Computers: What can’t they do!
Computers: What can’t they do?
Part 4: Undecidability

December 1, 2011
CS 42: Principles and Practice of Computer Science
A Strategy\(^1\) for CS Programming Assignments\(^2\)

Or: “How to succeed in CS 70 without programming at all!”

1. For each programming assignment, note that the desired program’s behavior can be rephrased as a function from the integers \(\mathbb{N}\) to the integers \(\mathbb{N}\) by encoding the input and output appropriately.

---

\(^1\) Please do not mention Prof. Stone when using the Strategy
\(^2\) This strategy not applicable in CS 42.
A Strategy\textsuperscript{1} for CS Programming Assignments\textsuperscript{2}

Or: “How to succeed in CS 70 without programming at all!”

1. For each programming assignment, note that the desired program’s behavior can be rephrased as a function from the integers $\mathbb{N}$ to the integers $\mathbb{N}$ by encoding the input and output appropriately.

2. Point out that \textbf{with 100\% probability}, a function chosen from the set of all functions from $\mathbb{N}$ to $\mathbb{N}$ is not computable by any program at all.

\textsuperscript{1} Please do not mention Prof. Stone when using the Strategy
\textsuperscript{2} This strategy not applicable in CS 42.
A Strategy\(^1\) for CS Programming Assignments\(^2\)

Or: “How to succeed in CS 70 without programming at all!”

1. For each programming assignment, note that the desired program’s behavior can be rephrased as a function from the integers \(N\) to the integers \(N\) by encoding the input and output appropriately.

2. Point out that **with 100% probability**, a function chosen from the set of all functions from \(N\) to \(N\) is not computable by any program at all.

3. Proceed to your choice of (a) the mountains, or (b) the beach.

4. Consult the fine print for disclaimers.

\(^1\) Please do not mention Prof. Stone when using the Strategy

\(^2\) This strategy not applicable in CS 42.
The TM Halting Problem

A specific undecidable problem. Is it obvious that it can’t be solved?

P is an input program, i.e., a *turing machine*, *encoded suitably*.

w is P's input data.

Note that this blue halt checker is *not* allowed to run forever.
The DFA Halting Problem

Could we solve the problem for DFAs?

The DFA Halt Checker Program

input

D, w

output

Yes, D halts on w

No, D doesn’t halt on input w.

Note that this blue halt checker is *not* allowed to run forever.
**The TM Halting Problem, Revisited**

P is an input program, i.e., a *turing machine, encoded suitably.*

w is P's input data.

The Halt Checker Program

- **Yes**, P halts on w
- **No**, P doesn't halt on input w.

Note that this blue halt checker is *not* allowed to run forever.

Weird things. Are these ok?

✓ The blue program takes two inputs, one of which is a program.

✓ If the blue program runs P(w), it may never finish!
Suppose a Halt-Checker exists:

Yes, P halts on w

No, P doesn’t halt on input w.

Our plan: Proof by Contradiction.

✓ Construct a new machine that uses this HC as a subroutine
✓ Show the new machine cannot possibly exist.
Halting is Undecidable

Suppose a Halt-Checker exists:

P, w

Halt Checker

Yes, P halts on w
No, P doesn’t halt on input w.

Cant. Machine: C

Key idea:

You can build whatever kind of program you want in here!

All usual computational techniques are OK and you can use H.C. as a subroutine!

looking for a contradiction...
Halting is Undecidable

Suppose a Halt-Checker exists:

Yes, \( P \) halts on \( w \)

No, \( P \) doesn't halt on input \( w \).

\[ P, w \]

Halt Checker

\[ Yes, P \text{ halts on } w \]

\[ No, P \text{ doesn't halt on input } w. \]

\[ P, \]

Cant. Machine: \( C \)

our CM's input is a program \( P \).

Duplicate \( P \) and use as both inputs to a halt checker.

if "Yes," then run while \((true)\);

if "No," halt and return \(42\).

\[ 42 \]

our CM's output is (sometimes) \( 42 \)

Claim: The Cant. Machine can't exist!
Beyond TM Halting

Now that we know that Halting is not decidable, we can use this to show other problems are undecidable.

