6. The Equivalence Theorem and the Church-Turing Thesis

This Sect. has two parts:

(a) Equivalence of the models of computation.

- Proof that sets of URM-computable functions, of TM-computable functions and of partial-recursive functions coincide.

(b) The Church Turing Thesis.

- Evidence why these models of computation define the computable functions.

(a) Equivalence of the Models of Computation

- We are going to show that
  - the set of URM-computable functions,
  - the set of TM-computable functions, and
  - the set of partial-recursive functions

  coincide.
- One direction we have already shown:
- **Lemma 5.13** All partial recursive functions are URM computable.
- **Proof:** By Lemma 3.3.

TM-Computable Implies Part.-Rec.

- We will define for TMs T a code $\mathrm{encode}(\mathrm{T}) \in \mathbb{N}$ and show:
  - For $n \in \mathbb{N}$ there exist partial rec. func.

$$f_n : \mathbb{N}^{n+1} \xrightarrow{\sim} \mathbb{N}$$

s.t. for every TM T we have

$$\forall \vec{m} \in \mathbb{N}^n . f_n(\mathrm{encode}(\mathrm{T}), \vec{m}) \simeq \mathrm{T}^{(n)}(\vec{m}) \quad .$$

- f_n will be called **universal partial recursive functions**. f_n can be seen as an interpreter for TM.

TM-Computable Implies Part.-Rec.

$f_n : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$,
$\forall \vec{m} \in \mathbb{N}^n . f_n(\mathrm{encode}(\mathrm{T}), \vec{m}) \simeq \mathrm{T}^{(n)}(\vec{m})$.

- We will later write $\{e\}^n(\vec{m})$ for $f_n(e, \vec{m})$.

Encoding of TM

- Several Steps needed for this.
- Use of **Gödel-brackets** in order to denote the code of an object:
  - E.g. $\lceil x \rceil$ for the code of x.
  - Then $\lceil x \rceil$ is called the **Gödel-number** of x.
- Assume encoding $\lceil x \rceil$ for symbols x of the alphabet of a TM s.t.
  - $\lceil 0 \rceil = 0$,
  - $\lceil 1 \rceil = 1$,
  - $\lceil \llcorner \lrcorner \rceil = 2$.

Kurt Gödel

Kurt Gödel (1906 – 1978)

Most important logician of the 20th century.

The use of Gödel numbers in order to encode complex data structures into natural numbers was one of his techniques for proving the famous Gödel's Incompleteness Theorems.

Encoding of TM

- We assume an encoding $\lceil s \rceil$ for states s s.t. $\lceil s_0 \rceil = 0$ for the initial state s_0 of the TM.
- We encode the directions in the instructions by
  - $\lceil \mathsf{L} \rceil = 0$,
  - $\lceil \mathsf{R} \rceil = 1$.
- We assume the alphabet consists of $\{0, 1, \llcorner\lrcorner\}$ and symbols occurring in the instructions.
  - $\Rightarrow$ no need to explicitly mention the alphabet in the code for a TM.
- We assume the states consists of $\{s_0\}$ and the states occurring in the instructions.
  - $\Rightarrow$ no need to explicitly mention the set of states in the code for a TM.

Encoding of TM

- No need to mention s_0, $\llcorner\lrcorner$.
  $\Rightarrow$ TM can be identified with its instructions.
- An instruction $\mathrm{I} = (s, a, s', a', D)$ will be encoded as

$$\mathrm{encode}(\mathrm{I}) = \pi^5(\lceil s \rceil, \lceil a \rceil, \lceil s' \rceil, \lceil a' \rceil, \lceil D \rceil) \ .$$

- A set of instructions $\{\mathrm{I}_0, \ldots, \mathrm{I}_{k-1}\}$ will be encoded as

$$\langle \mathrm{encode}(\mathrm{I}_0), \ldots, \mathrm{encode}(\mathrm{I}_{k-1}) \rangle \ .$$

This is as well $\mathrm{encode}(\mathrm{T})$ of the corresponding TM.

Encoding of a Configuration

Configuration of a TM given by:

- The code $\lceil s \rceil$ of its state s.
- A segment (i.e. a consecutive sequence of cells) of the tape, which includes
  - the cell at head position,
  - all cells which are not blank.

