Proof Theory and Computation: or, why study logic?

TKW

March 4, 2025

My goal here is to give the history and background of the area I've ended up studying, and in so doing try to give you an idea of why I think it's interesting. I tend to say that my field is "logic and computation", but it's not entirely obvious how those two are related, or at least the picture of logic gates on a microchip that might come to mind is not what I'm talking about. So if you can come away from this talk / document with a vague sense of what I mean then that will be a win.

Historically, logic was studied to provide a foundation — for reasoning, mathematics, philosophy, etc — the idea being that if we can formalise the way we reason, then we can be suitably sure of our conclusions. Arguably this goes back (at least in Europe) to the Greeks, but in a recognisably modern sense it probably starts with Gottlob Frege in the 19th century. His book *Begriffsschrift* (roughly "concept-writing", I am told) aimed to provide, per its subtitle, "a formal language for pure thought modeled on that of arithmetic". Arithmetic has a system of rules which guarantee the correct answer, why not the same for thinking?

This is picked up in the early 20th century by David Hilbert. The so-called "Hilbert program" was to find an axiomatic system to underlie all of mathematics. As he put it in a 1918 lecture (*Axiomatic Thought*), this axiomatic system should fulfil two conditions:

- the system should allow for "... the solvability in principle of every mathematical question...", and
- "... the theory of axioms must ... show that within every field of knowledge contradictions based on the underlying axiom-system are absolutely impossible."

I'll call these desiderata, respectively, *completeness* and *consistency*.

An early candidate for such a formal system was set theory, developed by Georg Cantor and others at the end of the 19th century. The, suprisingly effective, idea is that everything is a set, sets have things inside them (which are also sets), and we can combine them in various ways. For example, the set of counting numbers:

$$\mathbb{N} = \{0, 1, 2, \dots\}.$$

The key operation in Cantor's system is *comprehension*, which allows us to build new sets composed of all those elements satisfying a certain condition. For example, the set of even numbers (read the vertical line as "such that"):

$$\{x \mid x \text{ is an even counting number}\} = \{0, 2, 4, \dots\}.$$

This seems reasonable, but as it happens it doesn't work: the problem gets called *Russell's paradox.* Let's say that the barber shaves everyone (and only those) who don't shave themself. In Cantor's set theory, the "barber set" will be the set of all sets which are *not* a member of themself. To formalise this, we use the symbol $\in$ to denote "is an element of" — for our barber metaphor, $x \in y$ means x is shaved by y. As such, the barber set is defined by (the crossed-out symbol $\notin$ means "is not an element of", or in our case, "is not shaved by"):

$$B = \{x \mid x \notin x\}.$$

(Remember that because everything is a set, the elements of B will be sets, so it makes sense to talk about their elements.) Then, who shaves the barber? If $B \in B$, then B must satisfy the condition which defines it, so substituting $x = B$, we find $B \notin B$ (that is, if the barber shaves themself, $B \in B$, so they don't shave themself, $B \notin B$). But if $B \notin B$ then it can't satisfy the condition, so $B \in B$ (if they don't shave themself, then they do). But one of the two choices $B \in B$ or $B \notin B$ must be true[1], and in either case we have a

[1] This principle, called *tertium non datur* for "no third possibility", goes back to Aristotle. In fact, you can get very interesting systems of logic without it, but that's another talk...

contradiction. Now you can fix this system, and in fact modern maths is usually based on a version of it, but the problems with the Hilbert program run deeper.

The real killing blow to the Hilbert program came from the *incompleteness theorems*, due to Gödel in 1931. They are, roughly:

1. In any[2] consistent system, there is a formula G which is both true and not provable.

2. No consistent system can prove its own consistency.

In particular, the first theorem says that the two goals of Hilbert's program are incompatible, you can't have both consistency and completeness.

