Notes on Unprovable Statements

1 The Basic Idea

We prove that, in any formalization of mathematics which is sufficiently expressive to formalize statements about Turing machines, there are either false statements that can be proved (meaning that the formalism is *inconsistent*) or true statements that cannot be proved (meaning that the formalism is *incomplete*). This is Gödel's first incompleteness theorem. Furthermore the consistency of the formalism cannot proved within the formalism (Gödel's second incompleteness theorem), and the problem of checking whether a given statement has a proof is undecidable (the Entscheidungsproblem solved by Turing).

In the following a *formalization of mathematics* is a formal language to write mathematical statements and mathematical proofs. We will be using interchangeably the terms "formalization of mathematics," "formal system" and "system."

We begin with a very simple argument that contains all the main ideas that we will develop later.

For every Turing machine M and string x, let $S_{M,x}$ be the statement that, in our formalism, express the statement that M halts on input x, and let $\neg S_{M,x}$ be its negation. Then we can design the following algorithm for the halting problem:

- Input: $\langle M \rangle, x$
- construct the statement $S_{M,x}$
- for each string P in lexicographic order
  - if P is a valid proof of $S_{M,x}$ then accept
  - if P is a valid proof of $\neg S_{M,x}$ then reject

The algorithm correctly solves the halting problem provided that:

1. the statements $S_{M,x}$ and $\neg S_{M,x}$ exist and are efficiently computable given $\langle M \rangle, x$
2. given a statement S and a string P, it is decidable whether P is a proof of S
3. no false statement has a proof
4. all true statements have a proof

The requirements (1) and (2) are basic properties that we would like of any reasonable formalization of mathematics.

Perhaps we should justify these assumptions. If a system does not even allow you to formalize interesting theorems, then its limitations are clear from the start. In fact, one can come up with formal systems where are all true statements that you can write down are provable, but this is just because the system prevents you from writing down plenty of interesting statements. An example of such a system is the first-order theory of integers with addition discussed in Sipser's

book. From our perspective, as computer scientists, a formalization of mathematics that does not allow us to talk about algorithms and Turing machines is not useful. One may still object that important subjects of mathematics (for example calculus and number theory) could be captured by a formal system that would still be intuitively interesting but that may not allow a definition of Turing machine. As explained in Sipser's book, however, already the first-order theory of integers with addition and multiplication satisfies our first assumption (using a clever encoding of Turing machines as integers).

Regarding the second assumption, in all formalizations of mathematics proposed so far, valid proofs have a very simple structure, and can be described by a simple grammar. In any such formalization, one can construct a program to decide whether a proof is valid using yacc and a few hours of spare time. In general, a definition of "proof" that makes it undecidable to check the validity of a proof contradicts our intuitive notion that a proof is something that convinces us that a statement is true. If we allowed undecidable proofs, then it would be easy to come up with formalizations where all statements are provable: just allow the string "it's trivial" to be a valid proof for true statements.

Now, if a formalization of mathematics satisfies (1) and (2), it must either fail (3), meaning that the system is inconsistent, and hence useless, or, finally, fail (4), which means that in any sufficiently useful formalization of mathematics there are true statements that do not have a proof.

There are two undesirable feature in the above argument that we would like to eliminate. One is that the argument shows the existence of true statements that are not provable, but it does not exhibit a specific true and unprovable statement.

The other is a more subtle point. Whether a statement is true or not is not a property of a formalism to write mathematics, but of how we *interpret* the statements in the system as claims about actual mathematical objects. Logicians prefer to reason about the consistency and completeness of a formal system without reference to any interpretation. They do so by saying that a formal system is *inconsistent* if there is a statement S such that both S and $\neg S$ are provable, and by saying that a formal system is *incomplete* if there is a statement S such that neither S nor $\neg S$ are provable. Notice that if a formal system is inconsistent, then *for every interpretation* there is a false statement that is provable, because there is an S such that S and $\neg S$ are both provable, and one of them has to be false. Likewise, if a formal system is incomplete, then *for every interpretation* there is a true statement that is not provable.

The argument in this section did not show that every formal system to reason about Turing machines has to be either incomplete or inconsistent according to the syntactic definition of incompleteness and inconsistency: it could be that all the statements of the form $S_{M,x}$ are provable and none of the statements of the form $\neg S_{M,x}$ are provable.

2 A Refined Argument

Let us review the terminology we have introduced so far

- **Formalization of mathematics:** a set of mathematical statements S and proofs P, and a definition of when P is a proof of S. We will make the following assumptions:
  1. For every Turing machine M and string x, there is a statement $S_{M,x}$, which is computable given $\langle M \rangle$ and x, which is meant to encode the fact that M halts on input x.
  2. If M halts on input x, then there is a proof P of $S_{M,x}$, which is computable given $\langle M \rangle$ and x

3. If M on input x reaches a loop (meaning that the same configuration is encountered twice), then there is a proof of $\neg S_{M,x}$, which is computable given $\langle M \rangle$ and x

- **Consistency:** a formalization of mathematics is inconsistent if there is a statement S such that both S and $\neg S$ are provable; it is consistent if it is not inconsistent.

