NP: the ultimate definition.

An interesting begining to the philosophy which makes P the #P problems.

Abstract

As a result of years of study the best consideration about how to respond a question must be in the simplest way.We have two philosophies to get different results, and both could be used to create some kind of different technology. A digital technology and an analogical technology. But conflicting connections must be understanded to know the real limits of our engineering.

1. Introduction.

From the ancient Greeks, they has been giving to the role of human intelligence and reason an important role: it should be able to solve the mysteries that were presenting mythological. Imagination had offered sufficient housing to create different models whose stocks were impossible to verify in most cases. Using a formal and rational discourse seek ways to reduce our language capacity to avoid all we could not prove.

Indeed, scientific development would be doomed to have to be defined within an environment or idea where the truthful itself was clear, that is, before you know what we mean by scientifically valid you should know what constitutes convincing. I mean, the philosophy of accepting or rejecting ideas convincingly may be different to we'd like for all objectives should reach science, acourding with us.

So the first, and above all, was the philosophy of science: the way we choose to give fundamental postulates a role for science in our intellectual environment convincing. One must not forget that rationally get very far, but our nose is able to tell us a few things we can not prove where they came from. Knowing how to grant the smell is vital to understand what we mean by knowledge and superstition.

About the Popper postulate considering that all knowledge must be falsacionable to get housed from the occupation of scientific approaches, of course this would be unfeasible including himself, or unfeasible including real value granted to ethics that only skeptics will never accept. Perhaps that is why raising Popper is perfect for scientific knowledge, but not allow us to develop all of the intellectual work in regard to the last degree of certainty. Therefore, should we put there a limit to reason?

We also know from the work of Kuhn and paradigms: ensures that scientific life (as opposed to how the Vienna Circle got value) lived the kind of growth and destruction without any kind of

cumulative process of ideas but a reinterpretation of them. This model had to suffer their criticisms and modifications because science today is more than centuries ago.

We must understand, therefore, that there was (and is) a real concern of how works the truthful in a perfect information system, so that when Turing and his machine appeared then we could imagine how huge was the amount of pollution concepts from good faith and desire to help.

A Turing Machine is merely a tape that is modified according to an instruction set. When our instruction manual requires us to give an answer within a time expressible polynomial bound on the size of the input, we say that the machine is polynomially bounded. This is very important because it means that we could budget their construction so that one could study the feasibility of its use.

However, these machines can present a distortion within individual instructions describing the operation of the machine itself: if you are able to describe how to get to the final state regardless of the intermediate states through which must pass the machine, then the machine is called to be not deterministic. That is, with a nondeterministic machine we say what we want and it is concerned with how to get to that state.

When we combine the two concepts (dimension determinism and polynomial) the key statement will be displayed: polynomially bounded deterministic machines are often called P, nondeterministic machines polynomially bounded are often called NP. The point is, if we can express NP like P, then we could build machines that support more descriptive instructions without losing its deterministic character. So, P is equal to NP?

If we let our imaginations go on the we could begin to put more concepts to both machines (with all that that could mean), what if we could raise any kind of philosophical concept into a machine? How can we notationally fix an undefined state without losing the dimension polynomial?

The most important issue here is that there has been a lot of contributions that jumped several premises that had nothing to do with the original definition: neither can put all the questions that occur to us beyond a Turing machine, or we will not have the right to reduce the definition of what NP is for validating in a polynomial bound: both budgets are not set out in the definition.

Being strict, we might see how someone has tried to justify (or reject) even abiogenesis from Turing thesis, too imaginative because Turing thesis only says where we can put the border in any rigorous language that a human being is capable of put in mind. Certainly, the limit criterion of demarcation will mark our ability to configure a Turing machine.

And the main problem we really have is that many of the demonstrations surrounding studies of P and NP are stained with a philosophy unfit for scientific development concepts: mixed formalism with constructivism, and machine Turing does not extend beyond constructivism.

The incompleteness theorem of Gödel is essential to understand all these problems: when we combine concepts endless as omega-unification with an ability to demonstrate inductively we get inconsistent results. The omega-unification all it means is that we replace the parameters in the corresponding variable to infinity. And of course, there's something left over, can we really carry out an infinite number of assignments? Can we combine our results with arithmetic to prove any statement? Many statements of arithmetic can be proven in a Turing machine in a polynomial bound, so how long can we extent a nondeterministic machine definition for solving what we know we can not solve?

The mathematical formalism tells us that in order to prove we only the statements get in formulating coherent speech. But Gödel's theorem tells us that this language will accept consistent statements whose we do not know whether they are valid or not: so our model is victim to a possible error. This may be unacceptable when our schedules are accurate.

In any case, my intention here was only to show the difference between symbols constructivists (those from 1) and symbols formalists (those from logic). Understanding that the formalism will worry about consistency, while constructivism will worry about if it can be demonstrated / built. The formalism is more general and goes further with more claims, while constructivism is simply accurate.

It is, therefore, at this point we realize that the same statement for natural outcome and for the irrational could give the opposite result. But all our considerations will go to the wonderful demonstrations able to do this when the definition of natural was not necessary in the show.

