Lecture 20. Undecidable Problems

Reduction is the primary method for proving that a problem is computationally undecidable.

Reducing a problem A to problem B means a solution for problem B can be used to solve problem A.

To prove that a problem B is undecidable, we first assume on the contrary that B is decidable and show that, by making use of an algorithm for B (a black box), we would then be able to design an algorithm to solve another problem A that is already known to be undecidable, thus obtaining a contradiction.
Your task is to design an algorithm for A:
based on the input to problem A, decide what should be the input to the algorithm for B, and make use of the output of algorithm for B to decide what should be the output to problem A.

Let $L(M)$ be the language semidecided by M.

Theorem 1 *The following problems are all undecidable:*

1. *Given a Turing Machine M, does M halt on the empty tape? (i.e., $e \in L(M)$?)*
2. *Given a Turing Machine M, does M halt on every input string? (i.e., $L(M) = \Sigma^*$?)*
3. *Given a Turing Machine M, is there any string at all upon which M halts? (i.e., $L(M) = \emptyset$?)*
4. *Given two Turing Machines M_1 and M_2, do they halt on exactly the same input strings? (i.e., $L(M_1) = L(M_2)$?)*
5. *Given a Turing Machine M, is $L(M)$ regular?*

Theorem 2

$K_1 = \{$"M" : *Turing machine M halts on an empty tape* $\}$

is not recursive.

Proof:

- Suppose that K_1 is recursive.
- Then there exists a Turing machine M_{K_1} that *decides* it. (i.e. M_{K_1} on input "M" halts at y iff M halts on empty tape.)
- Then we would be able to use M_{K_1} to construct a Turing machine M_H that decides the language H (the halting problem). M_H behaves as follows:

  M_H: On input "M" "w"

  1. Construct a new Turing machine M_w, which is based on the given "M" and "w", that operates as follows:

     On an empty tape, write w on its tape and start simulating M. If M halts on w, then M_w halts (on e); otherwise M_w loops forever (i.e. M_w halts on e iff M halts on w).
  2. Run Turing machine M_{K_1} on input "M_w".
  3. If M_{K_1} halts at y, M_H halts at y;
     if M_{K_1} halts at n, M_H halts at n.
- This contradicts to the proven fact that the halting problem is undecidable.

Reduce from H to K_1:

Verify the correctness of M_H:

On input "M" "w"

- If "M" "w" $\in H$, i.e., M halts on w, then based on the construction of M_w, M_w will halt on the empty tape. So, M_{K_1} on input "M_w" will halt at y and so M_H will also halt at y.
- If "M" "w" $\notin H$, i.e., M does not halt on w, then based on the construction of M_w, M_w will not halt on the empty tape. So, M_{K_1} on input "M_w" will halt at n, and so M_H will also halt at n.

Theorem 3

$K_2 = \{$"M" : *Turing machine* M *halts on every input string*$\}$ *is not recursive.*

Proof:

- Suppose that K_2 is recursive.
- Then there exists a Turing machine M_{K_2} that decides it. (i.e. M_{K_2} on input "M" halts at y iff M halts on every string.)
- Then we would be able to use M_{K_2} to construct a Turing machine M_{K_1} that decides K_1 which is known to be undecidable!

  M_{K_1}: On input "M"

  1. Construct a new Turing machine M^*, which is based on M, that operates as follows:

     Given any input, erases the input, then start simulating M on the empty tape. If M halts (on empty tape), then M^* halts (on original input), else M^* loops. That is, M^* halts on every string iff M halts on the empty tape.
  2. Run Turing machine M_{K_2} on input "M^*".
  3. If M_{K_2} halts at y, M_{K_1} halts at y;
     if M_{K_2} halts at n, M_{K_1} halts at n.
- This contradicts to the proven result that K_1 is not recursive.

