Theory of Computer Science

C3. Turing-Computability

Gabriele Röger

University of Basel

May 22, 2022

Turing-Computable Functions

Hello World

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

Hello World

```
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.

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 intutive sense.

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 intutive sense.
- So far, we have only considered recognizability and decidability: Is a word in a language, yes or no?

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 intutive 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).

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 intutive 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' $(c = c_0 \vdash c_1 \vdash c_2 \vdash \cdots \vdash c_{n-1} \vdash c_n = c', \ n \geq 0)$

(Definition of $\vdash$, i.e., how a computation step changes the configuration, is not repeated here. $\rightsquigarrow$ Chapter B10)

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 $f : \Sigma^* \to_{\text{p}} \Sigma^*$ for which for all $x, y \in \Sigma^*$:

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

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

German: DTM berechnet f

- What happens if the computation does not reach q_{accept}?
- What happens if symbols from $\Gamma \setminus \Sigma$ (e. g., $\Box$) 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, $f : \Sigma^* \rightarrow_\mathrm{p} \Sigma^*$)

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

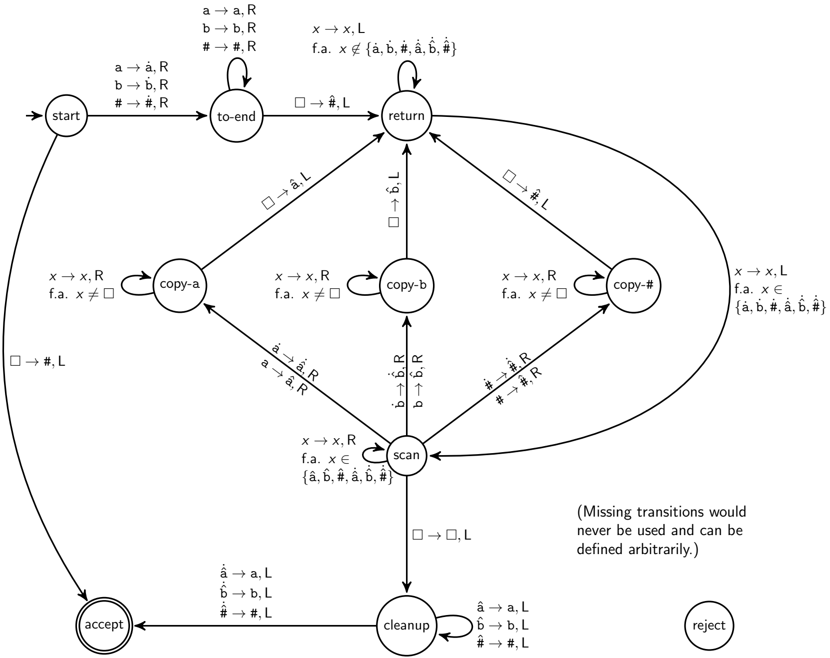
German: Turing-berechenbar

Example: Turing-Computable Functions on Words

Example

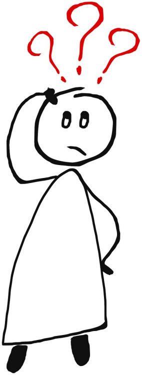
Let $\Sigma = \{a, b, \#\}$.
The function $f : \Sigma^* \rightarrow_p \Sigma^*$ with $f(w) = w\#w$ for all $w \in \Sigma^*$ is Turing-computable.

Idea: $\rightsquigarrow$ blackboard



(Missing transitions would never be used and can be defined arbitrarily.)

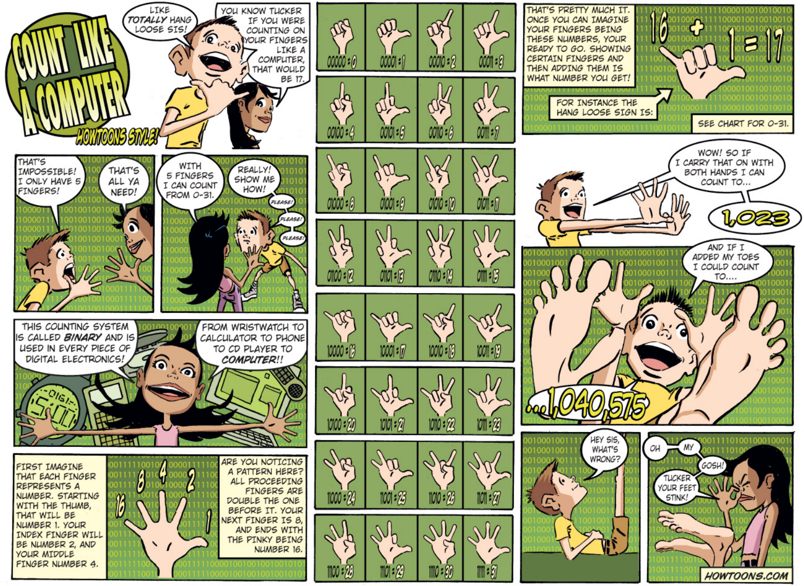
Questions



Questions?

Turing-Computable Numerical Functions

- We now transfer the concept to partial functions $f : \mathbb{N}_0^k \to_{\mathsf{p}} \mathbb{N}_0$.
- 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


COUNT LIKE A COMPUTER
HOWTOONS STYLE!
LIKE TOTALLY HANG LOOSE SIS!
YOU KNOW TUCKER IF YOU WERE COUNTING ON YOUR FINGERS LIKE A COMPUTER, THAT WOULD BE 17.
THAT'S IMPOSSIBLE! I ONLY HAVE 5 FINGERS!
THAT'S ALL YA NEED!
WITH 5 FINGERS I CAN COUNT FROM 0-31.
REALLY! SHOW ME HOW!
PLEASE!
PLEASE!
PLEASE!
THIS COUNTING SYSTEM IS CALLED BINARY AND IS USED IN EVERY PIECE OF DIGITAL ELECTRONICS!
FROM WRISTWATCH TO CALCULATOR TO PHONE TO CD PLAYER TO COMPUTER!!
FIRST IMAGINE THAT EACH FINGER REPRESENTS A NUMBER. STARTING WITH THE THUMB, THAT WILL BE NUMBER 1. YOUR INDEX FINGER WILL BE NUMBER 2, AND YOUR MIDDLE FINGER NUMBER 4.
