Computational Complexity; slides 3, HT 2019
Undecidability

Prof. Paul W. Goldberg (Dept. of Computer Science, University of Oxford)

HT 2019

Undecidable Languages

Aim of this section

Show that there are languages (problems) that cannot be decided no matter how long we are willing to wait for an answer.

A counting argument (sketch):

- The number of Turing machines is infinite but *countable*
- The number of different languages is infinite but *uncountable*
- Therefore, there are "more" languages than Turing machines

It follows that there are languages that are not decidable.
Indeed some aren't even semi-decidable.

The Halting Problem

previous argument shows that there are undecidable languages.

Can we find a concrete example?

Halting problem (HALT)

Input: A Turing machine $\mathcal{M}$ and an input string w
Question: Does $\mathcal{M}$ halt on w?

Theorem.

1. HALT is recursively enumerable (accepted by a TM).
2. HALT is undecidable.

details in e.g. Sipser Chapter 4.2

Undecidability of HALT

Theorem.

1. HALT is recursively enumerable (accepted by a TM).
2. HALT is undecidable.

Proof structure of 2nd part:

1. A decidable language can be decided by a 1-tape machine.
2. *universal Turing acceptor* – a TM $\mathcal{U}$ that can simulate other TMs given as input (an *interpreter* for TMs).
3. reduce halting (in general) to *halting of UTM* $\mathcal{U}$.
4. if halting of $\mathcal{U}$ is decidable, there exists a TM $\mathcal{D}$ that decides if a given TM $\mathcal{M}$ *running on itself* is non-terminating.
5. running $\mathcal{D}$ on itself reveals a *paradox*: running $\mathcal{D}$ on itself terminates (and accepts) iff running $\mathcal{D}$ on itself is non-terminating.
6. no such $\mathcal{D}$ can exist, so halting of $\mathcal{U}$ (and hence halting in general) is undecidable.

some details

I'll skip construction of UTM $\mathcal{U}$

HALT: Decide for any $\langle \mathcal{M}, w \rangle$ where $\mathcal{M}$ is a TM and $w \in \{0,1\}^*$:

$$\mathcal{M} \text{ halts on } w?$$

Reduce to: Decide for 1-tape TM $\mathcal{M}$ and $w \in \{0,1\}^*$

$$\mathcal{U} \text{ halts on } \langle \mathcal{M}, w \rangle$$

Note: $\mathcal{U}$ simulates the computation of $\mathcal{M}$ on w

In particular, $\mathcal{U}$ halts on $\langle \mathcal{M}, w \rangle$ iff $\mathcal{M}$ halts on w

some details (design of “contradictory” TM $\mathcal{D}$)

Assume $\mathcal{L} := \{\langle \mathcal{M}, w\rangle \mid \mathcal{U} \text{ halts on } \langle \mathcal{M}, w\rangle\}$ is decidable

I.e. we can predict with some TM for all $\mathcal{M}$ with $\Sigma = \{0, 1\}$ and $w \in \{0, 1\}^*$ whether or not $\mathcal{U}$ halts on $\langle \mathcal{M}, w\rangle$

i.e. there is a TM $\mathcal{H}$ such that

$$\mathcal{H}(\langle \mathcal{M}, w\rangle) := \begin{cases} \textit{accept} & \text{if } \mathcal{U} \text{ halts on } \langle \mathcal{M}, w\rangle \\ \textit{reject} & \text{otherwise} \end{cases}$$

We can use $\mathcal{H}$ to build another TM $\mathcal{D}$:

$$\mathcal{D}(\langle \mathcal{M}\rangle) := \begin{cases} \textit{accept} & \text{if } \mathcal{H} \text{ rejects } \langle \mathcal{M}, \langle \mathcal{M}\rangle \\ \textit{reject} & \text{otherwise} \end{cases}$$

i.e., $\mathcal{D}(\langle \mathcal{M}\rangle) = \textit{accept}$ iff $\mathcal{M}(\langle \mathcal{M}\rangle)$ does not halt

But what result does $\mathcal{D}$ compute for input $\langle \mathcal{D}\rangle$?

