Church's undecidability result

Alan Turing Birth Centennial Talk at IIT Bombay, Mumbai

Joachim Breitner

April 21, 2011

Welcome, and thank you for the invitation to speak about Church's lambda calculus and how he first showed that Hilbert's decision problem is not solvable. I will first try to give an idea of the time in which this result was published and then have a closer look at the calculus and the proof. Generally, I am happy to take questions during the talk, and I am even happier if the audience helps in answering them.

1 The early 20th century

The early 20th century must have been a very exciting time for mathematicians, although less so for computer scientists. The need for a formal language for mathematical statements and proofs had been recognized, but finding good formalisms has been hard. Already in the 19th century, George Boole had formalized propositional logic. Gottlob Frege had started a serious attempt to formalize predicate logic in a book called Begriffsschrift, using a strange two-dimensional syntax. Twenty years later, the 20th century has now just begun, Bertrand Russell pointed out in a letter to Frege what is now famous as Russell's paradox: We can define a set R as

$$s \in R \iff s \notin s$$

and then substitute R for s to find

$$R \in R \iff R \notin R$$

which is obviously quite a problem. Russell showed that the system of the Begriffsschrift is not consistent.

A later, enormous attempt to create a proper formal basis for mathematics was Russell and Alfred Whiteheads' principia mathematica. They now were aware of the problem of self-contradiction and tried hard to avoid these, using type theory: Individuals, sets of individuals, sets of sets of individuals were kept separate. I will refrain from telling after

how many pages of work the statement 1+1=2 was formalized – all in all it was a tedious, but worthwhile and well-respected approach.

Now David Hilbert enters the stage, and for some reason he always appears only posing questions, never giving answers. So I conclude that if you want to become one of the greatest mathematician of all times, do not waste your time with answers... Anyways, Hilbert's program asked for a formal system for all of mathematics, in which all true statements can be proven, no contradiction can occur and a decision method is available which tells us for any statement whether it is true or false.

That is a lot of wishes, and unfortunately, the world is not a nice enough place for these wishes to become true. The first setback was due to Kurt Gödel in 1931. His first incompleteness theorem states that any consistent logic strong enough to talk about arithmetic has formulas who are true, but cannot be proven true within the system. Shortly put: You cannot have consistence and completeness.

That was, understandably, quite a disappointment for Hilbert and the rest of mathematics. So what about Hilbert's last wish: Can we at least have a decision procedure, that is a mechanical method or algorithm, which can decide if a statement is provable? This question is called the Entscheidungsproblem and, poor Hilbert, that as well is not possible. This result was obtained by Alonzo Church in 1935 using his lambda calculus, by a nice argument by contradiction.

Turing, who at that time was working with Turing machines, although he certainly did not call them this way, published his result about the halting problem, which also implied the unsolvability of the Entscheidungsproblem, just months after Church. The fact that it is more famous now is probably due to computer science students having more exposure to Turing machines than to the lambda calculus.

Both these results, Church's and Turing's, have one problem: They are based on an unproven conjecture! Hilbert asked for a mechanical method of deciding the provability of a formula, something that was called to be effectively calculable. The idea he had in mind was, of course, an educated mathematician strictly following some clear rules. But – luckily – mathematicians, and humans in general, have not been formalized yet, so Church and Turing could not directly talk about such a hypothetical decision method. Instead they presented a model of computation – lambda calculus or Turing machines – and argued that anything effectively calculable could be implemented in these. To stress this, Church showed that lambda calculus is as expressive as general recursive functions, something defined by Gödel and currently discussed at that time, and he pleads: "If this interpretation or some similar one is not allowed, it is difficult to see how the notion of an algorithm can be given any exact meaning at all." But whether this really covers all that is effectively computable was doubted by some, including Gödel. Turing showed the equivalence of Turing machines and lambda calculus, and his arguing – but not proving – why Turing machines are a good model for effectively calculability at least convinced Gödel and so people started referring to it as the Church-Turing thesis – but naturally, it is not a real thesis, as it is not (and cannot) be proven.

2 Lambda calculus

Having set the stage let us now have a closer look at the lambda calculus. Lambda calculus talks about lambda terms, which are very simple: Given a set of variables V, a lambda calculus term is either a variable, an application of a lambda term to another or a lambda abstraction of an term with a free occurrence of the abstracted variable. Free means that the variable occurs in the term at a point where it is not bound by a lambda abstraction. Formally, the syntax is given by these rules, although we apply common sense when it comes to avoiding parenthesis and variable renaming.

$$\begin{aligned} v \in V &\implies v \in \Lambda \\ M, N \in \Lambda &\implies MN \in \Lambda \\ M \in \Lambda,\, x \in V,\; x \text{ free in } M &\implies \lambda x.M \in \Lambda \end{aligned}$$

