

# Theory of Computer Science

## C3. Turing-Computability

Gabriele Röger

University of Basel

April 21, 2021

# Theory of Computer Science

April 21, 2021 — C3. Turing-Computability

## C3.1 Turing-Computable Functions

## C3.2 Decidability vs. Computability

## C3.3 Summary

# C3.1 Turing-Computable Functions

# Hello World (slido)

```
def hello_world(name):  
    return "Hello " + name + "!"
```

When calling  
`hello_world("Florian")`  
we get the result `"Hello Florian!"`.

How could a Turing machine output a string as the result of a computation?



# Church-Turing Thesis Revisited

## Church-Turing Thesis

All functions that can be **computed in the intuitive sense** can be computed by a **Turing machine**.

- ▶ Talks about **arbitrary** functions that can be computed in the intuitive sense.
- ▶ So far, we have only considered **recognizability** and **decidability**: Is a word in a language, **yes or no**?
- ▶ We now will consider function values beyond yes or no (accept or reject).
- ▶  $\Rightarrow$  **consider the tape content** when the TM accepted.

# Computation

In the following we investigate

**models of computation** for **partial functions**  $f : \mathbb{N}_0^k \rightarrow_p \mathbb{N}_0$ .

- ▶ no real limitation: arbitrary information  
can be encoded as numbers

**German:** Berechnungsmodelle

# Reminder: Configurations and Computation Steps

## How do Turing Machines Work?

- ▶ **configuration:**  $\langle \alpha, q, \beta \rangle$  with  $\alpha \in \Gamma^*$ ,  $q \in Q$ ,  $\beta \in \Gamma^+$
- ▶ **one computation step:**  $c \vdash c'$  if one computation step can turn configuration  $c$  into configuration  $c'$
- ▶ **multiple computation steps:**  $c \vdash^* c'$  if 0 or more computation steps can turn configuration  $c$  into configuration  $c'$   
( $c = c_0 \vdash c_1 \vdash c_2 \vdash \dots \vdash c_{n-1} \vdash c_n = c'$ ,  $n \geq 0$ )

(Definition of  $\vdash$ , i.e., how a computation step changes the configuration, is not repeated here.  $\rightsquigarrow$  [Chapter B9](#))

# Computation of Functions?

## How can a DTM compute a function?

- ▶ “Input”  $x$  is the initial tape content
- ▶ “Output”  $f(x)$  is the tape content (ignoring blanks at the left and right) when reaching the accept state
- ▶ If the TM stops in the reject state or does not stop for the given input,  $f(x)$  is undefined for this input.

## Which kinds of functions can be computed this way?

- ▶ directly, only functions on **words**:  $f : \Sigma^* \rightarrow_p \Sigma^*$
- ▶ interpretation as functions on **numbers**  $f : \mathbb{N}_0^k \rightarrow_p \mathbb{N}_0$ :  
encode numbers as words



# Turing Machines: Computed Function

## Definition (Function Computed by a Turing Machine)

A DTM  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}} \rangle$  **computes** the (partial) function  $f : \Sigma^* \rightarrow_p \Sigma^*$  for which for all  $x, y \in \Sigma^*$ :

$$f(x) = y \text{ iff } \langle \varepsilon, q_0, x \rangle \vdash^* \langle \varepsilon, q_{\text{accept}}, y \square \dots \square \rangle.$$

(special case: initial configuration  $\langle \varepsilon, q_0, \square \rangle$  if  $x = \varepsilon$ )

**German:** DTM berechnet  $f$

- ▶ What happens if the computation does not reach  $q_{\text{accept}}$ ?
- ▶ What happens if symbols from  $\Gamma \setminus \Sigma$  (e. g.,  $\square$ ) occur in  $y$ ?
- ▶ What happens if the read-write head is not at the first tape cell when accepting?
- ▶ Is  $f$  uniquely defined by this definition? Why?

# Turing-Computable Functions on Words

Definition (Turing-Computable,  $f : \Sigma^* \rightarrow_p \Sigma^*$ )

A (partial) function  $f : \Sigma^* \rightarrow_p \Sigma^*$  is called **Turing-computable** if a DTM that computes  $f$  exists.