More technical aside! You can skip this bit if you like, but I think the interesting part of the proof of Gödel's theorems is actually pretty accessible. Roughly, $G =$ "the statement G is not provable". The self-reference seems suspicious, and indeed you don't write down the statement directly, but you can show that such a statement exists. If G were false, ie if G were provable, you could prove this fact by writing down a proof of G. But then your system has proven both G *and* "not G", which is not possible since our system is consistent! Therefore, G must be true, which also says that it's not provable.

For (2), you show that the proof of (1) can be done *inside* your chosen logic theory $\mathcal{T}$. That is, $\mathcal{T}$ proves the implication "$\mathcal{T}$ is consistent" $\implies G$, so given that it can't prove G it also can't prove its own consistency.

The "any" part of the statement refers to, roughly, "any system where you can do arithmetic". The hard part of the proof is showing that this is enough to talk about provability and so forth, meaning it makes sense to write down the statement G in your system $\mathcal{T}$. This happens by an unenlightening process which gets called Gödel numbering — Girard says that these theorems are like a Monet, they make more sense from further away. End technical aside.

The first incompleteness theorem says that completeness is doomed, under the reasonable assumption that we can't prove any contradictions. But Hilbert also wanted to *prove* that our axiomatic theory of logic is consistent. The second theorem says that if we want to do this, we have to use some other "bigger" theory. But to prove the new theory is consistent, we'll need a bigger one again, ... it's turtles all the way down.

So far I've demonstrated that one particular, sensible seeming, reason for studying logic doesn't work out — or at least doesn't work out as well as we might hope. So if logic can't, at least can't completely, tell us *why* things are true, our study of logic has to be grounded in an interest in logic itself. I'll call this an interest in *how* things are true, by which I mean a study of *proofs*.

So, what is a proof of a statement P? We'll follow the Brouwer-Heyting-Kolmogorov interpretation, and build it up from the pieces of the statement P. In this way, we break down our statement until we get to an "atomic" proposition, something like $13 \times 12 = 156$, which we know how to prove. If $P = A \wedge B$ — A and B — then a proof of P is a proof π_A of A and a proof π_B of B. In symbols[3]:

$$\dfrac{\begin{array}{c}\pi_A\\ \vdots\\ \vdash A\end{array} \qquad \begin{array}{c}\pi_B\\ \vdots\\ \vdash B\end{array}}{\vdash A \wedge B}\ (R\wedge)$$

If $P = A \vee B$ — A or B — a proof of P is a proof of one of A or B. For example:

$$\dfrac{\begin{array}{c}\pi_A\\ \vdots\\ \vdash A\end{array}}{\vdash A \vee B}\ (R\vee_1)$$

So far so straightforward. The interesting part of this interpretation comes when we consider a statement $P = A \to B$, the statement that A implies B. According to the BHK interpretation a proof of $A \to B$ should be a machine (function, algorithm) which transforms proofs of A into proofs of B. I'll draw this one a bit more heuristically as in fig. 1.

[2]The hard part is hidden in this word.

[3]The R in the label is for "right", because we are introducing the connective $\wedge$ on the right of the "turnstile" $\vdash$.

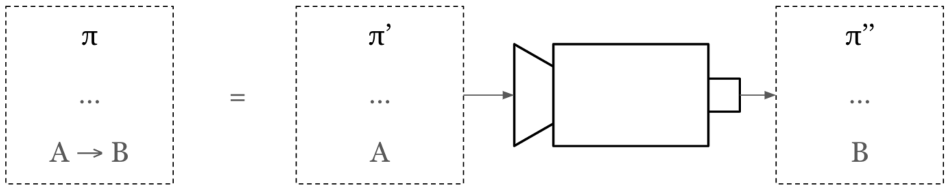

Figure 1: A proof of $A \to B$ is a machine (my little fish doodad) which transforms proofs of A into proofs of B.

This is starting to look a little closer to computation. What exactly, though, do we mean by *function*. One formal theory of functions is Alonzo Church's λ-calculus. More-or-less, Church's framework says we can do two things with functions. First, we can *apply* them to "arguments", this is what's pictured in fig. 1: we take the proof π' of A and substitute it into the function to get out a result, the proof π'' of B. The other key operation is *composition*, plugging the output of one function into another. So from $A \to B$ and $B \to C$ we get a new function $A \to C$, as in fig. 2.