16 8 4 2 1
ARE YOU NOTICING A PATTERN HERE? ALL PROCEEDING FINGERS ARE DOUBLE THE ONE BEFORE IT. YOUR NEXT FINGER IS 8, AND ENDS WITH THE PINKY BEING NUMBER 16.
00000 = 0
00001 = 1
00010 = 2
00011 = 3
00100 = 4
00101 = 5
00110 = 6
00111 = 7
01000 = 8
01001 = 9
01010 = 10
01011 = 11
01100 = 12
01101 = 13
01110 = 14
01111 = 15
10000 = 16
10001 = 17
10010 = 18
10011 = 19
10100 = 20
10101 = 21
10110 = 22
10111 = 23
11000 = 24
11001 = 25
11010 = 26
11011 = 27
11100 = 28
11101 = 29
11110 = 30
11111 = 31
THAT'S PRETTY MUCH IT. ONCE YOU CAN IMAGINE YOUR FINGERS BEING THESE NUMBERS, YOUR READY TO GO. SHOWING CERTAIN FINGERS AND THEN ADDING THEM IS WHAT NUMBER YOU GET!
16 + 1 = 17
FOR INSTANCE THE HANG LOOSE SIGN IS:
SEE CHART FOR 0-31.
WOW! SO IF I CARRY THAT ON WITH BOTH HANDS I CAN COUNT TO...
1,023
AND IF I ADDED MY TOES I COULD COUNT TO....
...1,040,575
HEY SIS, WHAT'S WRONG?
OH
MY
GOSH!
TUCKER YOUR FEET STINK!
HOWTOONS.COM

Encoding Numbers as Words

Definition (Encoded Function)

Let $f : \mathbb{N}_0^k \to_p \mathbb{N}_0$ be a (partial) function.
The encoded function f^{code} of f is the partial function $f^{\text{code}} : \Sigma^* \to_p \Sigma^*$ with $\Sigma = \{0, 1, \#\}$ and $f^{\text{code}}(w) = w'$ iff

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

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

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, $f : \mathbb{N}_0^k \to_{\mathrm{p}} \mathbb{N}_0$)

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

German: Turing-berechenbar

Exercise

The addition of natural numbers $+ : \mathbb{N}_0^2 \to \mathbb{N}_0$ 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:

- $succ : \mathbb{N}_0 \to_{\mathrm{p}} \mathbb{N}_0$ with $succ(n) := n + 1$
- $pred_1 : \mathbb{N}_0 \to_{\mathrm{p}} \mathbb{N}_0$ with $pred_1(n) := \begin{cases} n - 1 & \text{if } n \geq 1 \\ 0 & \text{if } n = 0 \end{cases}$
- $pred_2 : \mathbb{N}_0 \to_{\mathrm{p}} \mathbb{N}_0$ with $pred_2(n) := \begin{cases} n - 1 & \text{if } n \geq 1 \\ \text{undefined} & \text{if } n = 0 \end{cases}$

Example: Turing-Computable Numerical Function

Example

The following numerical functions are Turing-computable:

- $succ : \mathbb{N}_0 \rightarrow_{\mathrm{p}} \mathbb{N}_0$ with $succ(n) := n + 1$
- $pred_1 : \mathbb{N}_0 \rightarrow_{\mathrm{p}} \mathbb{N}_0$ with $pred_1(n) := \begin{cases} n - 1 & \text{if } n \geq 1 \\ 0 & \text{if } n = 0 \end{cases}$
- $pred_2 : \mathbb{N}_0 \rightarrow_{\mathrm{p}} \mathbb{N}_0$ with $pred_2(n) := \begin{cases} n - 1 & \text{if } n \geq 1 \\ \text{undefined} & \text{if } n = 0 \end{cases}$

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 ommitted)