$\mathcal{D}(\langle \mathcal{D}\rangle)$ halts and accepts iff $\mathcal{D}(\langle \mathcal{D}\rangle)$ does not halt

HALT, wrapping up

So, HALT is rec. enum. but not decidable, where HALT is $\{\langle\mathcal{M}, w\rangle : \mathcal{M} \text{ halts on } w\}$

Recall: A language $\mathcal{L} \subseteq \Sigma^*$ is decidable iff $\mathcal{L}$ and $\Sigma^* \setminus \mathcal{L}$ are recursively enumerable.

Proof: $\Rightarrow$ trivial. $\Leftarrow$ Let acceptors for $\mathcal{L}$ and $\Sigma^* \setminus \mathcal{L}$ run in parallel.

Corollary. $\overline{\text{HALT}}$ is not recursively enumerable.

$$\mathcal{L}(\overline{\text{HALT}}) := \{\langle\mathcal{M}, w\rangle : \ \mathcal{M} \text{ does not halt on input } w \ \}$$

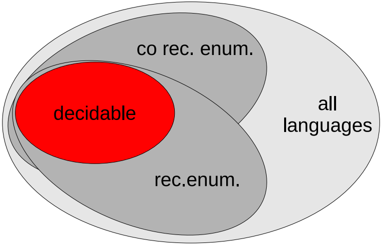
Proof. A decided for HALT can be modified to get a decider for $\overline{\text{HALT}}$.

Classification of Languages

Definition. A language $\mathcal{L} \subseteq \Sigma^*$ is *co-recursively enumerable*, or *co-r.e.*, if $\Sigma^* \setminus \mathcal{L}$ is recursively enumerable.

Example: $\mathcal{L}(\overline{\mathsf{HALT}})$ is co-r.e (but not r.e.).

Observation.[1] $\textsc{Decidable} = \textsc{r.e.} \cap \textsc{co-r.e.}$



[1]deserves more detailed explanation

We want to show that the following problems are also undecidable.

ε-Halting

Input:	Turing acceptor $\mathcal{M}$
Problem:	Does $\mathcal{M}$ halt on the empty input?

Equivalence

Input:	Turing acceptors $\mathcal{M}$ and $\mathcal{M}'$
Problem:	Is it true that $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$?

Emptiness

Input:	Turing acceptor $\mathcal{M}$
Problem:	Is $\mathcal{L}(\mathcal{M}) = \emptyset$?

Reductions

A major tool in analysing and classifying problems is the idea of "reducing one problem to another"

Reductions.

- Informally, a problem $\mathcal{A}$ is *reducible* to a problem $\mathcal{B}$ if we can use methods to solve $\mathcal{B}$ in order to solve $\mathcal{A}$.
- We want to capture the idea, that $\mathcal{A}$ is "no harder" than $\mathcal{B}$.

  (as we can use $\mathcal{B}$ to solve $\mathcal{A}$.)

Turing Reductions

Turing Reduction:

Informally, a problem $\mathcal{A}$ is *Turing reducible* to $\mathcal{B}$ if we can solve $\mathcal{A}$ using a program solving $\mathcal{B}$ as sub-program.

We write $\mathcal{A} \leq_T \mathcal{B}$.

Example: $\overline{\text{HALT}}$ is Turing reducible to HALT.

take a Turing acceptor accepting HALT as sub-program
and reverse its output

Turing Reductions

Turing Reduction:

Informally, a problem $\mathcal{A}$ is *Turing reducible* to $\mathcal{B}$ if we can solve $\mathcal{A}$ using a program solving $\mathcal{B}$ as sub-program.

We write $\mathcal{A} \leq_T \mathcal{B}$.

Example: $\overline{\text{HALT}}$ is Turing reducible to HALT.

take a Turing acceptor accepting HALT as sub-program and reverse its output

Turing reductions are free/unrestricted; sometimes too much so for our purposes.

$\leadsto$ **Many-One Reductions** (Sipser: "mapping reduction") *are more informative*: $\mathcal{A} \leq_T \mathcal{B}$ relates (un)decidability of problems; use $\mathcal{A} \leq_m \mathcal{B}$ (next slide) to find out if a problem (or its complement) is recursively enumerable.

