

# Theory of Computer Science

## C2. The Halting Problem

Gabriele Röger

University of Basel

April 13, 2026

# Turing-recognizable vs. decidable

## Plan for this Chapter

- We will first revisit the notions **Turing-recognizable** and **Turing-decidable** and identify a connection between the two concepts.
- Then we will get to know an important undecidable problem, the **halting problem**.
- We show that it **is Turing-recognizable**...
- ... but **not Turing-decidable**.
- From these results we can conclude that **there are languages that are not Turing-recognizable**.
- Some of the postponed results on the closure and decidability properties of type 0 languages are direct implications of our findings.

## Reminder: Turing-recognizable and Turing-decidable

### Definition (Turing-recognizable Language)

We call a language **Turing-recognizable** if some deterministic Turing machine recognizes it.

## Reminder: Turing-recognizable and Turing-decidable

### Definition (Turing-recognizable Language)

We call a language **Turing-recognizable** if some deterministic Turing machine recognizes it.

A Turing machine that halts on all inputs (entering  $q_{\text{reject}}$  or  $q_{\text{accept}}$ ) is a **decider**. A decider that recognizes some language also is said to **decide** the language.

### Definition (Turing-decidable Language)

We call a language **Turing-decidable** (or **decidable**) if some deterministic Turing machine decides it.

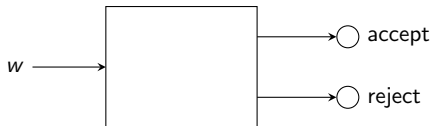
# Intuition

Are these two definitions meaningfully different?

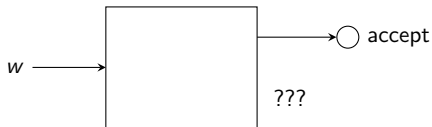
# Intuition

Are these two definitions meaningfully different? Yes!

(Turing-)decidable:



Turing-recognizable

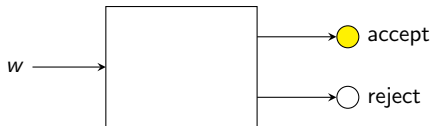


# Intuition

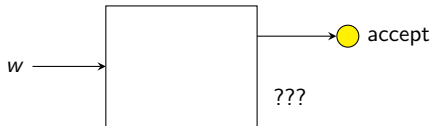
Are these two definitions meaningfully different? Yes!

Case 1:  $w \in L$

(Turing-)decidable:



Turing-recognizable

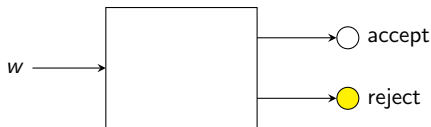


# Intuition

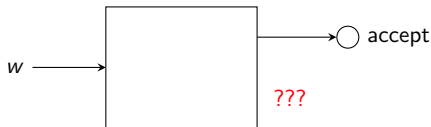
Are these two definitions meaningfully different? Yes!

Case 2:  $w \notin L$

(Turing-)decidable:



Turing-recognizable



# Connection Turing-recognizable and Turing-decidable (1)

**Reminder:** For language  $L$ , we write  $\bar{L}$  to denote its complement.

**Theorem (Decidable vs. Turing-recognizable)**

*A language  $L$  is decidable iff both  $L$  and  $\bar{L}$  are Turing-recognizable.*

**Proof.**

$(\Rightarrow)$ : obvious (**Why?**)

...

## Connection Turing-recognizable and Turing-decidable (2)

Proof (continued).

( $\Leftarrow$ ): Let  $M_L$  be a DTM that recognizes  $L$ ,  
and let  $M_{\bar{L}}$  be a DTM that recognizes  $\bar{L}$ .

The following algorithm decides  $L$ :

On a given input word  $w$  proceed as follows:

FOR  $s := 1, 2, 3, \dots$ :

IF  $M_L$  stops on  $w$  in  $s$  steps in the accept state:  
ACCEPT

IF  $M_{\bar{L}}$  stops on  $w$  in  $s$  steps in the accept state:  
REJECT



## Connection Turing-recognizable and Turing-decidable (2)

Proof (continued).