1. 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.
2. Move the head onto the last symbol of the input.
3. 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.
4. 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 ommitted)

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.

Predecessor Function

The Turing machine for $pred_1$ works as follows:

(Details of marking the first tape position ommitted)

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

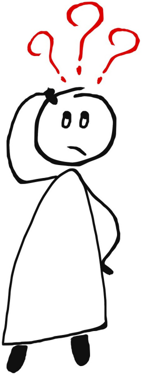
Example

The following numerical functions are Turing-computable:

- $add : \mathbb{N}_0^2 \to_{\mathrm{p}} \mathbb{N}_0$ with $add(n_1, n_2) := n_1 + n_2$
- $sub : \mathbb{N}_0^2 \to_{\mathrm{p}} \mathbb{N}_0$ with $sub(n_1, n_2) := \max\{n_1 - n_2, 0\}$
- $mul : \mathbb{N}_0^2 \to_{\mathrm{p}} \mathbb{N}_0$ with $mul(n_1, n_2) := n_1 \cdot n_2$
- $div : \mathbb{N}_0^2 \to_{\mathrm{p}} \mathbb{N}_0$ 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?

Questions



Questions?

Decidability vs. Computability

Decidability as Computability

Theorem

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

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

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

Decidability as Computability

Theorem

A language $L \subseteq \Sigma^*$ is decidable iff $\chi_L : \Sigma^* \to \{0, 1\}$, 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.

Decidability as Computability

Theorem

A language $L \subseteq \Sigma^*$ is decidable iff $\chi_L : \Sigma^* \to \{0, 1\}$, 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 χ_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 iff the following function $\chi'_L : \Sigma^* \to_{\mathrm{p}} \{0, 1\}$ is computable.

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

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

Turing-recognizable Languages and Computability

Theorem

A language $L \subseteq \Sigma^*$ is Turing-recognizable iff the following function $\chi'_L : \Sigma^* \to_{\mathrm{p}} \{0,1\}$ 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.

Turing-recognizable Languages and Computability

Theorem

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

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

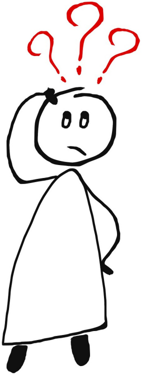
$$\chi'_L(w) = \begin{cases} 1 & \text{if } w \in L \\ \text{undefined} & \text{if } w \notin L \end{cases}$$

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 χ'_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.

Questions



Questions?

Summary

Summary

- Turing-computable function $f : \Sigma^* \to_{\mathrm{p}} \Sigma^*$:
  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 $f : \mathbb{N}_0^k \to_{\mathrm{p}} \mathbb{N}_0$:
  ditto; numbers encoded in binary and separated by #