German: Turing-berechenbar

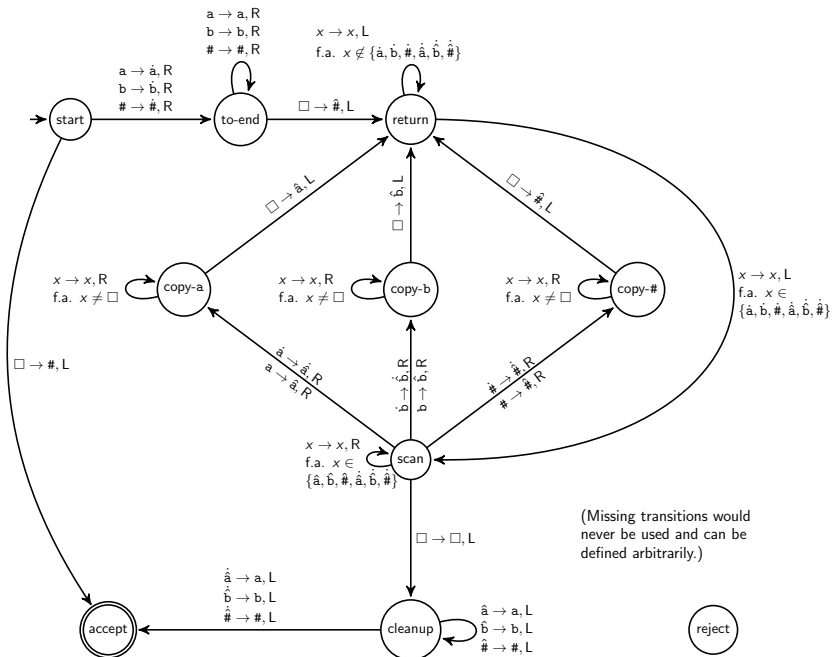
## Example: Turing-Computable Functions on Words

### Example

Let  $\Sigma = \{a, b, \#\}$ .

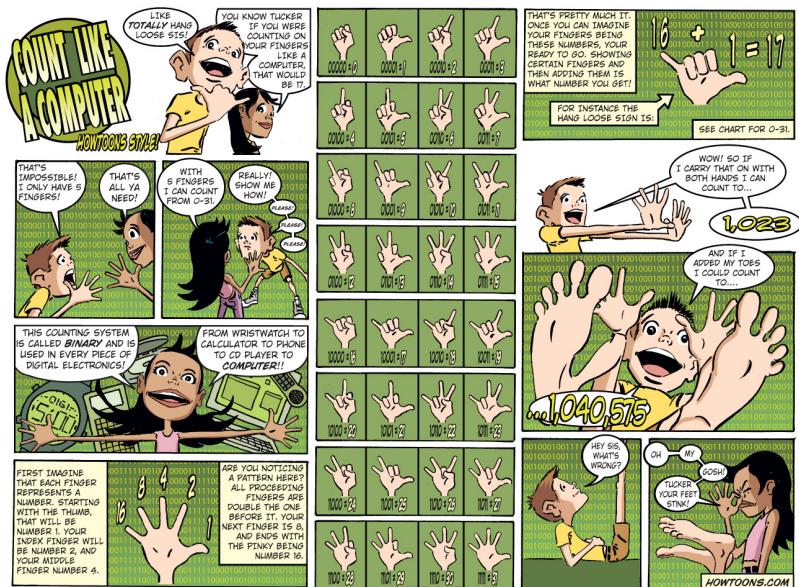
The function  $f : \Sigma^* \rightarrow_p \Sigma^*$  with  $f(w) = w\#w$  for all  $w \in \Sigma^*$  is Turing-computable.