Reduce from K_1 to K_2:

Verify the correctness of M_{K_1}:
On input "M":

- If "M" $\in K_1$, i.e., M halts on the empty tape. Then from the construction of M^*, M^* would halt on every string. So, M_{K_2} on input "M^*" will halt at y, and M_{K_1} will halt at y.
- If "M" $\notin K_1$, i.e., M does not halt on the empty tape. From the construction of M^*, M^* would not halt on any string. So, M_{K_2} on input "M^*" will halt at n, and M_{K_1} will halt at n.

Exercise: Try to reduce from H to K_2 directly.

Theorem 4

$K_3 = \{$"M" : *Turing machine M halts on some input string*$\}$

is not recursive.

Proof

The argument for proving K_2 is not recursive works here as well since M^* is constructed such that it accepts some input iff it accepts every input.
That is: M^* will halt on some string iff M^* halts on every string, and M^* halts on every string iff M halts on the empty tape.

Exercise:

Prove that $K_3' = \{$"M" : TM M does not halt on any string$\}$ is not recursive.

Theorem 5

$$K_4 = \{\text{“}M1\text{”}\text{“}M2\text{”} : \textit{ Turing machine } M1 \textit{ and } M2 \textit{ halt on the same input strings}\}$$

is not recursive.

Proof:

- Suppose on the contrary that K_4 is recursive.
- Then there exists a Turing machine M_{K_4} that decides it. (i.e. M_{K_4} on input "$M1$" "$M2$" halts at y iff $L(M1) = L(M2)$).
- Then we would be able to use M_{K_4} to construct a Turing machine M_{K_2} that decides K_2.

  M_{K_2}: on input "M"

  1. Construct M^* that operates as follows: Given any input, halts and accepts the string immediately. (Note that $L(M^*) = \Sigma^*$ and so $L(M) = L(M^*)$ iff M halts on all strings.)
  2. Run Turing machine M_{K_4} on input "M" "M^*".
  3. If M_{K_4} halt at y, M_{K_2} halt at y;
     if M_{K_4} halt at n, M_{K_2} halt at n.
- A contradiction occurs since K_2 is known to be not recursive.

K_2 is reduced to K_4:

Instead of proving 5., we will prove the following more generel result.

Theorem 6 (Rice's Theorem) *Suppose that $\mathcal{C}$ is a proper, nonempty subset of the class of all r.e. languages. Then the following problem is undecidable: Given a Turing machine M, is $L(M) \in \mathcal{C}$?*

Proof:

- Suppose we have a TM $M_{\mathcal{C}}$ that, given "M", decides whether $L(M) \in \mathcal{C}$. We will use it to solve the halting problem.
- Idea 1: On input "M" "w", we will construct a TM M_w such that, $L(M_w) \in \mathcal{C}$ iff M halts on w. Then we run $M_{\mathcal{C}}$ on M_w, which solves the halting problem.
- Idea 2: We will pick two languages $L_1 \in \mathcal{C}$ and $L_2 \notin \mathcal{C}$ such that $L(M_w) = L_1$ if M halts on w and $L(M_w) = L_2$ if M doesn't halt on w.
- Idea 3: We will pick $L_2 = \emptyset$. (If $\emptyset \in \mathcal{C}$, we use $\overline{\mathcal{C}}$ instead of $\mathcal{C}$ and repeat the entire argument.)
- We pick $L_1 = L$ to be any language in $\mathcal{C}$. Let M_L be the TM that semidecides L. Then M_w does the following, on input x (U is the UTM):

  $$\text{if } U(\text{“}M\text{” “}w\text{”}) \neq \nearrow \text{ then } M_L(x) \text{ else } \nearrow$$

- Verify correctness: If M halts on w, M_w will halt on any input x on which M_L halts, so $L(M_w) = L$. If $M(w) = \nearrow$, M_w doesn't halt on any input, so $L(M_w) = \emptyset$.

An Unsolvable Tiling Problem

This is a classical example where we use a seemingly entirely different computation model to simulate Turing machines (and where the simplicity of TMs becomes important).

See Sec 5.6 in textbook.