Computability Theory

Karl-Heinz Zimmermann

TUHH

September 7, 2020

TUHH
Technische Universität Hamburg

Prof. Dr. Karl-Heinz Zimmermann
Hamburg University of Technology
21071 Hamburg
Germany

This manuscript is listed in the GBV database and the TUHH library.

https://doi.org/10.15480/882.2897
http://hdl.handle.net/11420/7243
urn:nbn:gbv:830-882.0104943

Part I

Introduction

Preface

- How to show that a decision problem is not algorithmically decidable?[a]
- Rigorous formalism initialized by the work of David Hilbert (1900) and Kurt Gödel (1931).
- Turing machine captures the notion of computability (Alan Turing, 1936)
- Several mathematical problems are known to be undecidable (halting problem, word problems, string rewriting, solving diophantine equations).

[a] The problem whether two natural numbers are relatively prime is decidable by the Euclidean algorithm.

Preface

These notes are a development of class notes for a two-hour lecture including a two-hour lab held for second-year Bachelor students of Computer Science at the Hamburg University of Technology during the last few years.

The course aims to present the basic results of computability theory, including well-known mathematical models of computability such as the Turing machine, the unlimited register machine (URM) as well as GOTO and LOOP programs, the principal classes of computational functions like the classes of primitive recursive, recursive, and partial recursive functions, the famous Ackermann function and the Ackermann class, the main theoretical concepts of computability such as Gödel numbering, universal functions, parametrization, Kleene's normal form, Kleene's recursion theorem, the theorems of Rice, undecidable and semidecidable (or recursively enumerable) sets, computable real numbers, Hilbert's tenth problem, and last but not least several important undecidable word problems including those for semi-Thue systems, semigroups, and Post correspondence systems.

The corresponding manuscript provides a more detailled representation of the topic including the necessary mathematical background on semigroups and monoids, a brief introduction to the freely available command-line program `glu` which has been developed for the interpretation of GOTO, LOOP and URM programs, and a description of `Maple`^{TM} code for the encoding of strings of natural numbers and the Gödel numbering of SGOTO programs.

The manuscript has partly grown out of notes taken by the author during his studies at the University of Erlangen-Nuremberg. Therefore, I would like to express my thanks foremost to my teachers Martin Becker† and Volker Strehl for their wonderful and inspiring lectures in this field. Moreover, I would like to thank my colleague and friend Ralf Möller for valuable comments. I am also grateful to my collaborators Merve Nur Cakir, Mahwish Saleemi, Natalia Schmidt, and Robert Leppert for conducting the labs and Wolfgang Brandt for valuable technical support. Finally, I would like to thank my students for their attention, their stimulating questions, and their dedicated work.

The course took place in the first phase of the Corona pandemic and so we were forced to organize it as a full online course. At the end I can say that it has been a nice experience to present the topic by Zoom and explanatory videos. It appears that the participants were not fully unhappy with this situation. At the end, I'd like to add that the exam was fully organized in electronic form.

Contents

- Unlimited register machine (URM)
- Primitive recursive functions
- Partial recursive functions
- Ackermann function
- Turing machine
- Acceptable programming systems
- Undecidability
- Word problems

*Starred material can be safely skipped.

Literature

- Baader F, Nipkow T (1999) Term rewriting and all that. Cambridge Univ. Press, Cambridge
- Dyson G (2012) Turings Kathedrale. Ullstein, Berlin
- Enderton HB (2011) Computabilitiy Theory. Elsevier, Amsterdam
- Hopcraft JE, Ullman JD (1979) Introduction to automata theory, languages, and computation. Addison Wesley, New York
- Hofstadter DR (2008) Gödel, Escher, Bach: Ein endloses geflochtenes Band. 18. Auflage, Klett-Cotta, Stuttgart
- Hermes H (1978) Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit. 3. Auflage, Heidelberger Taschenbücher, Berlin
- Matiyasevich Y (1993) Hilbert's tenth problem. MIT Press, Boston
- Pour-El MB, Richards JI (1989) Computability in analysis and physics. Springer, New York
- Rogers H Jr (1987) Theory of recursive functions and effective computation. MIT Press, Boston
- Becker M, Strehl V (1984) Berechenbarkeit. Techn. Report, Erlangen
- Schöning U (2008) Theoretische Informatik – kurz gefasst. Spektrum, Heidelberg
- Zimmermann KH (2019) Computability theory. Techn. Report, Hamburg
- Zimmermann KH (2020) Berechenbarkeit. Springer *Essentials*, Heidelberg

Part II

Unlimited Register Machine

Unlimited Register Machine – Motivation

- URM introduced by Sheperdson and Sturgis (1963) as abstract computing machine.
- URM captures the notion of computability.
- URM has infinitely many, uniquely addressable registers, each of which holds a single natural number.
- URM programming (increment, decrement, composition, iteration).

E.g., addition of two numbers via URM program $(A1; S2)2$.

Unlimited Register Machine – Contents

- States and state transformations
- Syntax of URM programs
- Semantics of URM programs
- URM computable functions

Reading: Zimmermann, Chapter 1

States and Transformations – Contents

- Hardware
- State set and set of transformations
- Increment and decrement
- Composition
- Powers and iteration

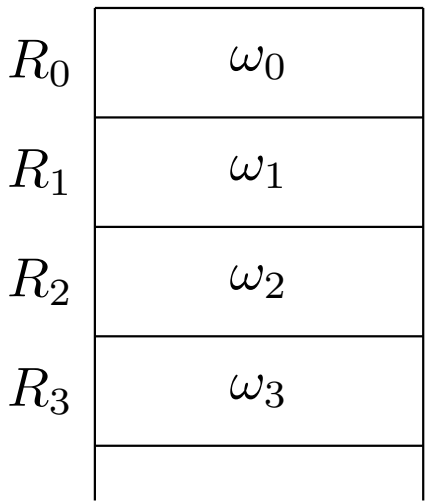
States and Transformations – Hardware

- URM contains an infinite number of registers named

$$R_0, R_1, R_2, R_3, \ldots. \tag{1}$$

- Each register stores a natural number, but only a finite number of registers can have nonzero entries.

R	0	1	2	3	4	5	...
ω	ω_0	ω_1	ω_2	ω_3	ω_4	ω_5	



States and Transformations – State Set

- *State set* of URM

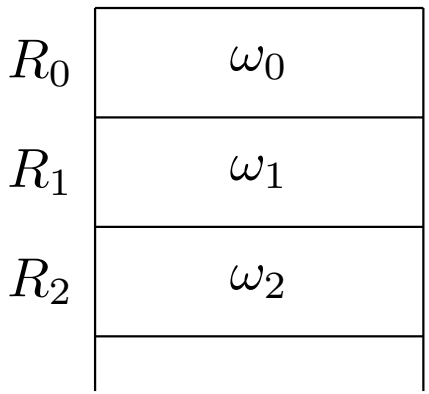
$$\Omega = \{\omega : \mathbb{N}_0 \to \mathbb{N}_0 \mid \omega \text{ is 0 almost everywhere}\}, \tag{2}$$

where *almost everywhere* means all but a finite number.

- *Almost everywhere* captures the notion of finite memory.
- The elements of Ω are denoted as sequences

$$\omega = (\omega_0, \omega_1, \omega_2, \omega_3, \ldots), \tag{3}$$

where $\omega_n = \omega(n)$ denotes the content of register R_n, $n \in \mathbb{N}_0$.



States and Transformations – $|\Omega|$

The set of states Ω of an URM is countable.

Proof.

Denote the infinite sequence of prime numbers as

$$(p_0, p_1, p_2, p_3, p_4, \ldots) = (2, 3, 5, 7, 11, \ldots).$$

The mapping

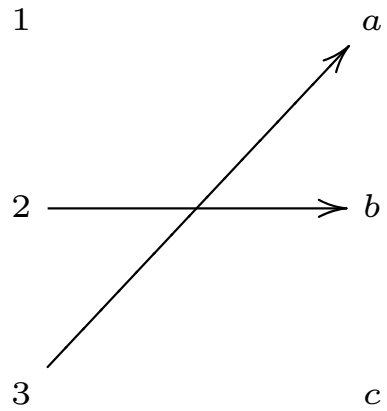
$$\Omega \to \mathbb{N} : \omega \mapsto \prod_{\substack{i \\ \text{finite}}} p_i^{\omega_i} \tag{4}$$

is bijective, since each natural number factors uniquely into a product of prime powers and the states are 0 almost everywhere, which means that the products are finite; e.g.,

$$(0, 5, 4, 0, 2, 0, \ldots) \mapsto p_0^0 p_1^5 p_2^4 p_3^0 p_4^2 p_5^0 \ldots = 3^5 5^4 11^2.$$

States and Transformations – Partial Functions

Partial function $f : \{1, 2, 3\} \to \{a, b, c\}$ has source set $\{1, 2, 3\}$, $\mathrm{dom}(f) = \{2, 3\}$ and $\mathrm{ran}(f) = \{a, b\}$:



Partial means that $\mathrm{dom}(f)$ is a proper subset of the source set. Note that total means that $\mathrm{dom}(f)$ equals the source set.

States and Transformations – Partial Functions

- A *partial function* $f : X \to Y$ is a function $f : X' \to Y$ for some subset X' of Y.

  Then the domain $\mathrm{dom}(f) = X'$ is a subset of the source set X of f.

- A partial function $f : X \to Y$ is *total* if the domain equals the source set, i.e., $\mathrm{dom}(f) = X$.

  Total functions are the functions in the usual sense (see Calculus, Linear Algebra).

- The square-root function

$$f : \mathbb{N}_0 \to \mathbb{N}_0 : x \mapsto \sqrt{x}$$

  is partial, since $\mathrm{dom}(f) = \{0, 1, 4, 9, 16, \ldots\}$ (perfect squares), e.g., $f(16) = 4$ but $f(17) \notin \mathbb{N}_0$.

States and Transformations – $E(\Omega)$

- $E(\Omega) = \{f \mid f : \Omega \to \Omega,\ f \text{ partial}\}$ denotes the set of all partial functions on Ω.

  $f \in E(\Omega)$ is a potential state transformation of the URM.

- Each partial function $f \in E(\Omega)$ has *domain*

$$\mathrm{dom}(f) = \{\omega \in \Omega \mid f(\omega) \text{ is defined}\} \tag{5}$$

  and *range*

$$\mathrm{ran}(f) = \{\omega' \in \Omega \mid \exists \omega \in \Omega :\ f(\omega) = \omega'\}. \tag{6}$$

- E.g., the square-root function $f : \mathbb{N}_0 \to \mathbb{N}_0 : x \mapsto \sqrt{x}$ has $\mathrm{dom}(f) = \{0, 1, 4, 9, 16, \ldots\}$ and $\mathrm{ran}(f) = \mathbb{N}_0$.

States and Transformations – $|E(\Omega)|$

The set of partial functions $E(\Omega)$ of an URM is uncountable.

*Proof

- The real interval $[0, 1)$ is uncountable.
- Represent each real number $a \in [0, 1)$ in binary format

$$a = 0.a_0a_1a_2\ldots, \quad a_i \in \{0, 1\}, i \geq 0.$$

- For each such number $a = 0.a_0a_1a_2\ldots$, define the function $f_a : \Omega \to \Omega$ by setting $f_a(\omega) = \omega'$, where

$$\omega'_n = a_n \oplus \omega_n, \quad n \geq 0, \tag{7}$$

  such that $a_n \oplus \omega_n = \omega_n$ if $a_n = 1$ and and $a_n \oplus \omega_n = 0$ if $a_n = 0$.
- f_a is well-defined, since a state ω with 0's almost everywhere is mapped to a state ω' with 0's almost everywhere.
- For different real numbers $a = 0.a_0a_1a_2\ldots$ and $b = 0.b_0b_1b_2\ldots$, the functions f_a and f_b are distinct. To see this, suppose $a_n = 1$ and $b_n = 0$ for some $n \geq 0$. Then for the state ω which is 1 at register R_n and 0 elsewhere, we have $f_a(\omega) = \omega$ and $f_b(\omega) = 0$, the zero state.
- E.g., let $a = 0.011000\ldots$ and $\omega = (2, 3, 4, 5, 0, 0, \ldots)$. Then

$$f_a(\omega) = (0, 3, 4, 0, 0, 0, \ldots).$$

States and Transformations – C-Programs

The set of programs in any programming language over a finite alphabet is countable.

Proof

- Each program written in any programming language over an alphabet $\Sigma = \{0, \ldots, s\}$ (keyboard alphabet) is a word in the free monoid Σ^*.
- The set Σ^* is countable, since the mapping

$$\phi : \Sigma^* \to \mathbb{N} : x_0 \ldots x_{n-1} \mapsto p_0^{x_0} \cdots p_{n-1}^{x_{n-1}}$$

is a bijection, where $(p_0, p_1, p_2, \ldots) = (2, 3, 5, \ldots)$ denotes the infinite sequence of primes (see (4)), e.g.,

$$(2, 0, 3, 1, 0) \mapsto p_0^2 p_1^0 p_2^3 p_3^1 p_4^0 = 2^2 5^3 7.$$

States and Transformations – Gaps

- There is a *gap* between the set of programs in a programming language and the set of state transformations of an abstract computing machine,

$$|\Sigma^*| < |E(\Omega)|.$$

- *Continuum hypothesis* says that there is no set whose cardinality is strictly between that of the natural numbers and the real numbers,

$$\exists\, X: \quad |\mathbb{N}_0| < |X| < |\mathbb{R}|\ ?$$

States and Transformations – Equality

Let $f, g \in E(\Omega)$ be partial functions.

- Function equality $f = g$ means
  - $\mathrm{dom}(f) = \mathrm{dom}(g)$ and
  - for all $\omega \in \mathrm{dom}(f)$, $f(\omega) = g(\omega)$.
- $f \in E(\Omega)$ *total* if $\mathrm{dom}(f) = \Omega$ (source set).

Total functions are the functions in the usual sense (see Calculus, Linear Algebra).

States and Transformations – Increment

Increment function $a_k \in E(\Omega)$ for k-th register is the total function $a_k : \Omega \to \Omega : \omega \mapsto \omega'$, where

$$\omega'_n = \begin{cases} \omega_n & \text{if } n \neq k, \\ \omega_k + 1 & \text{otherwise,} \end{cases} \tag{8}$$

$$(\omega_0, \ldots, \omega_{k-1}, \omega_k, \omega_{k+1}, \ldots) \mapsto (\omega_0, \ldots, \omega_{k-1}, \omega_k + 1, \omega_{k+1}, \ldots).$$

URM-programming primitive: Ak.

States and Transformations – Decrement

- *Decrement function* $s_k \in E(\Omega)$ for k-th register is the total function $s_k : \Omega \to \Omega : \omega \mapsto \omega'$, where

$$\omega'_n = \begin{cases} \omega_n & \text{if } n \neq k, \\ \omega_k \dot{-} 1 & \text{otherwise.} \end{cases} \tag{9}$$

$$(\omega_0, \ldots, \omega_{k-1}, \omega_k, \omega_{k+1}, \ldots) \mapsto (\omega_0, \ldots, \omega_{k-1}, \omega_k \dot{-} 1, \omega_{k+1}, \ldots).$$

- Dyadic operation $\dot{-} : \mathbb{N}_0^2 \to \mathbb{N}_0$ denotes the *asymmetric difference*,

$$x \dot{-} y = \begin{cases} x - y & \text{if } x \geq y, \\ 0 & \text{otherwise,} \end{cases} \tag{10}$$

e.g., $5 \dot{-} 3 = 2$ and $5 \dot{-} 7 = 0$.

URM-programming primitive: Sk.

States and Transformations – *Graph

- *Graph* of a partial function $f \in E(\Omega)$ is given by the relation

$$R_f = \{(\omega, f(\omega)) \mid \omega \in \mathrm{dom}(f)\}. \tag{11}$$

E.g., increment function a_k, $k \in \mathbb{N}_0$, gives

$$R_{a_k} = \{(\omega, \omega') \mid \omega \in \Omega, \omega' = (w_0, \ldots, w_{k-1}, w_k{+}1, w_{k+1}, \ldots)\}.$$

- Two partial functions $f, g \in E(\Omega)$ are *equal* if the corresponding graphs R_f and R_g are equal as sets, i.e.,

$$\mathrm{dom}(f) = \mathrm{dom}(g) \wedge \forall \omega \in \mathrm{dom}(f) : f(\omega) = g(\omega). \tag{12}$$

- E.g., the square-root function $f : \mathbb{N}_0 \to \mathbb{N}_0 : x \mapsto \sqrt{x}$ has the graph

$$R_f = \{(0,0), (1,1), (4,2), (9,3), (16,4), (25,5), \ldots\}.$$

States and Transformations – Composition

Composition of two partial functions $f, g \in E(\Omega)$ is a partial function $g \circ f$ with (inner function f and outer function g)

$$(g \circ f)(\omega) = g(f(\omega)), \tag{13}$$

where ω belongs to the domain of $g \circ f$,

$$\mathrm{dom}(g \circ f) = \{\omega \in \Omega \mid \omega \in \mathrm{dom}(f) \wedge f(\omega) \in \mathrm{dom}(g)\}. \tag{14}$$

- If f and g are total, then $g \circ f$ will be total.
- Composition is *associative*, i.e., for all $f, g, h \in E(\Omega)$,

$$f \circ (g \circ h) = (f \circ g) \circ h. \tag{15}$$

States and Transformations – Composition

Composition of increment functions a_k and a_l, $k, l \in \mathbb{N}_0$.

- Case $k = l$:

$$(a_k \circ a_l)(\omega) = (w_0, \ldots, w_k + 2, \ldots).$$

- Case $k \neq l$ with $k < l$:

$$(a_k \circ a_l)(\omega) = (w_0, \ldots, w_k + 1, \ldots, w_l + 1, \ldots).$$

URM-compound statement for composition $g \circ f$: $P_f; P_g$.

States and Transformations – Powers

Powers of partial function $f \in E(\Omega)$ are inductively defined,

$$f^0 = \mathrm{id}_\Omega, \quad \text{and} \quad f^{n+1} = f \circ f^n,\ n \in \mathbb{N}_0. \tag{16}$$

In particular,

$$f^1 = f \circ \mathrm{id}_\Omega = f.$$

The setting

$$f^{n+1} = f^n \circ f,\ n \in \mathbb{N}_0,$$

is equally applicable, since composition is associative.

E.g., n-th power of increment function a_k, $k \in \mathbb{N}_0$,

$$a_k^n(\omega) = (w_0, \ldots, w_{k-1}, w_k + n, w_{k+1}, \ldots), \quad n \in \mathbb{N}_0.$$

States and Transformations – Iteration

- *Iteration* of partial function $f \in E(\Omega)$ with initial state $\omega \in \Omega$ gives the sequence of states

$$\omega,\ f(\omega),\ f^2(\omega),\ f^3(\omega), \ldots. \tag{17}$$

- If f is total, each power f^n will be total and so all states $f^n(\omega)$ in the sequence (18) are defined.

States and Transformations – Iteration (Tricky)

- Idea is to iterate the sequence of states

$$\omega,\ f(\omega),\ f^2(\omega),\ f^3(\omega),\ldots. \tag{18}$$

  until register R_k becomes 0.

- *Iteration* of function $f \in E(\Omega)$ w.r.t. k-th register defines a new function $f^{*k} \in E(\Omega)$ such that
  - $f^{*k}(\omega) = f^n(\omega)$ if register R_k is 0 in state $f^n(\omega)$ and $n \geq 0$ is minimal.
  - $f^{*k}(\omega) = \uparrow$ (undefined) if
    - sequence (18) runs ad infinitum and R_k never becomes 0, or
    - sequence (18) aborts since $f^j(\omega)$ is undefined before R_k could become 0.

States and Transformations – Iteration

- Implementation of iteration as `while` loop:

Require: $\omega \in \Omega$
  while $\omega_k > 0$ **do**
    $w \leftarrow f(\omega)$
  end while

- URM-compound statement for iteration f^{*k}: $(P_f)k$.
- Iteration can be considered as unlimited search process.

States and Transformations – Example

In view of increment function $f = a_k$ and k-th register,

$$a_k^{*k}(\omega) = \begin{cases} \omega & \text{if } \omega_k = 0, \\ \uparrow & \text{otherwise.} \end{cases} \tag{19}$$

Take n-th power of increment function a_k,

$$a_k^n(\omega) = (w_0, \ldots, w_{k-1}, w_k + n, w_{k+1}, \ldots), \quad n \in \mathbb{N}_0.$$

Case $\omega_k = 0$:

$$(\omega_0, \ldots, \omega_{k-1}, 0, \omega_{k+1}, \ldots) \mapsto (\omega_0, \ldots, \omega_{k-1}, 0, \omega_{k+1}, \ldots).$$

Case $\omega_k \neq 0$ (runs ad infinitum):

$$\begin{aligned} &(\omega_0, \ldots, \omega_{k-1}, \omega_k, \omega_{k+1}, \ldots) \\ \mapsto \quad &(\omega_0, \ldots, \omega_{k-1}, \omega_k + 1, \omega_{n+1}, \ldots) \\ \mapsto \quad &(\omega_0, \ldots, \omega_{k-1}, \omega_k + 2, \omega_{k+1}, \ldots) \ldots \end{aligned}$$

Syntax and Semantics – Contents

- Alphabet
- Syntax of URM programs
- Semantics of URM programs

Syntax – Alphabet

- Set of (decimal) numbers over the alphabet of digits $\Sigma_{10} = \{0, 1, \ldots, 9\}$ is defined as

$$Z = (\Sigma_{10} \setminus \{0\})\Sigma_{10}^{+} \cup \Sigma_{10}, \tag{20}$$

  e.g., 0, 10010, 20000, but not 0111 (no leading 0's).

- URM programs are words over the alphabet

$$\Sigma_{\mathrm{URM}} = \{A, S, (,), ;\} \cup Z. \tag{21}$$

Syntax – URM Programs

Inductive definition of the set of *URM programs* $\mathcal{P}_{\mathrm{URM}}$:

- *Increment*: $A\sigma \in \mathcal{P}_{\mathrm{URM}}$ for each $\sigma \in Z$.
- *Decrement*: $S\sigma \in \mathcal{P}_{\mathrm{URM}}$ for each $\sigma \in Z$.
- *Composition*: If $P, Q \in \mathcal{P}_{\mathrm{URM}}$, then $P;Q \in \mathcal{P}_{\mathrm{URM}}$.
- *Iteration*: If $P \in \mathcal{P}_{\mathrm{URM}}$ and $\sigma \in Z$, then $(P)\sigma \in \mathcal{P}_{\mathrm{URM}}$.

n-fold composition of URM program $P \in \mathcal{P}_{\mathrm{URM}}$,

$$P^n = P; P; \ldots; P \quad (n \text{ times}),\ n \geq 0. \tag{22}$$

Syntax – Example

Well-formed URM programs:

- $A1$
- $(A1)1$
- $(A1)2$
- $A1; (A1)1$
- $(S1)1$
- $(A1; S2)2$

Semantics

Semantics of URM programs is a mapping

$$|\cdot| : \mathcal{P}_{\mathrm{URM}} \to E(\Omega)$$

defined inductively as follows:

- *Increment*: $|A\sigma| = a_\sigma$ for each $\sigma \in Z$.
- *Decrement*: $|S\sigma| = s_\sigma$ for each $\sigma \in Z$.
- *Composition*: If $P, Q \in \mathcal{P}_{\mathrm{URM}}$, then $|P;Q| = |Q| \circ |P|$.
- *Iteration*: If $P \in \mathcal{P}_{\mathrm{URM}}$ and $\sigma \in Z$, then $|(P)\sigma| = |P|^{*\sigma}$.

Interpretation of URM programs as functions in $E(\Omega)$.

$|(P)\sigma|$ means that P is executed until register R_σ becomes zero.

Semantics – Example

Trace of addition program $P_+ = (A1; S2)2$:

R_0	R_1	R_2	R_3	$\ldots$	Registers
0	3	2	0	$\ldots$	Init.
0	4	2	0	$\ldots$	$A1$
0	4	1	0	$\ldots$	$S2$
0	5	1	0	$\ldots$	$A1$
0	5	0	0	$\ldots$	$S2$

$|P_+| : \Omega \to \Omega$ is given by

$$|P_+| : (\omega_0, \omega_1, \omega_2, \omega_3, \ldots) \mapsto (\omega_0, \omega_1 + \omega_2, 0, \omega_3, \ldots).$$

URM Computable Functions – Contents

- URM computability
- Computation of functions
- Examples
- Normalization

URM Computable Functions – Computability

- Partial function $f \in E(\Omega)$ is *URM computable* if there is an URM program P such that

$$|P| = f. \tag{23}$$

- $\mathcal{P}_{\mathrm{URM}}$ is countable, while the set $E(\Omega)$ is not, i.e., there are partial functions in $E(\Omega)$ that are not URM computable.
- $\mathcal{F}_{\mathrm{URM}}$ denotes the class of all partial functions that are URM computable.
- $\mathcal{T}_{\mathrm{URM}}$ depicts the class of all total functions which are URM computable. Then $\mathcal{T}_{\mathrm{URM}} \subset \mathcal{F}_{\mathrm{URM}}$.