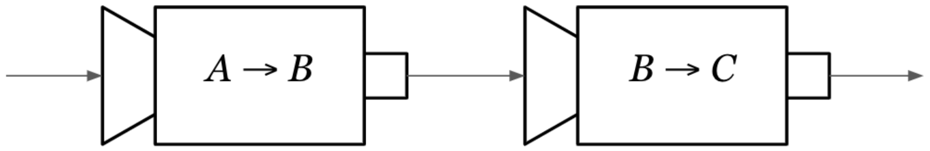

Figure 2: A composite function, combining $A \to B$ and $B \to C$ to get a function $A \to C$.

If we work with this framework for functions (and eventually computation), what does composition look like for proofs? If we write $A \vdash B$ for "A proves B", then the "composite" of two proofs π and π' looks like

$$\dfrac{\begin{array}{c}\pi \\ \vdots \\ A \vdash B\end{array} \qquad \begin{array}{c}\pi' \\ \vdots \\ B \vdash C\end{array}}{A \vdash C}\ \text{(cut)}$$

This proof system, called *sequent calculus* is due to Gerhard Gentzen — his cut rule splices together two proofs into a composite.

To get a genuine system of computation, we expect some kind of dynamic element: you start with some question, then crunch the numbers (run a program) to get the answer. In sequent calculus, the dynamic element is Gentzen's process of *cut elimination.* The idea is that some cuts are unneccesary. Proofs in the sequent calculus are structured like trees, and all the branches start with *axioms* of the form $A \vdash A$: this is the machine that, given a proof of A spits out the same proof — so it's a proof of $A \to A$. If we cut a proof of $A \vdash B$ against an axiom, we haven't added any new information, so we can eliminate that cut without loosing anything. This is pictured in table 1.

$$\dfrac{\dfrac{}{A \vdash A}\ \text{(axiom)} \qquad \begin{array}{c}\vdots \\ A \vdash B\end{array}}{A \vdash B}\ \text{(cut)} \quad \rightsquigarrow \quad \begin{array}{c}\vdots \\ A \vdash B\end{array}$$

Table 1: Eliminating a cut against an axiom.

Of course, not all cuts will be of this form, but others we can move around. For example, we can switch the order of the cut and the $L\vee$ rules in table 2.

$$\dfrac{\dfrac{\overset{\vdots}{A\vdash C}\qquad \overset{\vdots}{B\vdash C}}{A\lor B\vdash C}\,(L\lor)\qquad \overset{\vdots}{C\vdash D}}{A\lor B\vdash D}\,(\text{cut}) \quad\rightsquigarrow\quad \dfrac{\dfrac{\overset{\vdots}{A\vdash C}\qquad \overset{\vdots}{C\vdash D}}{A\vdash D}\,(\text{cut})\qquad \dfrac{\overset{\vdots}{B\vdash C}\qquad \overset{\vdots}{C\vdash D}}{B\vdash D}\,(\text{cut})}{A\lor B\vdash D}\,(L\lor)$$

Table 2: Moving the cut(s) upwards, past the $L\lor$ rule

.

The details of that example are not particularly important, the point is that we have proved the same sequent $A \lor B \vdash D$, and the cuts have moved up the tree. (Notice that the rule $L\lor$ has the same form in the second proof, but with C replaced by D.) Since the top of all the branches must be an axiom, after enough steps all the cuts will be of the form in table 1, and we can eliminate them. The fact that this algorithm always gets rid of all the cuts is Gentzen's *Hauptsatz*, for "main theorem" (you know a result is important when it gets a name in German).

Now we have given proof theory a dynamic element, how does this simulate computation? I've sketched an example in fig. 3. Say we have a proof which just spits out the number 3 — I've dotted the "input" to indicate it doesn't take any. We cut this proof "3" against a proof representing $\times 2$, that is, we plug the output "3" of the first proof into the input of the machine that doubles a number, so the composite represents 3×2. The proof we get has a cut in it, so we can apply Gentzen's algorithm to get a cut-free proof. If you set this all up right, that cut free proof will be the number six!

