COSC 341
Theory of Computing
Lecture 15
When will it stop?

Stephen Cranefield
stephen.cranefield@otago.ac.nz

Lecture slides (mostly) by Michael Albert

Keywords: Halting problem, Turing reduction

The halting problem: is HALT recursive?

Theorem

The halting language:

$$\textsc{Halt} = \{R(M)\#w \ : \ M \textit{ halts on } w\}$$

*is **not** recursive.*

- Where to begin in proving a negative like this?
- Contradiction seems the only hope, if the result were false then there would be a TM that "solves the halting problem". Perhaps we can do some engineering with it.

A detour via Russell's paradox

To give a hint of the type of thing we want to do consider Russell's paradox.

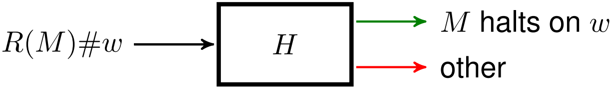
- Let Ω be the set of all sets and define:

  $$B = \{x \in \Omega \,:\, x \notin x\},$$

  the set of all sets that are not members of themselves.
- Is $B \in B$?
- Self-reference (e.g. $x \notin x$) creates problems – and a TM which analyses other TM's could also analyse (a representation of) itself ...

Can we build a halting machine?

- For the sake of contradiction suppose that a TM H (a halting machine) decides HALT.
- Remember, this means that H halts on all inputs and accepts *only and all* inputs of the form $R(M)\#w$ where:
  - $R(M)$ is the representation of a TM, and
  - $w \in \Sigma^*$,
  - M halts on w.



A diagonalisation argument

M_i \ $R(M_j)$	$R(M_1)$	$R(M_2)$	$R(M_3)$	$R(M_4)$	$\cdots$
M_1	accept	reject	reject	accept	$\cdots$
M_2	reject	reject	accept	reject	$\cdots$
M_3	reject	reject	accept	accept	$\cdots$
M_4	accept	accept	accept	reject	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Construct machine D such that the outcome of $D(RM(D))$ causes a contradiction if it appears as any element of the diagonal.

Alternative step-by-step approach: From H to AH

- From this, we can construct an anti-halting machine AH which, on input of the form $R(M)\#w$ behaves as follows:
  - First H is run on the input.
  - If H halts and rejects, i.e., M would loop on w, AH halts (and accepts)
  - If H halts and accepts, i.e., M would halt on w, AH enters an infinite loop.

From AH to D

- Now we introduce the self-reference.
- Define the machine D that does the following on input $R(M)$ (note, no w):
  - It writes a copy of $R(M)$ on the tape, so the tape is now $R(M)\#R(M)$.
  - It rewinds the read-write head to the left-hand end of the (modified) tape.
  - It runs AH.

What does D do?

- It tells us something about the behaviour of programs given their own source code as input.
- For any TM, M, if M would halt on input $R(M)$ then D loops on input $R(M)$.
- For any TM, M, if M would loop on input $R(M)$ then D halts on input $R(M)$.
- Well, D is a TM, so, "What would D do on input $R(D)$?"
- Oh dear.

What does it all mean?

- From the assumption that HALT was recursive we obtained a paradox/contradiction.
- So HALT is not recursive.
- But, we know HALT is recursively enumerable, since it's the language accepted by the universal TM.
- Therefore its complement is not even recursively enumerable.

Turing reducibility

- Carrying out this sort of diagonal argument from scratch is a pain.
- Can we build a tool that allows us to conclude that some languages are not recursive directly?
- Yes - the notion of Turing reducibility
- “If there is a mechanical procedure for converting all instances of a known undecidable problem into instances the problem we’re trying to solve (maintaining their positive or negative status), then our problem must be undecidable.”

Turing reducibility, the details

Given two languages L and K (or their associated decision problems), we say that L is Turing reducible to K, and write $L \xrightarrow{TR} K$, if there is a TM, M which always halts, and on input w leaves some other word $r(w)$ on the tape in such a way that:

$$w \in L \quad \text{if and only if} \quad r(w) \in K.$$

If L is an undecidable (i.e., non-recursive) language and $L \xrightarrow{TR} K$, then K must also be undecidable.

Otherwise we could run the reduction, and then apply the decision procedure for K. So, one way to show that K is undecidable is to show that $\textsc{Halt} \xrightarrow{TR} K$.

BLANK-HALT (example for reducibility)

$$\text{BLANK-HALT} = \{R(M) \mid M \text{ halts on a blank input tape}\}.$$

$$\text{HALT} \xrightarrow{TR} \text{BLANK-HALT}$$

The reducing machine, given an instance $R(M)\#w$ of HALT, creates (the representation of) a TM, M' whose behaviour is:

- Write w on a blank input tape
- Run M on w

The machine M' halts on blank input if and only if $R(M)$ halts on w. Therefore, if we could decide BLANK-HALT then we could also decide HALT. But we can't.

Rice's theorem

"All non-trivial semantic properties of programs are undecidable" [Wikipedia]

Theorem

Let $\mathcal{C}$ be a set of recursively enumerable languages that is non-trivial (neither $\emptyset$ nor the set of all RE languages). The set of Turing machines that accept some language in $\mathcal{C}$, written $\mathcal{L}_{\mathcal{C}} = \{R(M) : L(M) \in \mathcal{C}\}$ is undecidable.

- Consider some non-trivial $\mathcal{C}$ and assume that $\mathcal{L}_{\mathcal{C}}$ is decidable.
- Assume the empty language $\emptyset$ is not in $\mathcal{C}$ (if not, work with $\mathcal{L}_{\bar{\mathcal{C}}}$ instead).
- Choose some machine I whose language, $L(I)$, belongs to $\mathcal{C}$. This is possible because $\mathcal{C}$ is non-empty and contains RE languages.
- We produce a new Turing machine M_w that takes an input $R(M)\#w$ and operates as follows:
  - On any input x it first simulates M running on w.
  - If this halts, it then runs I on x.
- If M halts on w, then M_w behaves like I, so $L(M_w) = L(I) \in \mathcal{C}$ and therefore $R(M_w) \in \mathcal{L}_{\mathcal{C}}$.
- If M does not halt on w, then $L(M_w) = \emptyset \notin \mathcal{C}$ (assumption above) so $R(M_w) \notin \mathcal{L}_{\mathcal{C}}$.
- Thus, $R(M)\#w \in$ HALT if and only if $R(M_w) \in \mathcal{L}_{\mathcal{C}}$.
- We assumed that $\mathcal{L}_{\mathcal{C}}$ is decidable, which would mean that the halting problem is decidable—but we know it is not. Contradiction!

Modified from https://theory.stanford.edu/ trevisan/cs154-12/noterice.pdf

Examples of undecidable semantic properties of programs

Does a given TM:

- Accept a regular language?
- Accept a finite number of inputs?
- Accept only representations of prime numbers?
- Perform the same computation as another specified TM?

From https://theory.stanford.edu/ trevisan/cs154-12/noterice.pdf and Wikipedia.