URM Computable Functions – Functions

- Calculation of function $f : \mathbb{N}_0^k \to \mathbb{N}_0^m$ by URM program requires to load the registers with initial values and to read out the result.
- Define load function

$$\alpha_k : \mathbb{N}_0^k \to \Omega : (x_1, \ldots, x_k) \mapsto (0, x_1, \ldots, x_k, 0, 0, \ldots) \quad (24)$$

and result function

$$\beta_m : \Omega \to \mathbb{N}_0^m : (\omega_0, \omega_1, \omega_2, \ldots,) \mapsto (\omega_1, \omega_2, \ldots, \omega_m). \quad (25)$$

URM Computable Functions – Functions

- Given URM program P and integers $k, m \in \mathbb{N}_0$, define the partial function $\|P\|_{k,m} : \mathbb{N}_0^k \to \mathbb{N}_0^m$ by the composition

$$\|P\|_{k,m} = \beta_m \circ |P| \circ \alpha_k. \tag{26}$$

  $f = \|P\|_{k,m}$ loads argument $\boldsymbol{x} \in \mathbb{N}_0^k$, computes P, and outputs the result $f(\boldsymbol{x}) \in \mathbb{N}_0^m$.

- A function $f : \mathbb{N}_0^k \to \mathbb{N}_0^m$ is *URM computable* if there is an URM program P such that

$$f = \|P\|_{k,m}. \tag{27}$$

URM Computable Functions – Add

Addition of two natural numbers $f_+ : \mathbb{N}_0^2 \to \mathbb{N}_0 : (x, y) \mapsto x + y$ is URM computable.

URM program

$$P_+ = (A1; S2)2 \tag{28}$$

has the trace

R_0	R_1	R_2	R_3	$\ldots$	Registers
0	x	y	0	$\ldots$	Init.
0	$x+1$	y	0	$\ldots$	$A1$
0	$x+1$	$y-1$	0	$\ldots$	$S2$
0	$x+2$	$y-1$	0	$\ldots$	$A1$
0	$x+2$	$y-2$	0	$\ldots$	$S2$
$\ldots$					
0	$x+y-1$	1	0	$\ldots$	$S2$
0	$x+y$	1	0	$\ldots$	$A1$
0	$x+y$	0	0	$\ldots$	$S2$

Proof (Cont'd)

Thus $|P_+| : \Omega \to \Omega$ is given by

$$|P_+| : (\omega_0, \omega_1, \omega_2, \omega_3, \ldots) \mapsto (\omega_0, \omega_1 + \omega_2, 0, \omega_3, \ldots). \tag{29}$$

Hence,

$$\begin{aligned} \|P_+\|_{2,1}(x, y) &= (\beta_1 \circ |P_+| \circ \alpha_2)(x, y) \\ &= (\beta_1 \circ |P_+|)(0, x, y, 0, 0, \ldots) \\ &= \beta_1(0, x + y, 0, 0, \ldots) \\ &= x + y. \end{aligned} \tag{30}$$

URM Computable Functions – Difference

Asymmetric difference (9) is URM computable.

URM program

$$P_{\dot{-}} = (S1; S2)2 \tag{31}$$

has the trace

R_0	R_1	R_2	R_3	$\ldots$	Registers
0	x	y	0	$\ldots$	Init.
$\ldots$					
0	$x - y$	0	0	$\ldots$	$x \geq y$
$\ldots$					
0	0	0	0	$\ldots$	$x < y$

and so yields the function

$$\|P_{\dot{-}}\|_{2,1}(x, y) = x \dot{-} y, \quad x, y \in \mathbb{N}_0. \tag{32}$$

URM Computable Functions – Sign

Sign function $\operatorname{sgn} : \mathbb{N}_0 \to \mathbb{N}_0$ given by

$$\operatorname{sgn}(x) = \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{otherwise,} \end{cases} \tag{33}$$

is URM computable.

Consider the trace

R_0	R_1	R_2	R_3	R_4	$\ldots$	Registers
0	x	0	0	0	$\ldots$	Init.
0	0	x	x	0	$\ldots$	$(A2; A3; S1)1$
0	x	x	0	0	$\ldots$	$(A1; S3)3$
0	x	$x-1$	0	0	$\ldots$	$S2$
0	$\operatorname{sgn}(x)$	0	0	0	$\ldots$	$(S1; S2)2$

URM Computable Functions – Cosign

Cosign function $\operatorname{csg} : \mathbb{N}_0 \to \mathbb{N}_0$ given by

$$\operatorname{csg}(x) = \begin{cases} 1 & \text{if } x = 0, \\ 0 & \text{otherwise,} \end{cases} \tag{34}$$

is URM computable.

Note that

$$\operatorname{csg}(x) = 1 \dot{-} \operatorname{sgn}(x), \quad x \in \mathbb{N}_0. \tag{35}$$

Consider the trace

R_0	R_1	R_2	R_3	$\ldots$	Registers
0	x	0	0	$\ldots$	Init.
0	$\operatorname{sgn}(x)$	0	0	$\ldots$	pgm sgn
0	0	$\operatorname{sgn}(x)$	0	$\ldots$	$(A2; S1)1$
0	1	$\operatorname{sgn}(x)$	0	$\ldots$	$A1$
0	$1 \dot{-} \operatorname{sgn}(x)$	0	0	$\ldots$	pgm $\dot{-}$

URM Computable Functions – Weirdos

- $P = (A1)1$ computes the monadic function $f : \mathbb{N}_0 \to \mathbb{N}_0$ given by (see (19))

$$f(x) = \begin{cases} 0 & \text{if } x = 0, \\ \uparrow & \text{otherwise.} \end{cases} \tag{36}$$

- $P = A1; (A1)1$ provides the monadic function $f_\uparrow : \mathbb{N}_0 \to \mathbb{N}_0$ which is nowhere defined; i.e.,

$$f_\uparrow(x) = \uparrow, \quad x \in \mathbb{N}_0. \tag{37}$$

  $f_\uparrow$ is called the *empty function*.

- $P = (S1)1$ calculates the monadic zero-function $f : \mathbb{N}_0 \to \mathbb{N}_0$, i.e.,

$$f(x) = 0, \quad x \in \mathbb{N}_0. \tag{38}$$

URM Computable Functions – Normalization

Let $f : \mathbb{N}_0^k \to \mathbb{N}_0$ be an URM computable function.

An URM program P with $\|P\|_{k,1} = f$ is *normal* if for all inputs $(x_1, \ldots, x_k) \in \mathbb{N}_0^k$,

$$(|P| \circ \alpha_k)(x_1, \ldots, x_k) = \tag{39}$$
$$\begin{cases} (0, f(x_1, \ldots, x_k), 0, 0, \ldots) & \text{if } (x_1, \ldots, x_k) \in \mathrm{dom}(f), \\ \uparrow & \text{otherwise.} \end{cases}$$

A normal URM-program P computes a function f such that at the end of computation, all registers R_i with $i \neq 1$ are cleared.

URM Computable Functions – Add

URM program for the addition of two numbers

$$P_+ = (A1; S2)2$$

is already normal, since

$$\begin{aligned}(|P_+| \circ \alpha_2)(x, y) &= |P_+|(0, x, y, 0, 0, \ldots) \\ &= (0, x + y, 0, 0, 0, \ldots).\end{aligned}$$

URM Computable Functions – Normalization

For each URM-computable function $f : \mathbb{N}_0^k \to \mathbb{N}_0$ there is a normal URM-program P such that

$$\|P\|_{k,1} = f. \tag{40}$$

Proof.

Let Q be an URM-program such that $\|Q\|_{k,1} = f$.
Suppose R_σ is the largest register that contains a non-zero value in the final state of computation. Then the corresponding normal URM-program is given by

$$P = Q; (S0)0; (S2)2; \ldots; (S\sigma)\sigma. \tag{41}$$

Statement $(Si)i$ sets the value of the i-th register to zero. □

Skills

- Write and interpret an URM program
- Normalize an URM program

See the parser for URM programming.

Part III

Primitive Recursive Functions

Primitive Recursive Functions – Motivation

- Subclass of computable total functions $f : \mathbb{N}_0^k \to \mathbb{N}_0^m$ containing the arithmetic functions.
- Not contained are very fast growing total functions (superpowers).
- Inductive definition (base functions, composition, primitive recursion).
- LOOP programs as programming model for primitive recursive functions.

E.g., addition of two numbers by primitive recursion:

$$\begin{aligned} f_+(x,0) &= x \\ f_+(x,y+1) &= (\nu \circ \pi_3^{(3)})(x,y,f_+(x,y)). \end{aligned}$$

Primitive Recursive Functions

- Definition of primitive recursive functions
- Peano structures
- Closure properties
- Primitive recursive sets
- LOOP programs

Reading: Zimmermann, Chapter 2

Primitive Recursive Functions

- Primitive recursive functions form an important building block on the way to an abstract formalization of computability.
- Most of the functions studied in arithmetics are primitive recursive such as addition and multiplication.
- From the programming point of view, the primitive recursive functions can be implemented via `for` loops (bounded search process).

Primitive Recursive Functions – Contents

- Basic functions
- Composition
- Primitive Recursion
- URM computability

Primitive Recursive Functions – Basic Functions

The class of primitive recursive functions $\mathcal{P}$ is inductively defined.

Basic functions:

- *0-adic constant function* (constant 0)

$$c_0^{(0)} : \to \mathbb{N}_0 : \mapsto 0. \tag{42}$$

- *Monadic constant function*

$$c_0^{(1)} : \mathbb{N}_0 \to \mathbb{N}_0 : x \mapsto 0. \tag{43}$$

- *Successor function*

$$\nu : \mathbb{N}_0 \to \mathbb{N}_0 : x \mapsto x + 1. \tag{44}$$

- *Projection functions* for $n \geq 1$ and $1 \leq k \leq n$,

$$\pi_k^{(n)} : \mathbb{N}_0^n \to \mathbb{N}_0 : (x_1, \ldots, x_n) \mapsto x_k. \tag{45}$$

Primitive Recursive Functions – Composition I

If $g : \mathbb{N}_0^n \to \mathbb{N}_0^m$ and $h : \mathbb{N}_0^m \to \mathbb{N}_0^k$ lie in $\mathcal{P}$, the *composition*

$$f = (h \circ g) : \mathbb{N}_0^n \to \mathbb{N}_0^k \tag{46}$$

lies in $\mathcal{P}$, where for all $\boldsymbol{x} \in \mathbb{N}_0^n$,

$$f(\boldsymbol{x}) = (h \circ g)(\boldsymbol{x}) = h(g(\boldsymbol{x})). \tag{47}$$

E.g., if $g = \pi_1^{(n)}$ and $h = \nu$, then

$$f = h(g(\boldsymbol{x})) = h(x_1) = x_1 + 1, \quad \boldsymbol{x} \in \mathbb{N}_0^n.$$

Primitive Recursive Functions – Composition II

If $g_1, \ldots, g_m : \mathbb{N}_0^n \to \mathbb{N}_0$ and $h : \mathbb{N}_0^m \to \mathbb{N}_0^k$ lie in $\mathcal{P}$, the *composition*

$$f = h(g_1, \ldots, g_m) : \mathbb{N}_0^n \to \mathbb{N}_0^k \tag{48}$$

lies in $\mathcal{P}$, where for all $\boldsymbol{x} \in \mathbb{N}_0^n$,

$$f(\boldsymbol{x}) = h(g_1(\boldsymbol{x}), \ldots, g_m(\boldsymbol{x})). \tag{49}$$

If $m = 1$, then (see Composition I)

$$(h(g_1))(\boldsymbol{x}) = h(g_1(\boldsymbol{x})) = (h \circ g_1)(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{N}_0^n. \tag{50}$$

E.g., if $g_1 = \pi_1^{(n)}$, $g_2 = \pi_2^{(n)}$ and $h = +$, then

$$f = h(g_1(\boldsymbol{x}), g_2(\boldsymbol{x})) = h(x_1, x_2) = x_1 + x_2, \quad \boldsymbol{x} \in \mathbb{N}_0^n, n \geq 2.$$

Primitive Recursive Functions – Primitive Recursion

If $g : \mathbb{N}_0^n \to \mathbb{N}_0$ and $h : \mathbb{N}_0^{n+2} \to \mathbb{N}_0$ lie in $\mathcal{P}$, the *primitive recursion*

$$f = \mathrm{pr}(g, h) : \mathbb{N}_0^{n+1} \to \mathbb{N}_0 \tag{51}$$

lies in $\mathcal{P}$, where for all $\boldsymbol{x} \in \mathbb{N}_0^n$, $y \in \mathbb{N}_0$,

$$\begin{aligned} f(\boldsymbol{x}, 0) &= g(\boldsymbol{x}), & (52)\\ f(\boldsymbol{x}, y+1) &= h(\boldsymbol{x}, y, f(\boldsymbol{x}, y)). & (53) \end{aligned}$$

E.g., if $g = c_0^{(1)}$ and $h = \nu \circ \pi_3^{(3)}$ $(n = 1)$, then

$$\begin{aligned} f(x, 0) &= g(x) = 0,\\ f(x, y+1) &= h(x, y, f(x, y))\\ &= (\nu \circ \pi_3^{(3)})(x, y, f(x, y))\\ &= \nu(f(x, y)) = f(x, y) + 1. \end{aligned}$$

Primitive Recursive Functions – Class $\mathcal{P}$

The class of *primitive recursive functions* $\mathcal{P}$ contains the basic functions and is closed under composition and primitive recursion (Richard Dedekind, 1831-1916).

Well-definedness and uniqueness of function $f = \mathrm{pr}(g, h)$ defined by primitive recursion can be established via Peano structures.

Primitive Recursive Functions – Add

Addition is primitive recursive.

The addition function

$$f_+ : \mathbb{N}_0^2 \to \mathbb{N}_0 : (x, y) \mapsto x + y.$$

can be described by primitive recursion, i.e., for all $x, y \in \mathbb{N}_0$,

$$\begin{aligned} f_+(x, 0) &= x \\ &= \pi_1^{(1)}(x), \\ f_+(x, y+1) &= f_+(x, y) + 1 \\ &= \nu(f_+(x, y)) \\ &= (\nu \circ \pi_3^{(3)})(x, y, f_+(x, y)) \end{aligned}$$

and so $f_+ = \mathrm{pr}(g, h)$ with $g = \pi_1^{(1)} = \mathrm{id}_{\mathbb{N}_0}$ and $h = \nu \circ \pi_3^{(3)}$.

Primitive Recursive Functions – Totality

Each primitive recursive function is total.

Proof (Sketch)

- The basic functions are total.
- Let $f = h(g_1, \ldots, g_m)$. By induction, the functions $g_1, \ldots, g_m$ and h are total. Then it follows that the composition f is total.
- Let $f = \mathrm{pr}(g, h)$. By induction, the functions g and h are total. Then it follows that the function f is total. □

Primitive Recursive Functions – URM Computability

Each primitive recursive function is URM computable.

Proof (Sketch)

- The basic functions are URM computable.
- Let $f = h(g_1, \ldots, g_m)$. By induction, there are normal URM programs P_h and $P_{g_1}, \ldots, P_{g_m}$ such that $\|P_h\|_{m,k} = h$ and $\|P_{g_i}\|_{n,1} = g_i$, $1 \leq i \leq m$.

  Then there is a normal URM program P such that $\|P\|_{n,k} = f$.
- Let $f = \mathrm{pr}(g, h)$. By induction, there are normal URM programs P_g and P_h such that $\|P_g\|_{n,1} = g$ and $\|P_h\|_{n+2,1} = h$.

  Then there is a normal URM program P such that $\|P\|_{n+1,1} = f$. □

Primitive Recursive Functions – Function Classes

- $\mathcal{T}_{\mathrm{URM}}$ denotes the class of all *total* URM computable functions. Since each primitive recursive function is total and URM computable,

$$\mathcal{P} \subset \mathcal{T}_{\mathrm{URM}} \tag{54}$$

  with $A \in \mathcal{T}_{\mathrm{URM}} \setminus \mathcal{P}$ (Ackermann function, superpower).

- $\mathcal{F}_{\mathrm{URM}}$ denotes the class of URM computable functions. Then

$$\mathcal{T}_{\mathrm{URM}} \subset \mathcal{F}_{\mathrm{URM}} \tag{55}$$

  with $f_{\uparrow} = \|A1; (A1)1\|_{1,1} \in \mathcal{F}_{\mathrm{URM}} \setminus \mathcal{T}_{\mathrm{URM}}$ (nowhere defined function).

- Composition and primitive recursion can be defined for partial functions as well.

Peano Structures – Contents

- Semi-Peano structures
- Morphisms
- Peano structures
- Fundamental Lemma
- Uniqueness of Peano structures
- Primitive recursion (addition)

Peano Structures – Semi-Peano

A *semi-Peano structure* is a triple

$$\mathcal{A} = (A, \alpha, a),$$

where A is a non-empty set, $\alpha : A \to A$ is an operation, and $a \in A$ (distinguished element).

Examples

- $(\mathbb{N}_0, \nu, x_0)$ is a semi-Peano structure, where $\nu : \mathbb{N}_0 \to \mathbb{N}_0 : n \mapsto n + 1$ and $x_0 \in \mathbb{N}_0$.
- $(\Sigma^*, \circ_a, \epsilon)$ is a semi-Peano structure, where Σ is an alphabet, $\circ_a : \Sigma^* \to \Sigma^* : w \mapsto wa$ is the concatenation of a word w with a fixed letter $a \in \Sigma$, and $\epsilon \in \Sigma$ (empty word).

Peano Structures – Morphisms

Let $\mathcal{A} = (A, \alpha, a)$ and $\mathcal{B} = (B, \beta, b)$ be semi-Peano structures.

A mapping $\phi : A \to B$ is a *morphism* if

- ϕ assigns the distinguished elements

$$\phi(a) = b, \tag{56}$$

- ϕ commutes with the monadic operations,

$$\beta \circ \phi = \phi \circ \alpha, \tag{57}$$

i.e., the following diagram is commutative:

$$\begin{array}{ccc} A & \xrightarrow{\alpha} & A \\ \downarrow{\scriptstyle \phi} & & \downarrow{\scriptstyle \phi} \\ B & \xrightarrow{\beta} & B \end{array}$$

Peano Structures – Example

Consider the semi-Peano structures

$$(\mathbb{N}_0, \nu, 0) \quad \text{and} \quad (\Sigma^*, \circ_a, \epsilon),\ a \in \Sigma,$$

where ϵ denotes the empty word.

The mapping

$$\phi : \Sigma^* \to \mathbb{N}_0 : w \mapsto |w|$$

which assigns to each word $w \in \Sigma^*$ its length $|w|$ is a morphism, since

$$\phi(\epsilon) = |\epsilon| = 0$$

and for each $w \in \Sigma^*$,

$$\phi(\circ_a(w)) = \phi(wa) = |wa| = |w| + |a| = |w| + 1 = \nu(\phi(w)).$$

Peano Structures

A *Peano structure* is a semi-Peano structure $\mathcal{A} = (A, \alpha, a)$ such that

- α is injective,
- $a \notin \operatorname{ran}(\alpha)$, and
- A fulfills the *induction axiom:*

  If $T \subseteq A$ such that $a \in T$ and $\alpha(x) \in T$ whenever $x \in T$, then $T = A$.

Peano Structures

If $\mathcal{A} = (A, \alpha, a)$ is a Peano structure, then

$$A = \{\alpha^n(a) \mid n \in \mathbb{N}_0\}. \tag{58}$$

Proof.

Let $T = \{\alpha^n(a) \mid n \in \mathbb{N}_0\}$.

- We have $a = \alpha^0(a) \in T$.
- Let $x \in T$. Then $x = \alpha^n(a)$ for some $n \geq 0$. Thus $\alpha(x) = \alpha(\alpha^n(a)) = \alpha^{n+1}(a) \in T$.

Hence by the induction axiom, $T = A$. □

Peano Structures

If $\mathcal{A} = (A, \alpha, a)$ is a Peano structure, then for all $m, n \in \mathbb{N}_0$,

$$m \neq n \quad \Longrightarrow \quad \alpha^m(a) \neq \alpha^n(a). \tag{59}$$

Proof.

Define T as the set of all elements $\alpha^m(a)$ such that $\alpha^m(a) \neq \alpha^n(a)$ for all $n \in \mathbb{N}_0$ with $n > m$.

- Suppose that $\alpha^n(a) = \alpha^0(a) = a$ for some $n > 0$. Then $a \in \operatorname{ran}(\alpha)$ contradicting the definition. Thus $a \in T$.
- Let $x \in T$. Then $x = \alpha^m(a)$ for some $m \geq 0$. Suppose that $\alpha(x) = \alpha^{m+1}(x) \notin T$. Then there is a number $n > m$ such that $\alpha^{m+1}(a) = \alpha^{n+1}(a)$. But α is injective and so $x = \alpha^m(a) = \alpha^n(a)$ contradicting the hypothesis for x. Thus $\alpha(x) \in T$.

Hence by the induction axiom, $T = A$. □

Peano Structures – Fundamental Lemma

If $\mathcal{A} = (A, \alpha, a)$ is a Peano structure and $\mathcal{B} = (B, \beta, b)$ is a semi-Peano structure, there is a unique morphism $\phi : A \to B$.

*Proof

We have $A = \{\alpha^n(a) \mid n \in \mathbb{N}_0\}$.

- *Existence:* Define $\phi(\alpha^n(a)) = \beta^n(b)$ for all $n \in \mathbb{N}_0$. Then $\phi(a) = \phi(\alpha^0(a)) = \beta^0(b) = b$. Moreover, let $x \in A$. Then $x = \alpha^m(a)$ for some $m \in \mathbb{N}_0$ and so $(\phi \circ \alpha)(x) = \phi(\alpha^{m+1}(a)) = \beta^{m+1}(b) = \beta(\beta^m(b)) = \beta(\phi(\alpha^m(a))) = (\beta \circ \phi)(x)$. Thus ϕ is a morphism.
- *Uniqueness:* Suppose there is another morphism $\psi : A \to B$. Define $T = \{x \in A \mid \phi(x) = \psi(x)\}$. Then $\phi(a) = b = \psi(a)$ and so $a \in T$. Moreover, let $x \in T$. Then $\phi(\alpha(x)) = (\phi \circ \alpha)(x) = (\beta \circ \phi)(x) = (\beta \circ \psi)(x) = (\psi \circ \alpha)(x) = \psi(\alpha(x))$ and so $\alpha(x) \in T$. Hence by the induction axiom, $T = A$ and so $\phi = \psi$. □

Peano Structures – Guiseppe Peano, 1858-1932

$(\mathbb{N}_0, \nu, 0)$ is a Peano structure, where $\nu : \mathbb{N}_0 \to \mathbb{N}_0 : n \mapsto n + 1$.

Since $\mathbb{N}_0 = \{0, \nu(0), \nu^2(0), \nu^3(0), \ldots\}$, define

$$1 := \nu(0),\ 2 := \nu^2(0),\ 3 := \nu^3(0), \ldots.$$

Then $\mathbb{N}_0 = \{0, 1, 2, 3, \ldots\}$.

Peano Structures – Richard Dedekind (1831-1916)

Every Peano structure is isomorphic to $(\mathbb{N}_0, \nu, 0)$.

Proof.

- $(\mathbb{N}_0, \nu, 0)$ is a Peano structure.
- Let $\mathcal{A} = (A, \alpha, a)$ and $\mathcal{B} = (B, \beta, b)$ be Peano structures. By the Fundamental Lemma, there are morphisms $\phi : A \to B$ and $\psi : B \to A$. Since the composition of morphisms is also a morphism, $\psi \circ \phi : A \to A$ and $\phi \circ \psi : B \to B$ are morphisms.
- The identity mappings $\mathrm{id}_A : A \to A : x \mapsto x$ and $\mathrm{id}_B : B \to B : x \mapsto x$ are morphisms.
- By the uniqueness of morphisms, $\psi \circ \phi = \mathrm{id}_A$ and $\phi \circ \psi = \mathrm{id}_B$.
- Hence, ϕ and ψ are isomorphisms (bijective morphisms).

Peano Structures – Addition

There is a unique total function $f_+ : \mathbb{N}_0^2 \to \mathbb{N}_0$ such that for all $x, y \in \mathbb{N}_0$,

$$f_+(x, 0) = x, \tag{60}$$
$$f_+(x, y+1) = f_+(x, y) + 1. \tag{61}$$

Proof

- $(\mathbb{N}_0, \nu, 0)$ is a Peano structure.
- Consider the semi-Peano structure $(\mathbb{N}_0, \nu, x)$, where $x \in \mathbb{N}_0$.

  By the Fundamental Lemma, there is a **unique morphism** $f_x : \mathbb{N}_0 \to \mathbb{N}_0$ such that for all $y \in \mathbb{N}_0$,

$$\begin{aligned} f_x(0) &= x, \\ f_x(y+1) &= (f_x \circ \nu)(y) \overset{\text{morph}}{=} (\nu \circ f_x)(y) = f_x(y) + 1. \end{aligned}$$

Proof (Cont'd)

- For all $x, y \in \mathbb{N}_0$,

$$\begin{aligned} f_x(0) &= x, \\ f_x(y+1) &= f_x(y) + 1. \end{aligned}$$

- Define the function $f_+ : \mathbb{N}_0^2 \to \mathbb{N}_0$ by setting for all $x, y \in \mathbb{N}_0$,

$$f_+(x, y) = f_x(y).$$

Then for all $x, y \in \mathbb{N}_0$,

$$\begin{aligned} f_+(x, 0) &= f_x(0) = x, \\ f_+(x, y+1) &= f_x(y+1) = f_x(y) + 1 = f_+(x, y) + 1. \end{aligned}$$

Proof (Cont'd)

- For all $x, y \in \mathbb{N}_0$,

$$f_+(x, y) = x + y.$$

Indeed, by induction, for all $x, y \in \mathbb{N}_0$,

$$f_+(x, 0) = f_x(0) = x$$

and

$$f_+(x, y + 1) = f_x(y + 1) = f_x(y) + 1 = x + y + 1.$$

Closure Properties – Contents

- Transformation of variables
- Parametrization
- Definition by cases
- Bounded sum and product
- Iteration
- Bounded minimalization

Transformation of Variables

Given $\phi = \begin{pmatrix} 1 & \dots & n \\ \phi(1) & \dots & \phi(n) \end{pmatrix} : [n] \to [m]$ with $n, m \geq 1$ and total function $f : \mathbb{N}_0^n \to \mathbb{N}_0$.