**Reduction**: Show that if we could solve some problem $P$, then we could use that solution to build a Halt Checker HC! Hence, there’s no TM that solves $P$. 
The Blank-Tape Halting Problem

P is a program that takes no input (a TM that starts with a blank tape).

\[
\begin{array}{c}
\text{P} \\
\text{input}
\end{array}
\xrightarrow{\lambda-\text{Halt Checker}}
\begin{array}{c}
\text{Yes, P halts on } \lambda \\
\text{No, P does not halt on } \lambda
\end{array}
\]
The Blank-Tape Halting Problem

P is a program that takes no input (a TM that starts with a blank tape).

Claim: This machine cannot exist. Proof: Reduction from the Halting Problem: Use it to build HC.
**The Blank-Tape Halting Problem**

P is a program that takes no input (a TM that starts with a blank tape).

\[ \lambda \- Halt \ Checker \]

\[
\begin{array}{c}
P \\
\text{input}
\end{array}
\]

\[ \lambda \- Halt \ Checker \]

\[
\begin{array}{c}
\text{Yes, P halts on } \lambda \\
\text{output}
\end{array}
\]

\[
\begin{array}{c}
\text{No, P does not halt on } \lambda \\
\end{array}
\]

**Plan:** Build a H.C. machine!

using any programming we want, *plus*
our presumed no-input halt-checker.

HC

\[
\begin{array}{c}
P, w \\
\text{HC's input is a program P and input string w.}
\end{array}
\]

\[ \text{Yes} \]

\[ \text{No} \]
The Blank-Tape Halting Problem

P is a program that takes no input (a TM that starts with a blank tape).

\[ \lambda \text{-Halt Checker} \]

Input: P

Output:
- **Yes**, P halts on \( \lambda \)
- **No**, P does not halt on \( \lambda \)

**Plan:** Build a H.C. machine!

Using any programming we want, *plus* our presumed no-input halt-checker.

HC's input is a program P and input string w.

(1) Transform inputs
(2) Use (don't run) the newly created TM
(3) Transform outputs

Yes

No
**The Blank-Tape Halting Problem**

P is a program that takes no input (a TM that starts with a blank tape).

---

**HC**

Create a no-input Turing Machine that
- (1) writes w onto the blank tape
- (2) runs P on the tape when finished writing w

Call this machine $P_w$

Are all these steps doable?

---

Yes, P halts on $\lambda$

No, P does not halt on $\lambda$

---

Yes

No
Writeup for the Record

We show that a blank-tape-halt-checker cannot exist by reduction from the Halting Problem. Assume that a blank-tape-halt-checker BT(P) does exist. We build a program HC(P,w), which takes a program P and a string w as input, as follows:

1. Build a Turing Machine that takes no inputs. It first writes the string w to its blank tape, and then runs P on that tape. Call this no-input turing machine TM. Note that TM effectively runs P on w.

2. Call our blank-tape-halt-checker on this TM: BT(TM)

3. If BT(TM) reports that TM halts, halt and output “Yes”

4. If BT(TM) reports that TM does not halt, halt and output “No”

As long as BT exists, this constructed HC is a legitimate program. All of the steps are computable: writing a single, known string to a blank tape, running a turing machine’s program, and conditional-checking. However, note that HC(P,w) is a halt checker!

✓ If P halts on w, HC(P,w) returns “Yes” (because TM will halt on no input, so BT(TM) returns “Yes”)

✓ If P does not halt on w, HC(P,w) returns “No” (because TM will not halt on no input, so BT(TM) returns “No”)

Since a halt-checker cannot exist, we have reached a contradiction. Thus, our original assumption that the blank-tape-halt-checker exists was false. A blank-tape-halt-checker also cannot exist.
Show that All-Input Halting is Uncomputable

P is an input program. Assume that it returns a boolean “yes” or “no” (or runs forever) on its inputs.

Claim: This cannot exist. Proof: By Reduction from Halting.
Show that Says-Yes-To-A-Regular-Language is Uncomputable

Claim: This cannot exist. Proof: By Reduction from Halting.

HC's input is a program P and input string w.

P is an input program. Assume that it returns a boolean “yes” or “no” (or runs forever) on its inputs.

Is the collection of inputs where P says "yes" regular?

REG

Yes, it is

No, it's not.

Yes

No
Other Uncomputability Results

✓ Rice’s Theorem: No nontrivial property of program behavior is decidable.
✓ Equivalence of programs is not decidable.
✓ Various tiling/matching problems are not decidable.