( $\Leftarrow$ ): Let  $M_L$  be a DTM that recognizes  $L$ ,  
and let  $M_{\bar{L}}$  be a DTM that recognizes  $\bar{L}$ .

The following algorithm decides  $L$ :

On a given input word  $w$  proceed as follows:

FOR  $s := 1, 2, 3, \dots$ :

IF  $M_L$  stops on  $w$  in  $s$  steps in the accept state:  
ACCEPT

IF  $M_{\bar{L}}$  stops on  $w$  in  $s$  steps in the accept state:  
REJECT



Why don't we first entirely simulate  $M_L$  on the input  
and only afterwards  $M_{\bar{L}}$ ?

## Example: Decidable $\neq$ Known Algorithm

Decidability of  $L$  does not mean we know **how** to decide it:

- $L = \{n \in \mathbb{N} \mid \text{there are } n \text{ consecutive } 7\text{s}$   
in the decimal representation of  $\pi\}$ .

## Example: Decidable $\neq$ Known Algorithm

Decidability of  $L$  does not mean we know **how** to decide it:

- $L = \{n \in \mathbb{N} \mid \text{there are } n \text{ consecutive 7s}$   
in the decimal representation of  $\pi\}$ .
- $L$  is decidable.

## Example: Decidable $\neq$ Known Algorithm

Decidability of  $L$  does not mean we know **how** to decide it:

- $L = \{n \in \mathbb{N} \mid \text{there are } n \text{ consecutive 7s}$   
in the decimal representation of  $\pi\}$ .
- $L$  is decidable.
- There are either 7-sequences of arbitrary length in  $\pi$  (case 1)  
or there is a maximal number  $n_0$  of consecutive 7s (case 2).

## Example: Decidable $\neq$ Known Algorithm

Decidability of  $L$  does not mean we know **how** to decide it:

- $L = \{n \in \mathbb{N} \mid \text{there are } n \text{ consecutive } 7\text{s}$   
in the decimal representation of  $\pi\}$ .
- $L$  is decidable.
- There are either 7-sequences of arbitrary length in  $\pi$  (case 1)  
or there is a maximal number  $n_0$  of consecutive 7s (case 2).
  - Case 1: accept for all  $n$

## Example: Decidable $\neq$ Known Algorithm

Decidability of  $L$  does not mean we know **how** to decide it:

- $L = \{n \in \mathbb{N} \mid \text{there are } n \text{ consecutive 7s in the decimal representation of } \pi\}$ .
- $L$  is decidable.
- There are either 7-sequences of arbitrary length in  $\pi$  (case 1) or there is a maximal number  $n_0$  of consecutive 7s (case 2).
  - Case 1: accept for all  $n$
  - Case 2: accept if  $n \leq n_0$ , otherwise reject

## Example: Decidable $\neq$ Known Algorithm

Decidability of  $L$  does not mean we know **how** to decide it:

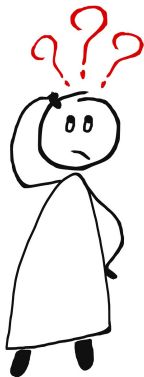
- $L = \{n \in \mathbb{N} \mid \text{there are } n \text{ consecutive 7s in the decimal representation of } \pi\}$ .
- $L$  is decidable.
- There are either 7-sequences of arbitrary length in  $\pi$  (case 1) or there is a maximal number  $n_0$  of consecutive 7s (case 2).
  - Case 1: accept for all  $n$
  - Case 2: accept if  $n \leq n_0$ , otherwise reject
- In both cases, we can decide the language.