Transformation of variables is the function

$$f^\phi : \mathbb{N}_0^m \to \mathbb{N}_0 : (x_1, \dots, x_m) \mapsto f(x_{\phi(1)}, \dots, x_{\phi(n)}), \tag{62}$$

i.e., $x_{\phi(1)}, \dots, x_{\phi(n)}$ is a list of variables from $x_1, \dots, x_m$.

Example

Let $\phi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 1 & 1 & 3 \end{pmatrix} : [5] \to [3]$. Then

$$f^\phi(x_1, x_2, x_3) = f(x_2, x_3, x_1, x_1, x_3).$$

Transformation of Variables

If f is primitive recursive, then f^ϕ is primitive recursive.

Proof.

Transformation of variables can be described by composition,

$$f^\phi = f(\pi^{(m)}_{\phi(1)}, \ldots, \pi^{(m)}_{\phi(n)}), \tag{63}$$

i.e., for all $x_1, \ldots, x_m \in \mathbb{N}_0$,

$$\begin{aligned} f(\pi^{(m)}_{\phi(1)}, \ldots, \pi^{(m)}_{\phi(n)})(x_1, \ldots, x_m) &= f(x_{\phi(1)}, \ldots, x_{\phi(n)}) \\ &= f^\phi(x_1, \ldots, x_m). \end{aligned}$$

Transformation of Variables – Example

Two important special cases:

- Permutation of variables:

$$f^{\phi} : (x_1, x_2) \mapsto f(x_2, x_1), \tag{64}$$

where $\phi = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} : [2] \to [2].$

- Identification of variables:

$$f^{\phi} : x_1 \mapsto f(x_1, x_1), \tag{65}$$

where $\phi = \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix} : [2] \to [1].$

Constant Functions

The k-adic constant function with value $i \in \mathbb{N}_0$ is given by

$$c_i^{(k)} : \mathbb{N}_0^k \to \mathbb{N}_0 : (x_1, \ldots, x_k) \mapsto i. \tag{66}$$

$c_i^{(k)}$ is primitive recursive.

Proof.

- If $k = 0$, $c_i^{(0)} = \nu^i \circ c_0^{(0)}$.
- If $k > 0$, $c_i^{(k)} = \nu^i \circ c_0^{(1)} \circ \pi_1^{(k)}$, i.e., for all $\boldsymbol{x} \in \mathbb{N}_0^k$,

$$\begin{aligned} c_i^{(k)}(\boldsymbol{x}) &= (\nu^i \circ c_0^{(1)} \circ \pi_1^{(k)})(\boldsymbol{x}) \\ &= (\nu^i \circ c_0^{(1)})(x_1) \\ &= \nu^i(0) = i. \end{aligned}$$

Parametrization

Given total function $f : \mathbb{N}_0^n \to \mathbb{N}_0$, integer m with $1 \leq m < n$ and $\boldsymbol{a} = (a_1, \ldots, a_m) \in \mathbb{N}_0^m$.

Parametrization of f by $\boldsymbol{a}$ is the function

$$\begin{aligned} f_{\boldsymbol{a}} \quad : \quad & \mathbb{N}_0^{n-m} \to \mathbb{N}_0 \\ & (x_1, \ldots, x_{n-m}) \mapsto f(x_1, \ldots, x_{n-m}, a_1, \ldots, a_m). \end{aligned} \tag{67}$$

Example

If $f : \mathbb{N}_0^5 \to \mathbb{N}_0$ and $a = (101, 2020)$, then

$$f_a(x_1, x_2, x_3) = f(x_1, x_2, x_3, 101, 2020).$$

Parametrization

If f is primitive recursive, then $f_{\boldsymbol{a}}$ is primitive recursive.

Proof.

Parametrization can be described by composition,

$$f_{\boldsymbol{a}} = f(\pi_1^{(n-m)}, \ldots, \pi_{n-m}^{(n-m)}, c_{a_1}^{(n-m)}, \ldots, c_{a_m}^{(n-m)}), \tag{68}$$

i.e., for all $x_1, \ldots, x_{n-m} \in \mathbb{N}_0$,

$$\begin{aligned} & f(\pi_1^{(n-m)}, \ldots, \pi_{n-m}^{(n-m)}, c_{a_1}^{(n-m)}, \ldots, c_{a_m}^{(n-m)})(x_1, \ldots, x_{n-m}) \\ = \;& f(\pi_1^{(n-m)}, \ldots, \pi_{n-m}^{(n-m)}, a_1, \ldots, a_m)(x_1, \ldots, x_{n-m}) \\ = \;& f(x_1, \ldots, x_{n-m}, a_1, \ldots, a_m) \\ = \;& f_{\boldsymbol{a}}(x_1, \ldots, x_{n-m}). \end{aligned}$$

Definition by Cases

Define a total function $f : \mathbb{N}_0^k \to \mathbb{N}_0$ by *case distinction*,

$$f : \boldsymbol{x} \mapsto \begin{cases} g_1(\boldsymbol{x}) & \text{if } \boldsymbol{x} \in A, \\ g_2(\boldsymbol{x}) & \text{if } \boldsymbol{x} \notin A, \end{cases} \tag{69}$$

where $A \subseteq \mathbb{N}_0^k$ and $g_1, g_2 : \mathbb{N}_0^k \to \mathbb{N}_0$ are total functions.

Primitive Sets (Interlude)

Let $A \subseteq \mathbb{N}_0^k$.

- A has the *characteristic function*

$$\chi_A : \mathbb{N}_0^k \to \mathbb{N}_0 : \boldsymbol{x} \mapsto \begin{cases} 1 & \text{if } \boldsymbol{x} \in A, \\ 0 & \text{otherwise,} \end{cases} \tag{70}$$

  i.e., for all $\boldsymbol{x} \in \mathbb{N}_0^k$,

$$\boldsymbol{x} \in A \iff \chi_A(\boldsymbol{x}) = 1. \tag{71}$$

- A is *primitive* if χ_A is primitive recursive.
- If $A \subseteq \mathbb{N}_0^k$ is primitive, then $\bar{A} = \mathbb{N}_0^k \setminus A$ is primitive, since

$$\chi_{\bar{A}} = 1 \dot{-} \chi_A \tag{72}$$

  is primitive recursive.
- E.g., $\mathbb{N} = \mathbb{N}_0 \setminus \{0\}$ is primitive, since $\chi_{\mathbb{N}} = \operatorname{sgn}$ (sign function) is primitive recursive.

Definition by Cases

If $A \subseteq \mathbb{N}_0^k$ is primitive and $g_1, g_2 : \mathbb{N}_0^k \to \mathbb{N}_0$ are primitive recursive, then $f : \mathbb{N}_0^k \to \mathbb{N}_0$ is primitive recursive.

Proof.

We have

$$f = (\chi_A \cdot g_1) + (\chi_{\bar{A}} \cdot g_2). \tag{73}$$

Indeed, for each $\boldsymbol{x} \in \mathbb{N}_0^k$,

$$f(\boldsymbol{x}) \quad = \quad \chi_A(\boldsymbol{x}) \cdot g_1(\boldsymbol{x}) + \chi_{\bar{A}}(\boldsymbol{x}) \cdot g_2(\boldsymbol{x}).$$

If $\boldsymbol{x} \in A$, then $f(\boldsymbol{x}) = g_1(\boldsymbol{x})$; otherwise, $f(\boldsymbol{x}) = g_2(\boldsymbol{x})$.

Moreover, f is a composition of primitive recursive functions and thus primitive recursive. □

Definition by Cases – Example

Consider the function

$$f : \mathbb{N}_0 \to \mathbb{N}_0 : x \mapsto \begin{cases} 2x & \text{if } x > 0, \\ 3 & \text{otherwise.} \end{cases}$$

Then

- $A = \mathbb{N} = \mathbb{N}_0 \setminus \{0\}$ with $\chi_A = \mathrm{sgn}$,
- $\bar{A} = \{0\}$ with $\chi_{\bar{A}} = 1 \dot{-} \mathrm{sgn} = \mathrm{csg}$,
- $g_1 : \mathbb{N}_0 \to \mathbb{N}_0 : x \mapsto 2x$,
- $g_2 : \mathbb{N}_0 \to \mathbb{N}_0 : x \mapsto 3$.

Thus for each $x \in \mathbb{N}_0$,

$$\begin{aligned} f(x) &= \chi_A(x) \cdot g_1(x) + \chi_{\bar{A}}(x) \cdot g_2(x) \\ &= \mathrm{sgn}(x) \cdot 2x + \mathrm{csg}(x) \cdot 3. \end{aligned}$$

Bounded Sum and Product

Let $f : \mathbb{N}_0^{n+1} \to \mathbb{N}_0$ be a total function.

- *Bounded sum* of f

$$f_\Sigma : \mathbb{N}_0^{n+1} \to \mathbb{N}_0 : (\boldsymbol{x}, y) \mapsto \sum_{i=0}^{y} f(\boldsymbol{x}, i). \tag{74}$$

- *Bounded product* of f

$$f_\Pi : \mathbb{N}_0^{n+1} \to \mathbb{N}_0 : (\boldsymbol{x}, y) \mapsto \prod_{i=0}^{y} f(\boldsymbol{x}, i). \tag{75}$$

Bounded Sum and Product

If f is primitive recursive, then f_Σ and f_Π are primitive recursive.

Proof.

The function f_Σ (case $n = 1$) satisfies

$$\begin{aligned} f_\Sigma(x, 0) &= f(x, 0), \\ f_\Sigma(x, y+1) &= f_\Sigma(x, y) + f(x, y+1). \end{aligned}$$

This is a primitive recursive scheme $f_\Sigma = \mathrm{pr}(g, h)$, where

$$\begin{aligned} g(x) &= f(x, 0), \\ h(x, y, f_\Sigma(x, y)) &= +(\pi_3^{(3)}, f(\pi_1^{(3)}, \nu \circ \pi_2^{(3)}))(x, y, f_\Sigma(x, y)). \end{aligned}$$

Similar for function f_Π. □

Bounded Sum – Example

- Identity function

$$f = \pi_1^{(1)} : \mathbb{N}_0 \to \mathbb{N}_0 : x \mapsto x$$

is primitive recursive.

- Bounded sum f_Σ (case $n = 0$),

$$\begin{aligned} f_\Sigma(0) &= 0, \\ f_\Sigma(y) &= \sum_{i=0}^{y} f(i) = \sum_{i=0}^{y} i = \frac{y(y+1)}{2} = \binom{y+1}{2}, \quad y \geq 1, \end{aligned}$$

(binomial coefficient) is also primitive recursive.

Bounded Product – Example

- The function

$$f : \mathbb{N}_0 \to \mathbb{N}_0 : x \mapsto \begin{cases} 1 & \text{if } x = 0, \\ x & \text{otherwise,} \end{cases}$$

is primitive recursive, since it is defined by cases,

$$f(x) = \operatorname{csg}(x) \cdot 1 + \operatorname{sgn}(x) \cdot x, \quad x \in \mathbb{N}_0.$$

- Bounded product f_Π (case $n = 0$),

$$f_\Pi(y) = \prod_{i=0}^{y} f(i) = \prod_{i=1}^{y} i = y!, \quad y \in \mathbb{N}_0,$$

(factorial function) is also primitive recursive.

Iteration

Given a total function $f : \mathbb{N}_0 \to \mathbb{N}_0$.

- *Powers* of function f,

$$f^0 = \mathrm{id}_{\mathbb{N}_0} \quad \text{and} \quad f^{n+1} = f \circ f^n,\ n \geq 0. \tag{76}$$

  In particular, $f^1 = f \circ f^0 = f \circ \mathrm{id}_{\mathbb{N}_0} = f$.

- *Iteration* of function f,

$$f_{\mathrm{it}} : \mathbb{N}_0^2 \to \mathbb{N}_0 : (x, y) \mapsto f^y(x). \tag{77}$$

Iteration can also be defined for multivariate functions.

Iteration

If f is primitive recursive, then f_{it} is primitive recursive.

Proof.

Primitive recursive scheme for f_{it},

$$\begin{aligned} f_{\mathrm{it}}(x,0) &= f^0(x) = x, \\ f_{\mathrm{it}}(x,y+1) &= f^{y+1}(x) = (f \circ f^y)(x) = f(f_{\mathrm{it}}(x,y)) \\ &= (f \circ \pi_3^{(3)})(x,y,f_{\mathrm{it}}(x,y)), \end{aligned}$$

where $f_{\mathrm{it}} = \mathrm{pr}(g,h)$ with $g = \pi_1^{(1)} = \mathrm{id}_{\mathbb{N}_0}$ and $h = f \circ \pi_3^{(3)}$. □

Iteration – Example

- Consider the primitive recursive function

$$f : \mathbb{N}_0 \to \mathbb{N}_0 : x \mapsto 2x.$$

- Iteration of function f,

$$f_{\text{it}} : \mathbb{N}_0^2 \to \mathbb{N}_0 : (x, y) \mapsto f^y(x) = 2^y \cdot x,$$

since by induction

$$f_{\text{it}}(x, 0) = f^0(x) = x = 2^0 x$$

and

$$\begin{aligned} f_{\text{it}}(x, y+1) &= f^{y+1}(x) = (f \circ f^y)(x) \\ &= f(f^y(x)) = f(2^y x) = 2^{y+1} x. \end{aligned}$$

Bounded Minimalization

Given total function $f : \mathbb{N}_0^{k+1} \to \mathbb{N}_0$.

Bounded minimalization of f,

$$\overline{\mu} f : \mathbb{N}_0^{k+1} \to \mathbb{N}_0 : (\boldsymbol{x}, y) \mapsto \overline{\mu} f(\boldsymbol{x}, y), \tag{78}$$

where for each $(\boldsymbol{x}, y) \in \mathbb{N}_0^{k+1}$,

$$\begin{aligned} \overline{\mu} f(\boldsymbol{x}, y) &= \begin{cases} j & \text{if } j = \min\{i \mid 0 \leq i \leq y \wedge f(\boldsymbol{x}, i) = 0\} \text{ exists,} \\ y + 1 & \text{otherwise} \end{cases} \\ &= \text{smallest index } j \text{ with } 0 \leq j \leq y \text{ such that} \\ &\quad f(\boldsymbol{x}, j) = 0; \text{ otherwise } y + 1. \end{aligned}$$

Bounded search process.

Bounded Minimalization – Bounded Search Process

Implementation of $\overline{\mu}f$ by `for` loop:

Require: $\boldsymbol{x} \in \mathbb{N}_0^k$, $y \in \mathbb{N}_0$
  for $i \leftarrow 0, y$ **do**
    if $f(\boldsymbol{x}, i) = 0$ **then**
      return i
    end if
  end for
  return $y + 1$

Kleene notation:

$$\mu(i \leq y)[f(\boldsymbol{x}, i) = 0] = \overline{\mu}f(\boldsymbol{x}, y). \tag{79}$$

"Smallest i with $i \leq y$ such that $f(x, i) = 0$."

Bounded Minimalization

If f is primitive recursive, then $\overline{\mu} f$ is primitive recursive.

*Proof

By definition,

$$\overline{\mu} f(\boldsymbol{x}, 0) = \operatorname{sgn}(f(\boldsymbol{x}, 0))$$

and

$$\overline{\mu} f(\boldsymbol{x}, y+1) = \begin{cases} \overline{\mu} f(x, y) & \text{if } \overline{\mu} f(\boldsymbol{x}, y) \leq y \text{ or } f(\boldsymbol{x}, y+1) = 0, \\ y+2 & \text{otherwise.} \end{cases}$$

This defines a primitive recursive scheme for $\overline{\mu} f$. □

First Case

Two cases for $\overline{\mu} f(x, y+1) = j \leq y+1$:
$j \leq y$: $f(x, j) = 0$, i.e., $\overline{\mu} f(x, y) = j \leq y$.
$j = y+1$: $f(x, y+1) = 0$, i.e., $\overline{\mu}(f, y) = y+1$.

Bounded Minimalization – Example

Consider the asymmetric difference (10),

$$f(x,y) = x \dot{-} y, \quad x,y \in \mathbb{N}_0.$$

Then

$$\overline{\mu} f(x,y) = \begin{cases} x & \text{if } x \leq y, \\ y+1 & \text{otherwise,} \end{cases}$$

since the search process seeking the smallest index j such that

$$f(x,j) = x \dot{-} j = 0$$

is limited to the range $0 \leq j \leq y$.

Example

$\overline{\mu} f(3,5) = 3$ and $\overline{\mu} f(5,3) = 4$.

Bounded Minimalization – *Integral Division

For integers m, n with $m \geq n \geq 1$, there are unique integers $q, r \geq 0$ such that

$$m = q \cdot n + r \quad \wedge \quad 0 \leq r \leq n - 1, \tag{80}$$

with *quotient*

$$q = \dot{-}(m, n) \tag{81}$$

and *remainder*

$$r = m \mod n. \tag{82}$$

E.g., $6 = 3 \cdot 2 + 0$ and $7 = 3 \cdot 2 + 1$.

Bounded Minimalization – *Quotient (Tricky)

Let $m \geq n \geq 1$. Then

$$\begin{aligned} \div(m,n) &= \chi_D(n,m) \cdot \mu(i \leq m)[m \dot{-} i \cdot n = 0] \\ &\quad + (\mathrm{csg} \circ \chi_D)(n,m) \cdot \{\mu(i \leq m)[m \dot{-} i \cdot n = 0] \dot{-} 1\}, \end{aligned} \tag{83}$$

where χ_D is the divisibility relation (93).
Two cases:

- If n divides m, then $\chi_D(n,m) = 1$ and $\mu(i \leq m)[m \dot{-} i \cdot n = 0]$ provides the quotient $\div(m,n)$.

  E.g., if $m = 6$ and $n = 2$, then $\mu(i \leq m)[m \dot{-} i \cdot n = 0]$ yields $i = 3$ and so $\div(6,2) = 3$.

- Otherwise, $\chi_D(n,m) = 0$, i.e., $\mathrm{csg}(\chi_D(n,m)) = 1$, and $\mu(i \leq m)[m \dot{-} i \cdot n = 0] \dot{-} 1$ provides the quotient.

  E.g., if $m = 7$ and $n = 2$, then $\mu(i \leq m)[m \dot{-} i \cdot n = 0]$ yields $i = 4$ and so $\div(7,2) = 4 \dot{-} 1 = 3$.

Primitive Recursive Sets – Contents

- Characteristic function
- Primitive recursive sets
- Set closure
- Divisibility relation
- Set of primes

Primitive Sets

Let $A \subseteq \mathbb{N}_0^k$.

- A has the *characteristic function*

$$\chi_A : \mathbb{N}_0^k \to \mathbb{N}_0 : \boldsymbol{x} \mapsto \begin{cases} 1 & \text{if } \boldsymbol{x} \in A, \\ 0 & \text{otherwise,} \end{cases} \tag{84}$$

  i.e., for all $\boldsymbol{x} \in \mathbb{N}_0^k$,

$$\boldsymbol{x} \in A \iff \chi_A(\boldsymbol{x}) = 1. \tag{85}$$

- A is *primitive* if χ_A is primitive recursive.

Primitive Recursive Sets – Example

- Equality relation

$$R_{=} = \{(x, y) \in \mathbb{N}_0^2 \mid x = y\} \tag{86}$$

is primitive, since $\chi_{R_{=}}(x, y) = \operatorname{csg}(|x - y|)$ with $|x - y| = (x \dot{-} y) + (y \dot{-} x)$ is primitive recursive.

- Inequality relation

$$R_{\neq} = \{(x, y) \in \mathbb{N}_0^2 \mid x \neq y\} \tag{87}$$

is primitive, since $\chi_{R_{\neq}}(x, y) = 1 \dot{-} \chi_{R_{=}}(x, y)$ is primitive recursive.

- Smaller relation

$$R_{<} = \{(x, y) \in \mathbb{N}_0^2 \mid x < y\} \tag{88}$$

is primitive, since $\chi_{R_{<}}(x, y) = \operatorname{sgn}(y \dot{-} x)$ is primitive recursive.

Primitive Recursive Sets - Set Closure

If $A, B \subseteq \mathbb{N}_0^k$ are primitive, then $A \cup B$, $A \cap B$, and $\bar{A} = \mathbb{N}_0^k \setminus A$ are primitive.

Proof.

We have

$$\chi_{A\cup B} = \mathrm{sgn} \circ (\chi_A + \chi_B),\ \chi_{A\cap B} = \chi_A \cdot \chi_B,\ \chi_{\bar{A}} = \mathrm{csg} \circ \chi_A. \quad (89)$$

i.e., for each $\boldsymbol{x} \in \mathbb{N}_0^k$,

$$\begin{aligned} \chi_{A\cup B}(\boldsymbol{x}) &= \mathrm{sgn}(\chi_A(\boldsymbol{x}) + \chi_B(\boldsymbol{x})), && (90)\\ \chi_{A\cap B}(\boldsymbol{x}) &= \chi_A(\boldsymbol{x}) \cdot \chi_B(\boldsymbol{x}), && (91)\\ \chi_{\bar{A}}(\boldsymbol{x}) &= \mathrm{csg}(\chi_A(\boldsymbol{x})). && (92) \end{aligned}$$

The functions on the right-hand sides are compositions of primitive recursive and so are also primitive recursive. □

Primitive Recursive Sets – Divisibility

The divisibility relation

$$D = \{(x, y) \in \mathbb{N}_0^2 \mid x \text{ divides } y\} \tag{93}$$

is primitive.

Proof

- Divisibility relation (x divides y),

$$x \mid y \iff \exists z[0 \leq z \leq y \wedge x \cdot z = y]. \tag{94}$$

- Characteristic function of D,

$$\chi_D(x, y) = \tag{95}$$
$$\operatorname{sgn}[\chi_=(x \cdot 0, y) + \chi_=(x \cdot 1, y) + \ldots + \chi_=(x \cdot y, y)],$$

which is primitive recursive as a composition of primitive recursive functions. Hence, D is primitive. □

Primitive Recursive Sets – Factorization

Consider the sequence of increasing primes

$$(p_0, p_1, p_2, p_3, p_4, \ldots) = (2, 3, 5, 7, 11, \ldots).$$

By the Fundamental Theorem of Arithmetic, each natural number $x \geq 1$ can be uniquely written as a product of prime powers, i.e.,

$$x = \prod_{i=0}^{r-1} p_i^{e_i}, \tag{96}$$

where $e_0, \ldots, e_{r-1} \in \mathbb{N}_0$.

Write $(x)_i = e_i$ for each $i \in \mathbb{N}_0$, and put $(0)_i = 0$ for all $i \in \mathbb{N}_0$.

E.g., $360 = 2^3 \cdot 3^2 \cdot 5^1$ and so $(360)_0 = 3$, $(360)_1 = 2$, $(360)_2 = 1$ and $(360)_i = 0$ for all $i \geq 3$.

Primitive Recursive Sets – Set of Primes

The set of primes P is primitive.

Proof.

By definition,

> $x \in \mathbb{N}_0$ *is prime iff* $x \geq 2$ *and* i *divides* x *implies* $i = 1$ *or* $i = x$ *for all* $i \leq x$.

Thus the characteristic function of P is

$$\chi_P(0) = 0, \quad \chi_P(1) = 0, \quad \chi_P(2) = 1,$$

and for all $x \geq 3$,

$$\chi_P(x) \quad = \quad \mathrm{csg}[\chi_D(2, x) + \chi_D(3, x) + \ldots + \chi_D(x - 1, x)],$$

which is a composition of primitive recursive functions. Thus χ_P is primitive recursive by case distinction. Hence, P is primitive. □

Primitive Recursive Sets

The function $p : \mathbb{N}_0 \to \mathbb{N}_0 : i \mapsto p_i$ is primitive recursive.

Proof

- Consider the function

$$\begin{aligned} g(z,y) &= |\chi_{R_<}(z,y) \cdot \chi_P(y) - 1| \\ &= \begin{cases} 0 & \text{if } z < y \text{ and } y \text{ prime,} \\ 1 & \text{otherwise.} \end{cases} \end{aligned} \tag{97}$$

  g is a composition of primitive recursive functions and so primitive recursive.

- Take the function

$$h(z) = \overline{\mu} g(z, z! + 1) = \mu(y \leq z! + 1)[g(z,y) = 0]. \tag{98}$$

  h is the bounded minimalization of a primitive recursive function and so primitive recursive.

Proof (Cont'd)

- We have $h(z) = \overline{\mu} g(z, z! + 1) = \mu(y \leq z! + 1)[g(z, y) = 0]$.
- If p is prime, then $h(p)$ is the smallest prime q such that $q > p$ and $q \leq p! + 1$.
- Theorem of Euclid:

  The $i + 1$-th prime p_{i+1} is bounded by the i-th prime p_i in a way that

  $$p_{i+1} \leq p_i! + 1, \quad i \geq 0.$$

- Thus $h(p_i) = p_{i+1}$.
- Hence, the sequence of prime numbers is given by the primitive recursive scheme

$$p_0 = 2 \quad \text{and} \quad p_{i+1} = h(p_i),\ i \geq 0. \tag{99}$$

Primitive Recursive Sets

The function $\tilde{p} : \mathbb{N}_0^2 \to \mathbb{N}_0 : (x, i) \mapsto (x)_i$ is primitive recursive.

Proof.