  A segment $a_0, \ldots, a_{n-1}$ is encoded as

$$\mathrm{encode}(a_0, \ldots, a_{n-1}) := \langle \lceil a_0 \rceil, \ldots, \lceil a_{n-1} \rceil \rangle \ .$$

- The position of the head on this tape.
  Given as a number i s.t. $0 \leq i < n$, if the segment represented is $a_0, \ldots, a_{n-1}$.

Encoding of a Configuration

- A configuration of a TM, given by
  - a state s,
  - a segment $a_0, \ldots, a_{n-1}$,
  - head position i,

  will be encoded as

$$\pi^3(\lceil s \rceil, \mathrm{encode}(a_0, \ldots, a_{n-1}), i) \quad .$$

- As we have seen in Sect. 4, at any time during the computation of a TM, only finitely many cells of the tape are not blank.
  - Therefore at any intermediate step the state of a TM can be encoded as a configuration.

Simulation of TM – symbol, state

- We define primitive recursive functions
  $\mathsf{symbol}, \mathsf{state} : \mathbb{N} \to \mathbb{N}$,
  which extract
  - the symbol at the head,
  - the state of the TM

  from the current configuration:

$$\begin{aligned} \mathsf{symbol}(a) &= (\pi_1^3(a))_{\pi_2^3(a)} \\ \mathsf{state}(a) &= \pi_0^3(a) \end{aligned}$$

Note that $a = \pi^3(\lceil s \rceil, \langle \lceil a_0 \rceil, \ldots, \lceil a_{n-1} \rceil \rangle, i)$.

Simulation of TM – lookup

- We define a prim. rec. function

$$\text{lookup} : \mathbb{N}^3 \to \mathbb{N} \ ,$$

s.t. if

- e is the code for a TM,
- s state,
- a a symbol,

then $\text{lookup}(e, s, a)$ is defined as

$$\pi^3(\lceil s' \rceil, \lceil a' \rceil, \lceil D \rceil)$$

where s', a', D are s.t. (s, a, s', a', D) is the first instruction e of the TM corresponding to state s and symbol a.

Simulation of TM – lookup

- If no such instruction exists, the result will be $0 (= \pi^3(0,0,0))$.
  lookup is defined as follows:
  - First find using bounded search the index of the first instruction starting with $\lceil s \rceil, \lceil a' \rceil$.
  - Then extract the corresponding values from this instruction.

  The details will be omitted in the lecture.
  Jump over details.

Definition of lookup

Formally the definition is as follows:

- Define an auxiliary primitive recursive function $g : \mathbb{N}^3 \to \mathbb{N}$,

$$g(e, q, a) = \mu i \leq \mathsf{lh}(e).\pi_0^5((e)_i) = q \wedge \pi_1^5((e)_i) = a \ ,$$

  which finds the index of the first instruction starting with q, a.

- Now define

$$\mathsf{lookup}(e, q, a) := \pi^3(\pi_2^5((e)_{g(e,q,a)}), \pi_3^5((e)_{g(e,q,a)}), \pi_4^5((e)_{g(e,q,a)}))$$

Simulation of TM – hasinstruction

- There exists a primitive recursive relation hasinstruction $\subseteq \mathbb{N}^3$, s.t.
  - hasinstruction$(e, \lceil s \rceil, \lceil a \rceil)$ holds, iff the TM corresponding to e has a instruction (s, a, s', a', D) for some s', a', D.
  - hasinstruction is defined as a byproduct of defining lookup.
  - The details will be omitted in the lecture. Jump over details.

Definition of hasinstruction

hasinstruction is defined, using the function g from the previous item, as

$$\chi_{\mathsf{hasinstruction}}(e,q,a) = \mathsf{sig}(\mathsf{lh}(e) \dot{-} g(e,q,a)) \ .$$

If there is such an instruction,

$$g(e,q,a) < \mathsf{lh}(e) \ ,$$

therefore

$$\mathsf{lh}(e) \dot{-} g(e,q,a) > 0 \ .$$

Otherwise

$$\begin{aligned} g(e,q,a) &= \mathsf{lh}(e) \ , \\ \mathsf{lh}(e) \dot{-} g(e,q,a) &= 0 \ . \end{aligned}$$

Simulation of TM – next

- There exists prim rec. function

$$\mathsf{next} : \mathbb{N}^2 \longrightarrow \mathbb{N}$$

s.t.