Here are some examples for lambda terms, with some arbitrary names:

$$\begin{aligned} I &= (\lambda x.x) \\ \omega &= (\lambda x.xx) \\ C_3 &= (\lambda fx.f(f(fx))) \end{aligned}$$

What can we do with these terms? Not much, really, there is only one rule, called β-reduction, that we can apply to a term or any of its subterms:

$$(\lambda x.M)N \longrightarrow M[x \coloneqq N]$$

This rule replaces an application of a lambda abstraction by the abstracted term, with the abstracted variable replaced by the second term of the application. If we can apply this rules repeatedly to the term M and end up having the term N, then we say that M can be reduced to N and write $M \longrightarrow^* N$. If M and N can both be reduced to some common term, we say M and N are convertible into each other and write $M \sim N$, which is an equivalence relation. I admit that I cheated with this definition, the original definition is more general, but then shown to be equivalent to this one.

Also important is the notion of a normal form: A lambda term N is in normal form if the β-reduction rule cannot be applied to it any more. A lambda term M has a normal form N if it can be reduced to it. Here is one example. We start with C_3Iz and find that it has the normal form z.

$$\begin{aligned} C_3Iz = (\lambda fx.f(f(fx)))(\lambda x.x)z &\longrightarrow (\lambda x.x)((\lambda x.x)((\lambda x.x)z)) \\ &\longrightarrow (\lambda x.x)((\lambda x.x)z) \\ &\longrightarrow (\lambda x.x)z \longrightarrow z \end{aligned}$$

Do all lambda terms have normal forms? Unfortunately not. Let us consider ω applied to itself:

$$\omega\omega = (\lambda x.xx)(\lambda x.xx) \longrightarrow (\lambda x.xx)(\lambda x.xx) \longrightarrow \cdots$$

No matter how often we apply the β reduction rule, we never obtain a term where we cannot apply it once more.

3 Numerical functions

As mentioned before, Church showed that the lambda calculus is as expressive as the theory of recursive numerical functions, where numerical function denotes a function from the natural numbers to the natural numbers. How can we express these in lambda calculus? To do so, we first have to define the natural numbers. You have already seen one, the three, and here is the general definition:

$$C_i = (\lambda f x.\, \underbrace{f(f(\cdots(f}_{i \text{ times}}\, x))), \quad i \in \mathbb{N}.$$

If you, for a moment, interpret the variable f to represent a function and x a value, then the lambda term representing the number 42 applies the function 42 times to the value. Note that all these numbers are normal forms.

Next we can define what it means that a lambda term represents a numerical function. We say that the lambda expression F represents the function f if the application of F to an encoded natural number reduces to the appropriate natural number representing the image under f:

$$\forall i, j \in \mathbb{N}: \quad f(i) = j \iff FC_i \longrightarrow^* C_j$$

Now we can let lambda terms work with numbers. For our purposes, we also need to represent arbitrary lambda terms as numbers. We do so by a Gödel encoding, which I will denote by G. The details are not interesting; we can use, for example, prime powers, where the power of the ith prime gives the symbol at the $i$ths position, with certain powers associated with certain symbols:

$$G[(\lambda x.xx)] = 2^{11} \cdot 3^1 \cdot 5^{17} \cdot 7^{17} \cdot 11^{17} \cdot 13^{13}$$

This mapping is reversible and both directions are, as Hilbert might have put it, effectively calculable.

Now we are able to create numerical functions that work on the Gödel encoding of lambda terms. For example, there is a function that takes a natural number and detects whether it is the Gödel encoding of a valid lambda term. Or a function that converts between natural numbers and Gödel encodings of lambda terms representing natural numbers (confusing, isn't it?). Or a function that can perform lambda reduction on a Gödel encoded lambda term. Or a function that checks whether a Gödel encoded lambda term is in normal form. Or a function that detects whether a Gödel encoded lambda term has a normal form...

$n \mapsto n \in G[\Lambda]?$
$n \mapsto G[C_n]$
$n \mapsto G[N]$, where $G^{-1}[n] \longrightarrow N$
$n \mapsto G^{-1}[n] \not\longrightarrow?$
$n \mapsto \exists N \in \Lambda.\ G^{-1}[n] \longrightarrow^* N \not\longrightarrow?$

4 The unsolvable problem

Or is there? We have finally hit the crux of the whole thing – that function does not exist! This is why Church's paper has the title "An unsolvable problem of number theory". Let me try to sketch the proof before we see the connection to the Entscheidungsproblem.

