hsrm Hochschule RheinMain

Programming Language Foundations

02 Computability

Prof. Dr. David Sabel

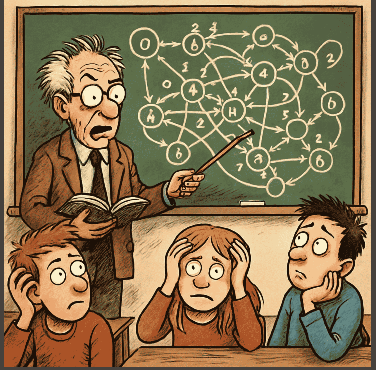
Winter term 2025/26 Last update: December 17, 2025

Image credits: All images are self-drawn or generated with AI from OpenAI at https://ki.hs-rm.de

Contents

- Intuitively computable functions
- Turing machines and Turing computability
- The Church-Turing thesis
- Turing completeness

INTUITIVE COMPUTABILITY



Intuitive Computability

- what is computable by a computer program compute, what not?
- do you have an intuition?
- we use the following definition:

Definition (Intuitive Computability)

A (partial) function $f : \mathbb{N}_0^k \to \mathbb{N}_0$ is computable iff there exists an algorithm (a program in a modern programming language) that computes f, i.e.
on input $(n_1, \ldots, n_k) \in \mathbb{N}_0^k$

- if $f(n_1, \ldots, n_k)$ is defined, then the program terminates after a finite number of steps and returns $f(n_1, \ldots, n_k)$ as result.
- if $f(n_1, \ldots, n_k)$ is undefined, then the program runs forever.

Intuitive Computability: Examples

Computing the sum of two numbers

Let $f : (\mathbb{N}_0 \times \mathbb{N}_0) \to \mathbb{N}_0$ be the function $f(x, y) = x + y$.

Is the function f computable?

Yes, the algorithm with inputs $n_1, n_2 \in \mathbb{N}_0$ and program code:

```
$result := n_1 + n_2;$
return $result;$
```

computes f.

Intuitive Computability: Examples

Always undefined function

Let $f : \mathbb{N}_0 \to \mathbb{N}_0$ be the partial function that is undefined for all inputs (often written as $f(x) = \bot$).

Is f computable?

Yes, the algorithm with input $n \in \mathbb{N}_0$ and program code

```
while true {skip};
```

computes f.

Intuitive Computability: Examples

Prefix of π

Let $f(n) = \begin{cases} 1, & \text{if } n \text{ is a prefix of the digits of the decimal representation of } \pi \\ 0, & \text{otherwise} \end{cases}$

For example, $f(31) = 1 \quad f(314) = 1 \quad f(2) = 0 \quad f(315) = 0.$

Is f computable?

Yes, the function f is computable:

- There are algorithms that can compute the first x digits of π
- Choose x large enough to capture the digits of n
- Compare the digits with the digits of n and return 1 if all match, and 0 otherwise

Intuitive Computability: Examples

Substring of π

Let $f(n) = \begin{cases} 1, & \text{if } n \text{ is a substring of the digits of the decimal representation of } \pi \\ 0, & \text{otherwise} \end{cases}$

Is f computable?

- There is no known algorithm to check the condition
- If we would know, that π contains every sequence of numbers (an open problem), then f is trivially computable (always return 1)

Intuitive Computability: Examples

Specific substring of π

Let $f(n) = \begin{cases} 1, & \text{if the digits of the decimal representation of } \pi \\ & \text{contains the substring } 3^m \text{ for some number } m \geq n \\ 0, & \text{otherwise} \end{cases}$

Is f computable?

Yes:

1. If π contains all strings 3^m, then f is the constant function 1, which is computable
2. If there is a bound M such that π contains 3^M, but π does not contain 3^x with $x > M$, then f can be computed:
   Check if $n \leq M$ holds. If yes, return 1, else return 0.

One of both algorithms computes f, and thus f is computable.

It is **not relevant**, that we do not know which one is the correct algorithm.

Intuitive Computability: Examples

Function depending on open question

Let f be $f(n) = \begin{cases} 1, & \text{if } P = NP \\ 0, & \text{if } P \neq NP \end{cases}$

Is f computable?

Yes, because either $P = NP$ holds (then $f(n) = 1$ for all n), or $P \neq NP$ holds (then $f(n) = 0$ for all n).

Once again, we cannot say which algorithm is correct, but we are certain that there is an algorithm that computes f.