- if e encodes a TM
- and c is a code for a configuration,

then

- $\mathsf{next}(e, c)$ is a code for the configuration after one step of the TM is executed,
- or equal to c, if the TM has halted.

Jump over details.

Simulation of TM – next

- Informal description of next:
  - Assume the configuration is

$$c = \pi^3(s, \langle a_0, \ldots, a_{n-1} \rangle, i) \quad .$$

  - Check using hasinstruction, whether the TM has stopped.
    If it has stopped, return c.
  - Otherwise, use lookup to obtain the codes for
    - the next state s',
    - next symbol a',
    - direction D.
  - Replace in $\langle a_0, \ldots, a_{n-1} \rangle$, the ith element by a'.
    Let the result be x.

Simulation of TM – next

Next state is s', next Symbol is a', direction is D.
x = result of substituting the symbol in original segment.

- Might be that head will leave segment.
  - Then extend segment.
- Case $D = \lceil L \rceil$ and $i = 0$:
  Extend the segment to the left by one blank.
  Result of next is

$$\pi^3(s', \langle \lceil \llcorner\lrcorner \rceil \rangle * x, 0) \quad .$$

- Case $D \neq \lceil L \rceil, i \geq n - 1$.
  Result of next is

$$\pi^3(s', x * \langle \lceil \llcorner\lrcorner \rceil \rangle, n) \quad .$$

Simulation of TM – next

Next state is s', next Symbol is a', direction is D.
x = result of substituting the symbol in original segment.

- Otherwise let
  - $i' := i - 1$, if $D = \lceil L \rceil$,
  - $i' := i + 1$, if $D \neq \lceil L \rceil$.

  Result of next is

$$\pi^3(s', x, i') \quad .$$

Simulation of TM – terminate

- There exists a prim. rec. predicate $\mathsf{terminate} \subseteq \mathbb{N}^2$, s.t. $\mathsf{terminate}(e, c)$ holds, iff the TM e with configuration c stops:

$$\mathsf{terminate}(e, c) :\Leftrightarrow \neg\mathsf{hasinstruction}(e, \mathsf{state}(c), \mathsf{symbol}(c)) \quad .$$

Simulation of TM – iterate

- There exists a primitive recursive function
  $\mathsf{iterate} : \mathbb{N}^3 \to \mathbb{N}$,
  s.t. $\mathsf{iterate}(e, c, n)$ is the result of iterating TM e with initial configuration c for n steps.
  The definition is as follows

$$\begin{aligned} \mathsf{iterate}(e, c, 0) &= c \ , \\ \mathsf{iterate}(e, c, n+1) &= \mathsf{next}(e, \mathsf{iterate}(e, c, n)) \ . \end{aligned}$$

Simulation of TM – init

- There exists a primitive recursive function
  $\mathsf{init}^n : \mathbb{N}^n \to \mathbb{N}$,
  s.t. for any TM T containing alphabet $\{0, 1, \llcorner\lrcorner\}$
  $\mathsf{init}^n(\vec{m})$ is the initial configuration for computing $\mathrm{T}^{(n)}(\vec{m})$
  for any TM T, the alphabet of which contains $\{0, 1, \llcorner\lrcorner\}$:

$$\begin{aligned}\mathsf{init}^n(m_0, \ldots, m_{n-1}) \\ &= \pi^3(\lceil s_0 \rceil, \mathsf{bin}(m_0) * \langle\lceil\llcorner\lrcorner\rceil\rangle * \mathsf{bin}(m_1) * \langle\lceil\llcorner\lrcorner\rceil\rangle * \cdots \\ &\qquad * \langle\lceil\llcorner\lrcorner\rceil\rangle * \mathsf{bin}(m_{n-1}), 0) \quad .\end{aligned}$$

Simulation of TM – extract

- There exists a primitive recursive function $\mathsf{extract} : \mathbb{N} \to \mathbb{N}$, s.t.
  - if c is a configuration in which the TM stops,
  - $\mathsf{extract}(c)$ is the natural number corresponding to the result returned by the TM.
- Assume $c = \pi^3(s, x, i)$.
- $\mathsf{extract}(c)$ obtained by applying bin^{-1} to the largest subsequence of x starting at i consisting of 0 and 1 only.