One example is Fermat's theorem. Through a known theorem in reals we can show that the equation $\mathbf{x}^{\mathbf{n}} = \mathbf{z}^{\mathbf{n}} + \mathbf{y}^{\mathbf{n}}$ with **n**, **z**, **y** and **x** is greater than 2 will find a solution for any **n**, **y**, **z** natural in a **x** real. And of Fermat's last theorem, we understand that this equality with **x** natural is impossible. There is a fake counterexample Fermat's theorem that is very famous: Those four

naturals work the equation on any calculator, but that can not be corroborated in a computer. That is, from the point of view of natural and constructivism the mathematical statement is exactly as enunciated by Fermat.However, in terms of approaches you can take the opposite case. We can formally give a practical reality that works coherently with those irrational numbers that makes impossible the statement. And that practical reality could be a short of mind calculator.

At this point, we are ready to understand the following demonstrations.

2. Constructively, P class is different to NP class.

Let's define our Turing machine as if it were a function, to get it more comfortable. Given a tape that can encode through a natural number called **E**, we can adopt a configuration set by another natural number which we call **C**, the result is the encoded tape **S**, where **S** = **C** (**E**).

If we define a function **XOR** natural, it could be defined as the result of applying the function itself bitwise **XOR** between such natural, so if there is **A XOR B** = **C** then **A** = **B XOR C**, for any **A**, **B**, **C** natural.

With the **XOR** function definition we will declare **X** so that **X** = **C XOR E**. And here is where we can ask the big question: given the values **X** and **S**, can we deduce **E**?

We can even reduce the question to a more recognizable within our problem: suppose that **C** is bounded polynomially (and we know that **XOR** is as well), is it possible to deduce the **E** within a polynomial bound validating **S** and **X**? If we had all Turing machines at our disposal and some ability to manage at once, the answer is yes: but because we would be using a non-deterministic Turing machine.

The answer that question ensures we are asked to find a **C** that meets: **S** = **C** (**C XOR X**). Then **C** will certificate the return of **E** with formula **E** = **C XOR X**. Now, it is intended that a Turing machine configuration is able to handle all these possibilities, discriminating configurations that cause the machine to crash, or even those that are not polynomially bounded? If we had such a good mechanism we could use it to index only the machines that are not hung and are polynomial. However, in the proof of Gödel there is no reference to the detail of the dimension polynomial, so finding the equivalence means accepting that we would have a mechanism for dealing with constructible configurations, where the incompleteness theorem ensures otherwise.

This explanation may not satisfy some other scholar, however you can expect an alternative proof even more rigorous and elaborated that demonstrates booster for those who still harbor doubts: it is the using of a function called **H**.

We say that **H** is an integer representing the configuration of a Turing machine, where if its entry is **B** then its output will match a single **X** that comply necessarily:

H1. **X** is encoded as an injective correspondence in a Turing machine,

ie if **X** (**a1**) = **b** and **X** (**a2**) = **b** then **a1** = **a2**.

H2. **X** (**X**) = **A**

H3. **X** (**X + A**) = **B**

In addition we will emphasize that **H** is also injective defined in a way, that is, this study is a study of permutations.

We know that this function makes sense because **X** always find configurations where for a coding system settings: **X** (cod (**X**)) = **A**

and, on the other hand, **X** (**A** + cod(**X**)) = **B**

so that **H** (**B**) will be defined, even it will be bounded if **X** was.

Now suppose that our coding is defined such that **H** (cod (**H**)) = cod (**H**), which is also easy to define because after calculating **H** (cod '(**H**)) is easy to change the coding on the other as encryption is independent of the way it has to be resolved **H** with the same settings Turing machine itself. Ie **H** (**A**) = cod (**X**) and **H** (**A'**) = cod' (**X**) must refer to the same **X** even altering the way to encode it.

Therefore:

1. **H** (**H**) = **H**, defining the selected coding
2. **H** (**H**) = **A**, condition H2
3. **H** (**H** + **A**) = **H**, by condition H3 combined with step **1**
4. Contradicts: **H** is not injective, for steps **1** and **3**.

We know that, so being able to define **H** injective so we would only set cod(**H**) = 0, which questions the computing power of the machine to not be able to be defined within the natural and undetermined manner.

As **H** is linked to the encoding mechanism, then we can not validate the existence of that function.

On the other hand, if there were a mechanism to determine which **C** holds for an **S** and **X** where **S** = **C** (**C XOR X**) then we would be able to implement **H**. In any case the link with the dimension polynomial is easy to adhere to the demonstration for the required response.

No need to impose this type of dimension, it can be argued more general statements about the impossibility of finding configurations from their outputs. Now, what this result tells us is that the algorithms zero-knowledge are valid, as long as you know how to implement them. For example, the NP-completeness of Cook should not be valid, because his theorem is proved with a philosophy formalist, that concept will be discussed later.