## Example: Decidable $\neq$ Known Algorithm

Decidability of  $L$  does not mean we know **how** to decide it:

- $L = \{n \in \mathbb{N} \mid \text{there are } n \text{ consecutive } 7\text{s}$   
in the decimal representation of  $\pi\}$ .
- $L$  is decidable.
- There are either 7-sequences of arbitrary length in  $\pi$  (case 1)  
or there is a maximal number  $n_0$  of consecutive 7s (case 2).
  - Case 1: accept for all  $n$
  - Case 2: accept if  $n \leq n_0$ , otherwise reject
- In both cases, we can decide the language.
- We just do not know what is the correct version  
(and what is  $n_0$  in case 2).

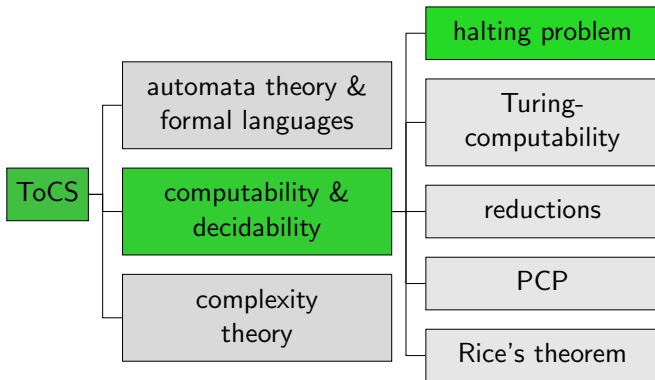
# Questions



Questions?

# The Halting Problem $H$

# Content of the Course



## Reminder: Encodings of Turing Machines

- We have seen how every deterministic Turing machine with input alphabet  $\{0, 1\}$  can be encoded as a word over  $\{0, 1\}$ .  
Can there be several words that encode the same DTM?

## Reminder: Encodings of Turing Machines

- We have seen how every deterministic Turing machine with input alphabet  $\{0, 1\}$  can be encoded as a word over  $\{0, 1\}$ .  
Can there be several words that encode the same DTM?
- Not every word over  $\{0, 1\}$  corresponds to such an encoding.

## Reminder: Encodings of Turing Machines

- We have seen how every deterministic Turing machine with input alphabet  $\{0, 1\}$  can be encoded as a word over  $\{0, 1\}$ .  
Can there be several words that encode the same DTM?
- Not every word over  $\{0, 1\}$  corresponds to such an encoding.
- To define for every  $w \in \{0, 1\}^*$  a corresponding TM, we use an arbitrary fixed DTM  $\hat{M}$  and define

$$M_w = \begin{cases} M' & \text{if } w \text{ is the encoding of some DTM } M' \\ \hat{M} & \text{otherwise} \end{cases}$$

## Reminder: Encodings of Turing Machines

- We have seen how every deterministic Turing machine with input alphabet  $\{0, 1\}$  can be encoded as a word over  $\{0, 1\}$ .  
Can there be several words that encode the same DTM?
- Not every word over  $\{0, 1\}$  corresponds to such an encoding.
- To define for every  $w \in \{0, 1\}^*$  a corresponding TM, we use an arbitrary fixed DTM  $\hat{M}$  and define

$$M_w = \begin{cases} M' & \text{if } w \text{ is the encoding of some DTM } M' \\ \hat{M} & \text{otherwise} \end{cases}$$

- $M_w =$  “Turing machine encoded by  $w$ ”

# Halting Problem

## Definition (Halting Problem)

The **halting problem** is the language

$$H = \{w\#x \in \{0, 1, \#\}^* \mid w, x \in \{0, 1\}^*, \\ M_w \text{ started on } x \text{ terminates}\}$$

“Does the computation of the TM encoded by  $w$  halt on input  $x$ ?”

“Does a given piece of code terminate on a given input?”

# The Halting Problem is Turing-recognizable

## Theorem

*The halting problem  $H$  is Turing-recognizable.*

The following Turing machine  $U$  recognizes language  $H$ :

On input  $w\#x$ :

- 1 If the input contains more than one  $\#$  then reject.
- 2 Simulate  $M_w$  (the TM encoded by  $w$ ) on input  $x$ .
- 3 If  $M_w$  halts, accept.