Intuitive Computability: Examples

A lot functions

Let f^r be the function and r be a real number

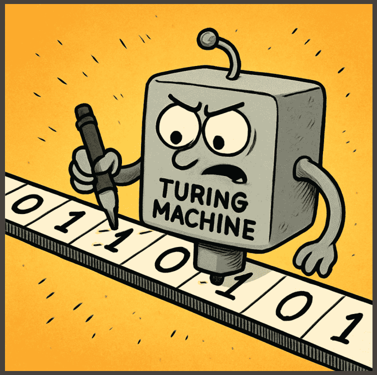
$$f^r(n) = \begin{cases} 1, & \text{if } n \text{ is prefix of the digits of the decimal representation of } r \\ 0, & \text{otherwise} \end{cases}$$

Are all functions f^r computable? (remember that for $r = \pi$, f is computable)

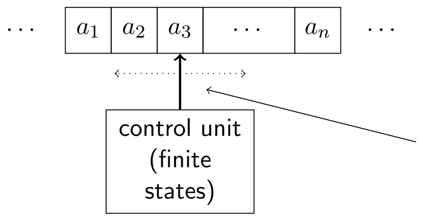
No, the argument is:

- $f^{r_1} \neq f^{r_2}$ for $r_1 \neq r_2$
- $|\mathbb{R}|$ different algorithms are required
- the set of algorithms is countable
- the real numbers are not countable

TURING COMPUTABILTY



Turing Machines: Informally



tape consisting of cells
(unbounded to the left and to the right)

read-/write-head
(may move to the left or to the right)

- introduced in 1936 by Alan Turing
- memory is represented by the infinite tape (divided into cells)
- in one step: TM reads the current cell, replaces the symbol, and may move the head by one cell

Turing Machine, formally

Definition

A Turing machine (TM) is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ where

- Q is a finite, non-empty set of states,
- Σ is a finite set of symbols, the input alphabet,
- $\Gamma \supset \Sigma$ is a finite set of symbols, the tape alphabet,
- δ is the state transition function where in the case of a deterministic Turing machine (DTM), $\delta : (Q \times \Gamma) \to (Q \times \Gamma \times \{L, R, N\})$, and in case of a non-deterministic Turing machine (NTM), $\delta : (Q \times \Gamma) \to \mathcal{P}(Q \times \Gamma \times \{L, R, N\})$,
- $q_0 \in Q$ is the start state,
- $\Box \in \Gamma \setminus \Sigma$ is the blank symbol,
- $F \subseteq Q$ is the set of final states.

For a deterministic Turing machine, an entry $\delta(q, a) = (q', b, x)$ means that in state q, current cell will be b and the read-/write-head moves into direction x (which is left if $x = L$, right if $x = R$ and no move if $x = N$).
For a non-deterministic Turing machine the same holds if $(q', b, x) \in \delta(q, a)$, but it means, that the TM can do this, but it can also do some other state transition in $\delta(q, a)$. It chooses non-deterministically between the choices in $\delta(q, a)$.

Configuration of TMs

Definition

A configuration of a Turing machine is a word $wqw' \in \Gamma^* Q \Gamma^*$

A configuration wqw' means

- the TM is in state q
- the tape content is ww' and infinitely many blank symbols left and right from ww'
- the current head position is on the first symbol of w'.

Initially the TM is in state q_0 and the head is on the first symbol of the input word:

Definition

For input w, the start-configuration of a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ is $q_0 w$.

Transition Relation

Definition (Transition relation on configurations)

For a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$, the relation $\vdash_M$ is defined as follows
(where $\delta(q, a) = (q', c, x)$ in case of an NTM means $(q', c, x) \in \delta(q, a)$):

$$\begin{array}{lll}
wqw' & \not\vdash_M & \text{if } q \in F \text{ (no transition for final states).} \\
b_1 \cdots b_m q a_1 \cdots a_n & \vdash_M b_1 \cdots b_m q' c a_2 \cdots a_n, & \text{if } \delta(q, a_1) = (q', c, N),\ m \geq 0, n \geq 1, q \notin F \\
b_1 \cdots b_m q a_1 \cdots a_n & \vdash_M b_1 \cdots b_{m-1} q' b_m c a_2 \cdots a_n, & \text{if } \delta(q, a_1) = (q', c, L),\ m \geq 1, n \geq 1, q \notin F \\
b_1 \cdots b_m q a_1 \cdots a_n & \vdash_M b_1 \cdots b_m c q' a_2 \cdots a_n, & \text{if } \delta(q, a_1) = (q', c, R),\ m \geq 0, n \geq 2, q \notin F \\
b_1 \cdots b_m q a_1 & \vdash_M b_1 \cdots b_m c q' \Box, & \text{if } \delta(q, a_1) = (q', c, R) \text{ and } m \geq 0, q \notin F \\
q a_1 \cdots a_n & \vdash_M q' \Box c a_2 \cdots a_n, & \text{if } \delta(q, a_1) = (q', c, L) \text{ and } n \geq 1, q \notin F
\end{array}$$

$\vdash_M^i$ is the i-fold application of $\vdash_M$
$\vdash_M^*$ the reflexive-transitive closure of $\vdash_M$
We omit the index M in $\vdash_M$ and write $\vdash$ is M is clear from the context.

