Then one could build a program that determined whether an input string S, when run as a program, halts in finite time, simply by searching for all proofs or disproofs of the statement "S halts in finite time"; this program is guaranteed to terminate with a correct answer by hypothesis.

Remark 1.15.24. While it is not possible for the halting problem for a given computing language to be computable in that language, it is certainly possible that it is computable in a strictly stronger language. When that is the case, one can then invoke *Newcomb's paradox* to argue that the weaker language does not have unlimited "free will" in some sense.

Remark 1.15.25. In the language of *recursion theory*, the halting theorem asserts that the set of programs that do not halt is not a *decidable set* (or a *recursive set*). In fact, one can make the slightly stronger assertion that the set of programs that do not halt is not even a *semidecidable set* (or a *recursively enumerable set*), i.e., there is no algorithm which takes a program as input and halts in finite time if and only if the input program does not halt. This is because the complementary set of programs that do halt is clearly semidecidable (one simply runs the program until it halts, running forever if it does not), and so if the set of programs that do not halt is also semidecidable, then it is decidable (by running both algorithms in parallel; this observation is a special case of *Post's theorem*).

Remark 1.15.26. One can use the halting theorem to exclude overly general theories for certain types of mathematical objects. For instance, one cannot hope to find an algorithm to determine the existence of smooth solutions to arbitrary nonlinear partial differential equations, because it is possible to simulate a Turing machine using the laws of classical physics, which in turn can be modeled using (a moderately complicated system of) nonlinear PDE. Instead, progress in nonlinear PDE has instead proceeded by focusing on much more specific classes of such PDE (e.g., elliptic PDE, parabolic PDE, hyperbolic PDE, gauge theories, etc.)

One can place the halting theorem in a more "quantitative" form. Call a function $f : \mathbf{N} \to \mathbf{N}$ *computable* if there exists a computer program which, when given a natural number n as input, returns $f(n)$ as output in finite time. Define the *Busy Beaver function* $BB : \mathbf{N} \to \mathbf{N}$ by setting $BB(n)$ to equal the largest output of any program of at most n characters in length (and taking no input), which halts in finite time. Note that there are only finitely many such programs for any given n, so $BB(n)$ is well defined. On the other hand, it is uncomputable, even to upper bound:

Proposition 1.15.27. *There does not exist a computable function f such that one has $BB(n) \leq f(n)$ for all n.*