Simulation of TM – T

- There exist primitive recursive predicates $\mathsf{T}^n_{\mathsf{Kleene}} \subseteq \mathbb{N}^{n+2}$ s.t.,
  if e encodes a TM T, then
  $\mathsf{T}^n_{\mathsf{Kleene}}(e, m_0, \ldots, m_{n-1}, k)$ holds, iff T,
  - started with the initial configuration corresponding to a computation of $\mathrm{T}^{(n)}(m_0, \ldots, m_{n-1})$,
  - stops after $\pi_0(k)$ steps,
  - and has final configuration $\pi_1(k)$.

$$\mathsf{T}^n_{\mathsf{Kleene}}(e, \vec{m}, k) :\Leftrightarrow$$
$$\mathsf{terminate}(e, \mathsf{iterate}(e, \mathsf{init}^n(\vec{m}), \pi_0(k)))$$
$$\wedge\, \mathsf{iterate}(e, \mathsf{init}^n(\vec{m}), \pi_0(k)) = \pi_1(k) \quad .$$

Simulation of TM – U

- There exists a prim. rec. function $\mathsf{U}_{\mathsf{Kleene}} : \mathbb{N} \to \mathbb{N}$ s.t. if
  - T is a TM with $\mathrm{encode}(\mathrm{T}) = e$,
  - $\mathsf{T}^n_{\mathsf{Kleene}}(e, \vec{m}, k)$ holds,

  then $\mathsf{U}_{\mathsf{Kleene}}(k)$ is the result of $\mathrm{T}^{(n)}(\vec{m})$.

$$\mathsf{U}_{\mathsf{Kleene}}(k) = \mathsf{extract}(\pi_1(k)) \quad .$$

- Therefore, if e encodes a TM T, then

$$\mathrm{T}^{(n)}(\vec{n}) \simeq \mathsf{U}_{\mathsf{Kleene}}(\mu z.\mathsf{T}^n_{\mathsf{Kleene}}(e, \vec{m}, z)) \quad .$$

Kleene's NF Theorem

Kleene's Normal Form Theorem 5.14:

(a) There exists partial recursive functions $f_n : \mathbb{N}^{n+1} \xrightarrow{\sim} \mathbb{N}$ s.t.
if e is the code of TM T then

$$f_n(e, \vec{n}) \simeq \mathrm{T}^n(\vec{n}) \quad .$$

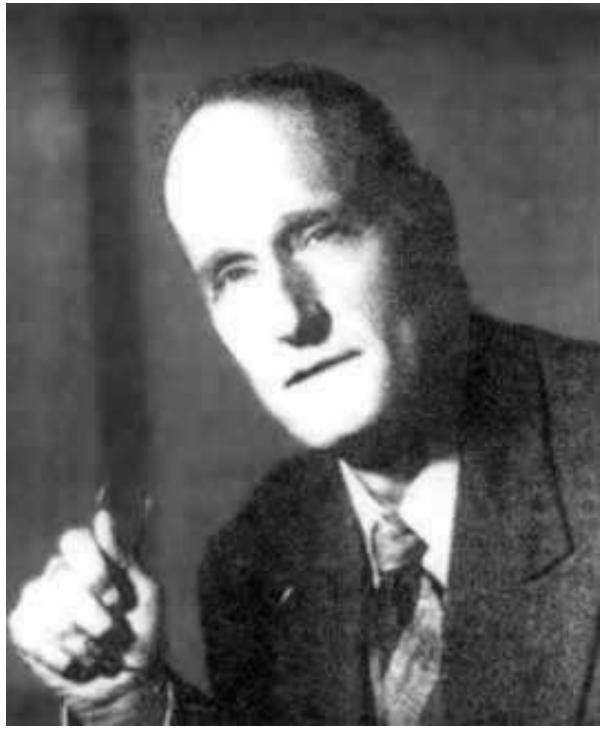
(b) There exist

- a primitive recursive function $\mathsf{U}_{\mathsf{Kleene}} : \mathbb{N} \to \mathbb{N}$,
- primitive recursive predicate $\mathsf{T}^n_{\mathsf{Kleene}} \subseteq \mathbb{N}^{n+2}$,

