The Halting Problem is Undecidable

We define the language HALT to be the set of all strings of the form P, w such that w is a string and P is a program which halts with input w. That is, HALT is a language equivalent to the halting problem.

Theorem 1 HALT *is not decidable.*

Proof: By contradiction. Suppose HALT is decidable. Then there exists a program H such that $H(P, w)$ returns true if P halts with input w, false otherwise. We assume that every prograqm is written in C++ (or whatever our favorite language is) and every string is over Σ, the alphabet of all symbols used for C++ programs.

Let Q be a C++ program such that, for any program P, $Q(P)$ halts if $P, P \notin$ HALT, and runs forever if $P, P \in$ HALT. We describe Q with pseudo-code:

$Q(P)$
If($H(P, P)$) run forever
Else halt.

We now ask whether $Q, Q \in$ HALT.

Case 1. $Q, Q \in$ HALT, that is, $H(Q, Q)$ is true. Then $Q(Q)$ runs forever, contradiction.

Case 2. $Q.Q \notin$ HALT. That is, $H(Q, Q)$ is false. Then $Q(Q)$ halts, contradiction.

In either case, we obtain a contradiction, hence HALT is undecidable. ∎

Theorem 2 HALT *is acceptable.*

Proof: Let A be the following program.

$A(P, w)$
Run P with input w
If (P ever halts)
  Accept P, w. ∎

Note that A does not decide HALT, If $P, w \notin$ HALT, it will just run forever and will never give an answer. Also, note that HALT is recursively enumerable, since it is accepted by some machine.