- **Completeness:** a formalization of mathematics is incomplete if there is a statement S such that neither S nor $\neg S$ are provable; it is complete if it is not incomplete.

Theorem 1 (Gödel) *Every consistent formalization of mathematics that satisfies the assumptions (1), (2) and (3) is incomplete.*

PROOF: Suppose towards a contradiction that there is a consistent and complete formal system that satisfies assumptions (1), (2) and (3). Then we can define the algorithm for the halting problem that we described in the previous section. When we proved that the halting problem is undecidable, we showed that if the halting problem was decidable we could devise an algorithm that accepts if and only if it rejects when given its own code as an input, which is impossible. We will apply the same ideas to the algorithm of the previous section, and we will end up with an algorithm M_G that, when given in input its own code, provably halts if and only if it provably loops. This means that either the statements $S_{M_G,\langle M_G \rangle}$ and $\neg S_{M_G,\langle M_G \rangle}$ are both provable, in which case the system is inconsistent, or neither of them is provable, in which case the system is incomplete. Details are below.

Consider the following algorithm, and call M_G the Turing machine that implements it:

- Input: a description $\langle M \rangle$ of a Turing machine M

- Construct the statement $S_{M,\langle M \rangle}$

- For every string P, in lexicographic order

  - If P is a proof of $S_{M,\langle M \rangle}$ then
    while (true)
    ;
  - If P is a proof of $\neg S_{M,\langle M \rangle}$ then halt

This algorithm is realized by a Turing machine M_G. Consider the behavior of M_G on input $\langle M_G \rangle$.

If it halts, then it is because it has found a proof of $\neg S_{M_G,\langle M_G \rangle}$, but if M_G halts on input $\langle M_G \rangle$ then a proof of $S_{M_G,\langle M_G \rangle}$ also exists, because of assumption (2), and so the system is inconsistent.

If it does not halt because it reaches the "while (true)" line, then it is because it has found a proof of $S_{M_G,\langle M_G \rangle}$. After that point, however, the computation of M_G continues for ever because of an infinite loop in which the same configuration is repeated again and again, and so by assumption (3) we have that $\neg S_{M_G,\langle M_G \rangle}$ is also provable, and so we violate consistency again.

The remaining case is that M_G runs for ever because it never finds any proof of $\neg S_{M_G,\langle M_G \rangle}$ or of $S_{M_G,\langle M_G \rangle}$, and thus we have a violation of completeness. □

Gödel's second incompleteness theorem is that the consistency of a formal system cannot be proved within the system, unless the system is actually inconsistent, in which case the proof means nothing. To make this theorem precise we need to make some more assumptions about what can be expressed and proved in our formal system. Instead of doing that, we will just give the proof,

and then you can see that any formal system in which the following argument can be carried out suffers from Gödel's second incompleteness theorem. (This includes all the known, and conceivable, approaches to formalizing mathematics.)

Let us look back at the machine M_G used to prove the first incompleteness theorem. We argued that, if our formal system is consistent, then M_G does not halt on input $\langle M_G \rangle$. The reasoning that we used is very basic, and it can itself be expressed in our formal system (under appropriate assumptions), which means that there is a proof, within the system, that if the system is consistent then M_G does not halt on input $\langle M_G \rangle$. But if we also have a proof, within the system, that the system is consistent, then the two proofs can be combined into a proof that M_G does not halt on input $\langle M_G \rangle$. But the existence of such a proof causes M_G to halt on input $\langle M_G \rangle$! This means that a proof of consistency cannot exist, unless the system is actually inconsistent.

Theorem 2 (Church-Turing) *In every consistent formalization $\mathcal{F}$ of mathematics that satisfies the assumptions (1), (2) and (3), the following language is undecidable:*

$$\text{Provability}_{\mathcal{F}} = \{S : \textit{ there is either a proof of } S \textit{ or a proof of } \neg S\}$$

PROOF: Suppose that for a formal system $\mathcal{F}$ that satisfies assumptions (1), (2) and (3) the language $\text{Provability}_{\mathcal{F}}$ were decidable by a machine M_P. Then consider the following algorithm M_{CT}

- Input: a description $\langle M \rangle$ of a Turing machine M
- Construct the statement $S_{M,\langle M \rangle}$
- If $M_P(S_{M,\langle M \rangle})$ rejects then halt
- For every string P, in lexicographic order
  - If P is a proof of $S_{M,\langle M \rangle}$ then
    while (true)
    ;
  - If P is a proof of $\neg S_{M,\langle M \rangle}$ then halt

We see that M_{CT} halts on input $\langle M \rangle$ if and only if there is no proof of $S_{M,\langle M \rangle}$.

Consider the computation of M_{CT} on input its own code $\langle M_{CT} \rangle$. If it halts, then there is no proof that it halts, in contradiction to assumption (2).

If it does not halt, it is because it goes into an infinite loop in which configurations are repeated, but then by assumption (3) there is a proof of $\neg S_{M_{CT},\langle M_{CT} \rangle}$, in which case it does not halt. □