Important note: *this document is inconsistent regarding to academic definitions of P and NP sets. Updated version of this document, but without this inconsistency, can be found here. This document is not deleted for conversational reasons.*

P not equal to NP

Summary: an approach is made by analyzing functions and their inverses mappings between domains and codomains elements. Drawn conclusions state that verifying data is an inverse function of finding solutions to the same problem. Further observation implies that P is not equal to NP.

"P versus NP" problem is one of the most intriguing problems in science due to an impact it would make in the world if P equals NP. While I was working on my new functional language, I decided to think about this problem in terms of computing functions, and I got some answers. Now, I can report some good news for bankers and bad news for scientists:

P ≠ NP

Solution to the problem could be seen in analyzing function domains versus their codomains and inverse functions, while extrapolating conclusions to function compositions. What does this have to do with solving problem and verifying solutions? Solving a problem and verifying a solution are actually two dual sides of the same function, and these dual sides relate as a function (solving) and its inverse (verifying). This duality can be seen in a simple example of a function $f(x) = x^2$ and its inverse. Their semantic table would look like this:

x **or** $f^{-1}(y)$	$f(x)$ **or** y
...	...
-2	4
-1	1
0	0
1	1
2	4
...	...

For f, we could say:

- Solving is happening from left to right: it is getting all right-hand values, according to already known left-hand value.
- Verifying is happening from right to left: it is finding corresponding left-hand value paired with already known right-hand side.

For f^{-1}, we could say exactly the opposite:

- Solving is happening from right to left, and it is analogous (if not the same) process to verifying f
- Verifying is happening from left to right, and it is analogous (if not the same) process to solving f

The only difference between solving and verifying is that in solving we return all of the answers, while in verifying we check the least of the answers (possibly all) against expected ones.

As any function (or its inverse) may assign multiple elements to the same parameters, function solving complexity is directly dependent of the amount of the assigned elements. More there are assigned elements, more time is needed to build up a set of solutions, and this is what makes a function (or its inverse) being a part of P or NP set. Moreover, functions can be composed, even recursively, like in the example of factorial function. A ratio between parameters and amount of solutions ranges from constant, over linear and polynomial, to exponential and even bigger complexity measure. Readers familiar to functional programming are aware of algorithmic completeness of using function compositions to form any kind of computation.

If we consider each function calculation step as a discrete unit of computation, we can enumerate three possibilities of function complexities:

- In a case of 1:N ratio between function domain elements and their mapped codomain elements, we can say that complexity measure is $O_D(d) + O_C(N)$,
- In a case of M:N ratio between function domain elements and their mapped codomain elements, the complexity is the same as in the case of 1:N ratio,
- In a case of N:1 ratio between function domain elements and their mapped codomain elements, we can say that complexity measure is $O_D(d) + O_C(1)$,
- In a case of 1:1 ratio between function domain elements and their mapped codomain elements, the complexity is the same as in the case of N:1 ratio,

where O_D is time needed for comparing function parameters to a domain element, O_C is time needed to construct a result, and d is a total number of domain elements. Analyzed function compositions give us compositions of these complexity functions.

The point of this short document is that the relation between two sides of the same function, a function f and its inverse f^{-1}, is being interpreted as solving and verifying, respectively, and that draws some conclusions which could formally be written as:

(1) $(f \in P) \leftrightarrow (f^{-1} \in NP)$
(2) $(f \notin P) \leftrightarrow (f^{-1} \notin NP)$
(3) $(f \in NP) \leftrightarrow (f^{-1} \in P)$
(4) $(f \notin NP) \leftrightarrow (f^{-1} \notin P)$

If we assume that there is such a function that:

(5) $(f \notin P)$
(6) $(f \in NP)$

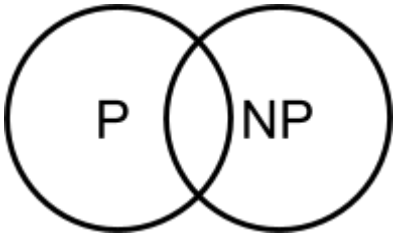
We can draw the following conclusion:

(from 2 and 5) $(f^{-1} \notin NP)$
(from 3 and 6) $(f^{-1} \in P)$

That changes even the mislead general assumption that every calculation of verifying should have a simpler complexity than corresponding calculation of solving, finally concluding that:

$P \not\supseteq$ NP

Considering all written, and according to the last observation, I propose the following Venn diagram for P versus NP situation:



Knowing that $P \neq NP$conclusion will not change the world as $P = NP$ would, I wrote this report mostly because of social reasons and my personal responsibility to report the result. I still didn't decide if this is generally a good or a bad news to all of us, but I guess we shouldn't look at things in black and white.

Author: Ivan Vodišek
May, 2017.
Special thanks go to friends from PL Forums for a fruitful conversation.