We have

$$(x)_i = \mu(y \leq x)[p_i^{y+1} \not| \, x] = \mu(y \leq x)[\chi_D(p_i^{y+1}, x) = 0], \quad (100)$$

i.e., $(x)_i$ is the smallest value y such that $p_i^{y+1} \not| \, x$, i.e., $p_i^y \mid x$ and $p_i^{y+1} \not| \, x$. □

E.g., $360 = 2^3 \cdot 3^2 \cdot 5^1$ and so $(360)_0 = 3$, $(360)_1 = 2$, $(360)_2 = 1$ and $(360)_i = 0$ for all $i \geq 3$.

LOOP Programs – Contents

- New programming primitives
- Inductive definition
- LOOP hierarchy
- Decrementation
- Equality with $\mathcal{P}$

LOOP Programs

- LOOP programming model provides an implementation of the primitive recursive functions.
- LOOP programs have restricted loop variables.
- Loops are of the form $(P; S\sigma)\sigma$, where the loop variable σ does not appear in the program P (explicit control over the loop).

LOOP Programs – New Primitives

- For each URM program P and each $\sigma \in Z$, the URM program $(P; S\sigma)\sigma$ is denoted by

$$[P]\sigma. \tag{101}$$

  E.g., $[A1]2 = (A1; S2)2$ increments R_1 exactly R_2 times.

- $[P]\sigma$ is executed exactly R_σ times if σ does not appear in P.
- URM program $(S\sigma)\sigma$ is denoted by

$$Z\sigma. \tag{102}$$

  Zero setting of register R_σ.

LOOP Programs – Class $\mathcal{P}_{\mathrm{LOOP}(0)}$

Definition of class of LOOP-0 programs:

- Primitives:

$$A\sigma,\ Z\sigma \in \mathcal{P}_{\mathrm{LOOP}(0)}, \quad \sigma \in Z. \tag{103}$$

- Copy:

$$\bar{C}(\sigma,\tau) = Z\tau; Z0; C(\sigma;\tau) \in \mathcal{P}_{\mathrm{LOOP}(0)}, \tag{104}$$

  where $\sigma, \tau \in Z$, $\sigma \neq \tau$, $\sigma \neq 0 \neq \tau$.

- Composition: If $P, Q \in \mathcal{P}_{\mathrm{LOOP}(0)}$, then

$$P; Q \in \mathcal{P}_{\mathrm{LOOP}(0)}. \tag{105}$$

For each $\omega \in \Omega$, $\bar{w} = \bar{C}(\sigma,\tau)(\omega)$ is given by

$$\bar{\omega}_n = \begin{cases} 0 & \text{if } n = 0, \\ \omega_\sigma & \text{if } n = \sigma \text{ or } n = \tau, \\ \omega_n & \text{otherwise,} \end{cases} \tag{106}$$

LOOP Programs – Class $\mathcal{P}_{\mathrm{LOOP}(n+1)}$

Suppose the class of LOOP-n programs $\mathcal{P}_{\mathrm{LOOP}(n)}$ has already been defined.

Definition of class of LOOP-$n+1$ programs:

- Inclusion: Each $P \in \mathcal{P}_{\mathrm{LOOP}(n)}$ belongs to $\mathcal{P}_{\mathrm{LOOP}(n+1)}$.
- Composition: If $P, Q \in \mathcal{P}_{\mathrm{LOOP}(n+1)}$, then

$$P;Q \in \mathcal{P}_{\mathrm{LOOP}(n+1)}. \tag{107}$$

- Iteration (restricted):

$$[P]\sigma \in \mathcal{P}_{\mathrm{LOOP}(n+1)} \tag{108}$$

where $\sigma \in Z$ does not appear in $P \in \mathcal{P}_{\mathrm{LOOP}(n)}$, e.g., $[A1]2 = (A1;S2)2 \in \mathcal{P}_{\mathrm{LOOP}(1)}$.

LOOP Programs – Example

The addition of two numbers is given by the URM program

$$(A1; S2)2$$

which by (101) corresponds to the LOOP-1 program

$$[A1]2.$$

LOOP Programs – Example

$S\sigma$ is not a LOOP program, but has an implementation as LOOP-1 program:

$$P_{\dot{-}1} = \bar{C}(1;3);[\bar{C}(2;1);A2]3. \tag{109}$$

Input $x = 0$:

0	1	2	3	4	...	registers
0	0	0	0	0	...	init
0	0	0	0	0	...	$\bar{C}(1;3)$
0	0	0	0	0	...	end

Input $x \geq 1$:

0	1	2	3	4	...	registers
0	x	0	0	0	...	init
0	x	0	x	0	...	$\bar{C}(1;3)$
0	0	0	x	0	...	$\bar{C}(2;1)$
0	0	1	x	0	...	$A2$
0	0	1	$x-1$	0	...	$S3$
...						
0	1	2	$x-2$	0	...	
...						
0	$x-1$	x	0	0	...	end

LOOP Programs – LOOP Hierarchy

- LOOP-n programs, $n \in \mathbb{N}_0$, form a strictly increasing sequence

$$\mathcal{P}_{\mathrm{LOOP}(0)} \subset \mathcal{P}_{\mathrm{LOOP}(1)} \subset \mathcal{P}_{\mathrm{LOOP}(2)} \subset \ldots, \tag{110}$$

  since each LOOP-n program is a LOOP-$n+1$ program and there are LOOP-$n+1$ programs which are not LOOP-n.

- $\mathcal{P}_{\mathrm{LOOP}}$ is the class of all *LOOP programs*,

$$\mathcal{P}_{\mathrm{LOOP}} = \bigcup_{n \geq 0} \mathcal{P}_{\mathrm{LOOP}(n)}. \tag{111}$$

LOOP Programs – Computability

- Function $f : \mathbb{N}_0^k \to \mathbb{N}_0$ is *LOOP-n computable* if there is a LOOP-n program P such that

$$\|P\|_{k,1} = f. \tag{112}$$

- $\mathcal{F}_{\mathrm{LOOP}(n)}$ denotes the class of all LOOP-n computable functions.
- $\mathcal{F}_{\mathrm{LOOP}}$ denotes the class of all LOOP computable functions,

$$\mathcal{F}_{\mathrm{LOOP}} = \bigcup_{n \geq 0} \mathcal{F}_{\mathrm{LOOP}(n)}. \tag{113}$$

- If P is a LOOP-n program and P' the corresponding normal program, then P' is a LOOP-n program (see (41) and make use of $Z\sigma$).

LOOP Programs – LOOP Hierarchy

- LOOP-n programs, $n \in \mathbb{N}_0$, form a strictly increasing sequence

$$\mathcal{P}_{\mathrm{LOOP}(0)} \subset \mathcal{P}_{\mathrm{LOOP}(1)} \subset \mathcal{P}_{\mathrm{LOOP}(2)} \subset \ldots, \tag{114}$$

- Thus the LOOP-n computable functions, $n \in \mathbb{N}_0$, form an increasing sequence

$$\mathcal{F}_{\mathrm{LOOP}(0)} \subseteq \mathcal{F}_{\mathrm{LOOP}(1)} \subseteq \mathcal{F}_{\mathrm{LOOP}(2)} \subseteq \ldots. \tag{115}$$

We will show that this sequence is strict (small Ackermann functions).

LOOP Programs – Main Result

The class of LOOP computable functions $\mathcal{F}_{\mathrm{LOOP}}$ is equal to the class of primitive recursive functions $\mathcal{P}$.

Proof (Sketch)

- $\mathcal{P} \subseteq \mathcal{F}_{\mathrm{LOOP}}$:

  Show that the basic functions are LOOP computable, and the class of LOOP computable functions is closed under composition and primitive recursion.

- $\mathcal{F}_{\mathrm{LOOP}} \subseteq \mathcal{P}$:

  Show that each LOOP computable function is primitive recursive; use induction on the inductive definition of LOOP programs. □

Skills

- Specify and interpret a primitive recursive function
- Define a function via Peano structure
- Establish a primitive set
- Write and interpret a LOOP program

Part IV

Partial Recursive Functions

Partial Recursive Functions – Motivation

- Full class of computable functions capturing the notion of computability.
- Contains the class of primitive recursive functions.
- GOTO programming as programming model for partial recursive functions.
- Church's thesis as working hypothesis for the programmer.

E.g., addition of two numbers by GOTO-2 program:

$$
\begin{array}{ll}
0: & T(p_2) \\
1: & \texttt{if } x_1 = 0 \texttt{ goto } 9 \\
2: & x_1 \leftarrow x_1 - 1 \\
3: & M(p_1) \\
4: & T(p_2) \\
5: & \texttt{if } x_1 = 0 \texttt{ goto } 8 \\
6: & x_1 \leftarrow x_1 - 1 \\
7: & D(p_2) \\
8: & \texttt{goto } 0 \\
9: &
\end{array}
$$

Partial Recursive Functions – Contents

- General definition
- GOTO programs and GOTO computable functions
- GOTO and URM programming
- GOTO-2 programming
- Church's thesis

Reading: Zimmermann, Chapter 3

Partial Recursive Functions – Inside

- Partial recursive functions form a class of partial functions that capture the notion of computability.
- Class of partial recursive functions equals the class of URM or GOTO computable functions.
- GOTO programs will be used later for Gödelization and reduction processes.
- GOTO programming can be restricted to using two registers only.
- Computability theory centers around Church's thesis, i.e., the partial recursive functions formalize the notion of computability.

Partial Recursive Functions – Contents

- Minimalization
- Class of partial recursive functions
- Class of recursive functions
- URM computability

Partial Recursive Functions – Minimalization

Given partial function $f : \mathbb{N}^{k+1} \to \mathbb{N}_0$.

(Unbounded) minimalization of f:

$$\mu f : \mathbb{N}_0^k \to \mathbb{N}_0 \tag{116}$$

with

$$\mu f(\boldsymbol{x}) = \begin{cases} y & \text{if } f(\boldsymbol{x}, y) = 0,\ f(\boldsymbol{x}, i) \neq 0 \text{ defined for } 0 \leq i < y, \\ \uparrow & \text{otherwise.} \end{cases} \tag{117}$$

μf provides an unbounded search process seeking the smallest $y \geq 0$ such that $f(x, y) = 0$.

Bounded minimalization

$\overline{\mu} f$ yields a bounded search process seeking the smallest $y \geq 0$ with $y \leq z$ such that $f(x, y) = 0$.

Partial Recursive Functions – Minimalization

Implementation of μf by `while` loop:

Require: $\boldsymbol{x} \in \mathbb{N}_0^n$
  $y \leftarrow -1$
  repeat
    $y \leftarrow y + 1$
    $z \leftarrow f(\boldsymbol{x}, y)$
  until $z = 0$
  return y

Kleene notation:

$$\mu y[f(\boldsymbol{x}, y) = 0] = \mu f(\boldsymbol{x}). \tag{118}$$

"Smallest y such that $f(\boldsymbol{x}, y) = 0$." Compare with bounded minimalization (79).

Partial Recursive Functions – Examples

- μf may be partial even if f is total.

  E.g., total function

  $$f(x, y) = (x + y) \dot{-} 3$$

  yields partial function $\mu f(x)$ with $\mathrm{dom}(\mu f) = \{0, 1, 2, 3\}$ and $\mu f(0) = \mu f(1) = \mu f(2) = \mu f(3) = 0$.

- μf may be total even if f is partial.

  E.g., partial function (see asymmetric difference $\dot{-}$)

  $$f(x, y) = \begin{cases} x - y & \text{if } y \leq x, \\ \uparrow & \text{otherwise,} \end{cases}$$

  provides total function $\mu f(x) = x$ with $\mathrm{dom}(\mu f) = \mathbb{N}_0$.

Partial Recursive Functions – Class $\mathcal{R}$

Class $\mathcal{R}$ of *partial recursive functions* is inductively defined:

- $\mathcal{R}$ contains the base functions.
- If partial functions $g_1, \ldots, g_m : \mathbb{N}_0^n \to \mathbb{N}_0$ and $h : \mathbb{N}_0^m \to \mathbb{N}_0$ lie to $\mathcal{R}$, the composite function $f = h(g_1, \ldots, g_m) : \mathbb{N}_0^n \to \mathbb{N}_0$ lies in $\mathcal{R}$.
- If partial functions $g : \mathbb{N}_0^n \to \mathbb{N}_0$ and $h : \mathbb{N}_0^{n+2} \to \mathbb{N}_0$ are in $\mathcal{R}$, the primitive recursion $f = \mathrm{pr}(g, h) : \mathbb{N}_0^{n+1} \to \mathbb{N}_0$ is in $\mathcal{R}$.
- If partial function $f : \mathbb{N}_0^{n+1} \to \mathbb{N}_0$ is in $\mathcal{R}$, the partial function $\mu f : \mathbb{N}_0^n \to \mathbb{N}_0$ is in $\mathcal{R}$.

Partial Recursive Functions – Class $\mathcal{T}$

- The *recursive functions* are the total partial recursive functions.
- $\mathcal{T}$ is the class of recursive functions.
- Each recursive function is partial recursive and so

$$\mathcal{T} \subset \mathcal{R} \tag{119}$$

with $f_\uparrow \in \mathcal{R} \setminus \mathcal{T}$ (empty function), say $f_\uparrow = \|A1; (A1)1\|_{k,1}$.

- Each primitive recursive function is total and built up along the same lines as $\mathcal{R}$, and so

$$\mathcal{P} \subset \mathcal{T} \tag{120}$$

with $A \in \mathcal{T} \setminus \mathcal{P}$ (Ackermann function).

Partial Recursive Functions – URM Computability

Each partial recursive function is URM computable.

Proof (Sketch)

- The basic functions are URM computable.
- Let $f = h(g_1, \ldots, g_m)$. By induction, there are URM programs P_h and $P_{g_1}, \ldots, P_{g_m}$ such that $\|P_h\|_{m,1} = h$ and $\|P_{g_i}\|_{n,1} = g_i$, $1 \leq i \leq m$. Then there is an URM program P such that $\|P\|_{n,1} = f$.
- Let $f = \mathrm{pr}(g, h)$. By induction, there are URM programs P_g and P_h such that $\|P_g\|_{n,1} = g$ and $\|P_h\|_{n+2,1} = h$. Then there is an URM program P such that $\|P\|_{n+1,1} = f$.
- Let $g = \mu f$. By induction, there is an URM program P_f such that $\|P_f\|_{n+1,1} = f$. Then there is an URM program P such that $\|P\|_{n,1} = g$. □

GOTO Programs – Contents

- Syntax
- Semantics
- Residual-step function

Syntax – Instructions

Set of variables $V = \{x_\sigma \mid \sigma \in \mathbb{N}\}$.

Instructions of GOTO programs:

- Incrementation:

$$(l, x_\sigma \leftarrow x_\sigma + 1, m), \quad l, m \in \mathbb{N}_0,\ x_\sigma \in V, \tag{121}$$

  l label and m next label.

- Decrementation:

$$(l, x_\sigma \leftarrow x_\sigma - 1, m), \quad l, m \in \mathbb{N}_0,\ x_\sigma \in V, \tag{122}$$

  l label and m next label.

- Branching:

$$(l, \texttt{if}\ x_\sigma = 0, k, m), \quad k, l, m \in \mathbb{N}_0,\ x_\sigma \in V, \tag{123}$$

  l label and k, m bifurcation labels.

Syntax – GOTO Programs

- Each *GOTO program* is a finite sequence of GOTO instructions

$$P = s_0; s_1; \ldots; s_q, \tag{124}$$

such that there is a unique instruction s_i which has the label $\lambda(s_i) = 0$, $0 \leq i \leq q$, and different instructions have distinct labels, i.e., for all $0 \leq i < j \leq q$, $\lambda(s_i) \neq \lambda(s_j)$.

- $\mathcal{P}_{\mathrm{GOTO}}$ denotes the class of all GOTO programs.
- For each GOTO program P, $V(P)$ denotes the set of variables occurring in P and $L(P) = \{\lambda(s_i) \mid 0 \leq i \leq q\}$ provides the set of labels in P.
- A GOTO program $P = s_0; s_1; \ldots; s_q$ is called *standard* if the i-th instruction s_i has the label $\lambda(s_i) = i$, $0 \leq i \leq q$.

GOTO Programs – Example

Standard GOTO program

$$P_+ = s_0; s_1; s_2$$

for the addition of two natural numbers:

s_0 :	0	if $x_2 = 0$	3	1
s_1 :	1	$x_1 \leftarrow x_1 + 1$	2	
s_2 :	2	$x_2 \leftarrow x_2 - 1$	0	

$V(P_+) = \{x_1, x_2\}$ and $L(P_+) = \{0, 1, 2\}$.

Semantics – Formalities

- *Memory:* variable x_σ is stored in register R_σ, $\sigma \geq 1$.
- *Instruction counter:* register R_0.
- *Initialization:* instruction counter is set to 0 such that the execution starts with the instruction having label 0.
- *Instructions:*
  - $(l, x_\sigma \leftarrow x_\sigma + 1, m)$ increments the content of register R_σ.
  - $(l, x_\sigma \leftarrow x_\sigma - 1, m)$ decrements the content of register R_σ if it contains a number greater than zero.
  - $(l, \texttt{if } x_\sigma = 0, k, m)$ provides a jump to the statement with label k if the content of register R_σ is 0; otherwise, to the statement with label m.
- *Termination:* program halts if the instruction counter does not correspond to an instruction label.

Semantics

Mimicking the computation of GOTO programs by partial recursive functions:

- Configuration
- Next configuration
- One-step function
- Multi-step function
- Runtime function
- Residual-step function

Semantics – Configuration

Given GOTO program

$$P = s_0; s_1; \ldots; s_q$$

with variables $V(P) = \{x_1, \ldots, x_n\}$.

- Each $n+1$-tuple $\boldsymbol{z} = (z_0, z_1, \ldots, z_n) \in \mathbb{N}_0^{n+1}$ is called a *configuration*.
- In each configuration $\boldsymbol{z} = (z_0, z_1, \ldots, z_n)$ of P,
  - z_0 is the content of the instruction counter,
  - z_i is the value of variable x_i, $1 \leq i \leq n$.

Semantics – Next Configuration

In program P, configuration $\boldsymbol{z} = (z_0, z_1, \ldots, z_n)$ transforms to configuration $\boldsymbol{z}' = (z_0', z_1', \ldots, z_n')$ *in one step*, written

$$\boldsymbol{z} \vdash_P \boldsymbol{z}', \tag{125}$$

if z_0 is a label in P, say $z_0 = \lambda(s_i)$ for some $0 \leq i \leq q$, and

- if $s_i = (z_0, x_\sigma \leftarrow x_\sigma + 1, m)$,

$$\boldsymbol{z}' = (m, z_1, \ldots, z_{\sigma-1}, z_\sigma + 1, z_{\sigma+1}, \ldots, z_n), \tag{126}$$

- if $s_i = (z_0, x_\sigma \leftarrow x_\sigma - 1, m)$,

$$\boldsymbol{z}' = (m, z_1, \ldots, z_{\sigma-1}, z_\sigma \dot{-} 1, z_{\sigma+1}, \ldots, z_n), \tag{127}$$

- if $s_i = (z_0, \texttt{if } x_\sigma = 0, k, m)$,

$$\boldsymbol{z}' = \begin{cases} (k, z_1, \ldots, z_n) & \text{if } z_\sigma = 0, \\ (m, z_1, \ldots, z_n) & \text{otherwise.} \end{cases} \tag{128}$$

Semantics – Example

Standard GOTO program P_+:

$$\begin{array}{llll} s_0: & 0 & \texttt{if } x_2 = 0 & 3 \quad 1 \\ s_1: & 1 & x_1 \leftarrow x_1 + 1 & 2 \\ s_2: & 2 & x_2 \leftarrow x_2 - 1 & 0 \end{array} \tag{129}$$

Tracing the computation with initial values $x_1 = 3$ and $x_2 = 2$:

$$\begin{array}{llll} \boldsymbol{z}_0 = (0,3,2) & 0 & \texttt{if } x_2 = 0 & 3 \quad 1 \\ \boldsymbol{z}_1 = (1,3,2) & 1 & x_1 \leftarrow x_1 + 1 & 2 \\ \boldsymbol{z}_2 = (2,4,2) & 2 & x_2 \leftarrow x_2 - 1 & 0 \\ \boldsymbol{z}_3 = (0,4,1) & 0 & \texttt{if } x_2 = 0 & 3 \quad 1 \\ \boldsymbol{z}_4 = (1,4,1) & 1 & x_1 \leftarrow x_1 + 1 & 2 \\ \boldsymbol{z}_5 = (2,5,1) & 2 & x_2 \leftarrow x_2 - 1 & 0 \\ \boldsymbol{z}_6 = (0,5,0) & 0 & \texttt{if } x_2 = 0 & 3 \quad 1 \\ \boldsymbol{z}_7 = (3,5,0) & & & \end{array} \tag{130}$$

*Semantics – One-Step Function

One-step function

$$E_P : \mathbb{N}_0^{n+1} \to \mathbb{N}_0^{n+1}$$

is defined as

$$E_P : \boldsymbol{z} \mapsto \begin{cases} \boldsymbol{z}' & \text{if } \exists \boldsymbol{z}' : \ \boldsymbol{z} \vdash_P \boldsymbol{z}', \\ \boldsymbol{z} & \text{otherwise.} \end{cases} \tag{131}$$

E_P is primitive recursive, it has as input $\boldsymbol{z}$ and an encoding of P and outputs the next configuration (if any).

*Semantics – Multi-Step Function

Multi-step function

$$M_P : \mathbb{N}_0^{n+2} \to \mathbb{N}_0^{n+1} : (\boldsymbol{z}, t) \mapsto E_P^t(\boldsymbol{z}), \tag{132}$$

i.e., for all $\boldsymbol{z} \in \mathbb{N}_0^{n+1}$ and $t \in \mathbb{N}_0$,

$$M_P(\boldsymbol{z}, t) = \underbrace{(E_P \circ \ldots \circ E_P)}_{t \text{ times}}(\boldsymbol{z}).$$

M_P is obtained from one-step function E_P by iteration and so is also primitive recursive.

*Semantics – Runtime Function

Runtime function

$$Z_P : \mathbb{N}_0^{n+1} \to \mathbb{N}_0$$

is given by

$$Z_P(\boldsymbol{z}) = \mu t \left[\chi_{L(P)}\left(\pi_1^{(n+1)} \circ E_P^t)(\boldsymbol{z})\right) = 0\right] \tag{133}$$

$$\begin{cases} \min\{t \in \mathbb{N}_0 \mid (\pi_1^{(n+1)} \circ E_P^t)(\boldsymbol{z}) \notin L(P)\} & \text{if } t \text{ exists,} \\ \uparrow & \text{otherwise,} \end{cases}$$

where

$$E_P^t(\boldsymbol{z}) = (z_0', \ldots, z_n')$$

is the configuration after t steps and

$$\pi_1^{(n+1)}(z_0', \ldots, z_n') = z_0'$$

is the label reached. If $z_0' \notin L(P)$, i.e., z_0' is not a program label, the program terminates. Z_P is (by minimalization) partial recursive, since $L(P)$ is finite and so $\chi_{L(P)}$ is primitive recursive.

Semantics – Example

Standard GOTO program P_+:

s_0 :	0	if $x_2 = 0$	3	1
s_1 :	1	$x_1 \leftarrow x_1 + 1$	2	
s_2 :	2	$x_2 \leftarrow x_2 - 1$	0	

Tracing the computation with initial values $x_1 = 3$ and $x_2 = 2$:

$\boldsymbol{z}_0 = (0, 3, 2)$	0	if $x_2 = 0$	3	1
$\boldsymbol{z}_1 = E_{P_+}(\boldsymbol{z}_0) = (1, 3, 2)$	1	$x_1 \leftarrow x_1 + 1$	2	
$\boldsymbol{z}_2 = E_{P_+}(\boldsymbol{z}_1) = (2, 4, 2)$	2	$x_2 \leftarrow x_2 - 1$	0	
$\boldsymbol{z}_3 = E_{P_+}(\boldsymbol{z}_2) = (0, 4, 1)$	0	if $x_2 = 0$	3	1
$\boldsymbol{z}_4 = E_{P_+}(\boldsymbol{z}_3) = (1, 4, 1)$	1	$x_1 \leftarrow x_1 + 1$	2	
$\boldsymbol{z}_5 = E_{P_+}(\boldsymbol{z}_4) = (2, 5, 1)$	2	$x_2 \leftarrow x_2 - 1$	0	
$\boldsymbol{z}_6 = E_{P_+}(\boldsymbol{z}_5) = (0, 5, 0)$	0	if $x_2 = 0$	3	1
$\boldsymbol{z}_7 = E_{P_+}(\boldsymbol{z}_6) = (3, 5, 0)$				

$M_{P_+}((0, 3, 2), 7) = E^7_{P_+}(0, 3, 2) = (3, 5, 0)$ with $3 \notin L(P_+)$ and $Z_{P_+}(0, 3, 2) = 7$.

Semantics – Example

Standard GOTO program P_{+1} (URM program $(A1)1$):

$$\begin{array}{llll} s_0: & 0 & \texttt{if } x_1 = 0 & 2 \quad 1 \\ s_1: & 1 & x_1 \leftarrow x_1 + 1 & 0 \end{array} \tag{134}$$

Tracing the computation with initial value $x_1 = 0$:

$$\begin{array}{llll} \boldsymbol{z}_0 = (0,0) & 0 & \texttt{if } x_1 = 0 & 2 \quad 1 \\ \boldsymbol{z}_1 = (2,0) & & & \end{array}$$

Then $Z_{P_{+1}} = 1$.