# The Halting Problem is Turing-recognizable

## Theorem

*The halting problem  $H$  is Turing-recognizable.*

The following Turing machine  $U$  recognizes language  $H$ :

On input  $w\#x$ :

- 1 If the input contains more than one  $\#$  then reject.
- 2 Simulate  $M_w$  (the TM encoded by  $w$ ) on input  $x$ .
- 3 If  $M_w$  halts, accept.

What does  $U$  do if  $M_w$  does not halt on the input?

# The Halting Problem is Turing-recognizable

## Theorem

*The halting problem  $H$  is Turing-recognizable.*

The following Turing machine  $U$  recognizes language  $H$ :

On input  $w\#x$ :

- 1 If the input contains more than one  $\#$  then reject.
- 2 Simulate  $M_w$  (the TM encoded by  $w$ ) on input  $x$ .
- 3 If  $M_w$  halts, accept.

What does  $U$  do if  $M_w$  does not halt on the input?

$U$  is an example of a so-called *universal Turing machine* which can simulate any other Turing machine from the description of that machine.

# Questions



Questions?

# $H$ is Undecidable

# Undecidability

- If some language or problem is **not Turing-decidable** then we call it **undecidable**.

# Undecidability

- If some language or problem is **not Turing-decidable** then we call it **undecidable**.
- Intuitively, this means that for this problem there is no algorithm that is correct and terminates on all inputs.

# Undecidability

- If some language or problem is **not Turing-decidable** then we call it **undecidable**.
- Intuitively, this means that for this problem there is no algorithm that is correct and terminates on all inputs.
- To establish the undeciability of the halting problem, we will consider a situation where we run a Turing machine/algorithm on its own encoding/source code.

# Undecidability

- If some language or problem is **not Turing-decidable** then we call it **undecidable**.
- Intuitively, this means that for this problem there is no algorithm that is correct and terminates on all inputs.
- To establish the undeciability of the halting problem, we will consider a situation where we run a Turing machine/algorithm on its own encoding/source code.
- We have seen something similar in the very first lecture. . .

# Uncomputable Problems?

Consider functions whose inputs are strings:

```
def program_returns_true_on_input(prog_code, input_str):  
    ...  
    # returns True if prog_code run on input_str returns True  
    # returns False if not  
  
def weird_program(prog_code):  
    if program_returns_true_on_input(prog_code, prog_code):  
        return False  
    else:  
        return True
```



What is the return value of `weird_program`  
if we run it on its own source code?

## Solution

- We can make a case distinction:

# Solution

- We can make a case distinction:
  - Case 1: `weird_program` returns True on its own source.  
Then `weird_program` returns False on its own source code.

# Solution

- We can make a case distinction:
  - Case 1: `weird_program` returns True on its own source.  
Then `weird_program` returns False on its own source code.
  - Case 2: `weird_program` returns False on its own source.  
Then `weird_program` returns True on its own source code.

# Solution

- We can make a case distinction:
  - Case 1: `weird_program` returns True on its own source.  
Then `weird_program` returns False on its own source code.
  - Case 2: `weird_program` returns False on its own source.  
Then `weird_program` returns True on its own source code.
- Contradiction in all cases, so `weird_program` cannot exist.

# Solution

- We can make a case distinction:
  - Case 1: `weird_program` returns True on its own source.  
Then `weird_program` returns False on its own source code.
  - Case 2: `weird_program` returns False on its own source.  
Then `weird_program` returns True on its own source code.
- Contradiction in all cases, so `weird_program` cannot exist.
- From the source we see that this can only be because subroutine `program_returns_true_on_input` cannot exist.

# Solution

- We can make a case distinction:
  - Case 1: `weird_program` returns True on its own source.  
Then `weird_program` returns False on its own source code.
  - Case 2: `weird_program` returns False on its own source.  
Then `weird_program` returns True on its own source code.
- Contradiction in all cases, so `weird_program` cannot exist.
- From the source we see that this can only be because subroutine `program_returns_true_on_input` cannot exist.
- Overall, we have proven that there cannot be a program with the behaviour described by the comments.