Many-One Reductions

Definition. A language $\mathcal{A}$ is *many-one reducible* to a language $\mathcal{B}$ if there exists a computable function f such that for all $w \in \Sigma^*$:

$$x \in \mathcal{A} \iff f(x) \in \mathcal{B}.$$

We write $\mathcal{A} \leq_m \mathcal{B}$.

Observation 1. If $\mathcal{A} \leq_m \mathcal{B}$ and $\mathcal{B}$ is decidable, then so is $\mathcal{A}$.

Proof. A many-one reduction is a Turing reduction, so it inherits that functionality

Observation 2. If $\mathcal{A} \leq_m \mathcal{B}$ and $\mathcal{B}$ is recursively enumerable, then so is $\mathcal{A}$.

Equivalently, if $\mathcal{A}$ is *not* decidable (resp. r.e.) then neither is $\mathcal{B}$; so, a tool for "negative results"

Properties of Many-One Reductions

1. $\leq_m$ is *reflexive* and *transitive*
   (if $\mathcal{A} \leq_m \mathcal{B}$ and $\mathcal{B} \leq_m \mathcal{C}$ then $\mathcal{A} \leq_m \mathcal{C}$, by composition of functions.)
2. If $\mathcal{A}$ is decidable and $\mathcal{B}$ is *any* language apart from $\emptyset$ and Σ^*, then $\mathcal{A} \leq_m \mathcal{B}$.

   As $\mathcal{B} \neq \emptyset$ and $\mathcal{B} \neq \Sigma^*$ there are $w_a \in \mathcal{B}$ and $w_r \notin \mathcal{B}$.

   For $w \in \Sigma^*$, define $f(w) := \begin{cases} w_a & \text{if } w \in \mathcal{A} \\ w_r & \text{if } w \notin \mathcal{A} \end{cases}$

   (Hence, many-one reductions are too weak to distinguish between decidable problems. later: "smarter" reductions)

Examples for Many-One Reductions

We will show the following chain of reductions:

HALT $\leq_m$ ε-HALT $\leq_m$ EQUIVALENCE

ε-HALT: Does $\mathcal{M}$ halt on the empty input?

EQUIVALENCE: $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$?

Hence, all these problems are undecidable.

Lemma. HALT $\leq_m \varepsilon$-HALT

Proof. Define function f such that $w \in \text{HALT} \iff f(w) \in \varepsilon\text{-HALT}$

For $w := \langle \mathcal{M}, v \rangle$ compute the following Turing machine $\mathcal{M}_w$:

1. Write v onto the input tape.
2. Simulate $\mathcal{M}$.

Clearly, $\mathcal{M}_w$ accepts the empty word if, and only if, $\mathcal{M}$ accepts v.

Let $\mathcal{M}_r$ be a TM that does not halt on the empty input.

Define $f(w) := \begin{cases} \mathcal{M}_w & \text{if } w = \langle \mathcal{M}, v \rangle \\ \mathcal{M}_r & \text{if } w \text{ is not of the correct input form } [2] \end{cases}$

[2] i.e. doesn't encode a TM with word

Lemma. ε-HALT $\leq_m$ EQUIVALENCE

Proof. Define f such that $w \in \varepsilon\text{-HALT} \iff f(w) \in \text{EQUIVALENCE}$

Let $\mathcal{M}_a$ be a Turing machine that accepts all inputs.

For a TM $\mathcal{M}$ compute the following Turing machine $\mathcal{M}^*$:

1. Run $\mathcal{M}$ on the empty input
2. If $\mathcal{M}$ halts, accept.

$\mathcal{M}^*$ is equivalent to $\mathcal{M}_a$ if, and only if, $\mathcal{M}$ halts on the empty input.

Define

$$f(w) := \begin{cases} (\langle \mathcal{M}^* \rangle, \langle \mathcal{M}_a \rangle) & \text{if } w = \langle \mathcal{M} \rangle \\ (w, \langle \mathcal{M}_a \rangle) & \text{if } w \text{ is not of the correct input form} \end{cases}$$

Rice's Theorem

Theorem. Every non-trivial property of Turing machines is undecidable.