Tracing the computation with initial value $x_1 > 0$:

$$\begin{array}{llll} \boldsymbol{z}_0 = (0,x_1) & 0 & \texttt{if } x_1 = 0 & 2 \quad 1 \\ \boldsymbol{z}_1 = (1,x_1) & 1 & \texttt{if } x_1 \leftarrow x_1 + 1 & 0 \\ \boldsymbol{z}_2 = (0,x_1+1) & 0 & \texttt{if } x_1 = 0 & 2 \quad 1 \\ \boldsymbol{z}_3 = (1,x_1+1) & 1 & \texttt{if } x_1 \leftarrow x_1 + 1 & 0 \end{array}$$

Then $Z_{P_{+1}} = \uparrow$.

*Semantics – Residual-Step Function

Residual-step function

$$R_P : \mathbb{N}_0^{n+1} \to \mathbb{N}_0^{n+1}$$

given by

$$R_P(\boldsymbol{z}) = M_P(\boldsymbol{z}, Z_P(\boldsymbol{z})) = \begin{cases} M_P(\boldsymbol{z}, Z_P(\boldsymbol{z})) & \text{if } \boldsymbol{z} \in \mathrm{dom}(Z_P), \\ \uparrow & \text{otherwise,} \end{cases} \quad (135)$$

is partial recursive.

If $\boldsymbol{z} \in \mathrm{dom}(Z_P)$, then $Z_P(\boldsymbol{z}) = t$ provides the number of steps P performed until it terminates and then $M_P(\boldsymbol{z}, t)$ is the **final configuration** reached.

Residual-step function R_P mimicks the execution of GOTO program P.

Semantics – Example

Standard GOTO program P_+:

$s_0:$	0	$\texttt{if } x_2 = 0$	3	1
$s_1:$	1	$x_1 \leftarrow x_1 + 1$	2	
$s_2:$	2	$x_2 \leftarrow x_2 - 1$	0	

Tracing the computation with initial values $x_1 = 3$ and $x_2 = 2$:

$\boldsymbol{z}_0 = (0, 3, 2)$	0	$\texttt{if } x_2 = 0$	3	1
$\boldsymbol{z}_1 = E_{P_+}(\boldsymbol{z}_0) = (1, 3, 2)$	1	$x_1 \leftarrow x_1 + 1$	2	
$\boldsymbol{z}_2 = E_{P_+}(\boldsymbol{z}_1) = (2, 4, 2)$	2	$x_2 \leftarrow x_2 - 1$	0	
$\boldsymbol{z}_3 = E_{P_+}(\boldsymbol{z}_2) = (0, 4, 1)$	0	$\texttt{if } x_2 = 0$	3	1
$\boldsymbol{z}_4 = E_{P_+}(\boldsymbol{z}_3) = (1, 4, 1)$	1	$x_1 \leftarrow x_1 + 1$	2	
$\boldsymbol{z}_5 = E_{P_+}(\boldsymbol{z}_4) = (2, 5, 1)$	2	$x_2 \leftarrow x_2 - 1$	0	
$\boldsymbol{z}_6 = E_{P_+}(\boldsymbol{z}_5) = (0, 5, 0)$	0	$\texttt{if } x_2 = 0$	3	1
$\boldsymbol{z}_7 = E_{P_+}(\boldsymbol{z}_6) = (3, 5, 0)$				

$M_{P_+}((0, 3, 2), 7) = E^7_{P_+}(0, 3, 2) = (3, 5, 0)$ with $3 \notin L(P_+)$, $Z_{P_+}(0, 3, 2) = 7$ and $R_{P_+}(0, 3, 2) = (3, 5, 0)$.

Semantics – Computable Functions

For each GOTO program P and $k \geq 0$, define the partial function

$$\|P\|_{k,1} = \beta_1 \circ R_P \circ \alpha_k, \tag{136}$$

where $\alpha_k : \mathbb{N}_0^k \to \mathbb{N}_0^{n+1}$ loads the registers with arguments,

$$\alpha_k : (x_1, x_2, \ldots, x_k) \mapsto (0, x_1, x_2, \ldots, x_k, 0, \ldots, 0) \tag{137}$$

R_P is the residual-step function

$$R_P : \mathbb{N}_0^{n+1} \to \mathbb{N}_0^{n+1} \tag{138}$$

and $\beta_1 : \mathbb{N}_0^{n+1} \to \mathbb{N}_0$ reads out the result,

$$\beta_1 : (x_0, x_1, \ldots, x_n) \mapsto x_1. \tag{139}$$

n must be large enough to provide sufficient workspace for the computation of P,

$$n \geq \max\{k, \max\{\sigma \mid x_\sigma \in V(P)\}\}. \tag{140}$$

Semantics – Example

Standard GOTO program P_+:

$$\begin{array}{llll} s_0: & 0 & \texttt{if } x_2 = 0 & 3 \quad 1 \\ s_1: & 1 & x_1 \leftarrow x_1 + 1 & 2 \\ s_2: & 2 & x_2 \leftarrow x_2 - 1 & 0 \end{array}$$

Computation of P_+ $(n = 2)$:

$$\begin{aligned} \|P_+\|_{2,1}(3,2) &= (\beta_1 \circ R_{P_+} \circ \alpha_2)(3,2) \qquad (141) \\ &= (\beta_1 \circ R_{P_+})(0,3,2) \\ &= \beta_1(3,5,0), \quad \text{by (130)}, \\ &= 5. \end{aligned}$$

GOTO Computable Functions – Contents

- GOTO computable functions
- URM computability implies GOTO computability

GOTO Computable Functions

- Partial function $f : \mathbb{N}_0^k \to \mathbb{N}_0$ is *GOTO computable* if there is a GOTO program P that computes f in the sense that by (136)

$$f = \|P\|_{k,1} = \beta_1 \circ R_P \circ \alpha_k. \tag{142}$$

- $\mathcal{F}_{\mathrm{GOTO}}$ denotes the class of all GOTO computable functions.
- $\mathcal{T}_{\mathrm{GOTO}}$ depicts the class of all total GOTO computable functions.

GOTO Computable Functions

Each GOTO computable function is partial recursive.

Proof.

Let $f : \mathbb{N}_0^k \to \mathbb{N}_0$ be GOTO computable. Then by (142), there is a GOTO program P such that

$$f = \|P\|_{k,1} = \beta_1 \circ R_P \circ \alpha_k.$$

f is the composition of primitive recursive functions β_1, α_k and partial recursive function R_P and so is also partial recursive. □

GOTO Computable Functions – Example

Standard GOTO program P_+:

0	$\texttt{if } x_2 = 0$	3	1
1	$x_1 \leftarrow x_1 + 1$	2	
2	$x_2 \leftarrow x_2 - 1$	0	

We have

$$\|P_+\|_{2,1}(x_1, x_2) = x_1 + x_2.$$

URM and GOTO Computable Functions

Each URM computable function is GOTO computable.

Proof

The idea is to translate each URM program P into a GOTO program $\phi(P)$ such that both compute the same function,

$$\|P\|_{k,1} = \|\phi(P)\|_{k,1}, \quad k \in \mathbb{N}_0.$$

Let P be an URM program which does not make use of register R_0. Write P as a string

$$P = \tau_0\tau_1 \dots \tau_q,$$

where each substring (token) τ_i is of the form

$$"A\sigma", "S\sigma", "(" \text{ or } ")\sigma" \text{ with } \sigma \in Z \setminus \{0\}.$$

Note that each opening parenthesis "(" has a unique closing parenthesis ")σ".

Proof (Cont'd)

Replace each token τ_i by a GOTO instruction s_i:

- If $\tau_i = "A\sigma"$, put

$$s_i = (i, x_\sigma \leftarrow x_\sigma + 1, i + 1),$$

- if $\tau_i = "S\sigma"$, set

$$s_i = (i, x_\sigma \leftarrow x_\sigma - 1, i + 1),$$

- if $\tau_i = "("$ and $\tau_j = ")\sigma"$ is the corresponding closing parenthesis, define

$$s_i = (i, \texttt{if } x_\sigma = 0, j + 1, i + 1),$$

- if $\tau_i = ")\sigma"$ and $\tau_j = "("$ is the associated opening parenthesis, put

$$s_i = (i, \texttt{if } x_\sigma = 0, i + 1, j + 1).$$

Proof (Cont'd)

In this way, from the URM program (written as a sequence of tokens)

$$P = \tau_0 \tau_1 \dots \tau_q$$

one obtains the GOTO program

$$\phi(P) = s_0; s_1; \dots; s_q$$

which has the required property

$$\|P\|_{k,1} = \|\phi(P)\|_{k,1}.$$

URM and GOTO Computable Functions – Example

The URM program

$$P_+ = (A1; S2)2$$

is given by the sequence of tokens

$$P_+ = \tau_0\tau_1\tau_2\tau_3,$$

where

$$\tau_0 = \text{"("},\ \tau_1 = \text{"}A1\text{"},\ \tau_2 = \text{"}S2\text{"},\ \tau_3 = \text{")2"}.$$

Translation yields the GOTO program

τ_0 :	s_0 :	0	if $x_2 = 0$	4	1
τ_1 :	s_1 :	1	$x_1 \leftarrow x_1 + 1$	2	
τ_2 :	s_2 :	2	$x_2 \leftarrow x_2 - 1$	3	
τ_3 :	s_3 :	3	if $x_2 = 0$	4	1

Compare this program with the GOTO program for addition (129) which has only three instructions.

Church's Thesis – Contents

- Big picture
- Church's thesis

Comparison of Function Classes - Big Picture

$$\begin{array}{c} \mathcal{R} = \mathcal{F}_{\mathrm{URM}} = \mathcal{F}_{\mathrm{GOTO}} = \mathcal{F}_{\mathrm{GOTO2}} \\ | \\ \mathcal{T} = \mathcal{T}_{\mathrm{URM}} = \mathcal{T}_{\mathrm{GOTO}} = \mathcal{T}_{\mathrm{GOTO2}} \\ | \\ \mathcal{P} = \mathcal{F}_{\mathrm{LOOP}} \end{array}$$

Proof.

We have explicitely proved

$$\mathcal{F}_{\mathrm{URM}} \subseteq \mathcal{F}_{\mathrm{GOTO}}, \quad \mathcal{F}_{\mathrm{GOTO}} \subseteq \mathcal{R}, \quad \mathcal{R} \subseteq \mathcal{F}_{\mathrm{URM}}.$$

By ring closure, all three classes are equal. □

Church's Thesis

Each computable partial function in the intuitive sense is partial recursive (Alonso Church, 1903-1995).

- What does *intuitive* mean?
  Just sit down and write an algorithm in your favorite programming language. Suppose it provides a function $f : \mathbb{N}_0^k \to \mathbb{N}_0$. Then by Church's thesis, this function is partial recursive.
- Church's thesis characterizes the nature of computation and cannot be formally proved. Nevertheless, it has reached universal acceptance.
- If the thesis could be disproved, the construction of a *hyper-computer* would be possible which would be superior to say GOTO computation.
- Church's thesis is often practically used in the sense that if a partial function is intuitively computable, it will be partial recursive. In this way, the thesis can lead to more intuitive and less rigorous proofs.

GOTO-2 Programs – Contents

- Encoding of URM states
- Basic GOTO-2 programs
- URM computability implies GOTO-2 computability
- Use of flow charts (diagrammatic representation of algorithm)

GOTO-2 Programs – Inside

- GOTO-2 programs are GOTO programs with two variables x_1 and x_2.
- Goal is to simulate URM programs by GOTO-2 programs.

GOTO-2 Programs - Encoding of URM States

- URM has countable state set

$$\Omega = \{\omega \mid \omega = (\omega_i)_{i \in \mathbb{N}_0} \text{ almost all entries } 0\}$$

by the bijection $G : \Omega \to \mathbb{N}$ given by

$$G(\omega) = \prod_{\substack{i \\ \text{finite}}} p_i^{\omega_i}, \tag{143}$$

where $(p_0, p_1, p_2, \ldots)$ is the sequence of primes.

- G is primitive recursive, since multiplication is primitive recursive.
- E.g., if $\omega = (0, 2, 3, 0, 5, 0, 0, \ldots)$, then $G(\omega) = p_1^2 p_2^3 p_4^5$.

GOTO-2 Programs - Encoding of URM States

- Collection of functions

$$G_i : \mathbb{N}_0 \to \mathbb{N}_0, \quad i \in \mathbb{N}_0,$$

given by

$$G_i(x) = (x)_i, \tag{144}$$

where $(x)_i$ is the exponent of p_i in the (unique) prime factorization of $x > 0$.

- In particular, set $G_i(0) = 0$ for all $i \in \mathbb{N}_0$.
- By (100), G_i is primitive recursive, $i \in \mathbb{N}_0$.
- E.g., if $\omega = (0, 2, 3, 0, 5, 0, 0, \ldots)$, then $x = G(\omega) = p_1^2 p_2^3 p_4^5$ and so $(x)_1 = 2$, $(x)_2 = 3$, $(x)_4 = 5$.

GOTO-2 Programs – Basic Programs

Consider three basis GOTO-2 programs:

$$|M(k)|(0,x) = (0,k\cdot x), \tag{145}$$

$$|D(k)|(0,k\cdot x) = (0,x), \tag{146}$$

$$|T(k)|(0,x) = \begin{cases} (1,x) & \text{if } k \text{ divides } x, \\ (0,x) & \text{otherwise.} \end{cases} \tag{147}$$

GOTO-2 Program $M(k)$

We have $|M(k)|(0, x) = (0, k \cdot x)$.

Implementation of $M(k)$ by two consecutive loops:

- Initially, $x_1 = 0$ and $x_2 = x$.
- 1st loop: x_2 is decremented, while in each decremention step, x_1 is incremented k times. After this loop, $x_1 = k \cdot x$ and $x_2 = 0$:

$$(A1^k; S2)2$$

- 2nd loop: x_2 is incremented and x_1 is decremented such that upon termination, $x_1 = 0$ and $x_2 = k \cdot x$ (swapping states):

$$(S1; A2)1$$

GOTO-2 Program $D(k)$

We have $|D(k)|(0, k \cdot x) = (0, x)$.

Implementation of $D(k)$ by two consecutive loops:

- Initially, $x_1 = 0$ and $x_2 = k \cdot x$.
- 1st loop: x_1 is incremented, while x_2 is decremented k times. After this loop, $x_1 = x$ and $x_2 = 0$:

$$(A1; S2^k)2$$

- 2nd loop: x_2 is incremented and x_1 is decremented such that upon termination, $x_1 = 0$ and $x_2 = x$ (swapping states):

$$(S1; A2)1$$

GOTO-2 Program – Main Result

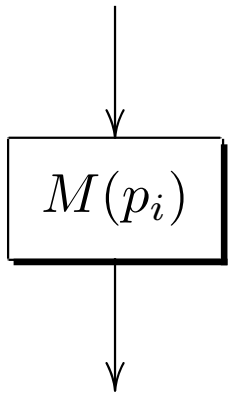
For each URM program P, there exists a GOTO-2 program $\overline{P}$ with the same semantics, i.e., for all states $\omega, \omega' \in \Omega$,

$$|P|(\omega) = \omega' \quad \Longleftrightarrow \quad |\overline{P}|(0, G(\omega)) = (0, G(\omega')). \tag{148}$$

The assignment $P \mapsto \overline{P}$ will be established by making use of the inductive definition of URM programs and flow diagrams to simplify notation.

GOTO-2 Program – Increment

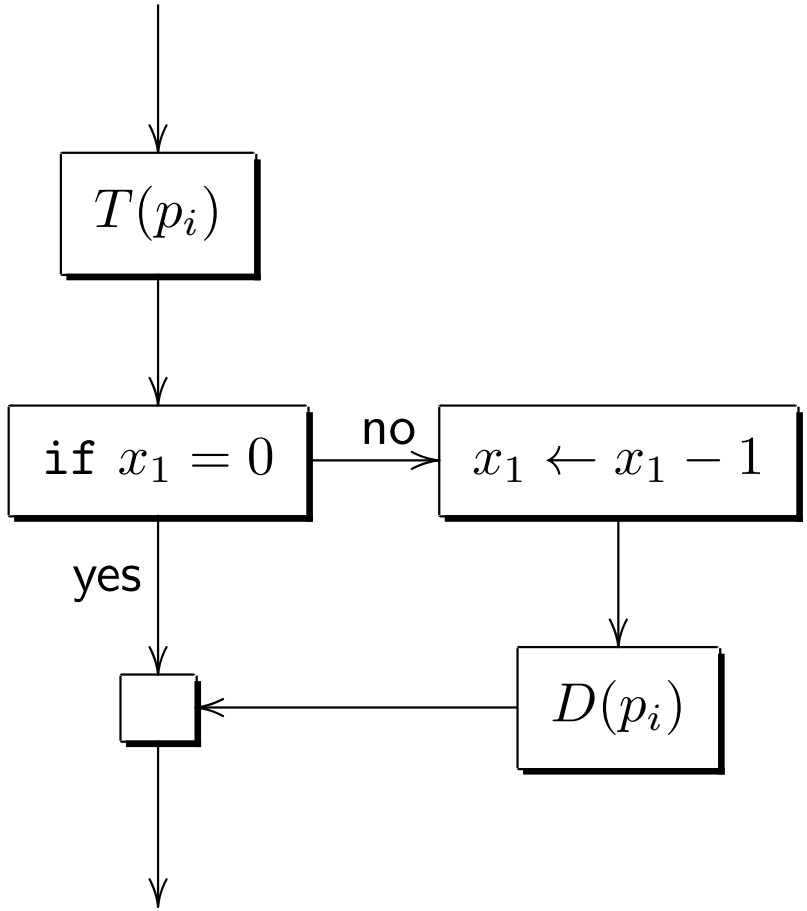
Program $\overline{Ai}$ can be realized by the flow chart



If $\omega = (\omega_0, \ldots, \omega_i, \ldots)$, then $\omega' = (\omega_0, \ldots, \omega_i + 1, \ldots)$, since $M(p_i)$ maps $(0, p_0^{\omega_0} \cdots p_i^{\omega_i} \cdots)$ to $(0, p_0^{\omega_0} \cdots p_i^{\omega_i+1} \cdots)$.

GOTO-2 Programs – Decrement

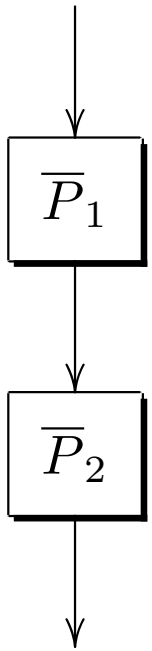
Program $\overline{Si}$ is given by the flow diagram



If $\omega = (\omega_0, \ldots, \omega_i, \ldots)$, then $\omega' = (\omega_0, \ldots, \omega_i \dot{-} 1, \ldots)$.

GOTO-2 Programs – Composition

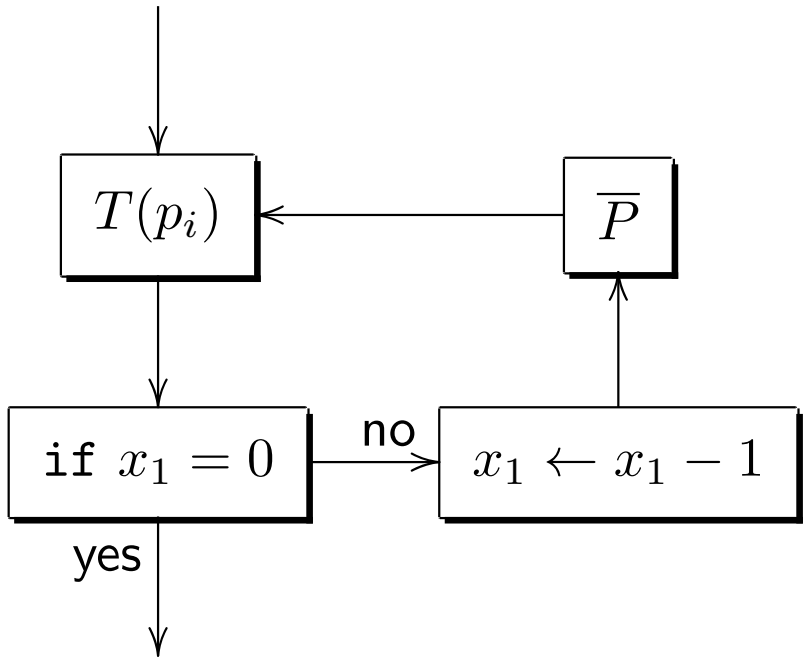
Program $\overline{P_1; P_2}$ can be decipted by the flow chart



Concatenation of GOTO-2 programs $\overline{P_1}$ and $\overline{P_2}$.

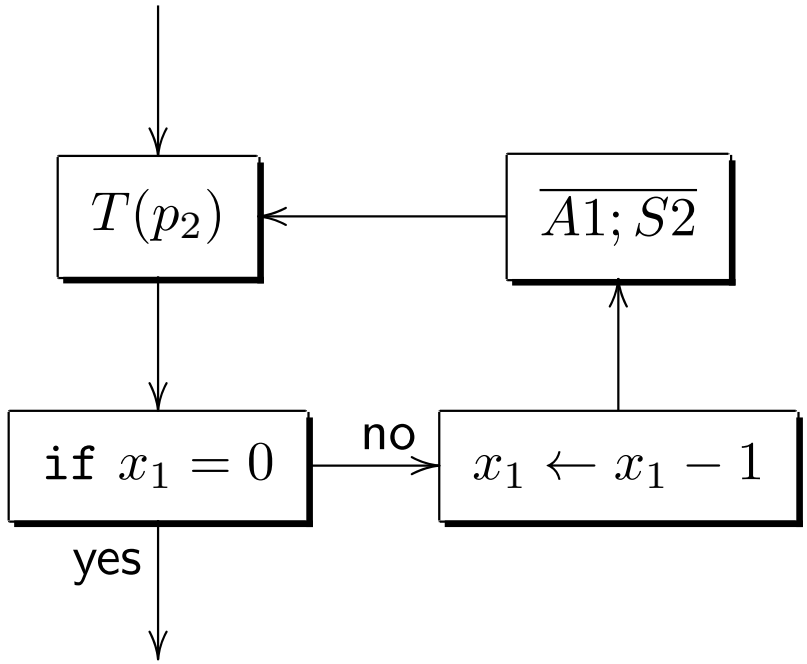
GOTO-2 Programs – Iteration

Program $\overline{(P)i}$ can be represented by the flow diagram



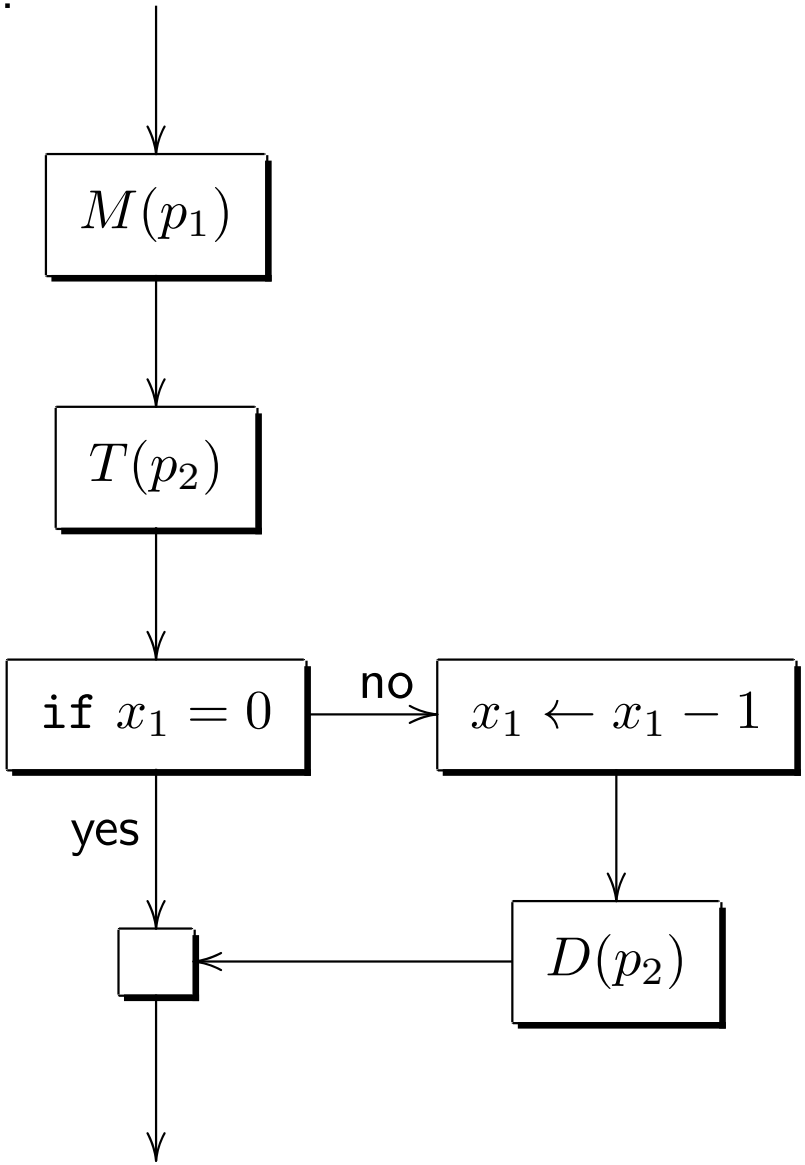
GOTO-2 Programs – Example

URM program $P_+ = (A1; S2)2$ translates into flow chart via iteration:



Example (Cont'd)

Program $\overline{A1; S2}$:



Example (Cont'd)

URM program $P_+ = (A1; S2)2$ translates into GOTO-2 program $\overline{P}_+$:

$$
\begin{array}{ll}
0: & T(p_2) \\
1: & \texttt{if } x_1 = 0 \texttt{ goto } 9 \\
2: & x_1 \leftarrow x_1 - 1 \\
3: & M(p_1) \\
4: & T(p_2) \\
5: & \texttt{if } x_1 = 0 \texttt{ goto } 8 \\
6: & x_1 \leftarrow x_1 - 1 \\
7: & D(p_2) \\
8: & \texttt{goto } 0 \\
9: &
\end{array}
$$

GOTO-2 Programs – Proof (End)

- By inductive definition of URM programs, the assignment $P \mapsto \overline{P}$ is well-defined.
- The computation of a function $f : \mathbb{N}_0^k \to \mathbb{N}_0^m$ by a GOTO-2 program requires to load the registers with the initial values and to identify the result.
- Define primitive recursive functions $\hat{\alpha}_k : \mathbb{N}_0^k \to \mathbb{N}_0^2$,

$$\hat{\alpha}_k : (x_1, \ldots, x_k) \mapsto (0, G(0, x_1, \ldots, x_k, 0, \ldots)) \tag{149}$$

and $\hat{\beta}_m : \mathbb{N}_0^2 \to \mathbb{N}_0^m$,

$$(a, b) \mapsto (G_1(b), \ldots, G_m(b)). \tag{150}$$

GOTO-2 Programs – Proof (End)

Each URM program P is (semantically) equivalent to the associated GOTO-2 program $\overline{P}$ in the sense that for all $k, m \in \mathbb{N}_0$:

$$\|P\|_{k,m} \overset{!}{=} \|\overline{P}\|_{k,m} \overset{\text{def}}{=} \hat{\beta}_m \circ R_{\overline{P}} \circ \hat{\alpha}_k, \tag{151}$$

where $R_{\overline{P}}$ is the residual-step function of GOTO-2 program $\overline{P}$.

