Collaborated with yc2454 (Yalu Cai)

1. Assume for the sake of contradiction that the halting problem is computable, so that given a machine and its input , we know whether halts on ( for yes, for no). If does halt on , then let be the set of all trivial functions where ; similarly, if does not halt on , then let be the set of all trivial functions where . Then in either these cases, would be computable, and the answer would be yes, the given machine computes one of the functions in . But Rice’s theorem says is incomputable, reaching a contradiction. Thus, the halting problem is incomputable.
2. We assume that is computable and reach the contradiction that this would allow the halting problem to be computable. Given a Turing machine and its input , the halting problem is concerned with whether halts on . Let a set be a non-trivial set of partial functions. We make only have one partial function , which is undefined for every input. We make a new Turing machine that loops infinitely for every input other than and just simulates on for , i.e. ’s pseudocode looks like this:

M\_b(input){

if (input!=x){

loop infinitely;

}

else{

return M(x);

}

}

If there were a Turing machine that computes , then we would know whether halts on , and thus whether halts on . Because if outputs on input , then we know doesn’t halt on because that means computes , which is undefined for . The complement of (let’s call it ) is also a non-trivial set of partial functions. If there were a Turing machine that computes and outputs on input , then we know (and thus, ) halts on , because then that means there’s a function in that is undefined for every input other than and is defined for , and has output . Either of these Turing machines that decide or is a decider for the halting problem, which we know is incomputable. Rice’s theorem is thus proven.