Rice's Theorem

Theorem. Every non-trivial property of Turing machines is undecidable.

Formally: Let $\mathcal{R}$ be a non-trivial subclass of the class of all recursively enumerable languages. ($\mathcal{R} \neq \emptyset$ and $\mathcal{R} \neq$ all r.e. lang.)

Then "$\mathcal{R}$-*ness*" is undecidable:

Given: Turing machine $\mathcal{M}$
Problem: Is $\mathcal{L}(\mathcal{M}) \in \mathcal{R}$?

(that is, $\mathcal{R}\text{-}ness = \{\langle \mathcal{M} \rangle : \mathcal{L}(\mathcal{M}) \in \mathcal{R}\}$ is undecidable.)

Rice's Theorem

Theorem. Every non-trivial property of Turing machines is undecidable.

Formally: Let $\mathcal{R}$ be a non-trivial subclass of the class of all recursively enumerable languages. ($\mathcal{R} \neq \emptyset$ and $\mathcal{R} \neq$ all r.e. lang.)

Then "$\mathcal{R}$-*ness*" is undecidable:

Given:	Turing machine $\mathcal{M}$
Problem:	Is $\mathcal{L}(\mathcal{M}) \in \mathcal{R}$?

(that is, $\mathcal{R}\text{-}ness = \{\langle\mathcal{M}\rangle : \mathcal{L}(\mathcal{M}) \in \mathcal{R}\}$ is undecidable.)

Proof. Define f such that $w \in \varepsilon\text{-HALT} \iff f(w) \in \mathcal{R}\text{-}ness$

Proof. Define f such that $w \in \varepsilon\text{-HALT} \iff f(w) \in \mathcal{R}\textit{-ness}$

W.l.o.g. assume $\emptyset \notin \mathcal{R}$. (We could always use $\overline{\mathcal{R}}$.)

Let $\mathcal{M}_L$ be a Turing machine that accepts some $L \in \mathcal{R}$.

For a TM $\mathcal{M}$ compute the following Turing machine $\mathcal{M}^*$:

On input $s \in \Sigma^*$

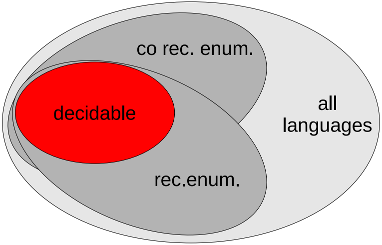
1. Simulate $\mathcal{M}$ on ε
2. If $\mathcal{M}$ halts, then simulate $\mathcal{M}_L$ on s

Clearly $\mathcal{L}(\mathcal{M}^*) = L \in \mathcal{R}$ if $\mathcal{M}$ halts on ε,
and $\mathcal{L}(\mathcal{M}^*) = \emptyset \notin \mathcal{R}$ if $\mathcal{M}$ does not halt on ε.

Let $\mathcal{M}_\emptyset$ be a TM that does not accept any input (i.e., $\mathcal{L}(\mathcal{M}_\emptyset) = \emptyset$).

Define $f(w) := \begin{cases} \mathcal{M}^* & \text{if } w = \langle \mathcal{M} \rangle \\ \mathcal{M}_\emptyset & \text{if } w \text{ is not of the correct input form} \end{cases}$

Decidable and Enumerable Languages

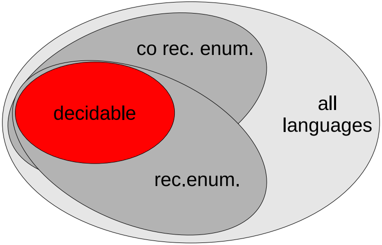


Recursion Theory:

Study the border between decidable and undecidable languages
Study the fine structure of undecidable languages.

(The work of Turing, Church, Post, ... was before computers existed.)

Decidable and Enumerable Languages



Recursion Theory:

Study the border between decidable and undecidable languages
Study the fine structure of undecidable languages.

(The work of Turing, Church, Post, ... was before computers existed.)

Complexity Theory:

Look at the fine structure of decidable languages.