# Solution

- We can make a case distinction:
  - Case 1: `weird_program` returns True on its own source.  
Then `weird_program` returns False on its own source code.
  - Case 2: `weird_program` returns False on its own source.  
Then `weird_program` returns True on its own source code.
- Contradiction in all cases, so `weird_program` cannot exist.
- From the source we see that this can only be because subroutine `program_returns_true_on_input` cannot exist.
- Overall, we have proven that there cannot be a program with the behaviour described by the comments.
- For the undecidability of the halting problem, we will use an analogous argument, only with Turing machines instead of code and termination instead of return values.

# Undecidability of the Halting Problem (1)

## Theorem (Undecidability of the Halting Problem)

*The halting problem  $H$  is undecidable.*

# Undecidability of the Halting Problem (1)

## Theorem (Undecidability of the Halting Problem)

*The halting problem  $H$  is undecidable.*

## Proof.

**Proof by contradiction:** we assume that the halting problem  $H$  was decidable and derive a contradiction.

# Undecidability of the Halting Problem (1)

## Theorem (Undecidability of the Halting Problem)

*The halting problem  $H$  is undecidable.*

## Proof.

**Proof by contradiction:** we assume that the halting problem  $H$  was decidable and derive a contradiction.

So assume  $H$  is decidable and let  $D$  be a DTM that decides it. ...

## Undecidability of the Halting Problem (2)

### Proof (continued).

Construct the following new machine  $M$  that takes a word  $x \in \{0, 1\}^*$  as input:

- 1 Execute  $D$  on the input  $x\#x$ .
- 2 If it rejects: accept.
- 3 Otherwise: enter an endless loop.

## Undecidability of the Halting Problem (2)

### Proof (continued).

Construct the following new machine  $M$  that takes a word  $x \in \{0, 1\}^*$  as input:

- 1 Execute  $D$  on the input  $x\#x$ .
- 2 If it rejects: accept.
- 3 Otherwise: enter an endless loop.

Let  $w$  be the encoding of  $M$ . **How will  $M$  behave on input  $w$ ?**

## Undecidability of the Halting Problem (2)

### Proof (continued).

Construct the following new machine  $M$  that takes a word  $x \in \{0, 1\}^*$  as input:

- 1 Execute  $D$  on the input  $x\#x$ .
- 2 If it rejects: accept.
- 3 Otherwise: enter an endless loop.

Let  $w$  be the encoding of  $M$ . **How will  $M$  behave on input  $w$ ?**

$M$  run on  $w$  stops

iff  $D$  run on  $w\#w$  rejects

## Undecidability of the Halting Problem (2)

### Proof (continued).

Construct the following new machine  $M$  that takes a word  $x \in \{0, 1\}^*$  as input:

- 1 Execute  $D$  on the input  $x\#x$ .
- 2 If it rejects: accept.
- 3 Otherwise: enter an endless loop.

Let  $w$  be the encoding of  $M$ . **How will  $M$  behave on input  $w$ ?**

$M$  run on  $w$  stops

iff  $D$  run on  $w\#w$  rejects

iff  $w\#w \notin H$

## Undecidability of the Halting Problem (2)

### Proof (continued).

Construct the following new machine  $M$  that takes a word  $x \in \{0, 1\}^*$  as input:

- 1 Execute  $D$  on the input  $x\#x$ .
- 2 If it rejects: accept.
- 3 Otherwise: enter an endless loop.

Let  $w$  be the encoding of  $M$ . **How will  $M$  behave on input  $w$ ?**

$M$  run on  $w$  stops

iff  $D$  run on  $w\#w$  rejects

iff  $w\#w \notin H$

iff  $M$  run on  $w$  does not stop (remember that  $w$  encodes  $M$ )

**Contradiction!** DTM  $M$  cannot exist.

## Undecidability of the Halting Problem (2)

### Proof (continued).

Construct the following new machine  $M$  that takes a word  $x \in \{0, 1\}^*$  as input:

- 1 Execute  $D$  on the input  $x\#x$ .
- 2 If it rejects: accept.
- 3 Otherwise: enter an endless loop.

