9.2
Introduction to the halting theorem

The halting problem

Halting problem: Given a program Q, if we run it would it stop?

Q: Can one build a program P, that always stops, and solves the halting problem.

Theorem ("Halting theorem")

There is no program that always stops and solves the halting problem.

The halting problem

Halting problem: Given a program Q, if we run it would it stop?
Q: Can one build a program P, that always stops, and solves the halting problem.

Theorem ("Halting theorem")

There is no program that always stops and solves the halting problem.

Intuition, why solving the Halting problem is really hard

Definition

An integer number n is a ***weird number*** *if*

- *the sum of the proper divisors (including 1 but not itself) of n the number is $> n$,*
- *no subset of those divisors sums to the number itself.*

70 is weird. Its divisors are $1, 2, 5, 7, 10, 14, 35$. $1 + 2 + 5 + 7 + 10 + 14 + 35 = 74$. No subset of them adds up to 70.

Open question: Are there are any odd weird numbers?

Write a program P that tries all odd numbers in order, and check if they are weird. The programs stops if it found such number.

If can solve halting problem $\implies$ can resolve this open problem.

Intuition, why solving the Halting problem is really hard

Definition

*An integer number n is a **weird number** if*

- *the sum of the proper divisors (including 1 but not itself) of n the number is $> n$,*
- *no subset of those divisors sums to the number itself.*

70 is weird. Its divisors are $1, 2, 5, 7, 10, 14, 35$. $1 + 2 + 5 + 7 + 10 + 14 + 35 = 74$. No subset of them adds up to 70.

Open question: Are there are any odd weird numbers?

Write a program P that tries all odd numbers in order, and check if they are weird. The programs stops if it found such number.

If can solve halting problem $\implies$ can resolve this open problem.

Intuition, why solving the Halting problem is really hard

Definition

An integer number n is a ***weird number*** *if*

- *the sum of the proper divisors (including 1 but not itself) of n the number is $> n$,*
- *no subset of those divisors sums to the number itself.*

70 is weird. Its divisors are $\mathbf{1, 2, 5, 7, 10, 14, 35}$. $\mathbf{1 + 2 + 5 + 7 + 10 + 14 + 35 = 74}$. No subset of them adds up to $\mathbf{70}$.

Open question: Are there are any odd weird numbers?

Write a program P that tries all odd numbers in order, and check if they are weird. The programs stops if it found such number.

If can solve halting problem $\implies$ can resolve this open problem.

Intuition, why solving the Halting problem is really hard

Definition

*An integer number n is a **weird number** if*

- *the sum of the proper divisors (including 1 but not itself) of n the number is $> n$,*
- *no subset of those divisors sums to the number itself.*

70 is weird. Its divisors are $1, 2, 5, 7, 10, 14, 35$. $1 + 2 + 5 + 7 + 10 + 14 + 35 = 74$. No subset of them adds up to 70.

Open question: Are there are any odd weird numbers?

Write a program P that tries all odd numbers in order, and check if they are weird. The programs stops if it found such number.

If can solve halting problem $\implies$ can resolve this open problem.

If you can halt, you can prove or disprove anything...

1. Consider any math claim C.
2. Prover algorithm P_C:
   (A) Generate sequence of all possible proofs (sequence of strings) into a pipe/queue.
   (B) $\langle p \rangle \leftarrow$ pop top of queue.
   (C) Feed $\langle p \rangle$ and $\langle C \rangle$, into a proof verifier ("easy").
   (D) If $\langle p \rangle$ valid proof of $\langle C \rangle$, then stop and accept.
   (E) Go to (B).
3. P_C halts $\iff$ C is true and has a proof.
4. If halting is decidable, then can decide if any claim in math is true.

If you can halt, you can prove or disprove anything...

1. Consider any math claim C.
2. Prover algorithm P_C:
   (A) Generate sequence of all possible proofs (sequence of strings) into a pipe/queue.
   (B) $\langle p \rangle \leftarrow$ pop top of queue.
   (C) Feed $\langle p \rangle$ and $\langle C \rangle$, into a proof verifier ("easy").
   (D) If $\langle p \rangle$ valid proof of $\langle C \rangle$, then stop and accept.
   (E) Go to (B).
3. P_C halts $\iff$ C is true and has a proof.
4. If halting is decidable, then can decide if any claim in math is true.

If you can halt, you can prove or disprove anything...

1. Consider any math claim C.
2. Prover algorithm P_C:
   (A) Generate sequence of all possible proofs (sequence of strings) into a pipe/queue.
   (B) $\langle p \rangle \leftarrow$ pop top of queue.
   (C) Feed $\langle p \rangle$ and $\langle C \rangle$, into a proof verifier ("easy").
   (D) If $\langle p \rangle$ valid proof of $\langle C \rangle$, then stop and accept.
   (E) Go to (B).
3. P_C halts $\iff$ C is true and has a proof.
4. If halting is decidable, then can decide if any claim in math is true.

If you can halt, you can prove or disprove anything...

1. Consider any math claim C.
2. Prover algorithm P_C:
   (A) Generate sequence of all possible proofs (sequence of strings) into a pipe/queue.
   (B) $\langle p \rangle \leftarrow$ pop top of queue.
   (C) Feed $\langle p \rangle$ and $\langle C \rangle$, into a proof verifier ("easy").
   (D) If $\langle p \rangle$ valid proof of $\langle C \rangle$, then stop and accept.
   (E) Go to (B).
3. P_C halts $\iff$ C is true and has a proof.
4. If halting is decidable, then can decide if any claim in math is true.

THE END

...

(for now)