Rice' theorem, the recursion theorem and why you should care!

Arno Pauly

March 22, 2021

The question for today

How does undecidability relate to actual programming stuff?

Equivalent TM's and semantic properties

Definition
We say that TM's M_1 and M_2 are equivalent ($M_1 \cong M_2$), if for any potential input $w \in \Sigma^*$ either M_1 and M_2 both do not halt, or they both halt and give the same answer.

Definition
We say that a formal language P is a *semantic property of Turing machines*, if $\langle M_1 \rangle \in P$ and $M_1 \cong M_2$ implies $M_2 \in P$. In words, descriptions of equivalent TMs are either both in P or both not in P.

This all immediately translates to "other" programming languages.

Rice' theorem

A language L is called trivial, if $L = \emptyset$ or $L = \Sigma^*$.

Theorem (Rice)

Any non-trivial semantic property of TMs is undecidable.

Example

As a consequence, it is undecidable whether executing a given program will erase your hard drive or not.

Rice' theorem – proof

1. Let P be a non-trivial semantic property. We show that the Halting problem (in the version without input) is Turing reducible to it.
2. Let M_{nothing} be a TM that does nothing, and let $M_{\text{something}}$ be a TM with $\langle M_{\text{nothing}} \rangle \in P \Leftrightarrow \langle M_{\text{something}} \rangle \notin P$.
3. Let I be the TM we receive as input for the Halting problem. Let $IM_{\text{something}}$ be "simulate I on an empty tape, suppressing any outputs; if I halts, proceed to simulate $M_{\text{something}}$ on the input". If I halts, then $IM_{\text{something}} \cong M_{\text{something}}$. If I does not halt, then $IM_{\text{something}} \cong M_{\text{nothing}}$.
4. So asking our oracle whether $\langle IM_{\text{something}} \rangle \in P$ lets us figure out whether I halts. QED.

Here is why programming is hard

- We can't decide whether a program is doing something bad (Rice).
- We can't decide whether a program does what it is supposed to do (Rice).
- We can't decide whether a program is as fast as possible (not Rice, but similar).
- We can't decide whether a program is the shortest one doing its job (not Rice, but similar).

So we need to rely on partial cases, heuristics, limited programming languages, etc.

The recursion theorem

Theorem

Recursion theorem Let $T : \Sigma^* \to \Sigma^*$ be a computable function. Then there is a Turing machine M such that $M \cong T(\langle M \rangle)$.

Corollary

Pick any conceivable computable transformation of Java programs. There is a program that does exactly the same as the transformed version.

Yes, there probably is black magic involved somehow.

An application

Let $\mathtt{Print}$ map the input Java program P to:
class HelloWorld { public static void main(String[] args) {
System.out.println(P); } }

Corollary

There is a Java program that prints it own source code.

Wait, what?

These programs are called *Quines*.

public class Quine { public static void main(String[] args) { char c=34; System.out.println(s+c+s+c+';'+'}'); } static String s="public class Quine { public static void main(String[] args) { char c=34; System.out.println(s+c+s+c+';'+'}'); } static String s=";}

Source: `https://introcs.cs.princeton.edu/java/54computability/Quine.java.html`

Outlook

- Tuesday – Register and counter machines
- Friday – another summary by Bertie
- Quiz results will be available tomorrow morning.
- Solutions to Coursework Part 2 will be posted next week.
- There will probably be new quizzes appearing over the recess, but deadlines will all be after lectures resume.