Idea:  $\rightsquigarrow$  blackboard



# Turing-Computable Numerical Functions

- ▶ We now transfer the concept to partial functions  
 $f : \mathbb{N}_0^k \rightarrow_p \mathbb{N}_0$ .
- ▶ Idea:
  - ▶ To represent a number as a word, we use its binary representation (= a word over  $\{0, 1\}$ ).
  - ▶ To represent tuples of numbers, we separate the binary representations with symbol #.
- ▶ For example:  $(5, 2, 3)$  becomes 101#10#11



# Encoding Numbers as Words

## Definition (Encoded Function)

Let  $f : \mathbb{N}_0^k \rightarrow_p \mathbb{N}_0$  be a (partial) function.

The **encoded function**  $f^{\text{code}}$  of  $f$  is the partial function  $f^{\text{code}} : \Sigma^* \rightarrow_p \Sigma^*$  with  $\Sigma = \{0, 1, \#\}$  and  $f^{\text{code}}(w) = w'$  iff

- ▶ there are  $n_1, \dots, n_k, n' \in \mathbb{N}_0$  such that
- ▶  $f(n_1, \dots, n_k) = n'$ ,
- ▶  $w = \text{bin}(n_1)\# \dots \# \text{bin}(n_k)$  and
- ▶  $w' = \text{bin}(n')$ .

Here  $\text{bin} : \mathbb{N}_0 \rightarrow \{0, 1\}^*$  is the binary encoding (e. g.,  $\text{bin}(5) = 101$ ).

**German:** kodierte Funktion

**Example:**  $f(5, 2, 3) = 4$  corresponds to  $f^{\text{code}}(101\#10\#11) = 100$ .

# Turing-Computable Numerical Functions

Definition (Turing-Computable,  $f : \mathbb{N}_0^k \rightarrow_p \mathbb{N}_0$ )

A (partial) function  $f : \mathbb{N}_0^k \rightarrow_p \mathbb{N}_0$  is called **Turing-computable** if a DTM that computes  $f^{\text{code}}$  exists.

German: Turing-berechenbar



## Exercise (slido)

The addition of natural numbers  $+ : \mathbb{N}_0^2 \rightarrow \mathbb{N}_0$  is Turing-computable. You have a TM  $M$  that computes  $+^{\text{code}}$ .

You want to use  $M$  to compute the sum  $3 + 2$ .

What is your input to  $M$ ?

## Example: Turing-Computable Numerical Function

### Example

The following numerical functions are Turing-computable:

- ▶  $\text{succ} : \mathbb{N}_0 \rightarrow_{\text{p}} \mathbb{N}_0$  with  $\text{succ}(n) := n + 1$
- ▶  $\text{pred}_1 : \mathbb{N}_0 \rightarrow_{\text{p}} \mathbb{N}_0$  with  $\text{pred}_1(n) := \begin{cases} n - 1 & \text{if } n \geq 1 \\ 0 & \text{if } n = 0 \end{cases}$
- ▶  $\text{pred}_2 : \mathbb{N}_0 \rightarrow_{\text{p}} \mathbb{N}_0$  with  $\text{pred}_2(n) := \begin{cases} n - 1 & \text{if } n \geq 1 \\ \text{undefined} & \text{if } n = 0 \end{cases}$

How does incrementing and decrementing binary numbers work?

# Successor Function

The Turing machine for *succ* works as follows:

(Details of marking the first tape position omitted)

- ❶ Check that the input is a valid binary number:
  - ▶ If the input is not a single symbol 0 but starts with a 0, reject.
  - ▶ If the input contains symbol #, reject.
- ❷ Move the head onto the last symbol of the input.
- ❸ While you read a 1 and you are not at the first tape position, replace it with a 0 and move the head one step to the left.
- ❹ Depending on why the loop in stage 3 terminated:
  - ▶ If you read a 0, replace it with a 1, move the head to the left end of the tape and accept.
  - ▶ If you read a 1 at the first tape position, move every non-blank symbol on the tape one position to the right, write a 1 in the first tape position and accept.

