Introduction to Theoretical Computer Science

Lecture 6: Universal RMs, Halting, and Turing

Dr. Liam O’Connor

University of Edinburgh

Semester 1, 2023/2024

Universality

We’ve seen *register machines*, a computational model that I have claimed is universal.

Universality

We've seen *register machines*, a computational model that I have claimed is universal.

Key Question

Can an RM *simulate* an RM?

Universality

We've seen *register machines*, a computational model that I have claimed is universal.

Key Question

Can an RM *simulate* an RM?

That is, can we write an RM that, given some encoding of a machine M, written $\ulcorner M \urcorner$, computes the result (if any) of the machine M?

Universality

We've seen *register machines*, a computational model that I have claimed is universal.

Key Question

Can an RM *simulate* an RM?

That is, can we write an RM that, given some encoding of a machine M, written $\ulcorner M \urcorner$, computes the result (if any) of the machine M?

Firstly, we need an *encoding*.

Encoding an RM

We have registers $R_0 \dots R_{m-1}$ and the program $I_0 \dots I_{n-1}$.

Pairing

Recall that pairing functions allow us to pack multiple numbers $\langle a, b \rangle$ into one number.

Encoding an RM

We have registers $R_0 \ldots R_{m-1}$ and the program $I_0 \ldots I_{n-1}$.

Pairing

Recall that pairing functions allow us to pack multiple numbers $\langle a, b\rangle$ into one number.

$$\begin{aligned}
\ulcorner \mathrm{INC}(i) \urcorner &= \langle 0, i\rangle \\
\ulcorner \mathrm{DECJZ}(i,j) \urcorner &= \langle 1, i, j\rangle \\
\ulcorner P \urcorner &= \langle \ulcorner I_0 \urcorner, \ldots, \ulcorner I_{n-1} \urcorner\rangle \\
\ulcorner R \urcorner &= \langle R_0, \ldots, R_{m-1}\rangle \\
\ulcorner M \urcorner &= \langle \ulcorner P \urcorner, \ulcorner R \urcorner\rangle
\end{aligned}$$

Exercise: Write an RM program that, given such an encoding, computes its result, if any (very tedious but achievable).

Halting

The Halting Problem

Given an RM encoding $\ulcorner M \urcorner$, can we write a program to determine if the simulated machine halts or not?

Halting

The Halting Problem

Given an RM encoding $\ulcorner M \urcorner$, can we write a program to determine if the simulated machine halts or not?

- Suppose H is such an RM, which takes a machine coding $\ulcorner M \urcorner$ in R_0 and halts with 1 if M halts, and halts with 0 if M doesn't halt.

Halting

The Halting Problem

Given an RM encoding $\ulcorner M \urcorner$, can we write a program to determine if the simulated machine halts or not?

- Suppose H is such an RM, which takes a machine coding $\ulcorner M \urcorner$ in R_0 and halts with 1 if M halts, and halts with 0 if M doesn't halt.
- Construct a new machine $L = (P_L, R_0 \ldots)$ which, given a program $\ulcorner P \urcorner$, runs H on [the program with itself as input], i.e. the machine $(P, \ulcorner P \urcorner)$, and loops iff it halts.

Halting

The Halting Problem

Given an RM encoding $\ulcorner M \urcorner$, can we write a program to determine if the simulated machine halts or not?

- Suppose H is such an RM, which takes a machine coding $\ulcorner M \urcorner$ in R_0 and halts with 1 if M halts, and halts with 0 if M doesn't halt.
- Construct a new machine $L = (P_L, R_0 \dots)$ which, given a program $\ulcorner P \urcorner$, runs H on [the program with itself as input], i.e. the machine $(P, \ulcorner P \urcorner)$, and loops iff it halts.
- What happens if we run L with input P_L?

Halting

The Halting Problem

Given an RM encoding $\ulcorner M \urcorner$, can we write a program to determine if the simulated machine halts or not?

- Suppose H is such an RM, which takes a machine coding $\ulcorner M \urcorner$ in R_0 and halts with 1 if M halts, and halts with 0 if M doesn't halt.
- Construct a new machine $L = (P_L, R_0 \dots)$ which, given a program $\ulcorner P \urcorner$, runs H on [the program with itself as input], i.e. the machine $(P, \ulcorner P \urcorner)$, and loops iff it halts.
- What happens if we run L with input P_L?

Contradiction!

If L halts on $\ulcorner P_L \urcorner$ that means that H says that $(P_L, \ulcorner P_L \urcorner)$ loops.
If L loops on $\ulcorner P_L \urcorner$ that means that H says that $(P_L, \ulcorner P_L \urcorner)$ halts.

