', \mathcal{L} \subseteq \mathcal{L}'$. Then $A \subseteq B$ and $\forall C$ $A^C \subseteq B^C$.*

E.g. it is true that $P^C \subseteq NP^C$. But for arbitrary complexity classes, the formalism is so general that it would be easy to construct a counterexample to (2). This still may not be too convincing because one could counterargue that there aren't enough restrictions on what a machine class can be. E.g. we could define that a machine class so that without oracles, the machines can decide powerful languages such as the halting problem, but as soon as you attach an oracle, they lose all power and can't decide anything but the empty language.

Maybe all complexity classes that anyone cares about have the property that (2) holds? Again, no, and we will construct a counterexample in the last section. In the next section we will see an example of why it is tempting to use (2).

3 A tempting argument

Consider the following problem: show that if $\text{NP} \subseteq \text{P/poly}$ then $\text{PH} \subseteq \text{P/poly}$. It may be tempting to argue as follows:

“We show by induction on n that $\Sigma_n \subseteq \text{P/poly}$. The base case holds because $\Sigma_0 = \text{P} \subseteq \text{P/poly}$. For the inductive case, $\Sigma_n = \text{NP}^{\Sigma_{n-1}} \subseteq \text{P/poly}^{\text{P/poly}} = \text{P/poly}$.”

Here is a correct argument: “We show by induction on n that $\Sigma_n \subseteq \text{P/poly}$. The base case holds because $\Sigma_0 = \text{P} \subseteq \text{P/poly}$. For the inductive case, let $L \in \Sigma_n = \text{NP}^{\Sigma_{n-1}} \subseteq \text{NP}^{\text{P/poly}}$ be decided by a NOTM N in time $p \in \text{poly}$ with oracle access to $A \in \text{P/poly}$, which itself can be decided in polytime given a polynomial amount of advice $a(n)$ for inputs of size n . Let $N'(x, a_1, \dots, a_{p(|x|)})$ be a nondeterministic machine that simulates $N(x)$ but uses a_i as advice when N makes a query of size i . Then $L' = \mathcal{L}(N') \in \text{NP} \subseteq \text{P/poly}$. So L' can be decided by a polytime machine M' if given polynomial advice $b(n)$ on inputs of size n . By giving M' the extra advice $a_i = a(i)$ for $i = 1, \dots, p(|x|)$, we can solve L in polytime with a polynomial amount of advice.”

The 2nd, correct argument is longer, and therefore may appear distasteful if (2) is available, so it is important to see that it is not.

4 Counterexample

We will construct our counterexample using the following intuition. We let A be defined by a powerful but deterministic machine class. By limiting the running time of such a machine, we effectively limit the number of oracle queries it can make once given an oracle, so that it can't possibly exploit all the information in the oracle. Then we let B be a relatively weak nondeterministic class. By making nondeterministic queries, such a machine can exploit information in the oracle that a deterministic machine would never be able to see because it wouldn't make the right queries. We then construct C to hide information that allows a B -type machine to decide some language that an A -type machine can't.

Claim 2. *Let D be any complexity class properly containing NP and containing complete languages under polytime Turing reductions (e.g. RE). Then $\exists C$ such that*

$$\text{NP} \subset \text{P}^D$$

and $\text{NP}^C \supset (\text{P}^D)^C$.

Proof. Let H be D -complete under polytime Turing reductions (e.g. the halting language if $D = \text{RE}$), $A = \text{NP}$, $B = \text{P}^H$. $A \subset D \subseteq B$ since H is D -hard. It is sufficient to construct languages C, L so that $L \in A^C - B^C$ since then taking $C' = C \times \{0\} \cup H \times \{1\}$, we have $A^{C'} \supset B^{C'}$.

Let P_1, P_2, \dots be an enumeration of the polytime oracle Turing machines that allow 2 oracles, one of which is always H , such that P_i runs in time $\leq n^{\lg i}$ independent of the 2nd oracle attached to it. Since H is always connected to P_i

we will suppress the superscript H in the notation, so P_i is understood to have H already connected. We will recursively construct sequences

$$\begin{aligned} C_1 &\subseteq C_2 \subseteq \dots \\ Q_1 &\subseteq Q_2 \subseteq \dots \\ L_1 &\subseteq L_2 \subseteq \dots \\ x_1, x_2, \dots \end{aligned}$$

where C_i, Q_i, L_i are finite sets of strings, x_i is a string, so that for each i , $|x_i| = i$,

$$P_i^{C_i}(x_i) \neq (x_i \in L_i), \quad (3)$$

and the only queries $P_i^{C_i}(x_i)$ makes are in Q_i , and $(C_i \Delta C_{i-1}) \cap Q_{i-1} = \emptyset$, and $L_i \Delta L_{i-1} \subseteq \{x_i\}$. These last 3 properties ensure that (3) continues to hold even if we replace C_i by $C = \cup C_i$ and L_i by $L = \cup L_i$, which will show that $L \notin B^C$. Suppose C_j, L_j, x_j have been constructed for $j < i$. We show how to construct C_i, Q_i, L_i, x_i .

Let $x_i = 0^i$. Let $Q_i = \{q_1, \dots, q_m\}$ be all the queries that the $P_j^{C_{i-1}}(x_j)$ make to C_{i-1} for $j \leq i$. Then $m \leq i \cdot i^{\lg i} \ll 2^i$, and so there is some string $q \in 2^i - Q_i$. Set $L_i = L_{i-1} \cup \begin{cases} \{x_i\} & \text{if } P_i^{C_{i-1}}(x_i) = 0 \\ \emptyset & \text{else} \end{cases}$. Then x_i is a witness to P_i^C not being able to decide L .

But we want to leave a backdoor open so that some NP machine given access to C can decide L . So we set $C_i = C_{i-1} \cup \begin{cases} \{q\} & \text{if } x_i \in L_i \\ \emptyset & \text{else} \end{cases}$. The following polytime NOTM decides L :

$N^C(x)$
if x is not of the form 0^n , reject
choose $q \in 2^n$ nondeterministically
return $q \in C$

So $L = \mathcal{L}(N^C) \in \text{NP}^C$, which completes the proof. □