Example

DTM $M = (\{q_0, q_1, q_2, q_3\}, \{0,1\}, \{0,1,\Box\}, \delta, q_0, \Box, \{q_3\})$ with

$$\begin{array}{lll}
\delta(q_0,0) = (q_0,0,R) & \delta(q_0,1) = (q_0,1,R) & \delta(q_0,\Box) = (q_1,\Box,L) \\
\delta(q_1,0) = (q_2,1,L) & \delta(q_1,1) = (q_1,0,L) & \delta(q_1,\Box) = (q_3,1,N) \\
\delta(q_2,0) = (q_2,0,L) & \delta(q_2,1) = (q_2,1,L) & \delta(q_2,\Box) = (q_3,\Box,R) \\
\delta(q_3,0) = (q_3,0,N) & \delta(q_3,1) = (q_3,1,N) & \delta(q_3,\Box) = (q_3,\Box,N)
\end{array}$$

- interprets the input as binary number
- In state q_0 it moves the head to the right end and switches to q_1
- In q_1 it adds 1 to the input, including a carryover
- If no more carryover occurs, it switches to q_2
- in q_2 it moves the head to the left end and switches to q_3
- it accepts in q_3

Example execution

$$q_0 0011 \vdash 0 q_0 011 \vdash 00 q_0 11 \vdash 001 q_0 1 \vdash 0011 q_0 \Box$$

$$\vdash 001 q_1 1 \Box \vdash 00 q_1 10 \Box \vdash 0 q_1 000 \Box \vdash q_2 0100 \Box$$

$$\vdash q_2 \Box 0100 \Box \vdash \Box q_3 0100 \Box$$

Turing Computability

Let $bin(n)$ be the binary representation of number $n \in \mathbb{N}_0$.

Definition

Function $f : \mathbb{N}_0^k \to \mathbb{N}_0$ is Turing computable,
if there exists a DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ such that for all $n_1, \ldots, n_k, m \in \mathbb{N}_0$:

$$f(n_1, \ldots, n_k) = m$$

iff

$$q_0\, bin(n_1) \# \ldots \# bin(n_k) \vdash^* \Box \ldots \Box q_f\, bin(m) \Box \ldots \Box \text{ with } q_f \in F.$$

Function $f : \Sigma^* \to \Sigma^*$ is Turing computable,
if there exists a DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ such that for all $u, v \in \Sigma^*$:

$$f(u) = v \text{ iff } q_0 u \vdash^* \Box \ldots \Box q_f v \Box \ldots \Box \text{ with } q_f \in F.$$

If $f(n_1, \ldots, n_k)$ is undefined, we assume that the TM loops.

Examples

- The successor function $f(x) = x + 1$ is Turing computable.

  We defined the corresponding TM in the last example.

- The identity $f(x) = x$ is Turing computable:

  DTM $M = (\{q_0\}, \{0,1,\#\}, \{0,1,\#\Box\}, \delta, q_0, \Box, \{q_0\})$ with $\delta(q_0, a) = (q_0, a, N)$
  for all $a \in \{0,1,\#,\Box\}$, we have $q_0\, bin(n) \vdash^* q_0\, bin(n)$ for all $n \in \mathbb{N}_0$.

- The function $f(x) = \bot$ which is undefined for every input is Turing computable:

  DTM $M = (\{q_0\}, \{0,1,\#\}, \{0,1,\#,\Box\}, \delta, q_0, \Box, \emptyset)$ with $\delta(q_0, a) = (q_0, a, N)$
  loops for every input and never reaches a final state.

Not Turing Computable Functions

- Turing machines and words can be encoded as numbers (called Gödel numbers)
- Let f be a function that gets a number n and
  - is undefined if n is not a valid encoding of a TM M and a word w
  - is 1, if the TM M holds on input w
  - is 0, otherwise
- Let f' be a function that gets a number n and
  - is undefined if n is not a valid encoding of a TM M and a word w
  - is 1, if the TM M holds on input w
  - is undefined, otherwise
- Function f is **not Turing computable**, because the TM that computes f has to solve the halting problem for Turing machines which is undecidable.
- Function f' is **Turing computable**, because f' can be computed by a Turing machine, by simulating M on input w.