Assume we have such a function. Then we certainly have a function that can check whether a given lambda term has one of the encoded numbers as a normal form, and which one. There is an enumeration A of all lambda terms which have a normal form – this is shown earlier in the paper. Then we can define a function e as follows:

$$e(n) = \begin{cases} i+1, & \text{if } A_n C_n \longrightarrow^* C_i \text{ for an } i \in \mathbb{N} \\ 1, & \text{otherwise} \end{cases}$$

Per our assumption e is effectively calculable, hence there is a lambda term E representing it. E has a normal form, which follows from a lemma that we skipped. Hence E is in the enumeration A, say $E = A_n$. Then we find that EC_n reduces to some number i, hence, by the definition of e, it reduces to $i+1$. That is the contradiction we were aiming for.

$$EC_n \longrightarrow^* C_i \implies A_n C_n \longrightarrow^* C_i \implies e(n) = i+1 \implies A_n C_n \longrightarrow^* C_{i+1}.$$

5 Towards the Entscheidungsproblem

Now that we have established that an effective method of deciding whether a lambda term has a normal form or not cannot exist, we can use this to show that the Entscheidungsproblem is not solvable.

So let us take any consistent logic strong enough to talk about natural numbers and recursive functions and other stuff we usually expect from a useful logic, for example the one in principia mathematica. In that logic, we can certainly give a formula which tells whether a Gödel encoded lambda term is in normal form, and also a formula which represents the β reduction relation. The transitive closure would also be formulateable, hence we would be able to express the predicate that a lambda term has a normal form:

$$\models \varphi(m) \iff \exists N \in \Lambda.\ G^{-1}[m] \longrightarrow^* N \not\longrightarrow$$

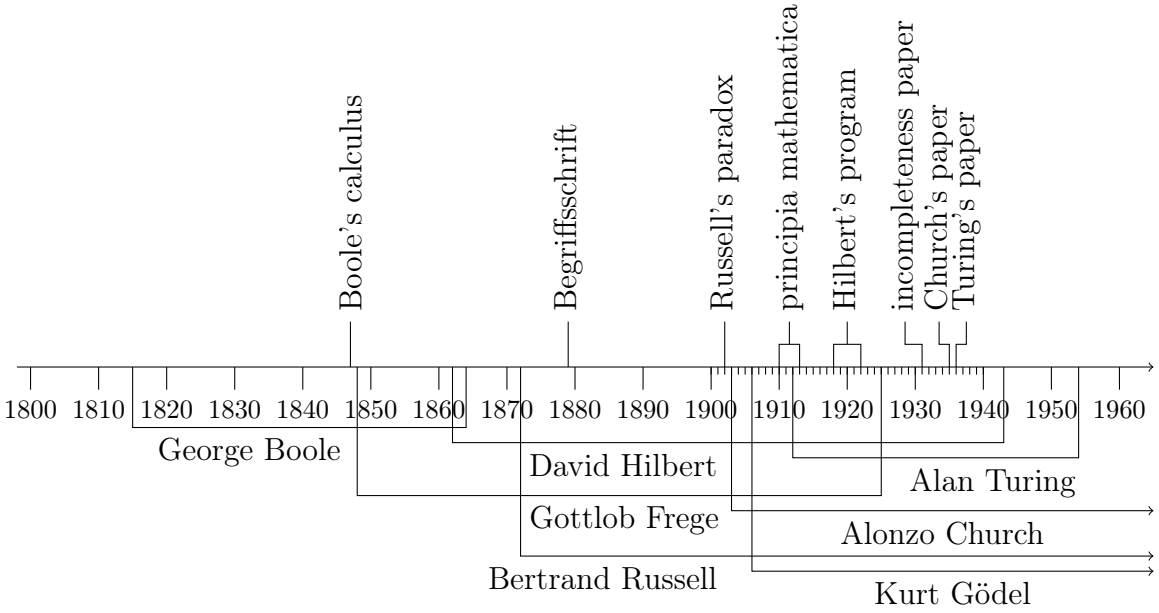
If a lambda term M has a normal form, then $\varphi(G[M])$ is provable in the logic, the proof could be given by the list of intermediate lambda terms and the final normal form. Conversely, if a lambda term does not have a normal form, then $\varphi(G[M])$ is not provable, because the logic is consistent.

Assume now the Entscheidungsproblem were solvable, so we have an effective method to decide whether $\varphi(G[M])$ is provable or not. Then we would have an effective method to decide whether M has a normal form or not. And that, as we have seen before, is not possible.

This implies that there is no hope of replacing mathematicians by computers and with this delectable result, I conclude my talk.

Timeline

For a visualization of the order of events, a time line has been prepared. The dates used are: Boole 1815-1864, calculus 1847. Frege 1848-1925, Begriffsschrift 1879. Hilbert 1862-1943, Hilbert's program 1918-1922. Russell 1872-1970, paradox 1902, principia mathematica 1910-1913. Church 1903-1995, publication 1935. Gödel 1906-1978, incompleteness theorem 1931, Turing 1912-1954, publication 1936.



Sources

- Alonzo Church: "An unsolvable Problem of Elementary Number Theory", American Journal of Mathematics, Vol. 58, No. 2 (April 1936), 345-363

- Yuri Gurevich: "The Church-Turing Thesis: Story and Recent Progress", Google Techtalk, June 8, 2009

- Wikipedia of course, what do you think?