□

GOTO-2 Programs – Example

Standard GOTO program P_+:

0	if $x_2 = 0$	3	1
1	$x_1 \leftarrow x_1 + 1$	2	
2	$x_2 \leftarrow x_2 - 1$	0	

We have

$$\|P_+\|_{2,1}(x_1, x_2) = x_1 + x_2.$$

Computation by GOTO-2 program $\overline{P}_+$:

$$\begin{aligned}
\|P_+\|_{2,1}(x_1, x_2) &= (\hat{\beta}_1 \circ R_{\overline{P}_+} \circ \hat{\alpha}_2)(x_1, x_2) \\
&= (\hat{\beta}_1 \circ R_{\overline{P}_+})(0, p_1^{x_1} \cdot p_2^{x_2}) \\
&= \hat{\beta}_1(0, p_0^3 \cdot p_1^{x_1+x_2}), \quad \text{by (130), (148),} \\
&= x_1 + x_2.
\end{aligned}$$

Note that by (141), $R_{P_+}(0, x_1, x_2) = (3, x_1 + x_2, 0)$.

Skills

- Determine μf from partial recursive function f
- Write and interpret a GOTO program
- Compile URM program into GOTO program
- Compile URM program into GOTO-2 program
- Explicate Church's thesis

Part V

Ackermann Function

Ackermann Function – Motivation

- Class of small Ackermann functions $B_n : \mathbb{N}_0 \to \mathbb{N}_0$, $n \geq 0$.
- Small Ackermann functions provide bound on the complexity of LOOP programs and establish the LOOP hierarchy.
- Definition of Ackermann function is based on small Ackermann functions.
- Ackermann function belongs to the class of fast-growing functions (superpowers).
- E.g., tetration:

$$A(4, y) = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{y+3} - 3.$$

Ackermann Function – Contents

- Small Ackermann functions
- Complexity of LOOP programs
- Ackermann function
- Superpowers

Reading: Zimmermann, Chapter 4

Ackermann Function – Inside

- David Hilbert (1926) posed the problem whether all total computable functions are primitive recursive or not.
- Wilhelm Ackermann (1928) introduced a function, which is recursive but not primitive recursive.

Tetration

$$
\begin{aligned}
A(4,n) &= 2 \uparrow^2 (n+3) - 3, \quad n \geq 0, \\
A(4,0) &= 2 \uparrow^2 3 - 3 = 2^4 - 3 = 13, \\
A(4,1) &= 2 \uparrow^2 4 - 3 = 2^{2^4} - 3 = 65533, \\
A(4,2) &= 2 \uparrow^2 5 - 3 = 2^{65536} - 3, \\
A(4,3) &= 2 \uparrow^2 6 - 3 = 2^{2^{65536}} - 3, \\
A(4,4) &= 2 \uparrow^2 7 - 3 = 2^{2^{2^{65536}}} - 3.
\end{aligned}
$$

Small Ackermann Functions – Contents

- General definition
- Primitive recursiveness
- Properties
- LOOP computability

Small Ackermann Functions – Definition

Sequence $(B_n)_{n\in\mathbb{N}_0}$ of *small Ackermann functions*

$$B_n : \mathbb{N}_0 \to \mathbb{N}_0 \tag{152}$$

is defined inductively:

$$B_0 : x \mapsto \begin{cases} 1 & \text{if } x = 0, \\ 2 & \text{if } x = 1, \\ x+2 & \text{otherwise,} \end{cases} \tag{153}$$

and for each $n \geq 0$,

$$B_{n+1} : x \mapsto B_n^x(1), \quad x \in \mathbb{N}_0, \tag{154}$$

where $B_n^x = \underbrace{B_n \circ \ldots \circ B_n}_{x}$ is the x-fold iteration of B_n.

Small Ackermann Functions – Primitive Recursiveness

Small Ackermann function B_n, $n \in \mathbb{N}_0$, is primitive recursive.

Proof

The function B_0 is defined by primitive case distinction:

$$B_0(x) = \begin{cases} \nu(x) & \text{if } \operatorname{sgn}(x) = 0, \\ \nu(x) & \text{if } \operatorname{sgn}(x) = 1 \text{ and } \operatorname{sgn}(x \dot{-} 1) = 0, \\ (\nu \circ \nu)(x) & \text{if } \operatorname{sgn}(x \dot{-} 1) = 1. \end{cases} \quad (155)$$

Then

$$B_0(x) = \nu(x) \cdot \operatorname{csg}(x) + \nu(x) \cdot [\operatorname{sgn}(x) \cdot \operatorname{csg}(x \dot{-} 1)] + \nu^2(x) \cdot \operatorname{sgn}(x \dot{-} 1).$$

Proof (Cont'd)

If $n \geq 1$, B_n is given by the primitive recursive scheme

$$B_n(0) = 1, \tag{156}$$
$$B_n(x+1) = B_{n-1}(B_n(x)), \quad x \in \mathbb{N}_0. \tag{157}$$

Note that $B_n(0) = B_{n-1}^0(1) = \mathrm{id}_{\mathbb{N}_0}(1) = 1$ and $B_n(x+1) = B_{n-1}^{x+1}(1) = B_{n-1}(B_{n-1}^x(1)) = B_{n-1}(B_n(x))$.

By induction, the small Ackermann functions are primitive recursive. □

Small Ackermann Functions – *Properties

For all $x, n, p \in \mathbb{N}_0$:

$$B_1(x) = \begin{cases} 1 & \text{if } x = 0, \\ 2x & \text{otherwise,} \end{cases} \tag{158}$$

$$B_2(x) = 2^x, \tag{159}$$

$$B_3(x) = \begin{cases} 1 & \text{if } x = 0, \\ 2^{B_3(x-1)} & \text{otherwise,} \end{cases} \tag{160}$$

$$x < B_n(x), \tag{161}$$

$$B_n(x) < B_n(x+1), \tag{162}$$

$$B_0^p(x) \leq B_1^p(x), \tag{163}$$

$$B_n(x) \leq B_{n+1}(x), \tag{164}$$

$$B_n^p(x) < B_n^p(x+1), \tag{165}$$

$$B_n^p(x) < B_n^{p+1}(x), \tag{166}$$

$$B_n^p(x) \leq B_{n+1}^p(x), \tag{167}$$

$$2^{p+1}x \leq B_1^{p+1}(x), \tag{168}$$

$$2B_n^p(x) \leq B_n^{p+1}(x), \quad n \geq 1. \tag{169}$$

Small Ackermann Functions – Growth

B_{n+1} grows faster than any power of B_n, i.e., for all n and p, there exists $x_0 \in \mathbb{N}_0$ such that for all $x \geq x_0$,

$$B_n^p(x) < B_{n+1}(x). \tag{170}$$

*Proof

- Let $n = 0$. For each number $x \geq 2$, $B_0^p(x) = x + 2p$.
  On the other hand, by (158), for each $x \geq 1$, $B_1(x) = 2x$.
  Put $x_0 = 2p + 1$. Then for each $x \geq x_0$,

$$B_0^p(x) = x + 2p \leq 2x = B_1(x).$$

- Let $n > 0$. First, let $p = 0$. Then for each $x \geq 0$,

$$B_n^0(x) = x < B_n(x) \leq B_{n+1}(x)$$

by (161) and (164).

Proof (Cont'd)

Second, let $p > 0$ and assume that $B_n^p(x) < B_{n+1}(x)$ holds for all $x \geq x_0'$. Put $x_0 = x_0' + 5$. Then

$$\begin{aligned}
B_n^{p+1}(x) &< B_n^{p+1}(2 \cdot (x \dot{-} 2)), && x \geq 5, \text{ by (165)},\\
&= B_n^{p+1}(B_1(x \dot{-} 2))\\
&\leq B_n^{p+1}(B_n(x \dot{-} 2)), && \text{by (167)},\\
&= B_n^{p+2}(x \dot{-} 2)\\
&= B_n^2(B_n^p(x \dot{-} 2))\\
&< B_n^2(B_{n+1}(x \dot{-} 2)), && \text{by induction, } x \geq x_0' + 2,\\
&= B_n^2(B_n^{x \dot{-} 2}(1))\\
&= B_n^x(1)\\
&= B_{n+1}(x), \quad x \geq 2.
\end{aligned}$$

Small Ackermann Functions – LOOP Computability

For each $n \geq 1$, B_n is LOOP-n computable.

Proof

First, define the LOOP-1 program

$$P_1 = \bar{C}(1;2); \bar{C}(1;3); Z1; A1; [Z1]3; [A1; A1]2. \tag{171}$$

For each input x, the program evaluates as follows:

0	1	2	3	4	...	registers
0	x	0	0	0	...	init
0	x	x	0	0	...	
0	x	x	x	0	...	
0	0	x	x	0	...	
0	1	x	x	0	...	
0	1	0	0	0	...	$x = 0$ end
0	$2x$	0	0	0	...	$x \neq 0$ end

By (158), the program satisfies $\|P_1\|_{1,1} = B_1$.

Proof (Cont'd)

Suppose there is a normal LOOP-n program P_n that computes B_n; that is, $\|P_n\|_{1,1} = B_n$, for $n \geq 1$. Put $m = n(P_n) + 1$[a] and consider the LOOP-$n+1$ program

$$P_{n+1} = [Am]1; A1; [P_n]m. \tag{172}$$

Note that the register R_m is unused in the program P_n. The program P_{n+1} computes $B_n^x(1)$ as follows:

0	1	2	3	4	...	m	...	registers
0	x	0	0	0	...	0	...	init
0	0	0	0	0	...	x	...	
0	1	0	0	0	...	x	...	
0	$B_n^x(1)$	0	0	0	...	0	...	end

Since $B_{n+1}(x) = B_n^x(1)$, we have $\|P_{n+1}\|_{1,1} = B_{n+1}$. □

[a] $n(P)$ is the largest register used in program P.

Complexity of LOOP Programs– Contents

- Complexity
- Bound on LOOP computable functions
- Proper LOOP hierarchy

Complexity of LOOP Programs

Function $\lambda : \mathcal{P}_{\mathrm{LOOP}} \to \mathbb{N}_0$ assigns to each LOOP program a measure of *complexity:*

$$\begin{aligned}
\lambda(A\sigma) &= 1, \quad \sigma \in \mathbb{N}_0, && (173)\\
\lambda(Z\sigma) &= 1, \quad \sigma \in \mathbb{N}_0, && (174)\\
\lambda(\bar{C}(\sigma;\tau)) &= 1, \quad \sigma \neq \tau,\ \sigma,\tau \in \mathbb{N}_0, && (175)\\
\lambda(P;Q) &= \lambda(P) + \lambda(Q), \quad P,Q \in \mathcal{P}_{\mathrm{LOOP}}, && (176)\\
\lambda([P]\sigma) &= \lambda(P) + 2, \quad P \in \mathcal{P}_{\mathrm{LOOP}},\ \sigma \in \mathbb{N}_0. && (177)
\end{aligned}$$

For each LOOP program P, the *complexity measure* $\lambda(P)$ is

$$\lambda(P) = x_0 + 2x_1, \tag{178}$$

where x_0 is the number of LOOP-0 subprograms of P and x_1 is the number of iterations of P.

Complexity of LOOP Programs – Example

LOOP program $P = \bar{C}(1;3);[\bar{C}(2;1);A2]3$ has the complexity

$$\begin{aligned}\lambda(P) &= \lambda(\bar{C}(1;3)) + \lambda([\bar{C}(2;1);A2]3)\\ &= 1 + \lambda(\bar{C}(2;1);A2) + 2\\ &= 3 + \lambda(\bar{C}(2;1)) + \lambda(A2)\\ &= 5.\end{aligned}$$

Complexity of LOOP Programs – Boundedness

A k-adic total function $f : \mathbb{N}_0^k \to \mathbb{N}_0$ is said to be *bounded* by a monadic total function $g : \mathbb{N}_0 \to \mathbb{N}_0$ if for each $\boldsymbol{x} \in \mathbb{N}_0^k$,

$$f(\boldsymbol{x}) \leq g(\max(\boldsymbol{x})), \tag{179}$$

where $\max(\boldsymbol{x}) = \max\{x_1, \ldots, x_k\}$ for $\boldsymbol{x} = (x_1, \ldots, x_k) \in \mathbb{N}_0^k$.

Complexity of LOOP Programs – Example

For each $\boldsymbol{x} \in \mathbb{N}_0^k$,

$$f(\boldsymbol{x}) \leq g(\max(\boldsymbol{x})).$$

- Addition function $f : \mathbb{N}_0^2 \to \mathbb{N}_0 : (x, y) \mapsto x + y$ is bounded by the function $g : \mathbb{N}_0 \to \mathbb{N}_o : x \mapsto 2x$, since $x + y \leq 2 \max\{x, y\}$ for all $x, y \in \mathbb{N}_0$.
- Multiplication function $f : \mathbb{N}_0^2 \to \mathbb{N}_0 : (x, y) \mapsto xy$ is bounded by the function $g : \mathbb{N}_0 \to \mathbb{N}_o : x \mapsto x^2$, since $xy \leq \max\{x, y\}^2$ for all $x, y \in \mathbb{N}_0$.
- Exponentiation function $f : \mathbb{N}_0^2 \to \mathbb{N}_0 : (x, y) \mapsto x^y$ is bounded by the function $g : \mathbb{N}_0 \to \mathbb{N}_o : x \mapsto x^x$, since $x^y \leq \max\{x, y\}^{\max\{x,y\}}$ for all $x, y \in \mathbb{N}_0$.

Complexity of LOOP Programs – Main Result

For each LOOP-n program P and each input $\boldsymbol{x} \in \mathbb{N}_0^k$ with $k \geq n(P)$,[a]

$$\max(\|P\|_{k,k}(\boldsymbol{x})) \leq B_n^{\lambda(P)}(\max(\boldsymbol{x})), \tag{180}$$

i.e., the maximum value of P computed in the registers $R_1, \ldots, R_k$ is bounded by the $\lambda(P)$-th power of the n-th small Ackermann function.

[a] $n(P)$ is the largest register used in program P.

*Proof

First, let P be a LOOP-0 program; that is, $P = P_1; P_2; \ldots; P_m$, where P_i, $1 \leq i \leq m$, is a LOOP-0 operation: $A\sigma$, $Z\sigma$ or $\bar{C}(\sigma, \tau)$. Then

$$\begin{aligned} \max(\|P\|_{k,k}(\boldsymbol{x})) &\leq m + \max(\boldsymbol{x}) \\ &= \lambda(P) + \max(\boldsymbol{x}) \\ &\leq B_0^{\lambda(P)}(\max(\boldsymbol{x})), \end{aligned}$$

since the "worst-case" operation is incrementation (of the register with the maximal initial value).

Proof (Cont'd)

Suppose the assertion holds for LOOP-n programs.

Let P be a LOOP-$n+1$ program of the form $P = Q;R$, where Q and R are LOOP-$n+1$ programs.

Then for $k \geq \max\{n(Q), n(R)\}$,

$$\begin{aligned}
\max(\|Q;R\|_{k,k}(\boldsymbol{x})) &= \max(\|R\|_{k,k}(\|Q\|_{k,k}(\boldsymbol{x}))) \\
&\leq B_{n+1}^{\lambda(R)}(\max(\|Q\|_{k,k}(\boldsymbol{x}))), \text{ by induction,} \\
&\leq B_{n+1}^{\lambda(R)}(B_{n+1}^{\lambda(Q)}(\max(\boldsymbol{x}))), \text{ by induction,} \\
&= B_{n+1}^{\lambda(R)+\lambda(Q)}(\max(\boldsymbol{x})) \\
&= B_{n+1}^{\lambda(P)}(\max(\boldsymbol{x})).
\end{aligned}$$

Proof (Cont'd)

Let P be a LOOP-$n+1$ program of the form $P = [Q]\sigma$, where Q is a LOOP-n program.

Then for $k \geq \max\{n(Q), \sigma\}$,

$$\begin{aligned}
\max(\|[Q]\sigma\|_{k,k}(\boldsymbol{x})) &= \max(\|Q; S\sigma\|_{k,k}^{x_\sigma}(\boldsymbol{x})) \\
&\leq \max(\|Q\|_{k,k}^{x_\sigma}(\boldsymbol{x})) \qquad (181) \\
&\leq B_n^{x_\sigma \cdot \lambda(Q)}(\max(\boldsymbol{x})), \text{ by induction,} \\
&\leq B_{n+1}^{\lambda(Q)+2}(\max(\boldsymbol{x})), \text{ see (182),} \\
&= B_{n+1}^{\lambda(P)}(\max(\boldsymbol{x})).
\end{aligned}$$

The last inequality follows from the assertion

$$B_n^{x \cdot l}(y) \leq B_{n+1}^{l+2}(y), \quad x \leq y,\ l \geq 0, \qquad (182)$$

which can be proved by using the above properties of the small Ackermann functions. □

LOOP Hierarchy – Main Result

For each $n \geq 0$, the class of LOOP-n functions is a proper subclass of the class of LOOP-$n+1$ functions.

Proof.

- The n-th small Ackermann function B_n is LOOP-n computable.
- Suppose B_{n+1} is LOOP-n computable. Then by (180), there is an integer $m \geq 0$ such that for all $x \in \mathbb{N}_0$,

$$B_{n+1}(x) \leq B_n^m(x).$$

However, by (170), B_{n+1} grows faster than any power of B_n. Contradiction!

Computable Total Functions

Hierarchy of total computable functions (all inclusions are strict):

$$\mathcal{T} = \mathcal{R}_{\text{recursive}}$$
$$\vdots \; ?$$
$$\mathcal{P} = \mathcal{F}_{\text{LOOP}}$$
$$\vdots$$
$$\cdots$$
$$\vdots$$
$$\mathcal{F}_{\text{LOOP}(n)}$$
$$\vdots$$
$$\cdots$$
$$\vdots$$
$$\mathcal{F}_{\text{LOOP}(0)}$$

Ackermann Function – Contents

- Definition
- Hierarchy of parametrized functions
- Redefinition using small Ackermann functions
- Ackermann function is recursive but not primitive recursive
- Hierarchy of computable total functions

Ackermann Function – Definition

Ackermann function $A : \mathbb{N}_0^2 \to \mathbb{N}_0$ is defined as

$$A(0, y) = y + 1, \tag{183}$$
$$A(x + 1, 0) = A(x, 1), \tag{184}$$
$$A(x + 1, y + 1) = A(x, A(x + 1, y)). \tag{185}$$

Original definition by Wilhelm Ackermann (1928) had three arguments; above definition by Rózsa Péter (1935).

Ackermann Function – Totality

Ackermann function is a total function.

Proof.

- Lexicographic ordering on $\mathbb{N}_0^2$:

$$(x, y) \succ_{\text{lex}} (x', y') \quad :\Longleftrightarrow \quad x > x' \vee (x = x' \wedge y > y').$$

- The evaluation of $A(x, y)$ yields in each step (183-185) a unique expression $A(x', y')$ which is evaluated next, if any.
- Here (x, y) is lexicographically larger than (x', y'), i.e., $(x + 1, 0) \succ_{\text{lex}} (x, 1)$ and $(x + 1, y + 1) \succ_{\text{lex}} (x + 1, y)$.
- Thus the derivation provides a strictly decreasing chain of pairs in $\mathbb{N}_0^2$.
- The lexicographic ordering on $\mathbb{N}_0^2$ is well-founded, i.e., there are no infinitely decreasing chains. Hence, each derivation must terminate.

Ackermann Function – Example

$$\begin{aligned} A(1,2) &= A(0,A(1,1)) \\ &= A(0,A(0,A(1,0))) \\ &= A(0,A(0,A(0,1))) \\ &= A(0,A(0,2)) \\ &= A(0,3) \\ &= 4. \end{aligned}$$

Here

$$(1,2) \succ_{\text{lex}} (1,1) \succ_{\text{lex}} (1,0) \succ_{\text{lex}} (0,1).$$

Ackermann Function – Partial Recursiveness

Ackermann function is partial recursive.

Proof.

- Ackermann function is a computable function in the intuitive sense.
- Intuitive means that we have not given a definition in terms of the construction of partial recursive functions (starting from the base functions and using a finite number of compositions, primitive recursions and minimalization).
- By Church's thesis, the Ackermann function is partial recursive (recursive, since total). □

Ackermann Function – Hierarchy of Operations

Monadic functions $A_x : \mathbb{N}_0 \to \mathbb{N}_0 : y \mapsto A(x,y)$, $x \in \mathbb{N}_0$:

- Successor: $A_0(y) = y + 1 = \nu(y+3) - 3$,
- Addition: $A_1(y) = y + 2 = f_+(2, y+3) - 3$,
- Multiplication: $A_2(y) = 2y + 3 = f_\cdot(2, y+3) - 3$,
- Exponentiation: $A_3(y) = 2^{y+3} - 3 = f_{\exp}(2, y+3) - 3$,
- Tetration: $A_4(y) = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{y+3} - 3 = f_{\mathrm{tet}}(2, y+3) - 3$.

*Proof

- $A_0(y) = A(0, y) = y + 1$.
- $A_1(0) = A(1, 0) = A(0, 1) = 2$. Suppose $A_1(y) = y + 2$. Then $A_1(y + 1) = A(1, y + 1) = A(0, A(1, y)) = A(1, y) + 1 = (y + 1) + 2$.
- $A_2(0) = A(2, 0) = A(1, 1) = 3$. Suppose $A_2(y) = 2y + 3$. Then $A_2(y + 1) = A(2, y + 1) = A(1, A(2, y)) = A(2, y) + 2 = (2y + 3) + 2 = 2(y + 1) + 3$.
- $A_3(0) = A(3, 0) = A(2, 1) = 5$. Suppose $A_3(y) = 2^{y+3} - 3$. Then $A_3(y + 1) = A(3, y + 1) = A(2, A(3, y)) = 2 \cdot (2^{y+3} - 3) + 3 = 2^{(y+1)+3} - 3$.

*Proof (Cont'd)

- $A_4(0) = A(4,0) = A(3,1) = 2^4 - 3 = 2^{2^2} - 3$. Suppose that

$$A_4(y) = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}_{y+3} - 3.$$

Then

$$\begin{aligned} A_4(y+1) &= A(4,y+1) = A(3,A(4,y)) = 2^{A(4,y)+3} - 3 \\ &= \underbrace{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}_{(y+1)-3} - 3. \end{aligned}$$

Pointless Large Number Stuff?

Ackermann function can be used as a benchmark of a compiler's ability to optimize recursion (Ygne Sundblad, 1971).

- The computation of $A(x, y)$ requires deep recursion, which can provoke stack overflow; A is largely immune against compiler optimization.
- The function
$$f(y) = A(3, y)$$
has $A(3, y) + 12^y - 2$ function calls.[a]
- How large can y be taken such that the compiler is able to compute $f(y)$?
- My `Maple`$^{\text{TM}}$ implemention works until $y = 8$. But $f(9)$ yields `too many levels of recursion`.

Compare this with the direct computation of $A(3, y) = 2^{y+3} - 3$ by a CAS.

[a] Wikipedia, 2020

Ackermann Function – Redefinition

Combination of the small Ackermann functions into the dyadic function $A : \mathbb{N}_0^2 \to \mathbb{N}_0$,

$$A(x, y) = B_x(y), \quad x, y \in \mathbb{N}_0. \tag{186}$$

By the properties of the small Ackermann functions, A is a total computable function.

Ackermann Function – Redefinition

Function A in (186) is given by the equations

$$\begin{aligned} A(0,y) &= B_0(y), && (187)\\ A(x+1,0) &= 1, && (188)\\ A(x+1,y+1) &= A(x,A(x+1,y)). && (189) \end{aligned}$$

Proof.