s.t. for the function f_n introduced in (a) we have

$$f_n(e, \vec{n}) \simeq \mathsf{U}_{\mathsf{Kleene}}(\mu z.\mathsf{T}^n_{\mathsf{Kleene}}(e, \vec{m}, z)) \quad .$$

Stephen Cole Kleene



Stephen Cole Kleene
(1909 – 1994)

Probably the most influential computability theorist up to now. Introduced the partial recursive functions.

Remark

- The operations for extracting instructions from a TM above were primitive recursive and therefore total.
- Therefore, even if e is not a code for a TM, e will be treated as if were a code for a TM, namely the one with the instructions computed by the above operations.
- Omit details

Details of the Prev. Remark

- A code for such a valid TM is obtained by
  - treating e as a sequence of instructions,
  - deleting from it any $\mathrm{encode}(q, a, q', a', D)$ s.t. there exists an instruction $\mathrm{encode}(q, a, q'', a'', D')$ occurring before it,
  - and by replacing all directions > 1 by $\lceil \mathsf{R} \rceil = 1$.

Definition 5.15

Let $\mathsf{U}_{\mathsf{Kleene}}$, $\mathsf{T}^n_{\mathsf{Kleene}}$ as in Kleene's Normal Form Theorem 5.14.
Then define for $e \in \mathbb{N}$ the partial recursive function

$$\{e\}^n : \mathbb{N}^n \xrightarrow{\sim} \mathbb{N}$$

by

$$\{e\}^n(\vec{m}) \simeq \mathsf{U}_{\mathsf{Kleene}}(\mu y.\mathsf{T}^n_{\mathsf{Kleene}}(e, \vec{m}, y)) \quad .$$

- $\{e\}^n$ is pronounced **Kleene-Bracket-e in n arguments**.

Notation

- We will often omit the superscript n in $\{e\}^n(m_0, \ldots, m_{n-1})$
  - i.e. write $\{e\}(m_0, \ldots, m_{n-1})$ instead of $\{e\}^n(m_0, \ldots, m_{n-1})$.
- Further $\{e\}$ not applied to arguments and without superscript means usually $\{e\}^1$.

Corollary 5.16

(a) For every TM-computable function $f : \mathbb{N}^n \xrightarrow{\sim} \mathbb{N}$ there exists $e \in \mathbb{N}$ s.t.

$$f = \{e\}^n \quad .$$

(b) Especially, all TM-computable functions are partial recursive.

Theorem 5.17

The following sets coincide:

- the set of URM-computable functions,
- the set of Turing-computable functions,
- the set of partial recursive functions.

Proof:

- URM-computable implies TM-computable by Theorem 4.3.
- TM-computable implies partial recursive by Corollary 5.16.
- Partial recursive implies URM computable by Lemma 5.13.

Theorem 5.18

The partial recursive functions $g : \mathbb{N}^n \xrightarrow{\sim} \mathbb{N}$ are exactly the functions $\{e\}^n$.

Proof:

- By Kleene's Normal Form Theorem every TM-computable function $g : \mathbb{N}^n \xrightarrow{\sim} \mathbb{N}$ is of the form $\{e\}^n$.
- The TM-computable functions are the partial rec. functions.
- Therefore the assertion follows.

Remark

- Theorem 5.18 means:
  - $\{e\}^n$ is partial recursive for every $e \in \mathbb{N}$.
  - For every partial recursive function $g : \mathbb{N}^n \xrightarrow{\sim} \mathbb{N}$ there exists an e s.t. $g = \{e\}^n$.
- The e s.t. $g = \{e\}^n$ for $g : \mathbb{N}^n \xrightarrow{\sim} \mathbb{N}$ is called the **Kleene-index** of g.