Diagonalization

We saw Cantor’s proof of the uncountability of infinite-length binary strings in the last lecture. This proof is another example of the same principle, which is called *diagonalization*.

Diagonalization

We saw Cantor's proof of the uncountability of infinite-length binary strings in the last lecture. This proof is another example of the same principle, which is called *diagonalization*.

Example (Gödel's first incompleteness theorem)

If a logic is capable of expressing basic (Peano) arithmetic, we can encode the provability of statements in the logic itself. Then, by the same diagonalisation trick, we can encode the statement *"This statement is not provable"* in the logic. If it is true, then it is not provable and thus the logic is *incomplete*. If it is false, then it is provable and thus the logic is *inconsistent*.

Consequences

We have sketched an argument that there are some programs that *cannot* be decided by register machines.

But what about *other* machines?

Turing Machines Reprisal

Recall a Turing Machine from prior courses. It is a machine with finite control, like an NFA or PDA, but with access to an unbounded *tape* $t_0 t_1 \ldots$ for storage. In each transition, we read and write to the tape, and move the tape head left or right.

Definition

A *Turing Machine* is a 7-tuple $\left(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}\right)$:

- Q : states
- Σ : input symbols
- $\Gamma \supseteq \Sigma$: *tape* symbols, including a blank symbol $\sqcup$.
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{-1, +1\}$
- $q_0, q_{\text{accept}}, q_{\text{reject}} \in Q$: start, accept, reject states.

Exercise: Construct a TM to recognise $\{0^n 1^n 2^n \mid n \in \mathbb{N}\}$

Programming Turing Machines

More examples are given in Sipser, ch. 3.

Programming Turing Machines

More examples are given in Sipser, ch. 3.

Question

How do RMs compare to TMs?

Programming Turing Machines

More examples are given in Sipser, ch. 3.

Question

How do RMs compare to TMs?

I claim they are equivalent in power. How would we demonstrate this?

Programming Turing Machines

More examples are given in Sipser, ch. 3.

Question

How do RMs compare to TMs?

I claim they are equivalent in power. How would we demonstrate this?

Exercise: Design a TM to simulate an RM
Exercise: Design a RM to simulate an TM

Upshot

The halting argument applies to TMs just as to RMs!

Extensions to Turing Machines

Do these modifications affect the expressivity of the machine?

- Adding the ability to stay put, i.e.:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$$

Extensions to Turing Machines

Do these modifications affect the expressivity of the machine?

- Adding the ability to stay put, i.e.:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$$

- Making the tape infinite in both directions?

Extensions to Turing Machines

Do these modifications affect the expressivity of the machine?

- Adding the ability to stay put, i.e.:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$$

- Making the tape infinite in both directions?
- Restricting to only two symbols?

Extensions to Turing Machines

Do these modifications affect the expressivity of the machine?

- Adding the ability to stay put, i.e.:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$$

- Making the tape infinite in both directions?
- Restricting to only two symbols?
- Allowing multiple tapes?

Extensions to Turing Machines

Do these modifications affect the expressivity of the machine?

- Adding the ability to stay put, i.e.:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$$

- Making the tape infinite in both directions?
- Restricting to only two symbols?
- Allowing multiple tapes?
- Allowing non-deterministic TMs? i.e.:

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{-1, +1\})$$

Extensions to Turing Machines

Do these modifications affect the expressivity of the machine?

- Adding the ability to stay put, i.e.:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$$

- Making the tape infinite in both directions?
- Restricting to only two symbols?
- Allowing multiple tapes?
- Allowing non-deterministic TMs? i.e.:

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{-1, +1\})$$

NO

Summary

We have found one problem (halting) that we *cannot* compute in either RMs or TMs.

Summary

We have found one problem (halting) that we *cannot* compute in either RMs or TMs.

The Church-Turing Thesis

Any problem is computable by any model of computation iff it is computable by a Turing Machine.

Summary

We have found one problem (halting) that we *cannot* compute in either RMs or TMs.

The Church-Turing Thesis

Any problem is computable by any model of computation iff it is computable by a Turing Machine.

Confirmed for: RMs, TMs, λ-calculus, combinator calculus, general recursive functions, pointer machines, counter machines, cellular automata, queue automata, enzyme-based DNA computers etc. etc. etc.

Summary

We have found one problem (halting) that we *cannot* compute in either RMs or TMs.

The Church-Turing Thesis

Any problem is computable by any model of computation iff it is computable by a Turing Machine.

Confirmed for: RMs, TMs, λ-calculus, combinator calculus, general recursive functions, pointer machines, counter machines, cellular automata, queue automata, enzyme-based DNA computers etc. etc. etc.

This means that for any model of computation we can think of, there are limits to what we can compute. Some problems are fundamentally *uncomputable* by any means. More on this next week.