We have

$$\begin{aligned} A(0,y) &= B_0(y),\\ A(x+1,0) &= B_{x+1}(0) = B_x^0(1) = 1,\\ A(x+1,y+1) &= B_{x+1}(y+1) = B_x^{y+1}(1) = B_x(B_x^y(1))\\ &= B_x(B_{x+1}(y)) = B_x(A(x+1,y)\\ &= A(x,A(x+1,y)). \end{aligned}$$

Ackermann Function – Main Result

Ackermann function A is recursive but not primitive recursive.

Proof.

Suppose A is primitive recursive. Then A is LOOP-n computable for some $n \geq 0$. By (180), there is an integer $p \geq 0$ such that for all $x, y \in \mathbb{N}_0$,

$$A(x, y) \leq B_n^p(\max\{x, y\}). \tag{190}$$

By (170), there is a number $y_0 \geq 0$ such that for all $y \geq y_0$,

$$B_n^p(y) < B_{n+1}(y). \tag{191}$$

Taking $y_0 \geq n + 1$, $x = n + 1$ and $y \geq y_0$ leads to a contradiction:

$$\begin{aligned} A(n+1, y) &\leq B_n^p(\max\{n+1, y\}) = B_n^p(y) \\ &< B_{n+1}(y) = A(n+1, y). \end{aligned} \tag{192}$$

Computable Total Functions

Hierarchy of total computable functions (all inclusions are strict):

$$\mathcal{T} = \mathcal{R}_{\text{recursive}}$$
$$\Big| \; !$$
$$\mathcal{P} = \mathcal{F}_{\text{LOOP}}$$
$$\Big| \; \ldots$$
$$\mathcal{F}_{\text{LOOP}(n)}$$
$$\Big| \; \ldots$$
$$\mathcal{F}_{\text{LOOP}(0)}$$

Superpowers – Contents

- Sequence of dyadic operations
- General definition
- Tetration and pentation
- *Infinite tetration
- Ackermann class and functional
- Representation of Ackermann function

Superpowers – Inside

- Ackermann function is a classical example of a recursive function, which is not primitive recursive.
- Closed form of Ackermann function will be developed using superpowers.
- Superpowers were introduced by Donald Knuth (1976).
- Knuth's superpower notation is based on the infinite sequence of operators

$$+, \cdot, \uparrow, \uparrow^2, \uparrow^3, \ldots,$$

where $\uparrow$ denotes exponentiation.

Tetration

$$a \uparrow^2 n = \underbrace{a^{a^{\cdot^{\cdot^{\cdot^{a}}}}}}_{n}. \tag{193}$$

Superpowers – Sequence

Consider the following sequence of dyadic functions,

$$\begin{aligned} a+n &= a+1+\ldots+1, \quad (n\ 1\text{'s}),\\ a\cdot n &= a+a+\ldots+a, \quad (n\ a\text{'s}),\\ a\uparrow n &= a\cdot a\cdot\ldots\cdot a, \quad (n\ a\text{'s}),\\ a\uparrow^2 n &= a\uparrow a\uparrow\ldots\uparrow a, \quad (n\ a\text{'s}),\\ a\uparrow^3 n &= a\uparrow^2 a\uparrow^2\ldots\uparrow^2 a, \quad (n\ a\text{'s}),\\ &\vdots \end{aligned}$$

where all operations $*$ are assumed to be *right associative*; i.e.,

$$a*b*c = a*(b*c).$$

Exponentiation is not associative, since
$2\uparrow(1\uparrow 2) = 2\uparrow 1 = 2^1 = 2$ and $(2\uparrow 1)\uparrow 2 = 2\uparrow 2 = 2^2 = 4$.

Superpowers – Definition

- *Superpowers* $a \uparrow^m n$ are defined inductively,

$$a \uparrow^1 n \quad = \quad a \uparrow n \tag{194}$$

and for $m \geq 2$,

$$\begin{aligned} a \uparrow^m n \quad &= \quad a \underbrace{\uparrow\uparrow \ldots \uparrow}_{m} n \\ &= \quad \underbrace{a \underbrace{\uparrow \ldots \uparrow}_{m-1} a \underbrace{\uparrow \ldots \uparrow}_{m-1} a \ldots \underbrace{\uparrow \ldots \uparrow}_{m-1} a}_{a \text{ appears } n\text{-times}} \\ &= \quad a \uparrow^{m-1} a \uparrow^{m-1} \ldots \uparrow^{m-1} a, \ (n \ a\text{'s}). \end{aligned} \tag{195}$$

- Evaluation by right associativity

$$\begin{aligned} & a \uparrow^m n \\ &= \quad a \uparrow^{m-1} (a \uparrow^{m-1} (\ldots (a \uparrow^{m-1} a) \ldots)), \quad (n \ a\text{'s}). \end{aligned} \tag{196}$$

Superpowers – Definition

- Definition of $a \uparrow^m n$ can be extended to include $m \in \{-2, -1, 0\}$ by setting

$$a \uparrow^{-2} n = n + 1, \tag{197}$$
$$a \uparrow^{-1} n = a + n, \tag{198}$$
$$a \uparrow^{0} n = a \cdot n. \tag{199}$$

- Definition (195) is only valid for $m \geq 0$, since e.g.

$$a \uparrow^{-1} 3 = a + 3$$

and

$$a \uparrow^{-1} 3 = a \uparrow^{-2} (a \uparrow^{-2} a) = a \uparrow^{-2} (a + 1) = a + 2.$$

Superpowers – Recursion

For all numbers $m \geq -1$ and $n \geq 2$,

$$a \uparrow^m n = a \uparrow^{m-1} a \uparrow^m (n-1). \tag{200}$$

Proof.

We have

$$\begin{aligned} a \uparrow^m n &= a \uparrow^{m-1} a \uparrow^{m-1} \ldots \uparrow^{m-1} a, \quad (n \text{ a's}), \\ &= a \uparrow^{m-1} (a \uparrow^{m-1} \ldots \uparrow^{m-1} a) \\ &= a \uparrow^{m-1} a \uparrow^m (n-1). \end{aligned}$$

Superpowers – Example

For all $m \geq -1$.

$$2 \uparrow^m 2 = 4. \tag{201}$$

Proof.

We have $2 \uparrow^{-1} 2 = 2 + 2 = 4$.
Suppose $2 \uparrow^m 2 = 4$ holds for some $m \geq -1$. Then by (200),

$$2 \uparrow^{m+1} 2 = 2 \uparrow^m (2 \uparrow^{m+1} 1).$$

But $2 \uparrow^{m+1} 1 = 2$ and so by induction,

$$2 \uparrow^{m+1} 2 = 2 \uparrow^m 2 = 4.$$

Tetration

Tetration is the dyadic function

$$a \uparrow^2 n = \underbrace{a^{a^{\cdot^{\cdot^{\cdot^{a}}}}}}_{n}. \tag{202}$$

We have $a \uparrow^2 0 = 1$, $a \uparrow^2 1 = a$, and by (200),

$$a \uparrow^2 (n+1) = a \uparrow (a \uparrow^2 n). \tag{203}$$

For $a = 2$, the first few values are

$$\begin{aligned}
2 \uparrow^2 1 &= 2, \\
2 \uparrow^2 2 &= 2 \uparrow 2 = 4, \\
2 \uparrow^2 3 &= 2 \uparrow (2 \uparrow 2) = 2 \uparrow 4 = 16, \\
2 \uparrow^2 4 &= 2 \uparrow (2 \uparrow (2 \uparrow 2)) = 2 \uparrow 16 = 65536, \\
2 \uparrow^2 5 &= 2 \uparrow 65536 \approx 2 \cdot 10^{19728}.
\end{aligned}$$

Pentation

Pentation is the expression

$$a \uparrow^3 n, \tag{204}$$

which is a power tower with $n-1$ power towers given by tetration. We have

$$\begin{aligned} 2 \uparrow^3 3 &= 2 \uparrow^2 (2 \uparrow^2 2) = 2^{16} = 65,536, \\ 3 \uparrow^3 3 &= 3 \uparrow^2 (3 \uparrow^2 3) \\ &= 3 \uparrow^2 7,625,597,848,987, \end{aligned}$$

where

$$\begin{aligned} 3 \uparrow^2 3 &= 3 \uparrow (3 \uparrow 3) = 3 \uparrow 27 \\ &= 7,625,597,848,987 \approx 7,6 \cdot 10^{12}. \end{aligned}$$

The number $3 \uparrow^3 3$ is called *tritri* and is a power tower with $7,625,597,848,987$ 3's.

*Infinite Tetration

Consider the infinite tetration of real numbers

$$x^{x^{x^{\cdot^{\cdot^{\cdot}}}}} = a \tag{205}$$

for $x > 0$ and its convergence to a number a.

Let $x > 0$. The tetration limit

$$x^{x^{x^{\cdot^{\cdot^{\cdot}}}}}$$

exists for $x \in [\frac{1}{e^e}, e^{\frac{1}{e}}]$.

*Infinite Tetration – Proof

The expression (205) has the form $a = x^a$. For this, we consider the real-valued function $f(t) = x^t$ for $x > 0$. The convergence $\lim_{t \to a} f(t) = f(a)$ means that

$$|f(t) - f(a)| < |t - a|$$

with $f(a) = a$ and t sufficiently close to a. Thus

$$\frac{|f(t) - f(a)|}{|t - a|} < 1.$$

By the mean value theorem,

$$|f'(\xi)| < 1$$

for some ξ between t and a. Since f' is continuous, we must have

$$|f'(a)| \leq 1.$$

Moreover, $f(t) = e^{\ln(x^t)} = e^{t \ln(x)}$ and so $f'(t) = e^{t \ln(x)} \ln(x) = f(t) \ln(x)$. Thus

$$f'(a) = x^a \ln(x) = a \ln(x) = \ln(x^a) = \ln(a)$$

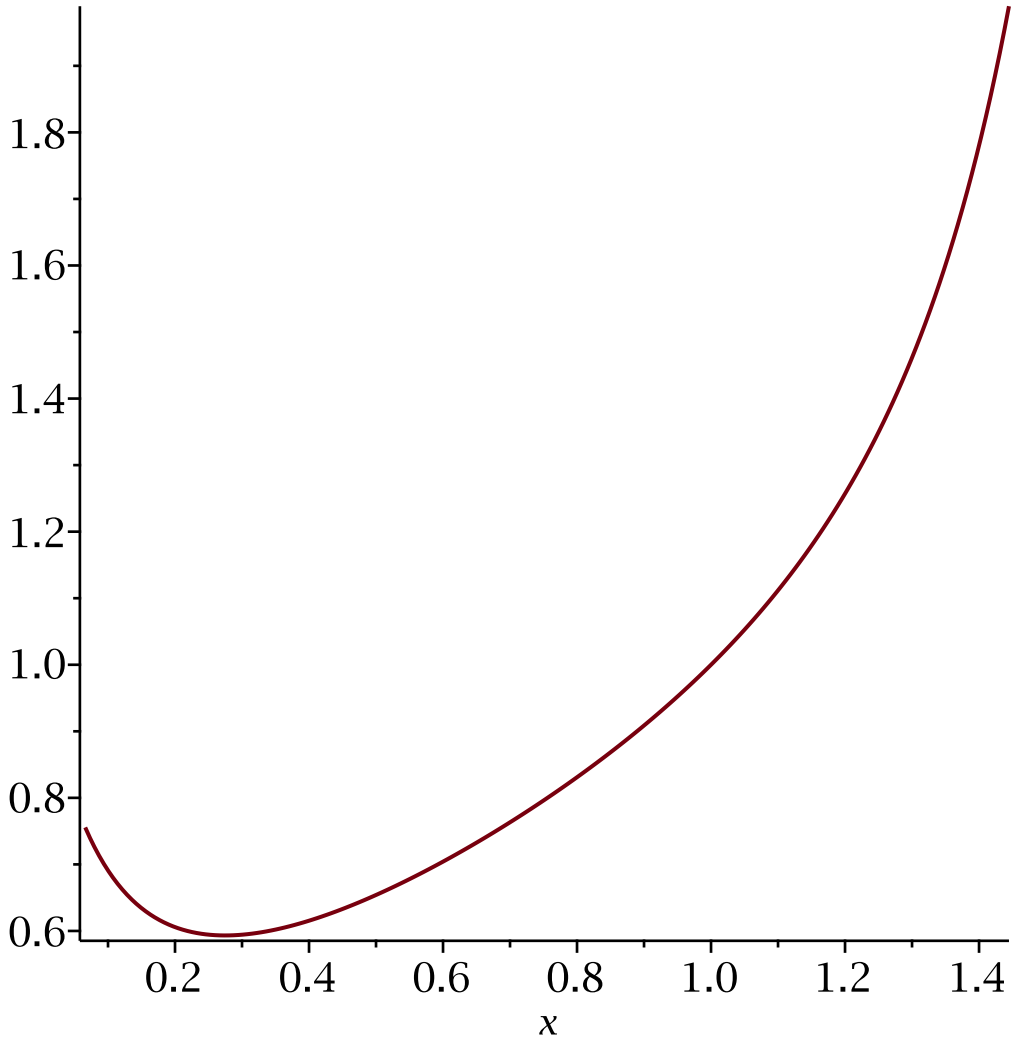
and therefore

$$|\ln(a)| \leq 1.$$

This leads to $a \in [1/e, e]$. But $a = x^a$ and so $x = a^{1/a}$. Hence $x \in [1/e^e, e^{1/e}]$ as claimed.

*Tetration (Maple^{TM})

```
> plot( x^(x^(x^x)), x = 1/e^e .. e^(1/e));
```



Ackermann Functional and Class

- *Ackermann functional* Γ is defined on the set of dyadic functions $f : \mathbb{N}_0^2 \to \mathbb{N}_0$ as follows,

$$\Gamma(f(m,n)) = f(m-1, f(m,n-1)), \quad m,n \in \mathbb{N}. \tag{206}$$

- *Ackermann class* $\mathcal{A}$ is the set of all dyadic functions $f : \mathbb{N}_0^2 \to \mathbb{N}_0$ which are fixed points of the Ackermann functional Γ for large enough values of m and n; i.e., there are m_0 and n_0 such that for all $m \geq m_0$ and $n \geq n_0$,

$$f(m,n) = \Gamma(f(m,n)). \tag{207}$$

By (185), the Ackermann function belongs to the class $\mathcal{A}$:

$$A(m+1, n+1) = A(m, A(m+1, n)).$$

Ackermann Class – Boundaries

- Definition of function in Ackermann class requires *boundary conditions*.
- Boundary conditions for the Ackermann function,

$$A(0,n) = n+1, \quad n \geq 0, \tag{208}$$

$$A(m,0) = A(m-1,1), \quad m \geq 1. \tag{209}$$

Ackermann Class – Superpowers

Consider the superpowers

$$f(a,m,n) = a \uparrow^m n. \tag{210}$$

Then by (200),

$$\begin{aligned} f(a,m,n) &= a \uparrow^m n = a \uparrow^{m-1} (a \uparrow^m (n-1)) \\ &= f(a, m-1, f(a,m,n-1)). \end{aligned} \tag{211}$$

For fixed $a \in \mathbb{N}_0$, define the dyadic function $f_a : \mathbb{N}_0^2 \to \mathbb{N}_0$ as

$$f_a(m,n) = f(a,m,n). \tag{212}$$

Then by (211),

$$f_a(m,n) = f_a(m-1, f_a(m,n-1)). \tag{213}$$

Hence, $f_a = f(a,\cdot,\cdot)$ belongs to the Ackermann class.

Ackermann Class – Main Result

Let a, α, k, m and n be numbers with $n + k \geq 2$ and $m + \alpha \geq 1$. Then the dyadic function

$$f(m, n) = a \uparrow^{m+\alpha} (n + k) - k \tag{214}$$

belongs to the Ackermann class for fixed values of a, α, and k.

Proof.

By (200),

$$\begin{aligned} f(m,n) &= a \uparrow^{m+\alpha} (n+k) - k \\ &= a \uparrow^{m+\alpha-1} (a \uparrow^{m+\alpha} (n+k-1)) - k \\ &= a \uparrow^{m-1+\alpha} (a \uparrow^{m+\alpha} (n-1+k) - k + k) - k \\ &= a \uparrow^{m-1+\alpha} (f(m, n-1) + k) - k \\ &= f(m-1, f(m, n-1)). \end{aligned}$$

Ackermann Class – Main Result

Ackermann function

$$A(m,n) = 2 \uparrow^{m-2} (n+3) - 3, \quad m,n \in \mathbb{N}_0. \tag{215}$$

Proof

- By (214), the function $A(m,n)$ with $a = 2$, $\alpha = -2$, and $k = 3$ belongs to the Ackermann class.
- Boundary condition $A(0,n) = n+1$ for all $n \geq 0$ is satisfied, since by definition

$$A(0,n) = 2 \uparrow^{-2} (n+3) - 3 = (n+4) - 3 = n+1.$$

Proof (Cont'd)

- Boundary condition $A(m,0) = A(m-1,1)$ for all $m \geq 1$ is also fulfilled. To see this, the left-hand side gives by (200),

$$2 \uparrow^{m-2} 3 - 3 = 2 \uparrow^{m-3} (2 \uparrow^{m-2} 2) - 3,$$

while the right-hand side gives

$$2 \uparrow^{m-3} 4 - 3.$$

Moreover, by (201), $2 \uparrow^{m-2} 2 = 4$ for each $m \geq 1$ and hence the boundary condition holds.

Ackermann Function – Example

By (215),

$$\begin{aligned}
A(0,n) &= 2 \uparrow^{-2} (n+3) - 3 = n+3+1-3 = n+1,\\
A(1,n) &= 2 \uparrow^{-1} (n+3) - 3 = n+3+2-3 = n+2,\\
A(2,n) &= 2 \uparrow^{0} (n+3) - 3 = 2(n+3) - 3 = 2n+3,\\
A(3,n) &= 2 \uparrow^{1} (n+3) - 3 = 2^{n+3} - 3,\\
A(4,n) &= 2 \uparrow^{2} (n+3) - 3 = \underbrace{a^{a^{\cdot^{\cdot^{\cdot^{a}}}}}}_{n+3} - 3.
\end{aligned}$$

Ackermann Function – Example

Tetration

$$
\begin{aligned}
A(4,n) &= 2 \uparrow^2 (n+3) - 3, \quad n \geq 0, \\
A(4,0) &= 2 \uparrow^2 3 - 3 = 2 \uparrow (2 \uparrow 2) - 3 = 2^4 - 3 = 13, \\
A(4,1) &= 2 \uparrow^2 4 - 3 = 2 \uparrow (2 \uparrow (2 \uparrow 2)) - 3 = 2^{2^4} - 3 = 65533, \\
A(4,2) &= 2 \uparrow^2 5 - 3 = 2 \uparrow (2 \uparrow (2 \uparrow (2 \uparrow 2))) - 3 = 2^{65536} - 3, \\
A(4,3) &= 2 \uparrow^2 6 - 3 = 2 \uparrow (2 \uparrow (2 \uparrow (2 \uparrow (2 \uparrow 2)))) - 3 \\
&= 2^{2^{65536}} - 3, \\
A(4,4) &= 2 \uparrow^2 7 - 3 = 2 \uparrow (2 \uparrow (2 \uparrow (2 \uparrow (2 \uparrow (2 \uparrow 2))))) - 3 \\
&= 2^{2^{2^{65536}}} - 3.
\end{aligned}
$$

The computation of $A(4,1)$ may already abort due to too many levels of recursion.

Skills

- Evaluation of small Ackermann function
- Computation of complexity of LOOP program
- Evaluation of Ackermann function for small values
- Evaluation and interpretation of superpowers for small values

Part VI

Programming Systems

Programming Systems – Motivation

- Entry point into the theory of computation.
- Gödel numbering of partial recursive functions.
- Main theorems (parametrization, universal computation, normalization of minimalization).

Addition program P_+

s_0 :	0	if $x_2 = 0$	3	1
s_1 :	1	$x_1 \leftarrow x_1 + 1$	2	
s_2 :	2	$x_2 \leftarrow x_2 - 1$	0	

has Gödel number

$$\rho(P_+) = 1,269,420,373,033,475,160,642,134.$$

Programming Systems – Contents

- Gödel numbering of GOTO programs and partial recursive functions
- Kleene's smn theorem (parametrization)
- Univeral functions
- Kleene's normal form

Reading: Zimmermann, Section 2.3 (Pairing Functions), Chapter 5

Gödel Numbering – Contents

- Cantor's pairing function
- Gödel numbering of $\mathbb{N}_0^*$
- Gödel numbering of GOTO programs
- Gödel numbering of partial recursive functions

In mathematical logic, *Gödel numbering* refers to a function that assigns to each well-formed formula of some formal language a unique natural number called its *Gödel number* (Kurt Gödel, 1906-1978).

`Maple`$^{\text{TM}}$ demo software: `cc-J-proc.mw` (clickable worksheet)

Cantor's Pairing Function – Definition

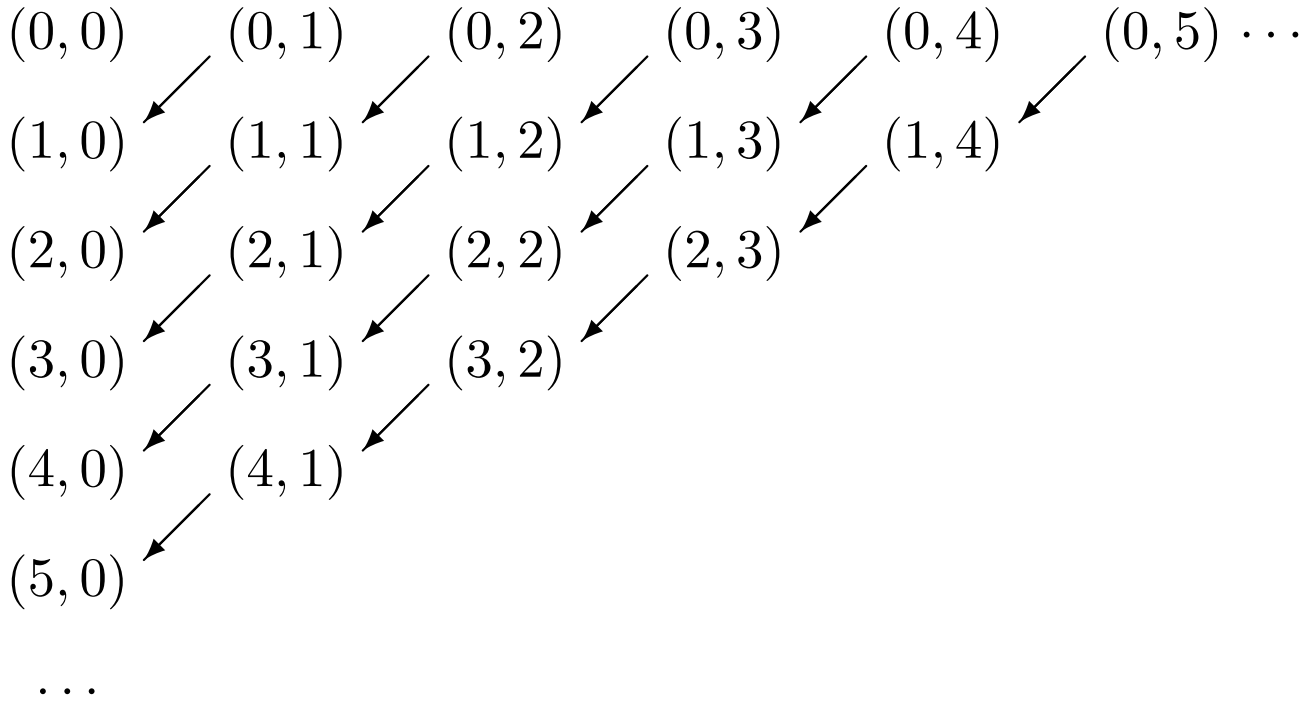
- Each primitive recursive bijection $f : \mathbb{N}_0^2 \to \mathbb{N}_0$ is called a *pairing function*.
- *Cantor's pairing function* $\sigma_2 : \mathbb{N}_0^2 \to \mathbb{N}_0$ is defined as

$$\begin{aligned} \sigma_2(m,n) &= \binom{m+n+1}{2} + m \qquad (216) \\ &= \frac{(m+n)(m+n+1)}{2} + m. \end{aligned}$$

As a consequence, Cartesian product $\mathbb{N}_0^2$ is denumerable since $\mathbb{N}_0$ is denumerable.

Proof.

- List the elements of $\mathbb{N}_0^2$ in form of a table:



- σ_2 provides a numbering of the elements $\mathbb{N}_0^2$ by running through the anti-diagonals starting with $(0, 0)$. Hence, σ_2 is bijective.

Cantor's Pairing Function

Cantor's pairing function σ_2 is primitive recursive.

Proof

- By (83), integral division function $\div$ is primitive recursive by bounded minimalization.
- By the representation of the binomial coefficient,

$$\binom{n}{2} = \frac{(n-1)n}{2} = \div((n-1)\cdot n, 2), \quad n \geq 2,$$

we have

$$\sigma_2(m,n) = \div((m+n)\cdot(m+n+1), 2) + m.$$

Thus σ_2 is a composition of primitive recursive function and hence also primitive recursive. □

Cantor's Pairing Function – Inverse

There are the coordinate functions $\kappa_2, \tau_2 : \mathbb{N}_0 \to \mathbb{N}_0$ such that for all $m, n \in \mathbb{N}_0$:

$$\kappa_2(\sigma_2(m,n)) = m, \tag{217}$$
$$\tau_2(\sigma_2(m,n)) = n, \tag{218}$$
$$\sigma_2(\kappa_2(n), \tau_2(n)) = n. \tag{219}$$

Proof

Let $n \in \mathbb{N}_0$.