Remark

- Therefore

$$f_n : \mathbb{N}^{n+1} \stackrel{\sim}{\rightarrow} \mathbb{N} \ , \qquad f_n(e, \vec{x}) \simeq \{e\}^n(\vec{x})$$

  forms a **universal n-ary partial recursive function**:
  It encodes all n-ary partial recursive function.

- So we can assign to each partial recursive function g a number, namely an e s.t. $g = \{e\}^n$.
  - Each number e denotes one partial recursive function $\{e\}^n$.
  - However, several numbers denote the same partial recursive function:
    There are e, e' s.t. $e \neq e'$ but $\{e\}^n = \{e'\}^n$.

Proof of Last Fact

Proof, that different e compute the same function:

- There are several algorithms for computing the same function.
- Therefore there are several Turing machines which compute the same function.
- These Turing machines have different codes e.

Lemma 5.19

The set F of partial recursive functions

(and therefore as well of Turing-computable and of URM-computable functions),

i.e.

$$F := \{f : \mathbb{N}^k \xrightarrow{\sim} \mathbb{N} \mid k \in \mathbb{N} \wedge f \text{ partial recursive}\}$$

is countable.

Proof of Lemma 5.19

Every partial recursive function is of the form $\{e\}^n$ for some $e, n \in \mathbb{N}$.

Therefore

$$f : \mathbb{N}^2 \to F \ , \qquad f(e,n) := \{e\}^n$$

is surjective.

$\mathbb{N}^2$ is countable.

Therefore by Corollary 2.15 (a), F is countable as well.

(b) The Church-Turing Thesis

We have introduced three models of computations:

- The URM-computable functions.
- The Turing-computable functions.
- The partial recursive functions.

Further we have shown that all three models compute the same partial functions.

The Church-Turing Thesis

Lots of other models of computation have been studied:

- The while programs.
- Symbol manipulation systems by Post and by Markov.
- Equational calculi by Kleene and by Gödel.
- The λ-definable functions.
- Any of the programming languages Pascal, C, C++, Java, Prolog, Haskell, ML (and many more).
- Lots of other models of computation.

The Church-Turing Thesis

- One can show that the partial functions computable in these models of computation are again exactly the partial recursive functions.
- So all these attempts to define a complete model of computation result in the same set of partial recursive functions.
- Therefore we arrive at the Church Turing Thesis

The Church-Turing Thesis

Church-Turing Thesis: *The (in an intuitive sense) computable partial functions are exactly the partial recursive functions (or equivalently the URM-computable or Turing-computable functions).*

Philosophical Thesis

- This thesis is **not a mathematical theorem**.
- It is a **philosophical thesis**.
- Therefore the Church-Turing thesis **cannot be proven**.
- We can only provide **philosophical evidence** for it.
- This evidence comes from the following **considerations and empirical facts**:

Empirical Facts

- All complete models of computation suggested by researchers define the same set of partial functions.
- Many of these models were carefully designed in order to capture intuitive notions of computability:
  - The Turing machine model captures the intuitive notion of **computation on a piece of paper** in a general sense.
  - The URM machine model captures the general notion of **computability by a computer**.
  - Symbolic manipulation systems capture the general notion of computability by **manipulation of symbolic strings**.

Empirical Facts

- No intuitively computable partial function, which is not partial recursive, has been found, despite lots of researchers trying it.
- A strong intuition has been developed that in principal programs in any programming language can be simulated by Turing machines and URMs.

Because of this, only few researchers doubt the correctness of the Church-Turing thesis.

Decidable Sets

- A predicate A is URM-/Turing-decidable iff χ_A is URM-/Turing-computable.
- A predicate A is decidable iff χ_A is computable.
- By Church's thesis to be computable is the same as to be URM-computable or to be Turing-computable.
- So the decidable predicates are exactly the URM-decidable and exactly the Turing-decidable predicates.

Halting Problem

- Because of the equivalence of the 3 models of computation, the halting problem for any of the above mentioned models of computation is undecidable.
- Especially it is undecidable, whether a program in one of the programming languages mentioned terminates:
  - Assume we had a decision procedure for deciding whether or not say a Pascal program terminates for given input.
  - Then we could, using a translation of URMs into Pascal programs, decide the halting problem for URMs, which is impossible.