Are Undecidability Results Applicable in the Real World?

Jeffrey Shallit
School of Computer Science
University of Waterloo
shallit@uwaterloo.ca
https://cs.uwaterloo.ca/~shallit
Alan Turing proved that there exist natural computational problems for which there exists no algorithm that will unerringly solve them.

For example, the problem

Given Turing machine $T$ and input $w$, does $T$ halt on input $w$? is unsolvable.

Today, dozens of problems are known which are unsolvable.
There are several possible objections to the applicability of this result to real-world programs.

Turing’s result does not rule out the possibility that one might be able to solve the halting problem for all programs that one encounters in real life.

It is at least conceivable that only enormously large programs create problems.
Answering objections

But a Turing machine can simulate any other Turing machine, so if one could solve the halting problem for a universal TM, then one could solve the halting problem for any TM.

Some very “small” universal Turing machines are known: for example, there are universal TM’s with 24 states on 2 symbols, and 2 states on 18 symbols.

So the halting problem is unsolvable even for one specific, reasonably small program.

And the existence of small universal TM’s strongly suggests that there are short real-world programs for which it will not be easy to decide whether or not they halt.
Answering objections

Another objection: Turing’s result does not apply to real-world programs since it is based on Turing machines, a purely theoretical model that has access to an unbounded storage tape.

Since real machines do not have access to arbitrarily large amounts of storage, perhaps Turing’s result says nothing about real-world programs.

This criticism is technically correct, since every real-world machine is a finite-state machine.

However, if a program has access to even a very small amount of storage, say 5000 bytes, then the number of possible states is so large as to preclude solving the halting problem before the Sun burns out.
In fact, there exist very short programs for which we cannot currently determine, in practice, whether or not they halt – even if they are restricted to using some very small amount of storage, on the order of several thousand bytes.

Of course, in theory the halting problem is solvable for any machine with access to only a finite amount of storage.

More precisely, there exist very short inputless Python programs (where operations can be done on integers of arbitrary size) for which no one can currently decide whether or not they halt.
Consider the following python program, whose idea was suggested to me by Robert Israel:

```python
n = 1
s = 0

while s != n:
    n = n + 2
    s = 0
    for d in range(1, n):
        if n%d == 0:
            s = s + d
```

This program attempts to find an odd perfect number.
A perfect number is a number $n$ which is equal to the sum of its positive divisors, excluding $n$ itself. For example, 6 is a perfect number because $6 = 1 + 2 + 3$.

No odd perfect number is currently known, and it is known that the least such number is at least $10^{1500}$.

So it is currently unknown whether or not this program will ever terminate, and even if it can be proved to terminate, the running time on the fastest known computer will exceed the current age of the universe by a huge factor.
Our second example is based on Goldbach’s conjecture. This conjecture states that every even number \( > 2 \) is the sum of two primes.

Goldbach’s conjecture is known to hold for all even numbers \( < 4 \cdot 10^{18} \).

The Python program on the next slide attempts to verify Goldbach’s conjecture by testing each even number in turn. It halts if and only if Goldbach’s conjecture is false.
Example #2

def is_prime(n):
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

n = 4
flag = False
while not flag:
    n = n + 2
    i = 3
    flag = True
    while flag and 2 * i <= n:
        if is_prime(i) and is_prime(n - i):
            flag = False
        i = i + 2
Other evidence is supplied by the “$3n + 1$” problem and its variants.

In this problem, you start with a positive integer $n$ and replace $n$ by $3n + 1$ if $n$ is odd, and $n/2$ if $n$ is even.

The question is, does this procedure started with $n$ eventually reach 1, for all positive integer starting points $n$? No one currently knows.

In fact, no one knows an example of a number $n > 1$ for which more than $50 \log n$ iterations are needed.
In fact, Conway proved that a simple generalization of the “$3n + 1$” problem is unsolvable.

Dean Hickerson suggests the following variant:

```python
n = 7

while n != 1:
    if n%2==0:
        n = n//2
    else:
        n = 5*n + 1
```

Does this Python program halt? The answer is believed to be “no”, but no one currently knows how to prove it.
An even shorter version of this program was found by Artem Kholodov in 2019:

```
n=7
while~-n:n=(n//2,5*n+1)[n%2]
```

With only 32 characters, this is the shortest Python program I know for which no one currently knows whether it halts. Can you find a shorter one?

These examples should convince you that the theory of undecidability says something deep and profound, even about very short programs in the real world.