# Predecessor Function

The Turing machine for  $pred_1$  works as follows:

(Details of marking the first tape position omitted)

- ① Check that the input is a valid binary number (as for *succ*).
- ② If the (entire) input is 0 or 1, write a 0 and accept.
- ③ Move the head onto the last symbol of the input.
- ④ While you read symbol 0 replace it with 1 and move left.
- ⑤ Replace the 1 with a 0.
- ⑥ If you are on the first tape cell, eliminate the trailing 0 (moving all other non-blank symbols one position to the left).
- ⑦ Move the head to the first position and accept.

What do you have to change to get a TM for  $pred_2$ ?

# More Turing-Computable Numerical Functions

## Example

The following numerical functions are Turing-computable:

- ▶  $add : \mathbb{N}_0^2 \rightarrow_p \mathbb{N}_0$  with  $add(n_1, n_2) := n_1 + n_2$
- ▶  $sub : \mathbb{N}_0^2 \rightarrow_p \mathbb{N}_0$  with  $sub(n_1, n_2) := \max\{n_1 - n_2, 0\}$
- ▶  $mul : \mathbb{N}_0^2 \rightarrow_p \mathbb{N}_0$  with  $mul(n_1, n_2) := n_1 \cdot n_2$
- ▶  $div : \mathbb{N}_0^2 \rightarrow_p \mathbb{N}_0$  with  $div(n_1, n_2) := \begin{cases} \left\lceil \frac{n_1}{n_2} \right\rceil & \text{if } n_2 \neq 0 \\ \text{undefined} & \text{if } n_2 = 0 \end{cases}$

$\rightsquigarrow$  sketch?

## C3.2 Decidability vs. Computability

# Decidability as Computability

## Theorem

A language  $L \subseteq \Sigma^*$  is *decidable* iff  $\chi_L : \Sigma^* \rightarrow \{0, 1\}$ , the *characteristic function of  $L$* , is computable.

Here, for all  $w \in \Sigma^*$ :

$$\chi_L(w) := \begin{cases} 1 & \text{if } w \in L \\ 0 & \text{if } w \notin L \end{cases}$$

## Proof sketch.

“ $\Rightarrow$ ” Let  $M$  be a DTM for  $L$ . Construct a DTM  $M'$  that simulates  $M$  on the input. If  $M$  accepts,  $M'$  writes a 1 on the tape. If  $M$  rejects,  $M'$  writes a 0 on the tape. Afterwards  $M'$  accepts.

“ $\Leftarrow$ ” Let  $C$  be a DTM that computes  $\chi_L$ . Construct a DTM  $C'$  that simulates  $C$  on the input. If the output of  $C$  is 1 then  $C'$  accepts, otherwise it rejects.

# Turing-recognizable Languages and Computability

## Theorem

A language  $L \subseteq \Sigma^*$  is **Turing-recognizable** if the following function  $\chi'_L : \Sigma^* \rightarrow_p \{0, 1\}$  is computable.

Here, for all  $w \in \Sigma^*$ :

$$\chi'_L(w) = \begin{cases} 1 & \text{if } w \in L \\ \text{undefined} & \text{if } w \notin L \end{cases}$$

## Proof sketch.

“ $\Rightarrow$ ” Let  $M$  be a DTM for  $L$ . Construct a DTM  $M'$  that simulates  $M$  on the input. If  $M$  accepts,  $M'$  writes a 1 on the tape and accepts. Otherwise it enters an infinite loop.

“ $\Leftarrow$ ” Let  $C$  be a DTM that computes  $\chi'_L$ . Construct a DTM  $C'$  that simulates  $C$  on the input. If  $C$  accepts with output 1 then  $C'$  accepts, otherwise it enters an infinite loop.



## C3.3 Summary

# Summary

- ▶ **Turing-computable** function  $f : \Sigma^* \rightarrow_p \Sigma^*$ :  
there is a DTM that transforms every input  $w \in \Sigma^*$   
into the output  $f(w)$  (undefined if DTM does not stop  
or stops in invalid configuration)
- ▶ **Turing-computable** function  $f : \mathbb{N}_0^k \rightarrow_p \mathbb{N}_0$ :  
ditto; numbers encoded in binary and separated by #