CHURCH-TURING-THESIS

Computability

In the 1930s also other notions of computability were invented, e.g.:

- Kurt Geödel and Jacques Herbrand: General recursive functions
- Alonzo Church and Stephen Kleene: λ-definable functions

Remarkable result:

> *All of the formalisms were shown to be equivalent,*
> *i.e. they define the same class of functions.*

Church-Turing Thesis

Church-Turing Thesis

The class of Turing computable functions is identical to the class of intuitively computable functions.

- Thesis cannot be proved, since there is no formal definition of “intuitively computable”.

Turing Completeness

Definition (Turing completeness

A formalism (a programming language, an instruction set of a computer, a rewrite system etc.) is called Turing complete iff it can simulate a Turing machine.

Turing completeness $\implies$ all Turing computable function can be computed

- Turing complete formalism can be replaced in the Church-Turing thesis
- Turing complete formalisms: all modern programming languages, the lambda-definable functions, the general recursive functions, WHILE-programs, GOTO-programs, the RAM-model, etc.
- Convince yourself that your favourite programming language is Turing complete by programming a simulation of Turing machines.

Conclusion

- We recalled Turing machines and Turing computability
- Several other formalisms are Turing-complete
- Church-Turing-Thesis: all these formalisms match the class of intuitively computable functions
- For considering foundational models of programming languages, we have several choices as long as the model is Turing-complete

APPENDIX

- Gödel-numbering of Turing machines
- Undecidability of the halting problem

Gödel-Numbering of Turing Machines

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ be a DTM with $\Sigma = \{0, 1\}$ and

- $\Gamma = \{a_0, \dots, a_k\}$ where $a_0 = \Box$, $a_1 = \#$, $a_2 = 0$, $a_3 = 1$
- $Q = \{q_0, \dots, q_n\}$
- $F = \{q_n\}$

For $\delta(q_r, a_s) = (q_t, a_u, D)$ generate a word over $\{0, 1, \#\}$:

$$w_{r,s,t,u,D} = \#\#bin(r)\#bin(s)\#bin(t)\#bin(u)\#bin(val(D))$$

with $val(L) = 0, val(R) = 1$, and $val(N) = 2$

For M we generate w_M:

- Concatenate all words $w_{r,s,t,u,D}$ for $r \in \{0, \dots, n\}, s \in \{0, \dots, k\}$ and t, u, D given by $\delta(q_r, a_s) = (q_t, a_u, D)$
- Apply the following encoding to each symbol $\{0 \mapsto 00, 1 \mapsto 01, \# \mapsto 11\}$

Gödel-Numbering of Turing Machines (Cont'd)

- Not every word over $\{0,1\}$ is an encoding of a Turing machine (i.e. there exists w such that $w \neq w_M$ for all TMs M)
- To fix this: Let $\widehat{M}$ be a fixed (but arbitrary) Turing machine.
- For $w \in \{0,1\}^*$ let M_w be:

$$M_w := \begin{cases} M, & \text{if } w = w_M \\ \widehat{M}, & \text{otherwise} \end{cases}$$

Undecidability of the Halting-Problem

The halting problem is $H := \{w_M\#w \mid \text{TM } M \text{ halts on input } w\}$

- Assumption: H is decidable, i.e. there exists a TM M_H that terminates for any input $w_M\#w$ with output
  - 1 (=Yes) if M halts on input w
  - 0 (=No) if M does not halt on input w

Using M_H and the Gödel-numbering we can fill an (infinite) table, with entries Yes or No, depending on whether or not M_i halts on input w_{M_j}

	w_{M_1}	w_{M_2}	w_{M_3}	$\ldots$
M_1	Yes	No	$\ldots$	
M_2	No	No	$\ldots$	
M_3	Yes	No	$\ldots$	
$\ldots$	$\ldots$	$\ldots$	$\ldots$	

We construct a TM M_K:

- On input w, it checks whether M_w holds on w by using M_H.
- If yes, then M_K loops
- If no, then M_K stops successfully.

By construction:

$$M_K \text{ halts on } w_{M_j} \text{ iff } M_j \text{ does not halt on } w_{M_j}$$

Since all TMs are in the table: there is a j such that $M_j = M_K$.

$$M_j \text{ halts on input } w_{M_j} \text{ iff } M_j \text{ does not halt on input } w_{M_j}$$

This is a contradiction!
Our assumption was wrong: The halting problem is not decidable.