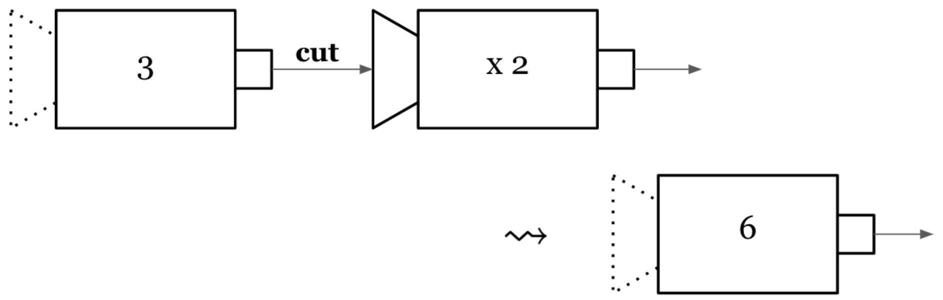

Figure 3: Cut elimination simulating the computation $3 \times 2 = 6$.

This to me is remarkable. Gentzen proves his result for purely logical reasons (in the 1930s mind you), and it turns out to be enough to simulate *any* computation. To make that claim we appeal to the *Church-Turing thesis*: a computable function is one that can be computed by a Turing machine, a kind of abstract computer. Church's λ-calculus has the same power as a Turing machine (it's "Turing complete"), and it's relatively straightforward to encode the λ-calculus into Gentzen's sequent calculus. In fact, in a precise way the *Curry-Howard correspondence* says that proofs and the λ-calculus are two perspectives on the same thing. In a nutshell:

proofs are programs, and

cut elimination is execution.

By execution I mean pressing "run" on the program. (For anyone with a computer science background, the formulas which make up the propositions in our proofs correspond to types under the Curry-Howard correspondence. I'm gliding over some subtleties here: usually the correspondence is with typed versions of the lambda calculus, which are not Turing complete since every program terminates — this fact is the analogue of Gentzen's *Hauptsatz*.)

Technically, the Curry-Howard correspondence is not particularly interesting or difficult — once you write down everything you're talking about it's almost tautological. On the other hand, the philosophical implications are quite profound, and open up new fields of study based on what proofs can tell us about programs, and vice-versa.

I'll sketch here an example. On the proof side, recall Russell's paradox, where we had a chain of implications between a statement and its negation:

$$B \in B \implies B \notin B \implies B \in B \implies \cdots$$

If we forgot for a moment that this gives us a contradiction (because one of $B \in B$ or $B \notin B$ must be true), this looks something like a computation (program, algorithm) which runs forever. You start with one statement, and loop around and around forever. On the program side, this corresponds roughly to the *Halting problem*: find an algorithm which, given a specification of a program, tells you whether it will spit out an answer (halt) or run forever. It's a theorem of Church and Turing that this is impossible. What's remarkable to me is that the methods of proof for these two facts are the same! Under the hood, they work by a kind of self reference — bringing about a contradiction by applying something to itself, in the manner of the Epimenides paradox:

Epimenides was a Cretan who made the immortal statement: "All Cretans are liars."

In fact, the Gödel sentence $G \approx$ "The sentence G is not provable" which features in the incompleteness theorems comes about in the same way again.

Much of this talk follows the work of Jean-Yves Girard, who invented *linear logic*, which I have found myself studying. If you're interested in further reading, here's some work of his you could try:

- *La Théorie de la Démonstration: du programme de Hilbert à la logique linéaire* (1997), for more detailed history, leading up to his definition of linear logic in the 80s,
- *Proofs and Types* (1989), a textbook account of the proof-program connection,
- *The Blind Spot: lectures on logic* (2010)[4], for philosophising — if you're feeling brave and willing to bear with his eccentricity...

[4]In fact, based on a course held at Roma Tre.