Let  $w$  be the encoding of  $M$ . **How will  $M$  behave on input  $w$ ?**

$M$  run on  $w$  stops

iff  $D$  run on  $w\#w$  rejects

iff  $w\#w \notin H$

iff  $M$  run on  $w$  does not stop (remember that  $w$  encodes  $M$ )

**Contradiction!** DTM  $M$  cannot exist.

$\Rightarrow$  DTM  $D$  cannot exist, thus  $H$  is not decidable. □

# A Language that is not Turing-recognizable

We have the following results:

- A language  $L$  is decidable iff both  $L$  and  $\bar{L}$  are Turing-recognizable.
- The halting problem  $H$  is Turing-recognizable but not decidable.

## Corollary

*The complement  $\bar{H}$  of the halting problem  $H$  is **not Turing-recognizable**.*

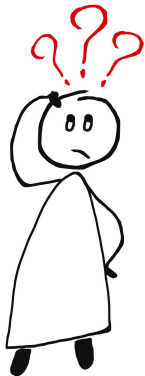
## Exercises

- True or false? There is a grammar that generates  $H$ .
- True or false? Not all languages are of type 0.

Justify your answers.



# Questions



Questions?

# Reprise: Type-0 Languages

## Back to Chapter B13: Closure Properties

	Intersection	Union	Complement	Concatenation	Star
Type 3	Yes	Yes	Yes	Yes	Yes
Type 2	No	Yes	No	Yes	Yes
Type 1	Yes <sup>(2)</sup>	Yes <sup>(1)</sup>	Yes <sup>(2)</sup>	Yes <sup>(1)</sup>	Yes <sup>(1)</sup>
Type 0	Yes <sup>(2)</sup>	Yes <sup>(1)</sup>	No <sup>(3)</sup>	Yes <sup>(1)</sup>	Yes <sup>(1)</sup>

Proofs?

(1) proof via grammars, similar to context-free cases

(2) without proof

(3) proof in later chapters (part C)

# Back to Chapter B13: Decidability

	Word problem	Emptiness problem	Equivalence problem	Intersection problem
Type 3	Yes	Yes	Yes	Yes
Type 2	Yes	Yes	No	No
Type 1	Yes <sup>(1)</sup>	No <sup>(3)</sup>	No <sup>(2)</sup>	No <sup>(2)</sup>
Type 0	No <sup>(4)</sup>	No <sup>(4)</sup>	No <sup>(4)</sup>	No <sup>(4)</sup>

Proofs?

- (1) same argument we used for context-free languages
- (2) because already undecidable for context-free languages
- (3) without proof
- (4) proofs in later chapters (part C)

# Answers to Old Questions

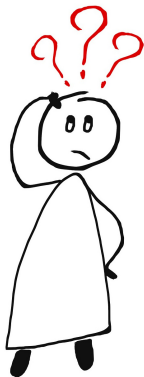
## Closure properties:

- $H$  is Turing-recognizable (and thus type 0) but not decidable.
- ↪  $\bar{H}$  is **not** Turing-recognizable, thus **not** type 0.
- ↪ Type-0 languages are **not** closed under complement.

## Decidability:

- $H$  is type 0 but not decidable.
- ↪ **word problem** for type-0 languages not decidable
- ↪ emptiness, equivalence, intersection problem: **later in exercises**  
(We are still missing some important results for this.)

# Questions



Questions?

# Summary

# Summary

- A language  $L$  is **decidable** iff both  $L$  and  $\bar{L}$  are **Turing-recognizable**.
- The **halting problem** is the language

$$H = \{w\#x \in \{0, 1, \#\}^* \mid w, x \in \{0, 1\}^*, \\ M_w \text{ started on } x \text{ terminates}\}$$

- The halting problem is **Turing-recognizable** but **undecidable**.
- The complement language  $\bar{H}$  is an example of a language that is **not even Turing-recognizable**.