3. Formalist philosophy and # P = P.

As we mentioned at the beginning, formalism gets offer us more vague and general results. The inaccuracy comprises the inclusion of infinity like an enumerable value, or even it makes use of transfinite numbers thanks to endless expressions that allow us to work on some kind of algebra.

All of this will lead us to models with a long variety of degrees of accuracy. And it takes us to the possibility that certain statements completely change sign: whatever could be false before now could be true.

That's why we study **#P**: **#P** resolutions returns the number of solutions behind a statement which can be validated within a narrow polynomially Turing machine.

The formalism allows us to find a secure connection between these two classes if we accept as valid any correspondence that may exist in that their existence is consistent and not in terms of what are we capable of doing. That is, this way of answering questions will tell us that there will be a setup, but we do not know what is.

At this point, we will need to get something more close to the original notation of the Turing Machine: we say that a Turing machine tape is composed of cells.

In each one of them we can read a symbol belonging to a finite set of symbols, which is its alphabet. The configuration of the machine consist of a description of what state to what state transitions, depending on the symbol read from a pointer to the ribbon, to determine whether to change the symbol or whether to move the pointer to the left or right.

From this original definition to Turing, we will define what we want to solve: **#SAT** is equivalent to how many solutions has a Boolean equation. To put it in our Turing machine we must stay with the definitions:

1. **Boolean variable**: Variable that takes the values 0 or 1, but not both.

2. **Literal**: Variable affirmed or denied within Boolean logic.

3. **Clause**: Sum of Boolean literals.

Any Boolean function can be expressed easily and quickly in product clauses: for this, every time we see an expression other than a sum between literals, then the expression is changed by a new variable that is equivalent to the expression. As the only operation that sum is not the product (for the denial of a product is the sum of its denied), then the process is quite simple

Lemma: (**Z** = **XY**) ↔ (¬**Z** + **X**) (¬**Z** + **Y**) (¬**X** +¬**Y** + **Z**)

To see it you must verify that

1. (**XY** → **Z)** ↔ (¬**X** +¬**Y** + **Z**)
2. (**Z** → **XY**) ↔ (¬**Z** + **X**) (¬**Z** + **Y**)

It is not of interest to do it when you only can prove every combination.

In any event, applying Demorgan rule when it were necesary, you will have a very simple sums of a product which should take a value of **1**. And we wonder how many solutions has the formula.

To solve it in our setup formalist, we only have to enter in each cell a clause in full. So if our formula has 5 clauses then the size of the input could be exactly 5, then we will not need

intermediate cells. Our alphabet, therefore, must understand the variables **V** in 3^V combinations representing a clause in full, plus the relevant symbols to manage the output of the machine. The configuration will perform a linear path to meet an invariant: in the current state of the same subscript solutions encrypt all clauses read so far.

That is why we must label the states of the Turing machine with an aditional integer: the subscript encodes a binary matrix with as many columns as variables and as many rows as possible solutions to the clauses that have been read. When the pointer moves to the right to read the next clause, which will be intersecting rows of the matrix to be recognized in that clause with the rows of the matrix that has been building solutions. Thus is the why we have defined a state transition function consistent with the required solution.

Because the size of the alphabet and the number of states is not related to the complexity of the decision, we have to understand that there will be a linear solution to problem **#SAT** in the same instant that all clauses are read and counted the number of rows, to encode that value in the tape output mode.

This result tells us that **#SAT** is a **P** class, from a formal point of view. However, the curious result goes far beyond ...

Because *Cook's theorem* defined formalistic way, we know that an **NP** language can be expressed by **SAT** in a Turing machine. So from this point of view we might well say that it is shown that **#P** = **P**. However, to reach this conclusion must overcome a double inaccuracy: the existence of a map of any **NP** to **SAT**, and the existence of the correspondence described herein.

Another thing to consider is the existence of machines about tend to produce results increasingly polynomial. That is, if there is a correspondence **P** that solves a **#P**, so it could not be unreasonable to think that there could be slow training mechanisms which would help us pull together to resolve those configurations.

One consideration to take into account is the fact that it has taken to establish a maximum number of variables. This means that if the problem would have taken any number of variables, then if we want to solve it for any type of entry and configuration it would have needed to use the "trick Cook" to solve the other problems.

These inaccuracies lead to a degree "additional" correlation search, but we only need to find two. These settings will establish the way our information system uses to evolve into **#P** and, combined with technology **#P** I'm about to reveal, will lead to an understanding of how it should evolve the software and hardware.

4. Technology # P. Spintronics and cybernetics.

Investigations remain open. Be left for future conferences, you have to eat.

1. Computers and Intractability. A guide to the theory of NP-Completeness. Michael R. Garey / David S. Johnson.

2. http:// www.claymath.org/millenium/P vs NP/ S.A. Cook

3. Clay Millenium Problem P = NP. By Harvey M. Friedman. Mathematics Colloquium in Ohio State University October 20, 2005

4. On computable numbers, with an application to the entscheidunsproblem. Alan M. Turing

5. Sobre la certeza. Ludwig Wittgenstein

6. La ciencia contemporánea y sus implicaciones filosóficas. A. Pérez de Laborda