

Introduction to Theoretical Computer Science

Lecture 8: Semi-decidability

Richard Mayr

University of Edinburgh

Semester 1, 2025/2026

A Review of Halting

The halting problem H is **not symmetric**:

- if $M \in H$ we **can** determine this: run M —if it halts say “yes”
- if $M \notin H$ we **can't** determine this by running M —it will run forever.

Definition

A problem (D, Q) is **semi-decidable** if there is a TM/RM that returns “yes” for any $d \in Q$, but may return “no” or **loop forever** when $d \notin Q$.

Semi-decidable problems are sometimes called **recognisable**.

A Review of Looping

The looping problem L is **not symmetric**:

- if $M \notin L$ we **can** determine this: run M —if it halts say “no”
- if $M \in L$ we **can't** determine this by running M —it will run forever.

Definition

A problem (D, Q) is **co-semi-decidable** if there is a TM/RM that returns “no” for any $d \notin Q$, but may return “yes” **or loop forever** when $d \in Q$.

Theorem

Any problem that is both **semi-decidable** and **co-semi-decidable** is decidable. **Why?**

Therefore L cannot be semi-decidable. **Why?**

Unrecognisable languages

We have seen that H and L are semi-decidable and co-semi-decidable, respectively.

Theorem

If a problem P is semi-decidable then its complement \overline{P} is co-semi-decidable, and vice versa.

Question

Are there any problems that are neither semi-decidable **nor** co-semi-decidable?

Yes, by the counting argument from last lecture.

Enumeration

Recall

A set S is *enumerable* if there is a bijection between S and \mathbb{N} .

A set S is called *computably enumerable* (or c.e.)¹ if the enumeration function $f : \mathbb{N} \rightarrow S$ is computable.

Example

We can think of an enumerating RM/TM as outputting an “infinite list” as it executes forever.

Question: H is not decidable. But can we *enumerate* it?

¹Sometimes called *recursively enumerable* or r.e.

Interleaving

Observe that the set of valid RM descriptions is decidable: Given $n \in \mathbb{N}$, we can check whether $n = \lceil M \rceil$ for some machine M . Therefore, we can enumerate all machines $\lceil M_0 \rceil, \lceil M_1 \rceil, \dots$

Interleaving

```
ms :=  $\langle \rangle$ ;    (A list of RMs)
i := 0
while true do
  ms := ms ++  $\langle \lceil M_i \rceil \rangle$ 
  for  $\lceil M \rceil \in$  ms do
    run M for one step and update ms
    if M has halted :
      output  $\lceil M \rceil$ ; delete M from ms
  od
  i := i + 1
od
```

Interleaving

Interleaving shows that H is **computably enumerable**.

Theorem

Any semi-decidable problem P is computably enumerable.

Why?

Any computably-enumerable problem P is semi-decidable.

Why?

Therefore, semidecidability is **the same** as c.e.-ness.

Reductions and c.e

Recall:

To prove that a problem P_2 is *hard*, show that there is an *easy* reduction from a known hard problem P_1 to P_2 .

Theorem

To prove that a problem P_2 is **not c.e.**, show that there is a **mapping** reduction from a known **not-c.e.** problem P_1 to P_2 .

Note we must use **mapping** reductions, not **Turing** reductions. Why? H is c.e. but its complement L is not. But H is Turing-reducible to L and L is Turing-reducible to H by flipping the answers.

Returning to UH

- Recall that Uniform Halting is the undecidable problem that contains all RM/TMs that halt on **all** inputs.
- We have a mapping reduction from H to UH (last lecture), so we know that UH is not co-semi-decidable. **Why?**
- We also have a mapping reduction from L to UH , in the next slide.

Conclusion

UH is neither semi-decidable nor co-semi-decidable.

We showed that such problems must exist earlier by counting.

Reducing L to UH

We want a transducer $f : RM \times \text{Input} \rightarrow RM$ such that $f(M, i)$ halts on all inputs iff M loops on i .

As with H to UH , we can make a machine that replaces any input with i and then runs M , but how do we make it stop?

Solution

Our machine will **measure** how long M runs for, with a timeout as its input.

- $f(M, i)$ will take a number n as input, and run M on i **for at most n steps**.
- If M halts on i before n steps, $f(M, i)$ goes into a loop.
- If M goes n steps without halting, our $f(M, i)$ just halts.

If M loops on i , then $f(M, i)$ will halt on all inputs n , and if M halts, then $f(M, i)$ loops on some sufficiently large n .