- There exists a number $t \geq 0$ such that

$$\frac{1}{2}t(t+1) \leq n < \frac{1}{2}(t+1)(t+2). \tag{220}$$

- Put

$$m = n - \frac{1}{2}t(t+1). \tag{221}$$

Proof (Cont'd)

Put $m = n - \frac{1}{2}t(t+1) = n - \binom{t+1}{2}$.

- Then

$$\begin{aligned} m &= n - \frac{1}{2}t(t+1) \\ &\leq \left[\frac{1}{2}(t+1)(t+2) - 1\right] - \left[\frac{1}{2}t(t+1)\right] = t. \end{aligned}$$

- Put

$$\kappa_2(n) = m \quad \text{and} \quad \tau_2(n) = t - m. \tag{222}$$

- Then

$$\begin{aligned} \sigma_2(\kappa_2(n), \tau_2(n)) &= \sigma_2(m, t-m) \\ &= \binom{m + (t-m) + 1}{2} + m = n. \end{aligned}$$

□

Cantor's Pairing Function – Example

Let $n = 17$.

- Then
$$\frac{1}{2}t(t+1) \leq 17 < \frac{1}{2}(t+1)(t+2)$$
is satisfied by $t = 5$, since $15 \leq 17 < 21$.
- Put $m = n - \frac{1}{2}t(t+1) = 2$.
- Then $\kappa_2(17) = m = 2$ and $\tau_2(17) = t - m = 3$.
- Check:
$$\sigma_2(2,3) = \binom{2+3+1}{2} + 2 = \binom{6}{2} + 2 = 17. \qquad (223)$$

Cantor's Pairing Function

Coordinate functions κ_2 and τ_2 are primitive recursive.

Proof.

Let $n \in \mathbb{N}_0$.

- The number t in (220) can be obtained by bounded minimalization.
- Thus $\kappa_2(n) = n - \frac{1}{2}t(t+1)$ and $\tau_2(n) = t - \kappa_2(n)$ are compositions of primitive recursive functions and hence primitive recursive.

Gödel Numbering of $\mathbb{N}_0^*$

$\mathbb{N}_0^*$ denotes the union of all Cartesian products $\mathbb{N}_0^\ell$, $\ell \geq 0$, i.e.,

$$\mathbb{N}_0^* = \bigcup_{\ell \geq 0} \mathbb{N}_0^\ell. \tag{224}$$

In particular, $\mathbb{N}_0^0 = \{\epsilon\}$, where ϵ is the empty string, and $\mathbb{N}_0^1 = \mathbb{N}_0$.

Gödel Numbering of $\mathbb{N}_0^*$ – Definition

Define encoding

$$\sigma : \mathbb{N}_0^* \to \mathbb{N}_0 \tag{225}$$

using Cantor's pairing function $\sigma_2 : \mathbb{N}_0^2 \to \mathbb{N}_0$, i.e., for all $\boldsymbol{x} \in \mathbb{N}_0^*$ and $y \in \mathbb{N}_0$,

$$\begin{aligned} \sigma(\epsilon) &= 0, && (226)\\ \sigma(x) &= \sigma_2(0, x) + 1, && (227)\\ \sigma(\boldsymbol{x}, y) &= \sigma_2(\sigma(\boldsymbol{x}), y) + 1. && (228) \end{aligned}$$

Second equation (227) is a special case of the third one (228), since for each $y \in \mathbb{N}_0$,

$$\sigma(\epsilon, y) = \sigma_2(\sigma(\epsilon), y) + 1 = \sigma_2(0, y) + 1 = \sigma(y). \tag{229}$$

$\sigma(\boldsymbol{x})$ is called the *Gödel number* of the string $\boldsymbol{x} \in \mathbb{N}_0^*$.

Gödel Numbering of $\mathbb{N}_0^*$ – Example

We have

$$\begin{aligned} \sigma(1,3) &= \sigma_2(\sigma(1),3)+1 \\ &= \sigma_2(\sigma_2(0,1)+1,3)+1 \\ &= \sigma_2(2,3)+1 \\ &= 17+1=18. \end{aligned} \tag{230}$$

Gödel Numbering of $\mathbb{N}_0^*$

Encoding function σ is well-defined.

Proof.

In each step of the evaluation of σ, the length of the string decreases by 1.

More concretely, by (228), the evaluation of $\sigma(\boldsymbol{x}, y)$ is replaced by the evalution of $\sigma(\boldsymbol{x})$, where $\boldsymbol{x} \in \mathbb{N}_0^*$ and $y \in \mathbb{N}_0$. □

Gödel Numbering of $\mathbb{N}_0^*$

Encoding function σ is primitive recursive.

Proof.

- For strings of length ≤ 1, by (226)-(227), σ is primitive recursive, since σ_2 is primitive recursive.
- Suppose σ is primitive recursive for strings of length $\leq \ell$, where $\ell \geq 1$.

  For strings of length $\ell + 1$, by (228), σ can be written as composition of primitive recursive functions:[a]

$$\sigma_{|\mathbb{N}_0^{\ell+1}} = \nu \circ \sigma_2(\sigma_{|\mathbb{N}_0^{\ell}}(\pi_1^{(\ell+1)}, \ldots, \pi_\ell^{(\ell+1)}), \pi_{\ell+1}^{(\ell+1)})_{|\mathbb{N}_0^{\ell+1}}. \quad (231)$$

  By induction hypothesis, σ is primitive recursive for strings of length $\leq \ell$ and so the right-hand side is primitive recursive. □

[a] $\sigma(\boldsymbol{x}, y) = \sigma_2(\sigma(\boldsymbol{x}), y) + 1$.

Gödel Numbering of $\mathbb{N}_0^*$

Encoding function σ is bijective.

*Proof

Let A be the set of $n \in \mathbb{N}_0$ such that there is a unique $\boldsymbol{x} \in \mathbb{N}_0^*$ with $\sigma(\boldsymbol{x}) = n$. Claim that $A = \mathbb{N}_0$.

- 0 lies in A since $\sigma(\epsilon) = 0$ and $\sigma(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \neq \epsilon$
- Let $n > 0$. Define

$$u = \kappa_2(n-1) \quad \text{and} \quad v = \tau_2(n-1). \tag{232}$$

Then by (219), $\sigma_2(u, v) = \sigma_2(\kappa_2(n-1), \tau_2(n-1)) = n - 1$.

By construction, $\kappa_2(z) \leq z$ and $\tau_2(z) \leq z$ for all $z \in \mathbb{N}_0$.

Thus $u = \kappa_2(n-1) < n$ and hence $u \in A$, i.e., there is exactly one string $\boldsymbol{x} \in \mathbb{N}_0^k$ such that $\sigma(\boldsymbol{x}) = u$.

Then $\sigma(\boldsymbol{x}, v) = \sigma_2(\sigma(\boldsymbol{x}), v)) + 1 = \sigma_2(u, v) + 1 = n$.

Proof (Cont'd)

Suppose for some $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{N}_0^*$ and $v, w \in \mathbb{N}_0$,

$$\sigma(\boldsymbol{x}, v) = n = \sigma(\boldsymbol{y}, w).$$

Then by (228),

$$\sigma_2(\sigma(\boldsymbol{x}), v) = \sigma_2(\sigma(\boldsymbol{y}), w).$$

But σ_2 is bijective and so $\sigma(\boldsymbol{x}) = \sigma(\boldsymbol{y})$ and $v = w$.
Since $\sigma(\boldsymbol{x}) < n$, by induction $\boldsymbol{x} = \boldsymbol{y}$.
Thus $n \in A$ and hence by the induction axiom, $A = \mathbb{N}_0$. □

Gödel Numbering of $\mathbb{N}_0^*$ – Functions κ, τ

Define the functions $\kappa, \tau : \mathbb{N}_0 \to \mathbb{N}_0$ such that for all $n \in \mathbb{N}_0$,

$$\kappa(n) = \kappa_2(n \dot{-} 1) \quad \text{and} \quad \tau(n) = \tau_2(n \dot{-} 1), \tag{233}$$

where κ_2, τ_2 form the inverse functions of σ_2, i.e.,

$$\sigma_2(\kappa_2(n), \tau_2(n)) = n, \quad n \in \mathbb{N}_0. \tag{234}$$

Marginal conditions:

$$\kappa(1) = \kappa(0) = 0 \quad \text{and} \quad \tau(1) = \tau(0) = 0. \tag{235}$$

Gödel Numbering of $\mathbb{N}_0^*$ – Functions κ, τ

Functions κ, τ are primitive recursive, and for each $n \geq 1$ there are unique $\boldsymbol{x} \in \mathbb{N}_0^*$ and $y \in \mathbb{N}_0$ with $\sigma(\boldsymbol{x}, y) = n$ such that

$$\kappa(n) = \sigma(\boldsymbol{x}) \quad \text{and} \quad \tau(n) = y. \tag{236}$$

Proof

- κ and τ are compositions of primitive recursive functions and thus also primitive recursive.
- Let $n \geq 1$. Since σ is bijective, there are unique $\boldsymbol{x} \in \mathbb{N}_0^*$ and $y \in \mathbb{N}_0$ such that $\sigma(\boldsymbol{x}, y) = n$.

  Thus by (228), $\sigma_2(\sigma(\boldsymbol{x}), y) = n - 1$.

  But by (219), $\sigma_2(\kappa_2(n-1), \tau_2(n-1)) = n - 1$.

  Since σ_2 is bijective, $\kappa_2(n-1) = \sigma(\boldsymbol{x})$ and $\tau_2(n-1) = y$.

  Hence $\kappa(n) = \sigma(\boldsymbol{x})$ and $\tau(n) = y$. □

Gödel Numbering of $\mathbb{N}_0^*$ – String Length

A string $\boldsymbol{x} \in \mathbb{N}_0^*$ has length $\ell \geq 0$ if ℓ is the smallest number such that

$$\kappa^{\ell}(\sigma(\boldsymbol{x})) = 0. \tag{237}$$

Proof

- For the empty string, $\kappa^0(\sigma(\epsilon)) = \sigma(\epsilon) = 0$, since $\kappa^0 = \mathrm{id}_{\mathbb{N}_0}$.
- Let $\boldsymbol{x} = x_1 \ldots x_\ell \in \mathbb{N}_0^\ell$ with $\ell \geq 1$.

  We have $\sigma(\boldsymbol{x}) = n$ for some $n \geq 1$.

  By (236), $\sigma(x_1 \ldots x_{\ell-1}) = \kappa(n)$, and by induction, $\kappa^{\ell-1}(\sigma(x_1 \ldots x_{\ell-1})) = 0$, where $\ell - 1$ is minimal with this property. Thus

$$\begin{aligned} \kappa^{\ell}(\sigma(\boldsymbol{x})) &= \kappa^{\ell}(n) = \kappa^{\ell-1}(\kappa(n)) \\ &= \kappa^{\ell-1}(\sigma(x_1 \ldots x_{\ell-1})) = 0 \end{aligned}$$

  and ℓ is minimal with this property. □

Gödel Numbering of $\mathbb{N}_0^*$ – Example

Take a number $x \in \mathbb{N}_0$, i.e., a string of length 1.

Then by definition $\sigma(x) = n \geq 1$. By (229),

$$\sigma(x) = \sigma(\epsilon, x)$$

and therefore by (236),

$$\kappa^1(n) = \sigma(\epsilon) = 0 \quad \text{and} \quad \tau(n) = x.$$

Gödel Numbering of $\mathbb{N}_0^*$ – Length

The *length function* $\lg : \mathbb{N}_0 \to \mathbb{N}_0$ given by

$$\lg(n) = \mu\ell \left(\kappa^\ell(n) = 0\right) \tag{238}$$

is primitive recursive.

Proof.

Since κ is primitive recursive, κ^ℓ is primitive recursive. Minimalization is restricted as we have $\ell \leq n$; i.e., the string is not longer than the number it encodes.

In view of minimalization, the correct definition would be $f : \mathbb{N}_0^2 \to \mathbb{N}_0 : (n, \ell) \mapsto \kappa^\ell(n)$ and then $\lg(n) = \mu\ell \left[f(n, \ell) = 0\right]$. □

By (237), $\lg$ assigns to each Gödel number n the length ℓ of the string $\boldsymbol{x}$ with $\sigma(\boldsymbol{x}) = n$.

Gödel Numbering of $\mathbb{N}_0^*$ – Inverse

The inverse value of $n \geq 1$ is given by

$$\sigma^{-1}(n) = (\tau \circ \kappa^{\ell-1}(n), \ldots, \tau \circ \kappa(n), \tau(n)) \in \mathbb{N}_0^\ell, \tag{239}$$

where $\lg(n) = \ell$.

Proof

Let $x \in \mathbb{N}_0$.

- Then $\sigma(x) = n \geq 1$ and by (229), $\sigma(x) = \sigma(0, x)$.
- Thus by (236),

$$\kappa(n) = 0 \quad \text{and} \quad \tau(n) = x.$$

- Hence, $\sigma^{-1}(n) = x = \tau(n)$.

Proof (Cont'd)

Let $\boldsymbol{x} \in \mathbb{N}_0^\ell$, $\ell \geq 1$, and $y \in \mathbb{N}_0$.

- Then $\sigma(\boldsymbol{x}, y) = n \geq 1$.
- By (236),

$$\kappa(n) = \sigma(\boldsymbol{x}) \quad \text{and} \quad \tau(n) = y.$$

Since $\kappa(n) < n$, by induction $\boldsymbol{x} = \sigma^{-1}(\kappa(n))$ is given by

$$\begin{aligned} &((\tau \circ \kappa^{\ell-1})(\kappa(n)), \ldots, \tau(\kappa(n))) \\ = \quad &(\tau \circ \kappa^{\ell}(n), \ldots, \tau \circ \kappa(n)). \end{aligned}$$

- Hence, $(\boldsymbol{x}, y)$ has the form

$$(\tau \circ \kappa^{\ell}(n), \ldots, \tau \circ \kappa(n), \tau(n))$$

as required. □

Gödel Numbering of $\mathbb{N}_0^*$ – Example

Let $n = 18$.

- Computation of length:

$$\kappa^2(18) = \kappa(\kappa(18)) = \kappa(\kappa_2(17)) = \kappa(2) = \kappa_2(1) = 0,$$

since by (223), $\sigma_2(2,3) = 17$ and so $\kappa_2(17) = 2$, and $\kappa(2) = \kappa_2(1) = 0$, since $\sigma_2(0,1) = 1$ and so $\kappa_2(1) = 0$.

Thus by (238), $\lg(18) = 2$.

- Computation of string:

$$\tau(18) = \tau_2(17) = 3,$$

since $\sigma_2(2,3) = 17$ and so $\tau_2(17) = 3$, and

$$\tau(\kappa(18)) = \tau(\kappa_2(17)) = \tau(2) = \tau_2(1) = 1,$$

since $\sigma_2(0,1) = 1$ and so $\tau_2(1) = 1$.

Hence, $\sigma(1,3) = 18$.

Gödel Numbering of GOTO Instructions – Definition

Let $P = s_0; s_1; \ldots; s_m$ be a standard GOTO (SGOTO) program.

The l-th instruction s_l has label l, $0 \leq l \leq q$, and is encoded by

$$\sigma'(s_l) = \begin{cases} 3 \cdot \sigma(i,k) & \text{if } s_l = (l, x_i \leftarrow x_i + 1, k), \\ 3 \cdot \sigma(i,k) + 1 & \text{if } s_l = (l, x_i \leftarrow x_i - 1, k), \\ 3 \cdot \sigma(i,k,m) + 2 & \text{if } s_l = (l, \texttt{if } x_i = 0, k, m). \end{cases} \quad (240)$$

$\sigma'(s_l)$ is called the *Gödel number* of instruction s_l, $0 \leq l \leq m$.

Function σ' is primitive recursive, since it is defined by primitive case distinction and the functions involved are primitive recursive.

Gödel Numbering of GOTO Instructions – Inverse

Given the Gödel number $e = \sigma'(s_l) \in \mathbb{N}_0$ of l-th instruction s_l.

- By division with remainder, write

$$e = 3n + t, \tag{241}$$

  where $n = \div(e, 3)$ and $t = \mod(e, 3)$.

- By (239), $\sigma^{-1}(n) = (\tau(\kappa(n)), \tau(n))$ for increment and decrement, and $\sigma^{-1}(n) = (\tau(\kappa^2(n)), \tau(\kappa(n)), \tau(n))$ for branching. Then by (240),

$$s_l = \begin{cases} (l, x_{\tau(\kappa(n))} \leftarrow x_{\tau(\kappa(n))} + 1, \tau(n)) & \text{if } t = 0, \\ (l, x_{\tau(\kappa(n))} \leftarrow x_{\tau(\kappa(n))} - 1, \tau(n)) & \text{if } t = 1, \\ (l, \texttt{if } x_{\tau(\kappa^2(n))} = 0, \tau(\kappa(n)), \tau(n)) & \text{if } t = 2. \end{cases} \tag{242}$$

Gödel Numbering of GOTO Instructions – Example

Suppose $e = 55$ is the Gödel number of l-th instruction s_l.

- Write
$$55 = 3 \cdot 18 + 1$$
and so $n = 18$ and $t = 1$ (decrement).
- By (230),
$$\sigma(1, 3) = 18$$
and so
$$s_l = (l, x_1 \leftarrow x_1 - 1, 3).$$

Gödel Numbering of GOTO Programs – Definition

SGOTO program $P = s_0; s_1; \ldots; s_m$ has the *Gödel number*

$$\rho(P) = \sigma(\sigma'(s_0), \sigma'(s_1), \ldots, \sigma'(s_m)). \tag{243}$$

$\rho : \mathcal{P}_{\mathrm{SGOTO}} \to \mathbb{N}_0$ is bijective and primitive recursive.

Proof.

- Function ρ is bijective since σ is bijective and the instructions encoded by σ' are uniquely determined.
- Function ρ is a composition of primitive recursive functions and thus also primitive recursive.

Gödel Numbering of GOTO Programs – Example

Consider the SGOTO program P_+ given by

$$\begin{array}{llll} s_0: & 0 & \texttt{if } x_2 = 0 & 3 \quad 1 \\ s_1: & 1 & x_1 \leftarrow x_1 + 1 & 2 \\ s_2: & 2 & x_2 \leftarrow x_2 - 1 & 0 \end{array}$$

Encoding of instructions:

$$\begin{aligned} \sigma'(s_0) &= 3 \cdot \sigma(2,3,1) + 2 = 1889, \\ \sigma'(s_1) &= 3 \cdot \sigma(1,2) = 39, \\ \sigma'(s_2) &= 3 \cdot \sigma(2,0) + 1 = 46. \end{aligned}$$

Encoding of program:

$$\begin{aligned} \rho(P_+) &= \sigma(1889, 39, 46) \\ &= 1,269,420,373,033,475,160,642,134. \end{aligned}$$

Gödel Numbering of GOTO Programs – List

- P_e denotes the SGOTO program P with Gödel number e.
- Gödel numbering provides a list of all SGOTO programs

$$P_0, P_1, P_2, \ldots. \tag{244}$$

- Conversely, each $e \in \mathbb{N}_0$ can be uniquely assigned an SGOTO program P such that $\rho(P) = e$ (by the above procedure).

Gödel Numbering of $\mathcal{R}$ – List

Let $n \geq 0$,

- The n-adic partial recursive function computing the SGOTO program P_e is

$$\phi_e^{(n)} = \|P_e\|_{n,1}. \tag{245}$$

  Then e is the *Gödel number* or *index* of $\phi_e^{(n)}$.

- The list of all SGOTO programs in (244) yields a list of all n-adic partial recursive functions:

$$\phi_0^{(n)}, \phi_1^{(n)}, \phi_2^{(n)}, \ldots. \tag{246}$$

  This list contains many repetitions, since each n-adic partial recursive function has infinitely many indices; e.g., add zero to the result.

Parametrization

- Parametrization theorem (smn theorem) is a cornerstone of computability theory (Kleene, 1943)
- Refers to computable functions in which some arguments are considered as parameters.
- Applications in reduction steps and recursion theory.

Parametrization – Dyadic Case

For each dyadic partial recursive function $f : \mathbb{N}_0^2 \to \mathbb{N}_0$, there is a monadic primitive recursive function g such that for all $x \in \mathbb{N}_0$,

$$f(x, \cdot) = \phi^{(1)}_{g(x)}, \tag{247}$$

i.e., for all $x, y \in \mathbb{N}_0$,

$$f(x, y) = \phi^{(1)}_{g(x)}(y).$$

Proof

- Given SGOTO program $P_e = s_0; s_1; \ldots; s_m$ with

$$\|P_e\|_{2,1} = f.$$

- For each $x \in \mathbb{N}_0$, consider the following SGOTO program Q_x:

0	if $x_1 = 0$	3	1
1	$x_2 \leftarrow x_2 + 1$	2	
2	$x_1 \leftarrow x_1 - 1$	0	
3	$x_1 \leftarrow x_1 + 1$	4	
4	$x_1 \leftarrow x_1 + 1$	5	
	$\vdots$		
$2 + x$	$x_1 \leftarrow x_1 + 1$	$3 + x$	
	s'_0		
	s'_1		
	$\vdots$		
	s'_m		

where $P'_e = s'_0; s'_1; \ldots; s'_m$ is the SGOTO program that is derived from P_e by replacing each occurring label j with $j + 3 + x$, $0 \leq j \leq q$.

Proof (Cont'd)

- Milestones of the computation of Q_x:

$$\begin{array}{cccccccl} 0 & y & 0 & 0 & 0 & \dots & \text{init} \\ 0 & 0 & y & 0 & 0 & \dots & \text{step } 3 \\ 0 & x & y & 0 & 0 & \dots & \text{step } 3+x \\ 0 & f(x,y) & . & . & . & \dots & \text{end} \end{array}$$

- Thus

$$\|Q_x\|_{1,1}(y) = f(x,y), \quad x, y \in \mathbb{N}_0.$$

- Take the function $g : \mathbb{N}_0 \to \mathbb{N}_0$ defined by

$$g(x) = \rho(Q_x), \quad x \in \mathbb{N}_0,$$

  i.e., $g(x)$ is the Gödel number of the program Q_x. By (243), this function is primitive recursive.

- Hence, $\phi^{(1)}_{g(x)} = \|Q_x\|_{1,1} = f(x,\cdot)$ for each $x \in \mathbb{N}_0$. □

Parametrization – Example

Consider the dyadic addition

$$f_+ : \mathbb{N}_0^2 \to \mathbb{N}_0 : (x, y) \mapsto x + y$$

and the corresponding SGOTO program P_+

s_0 :	0	if $x_2 = 0$	3	1
s_1 :	1	$x_1 \leftarrow x_1 + 1$	2	
s_2 :	2	$x_2 \leftarrow x_2 - 1$	0	

with

$$\|P_+\|_{2,1} = f.$$

Fixing the first argument to $x = 3$ yields the monadic function

$$f_+(3, \cdot) : \mathbb{N}_0 \to \mathbb{N}_0 : y \mapsto 3 + y.$$

Parametrization – Example (Cont'd)

SGOTO program Q_3 computing $f_+(3,\cdot)$:

0	`if` $x_1 = 0$	3	1
1	$x_2 \leftarrow x_2 + 1$	2	
2	$x_1 \leftarrow x_1 - 1$	0	
3	$x_1 \leftarrow x_1 + 1$	4	
4	$x_1 \leftarrow x_1 + 1$	5	
5	$x_1 \leftarrow x_1 + 1$	6	
6	`if` $x_2 = 0$	9	7
7	$x_1 \leftarrow x_1 + 1$	8	
8	$x_2 \leftarrow x_2 - 1$	6	

The last three statements comprise P_+.

Parametrization – Example (Cont'd)

Milestones of the computation of Q_3:

0	y	0	0	0	$\ldots$	init
0	0	y	0	0	$\ldots$	step 3
0	3	y	0	0	$\ldots$	step 6
0	$3+y$	$.$	$.$	$.$	$\ldots$	end

Let $g(3) = \rho(Q_3)$ denote the Gödel number of Q_3. Then

$$\phi^{(1)}_{g(3)} = \|Q_3\|_{1,1} = f_+(3, \cdot).$$

Parametrization – Example

Consider the dyadic addition

$$f_+ : \mathbb{N}_0^2 \to \mathbb{N}_0 : (x, y) \mapsto x + y$$

and the corresponding URM program

$$P_+ = (A1; S2)2,$$

i.e., $\|P_+\|_{2,1} = f$.
Fixing the first argument to $x = 3$ yields the monadic function

$$f_+(3, \cdot) : \mathbb{N}_0 \to \mathbb{N}_0 : y \mapsto 3 + y.$$

Consider the URM program

$$P_{+3} = (S1; A2)2; A1; A1; A1; P_+.$$

Parametrization – Example (Cont'd)

Consider the URM program

$$P_{+3} = (S1; A2)2; A1; A1; A1; P_{+}.$$

Milestones of the computation of P_{+3}:

0	y	0	0	0	$\ldots$	init
0	0	y	0	0	$\ldots$	reload y
0	3	y	0	0	$\ldots$	generate $x = 3$
0	$3 + y$	$.$	$.$	$.$	$\ldots$	addition

Gödel number of P_{+3} is obtained by converting into SGOTO program (by tokenization) and then using (243).

Parametrization – General Case

For each pair $m, n \geq 1$, there is an $m+1$-adic primitive recursive function $s_{m,n}$ such that for all $e \in \mathbb{N}_0$ and $\boldsymbol{x} \in \mathbb{N}_0^m$,

$$\phi_e^{(m+n)}(\boldsymbol{x}, \cdot) = \phi^{(n)}_{s_{m,n}(e,\boldsymbol{x})}, \tag{248}$$

i.e., for all $e \in \mathbb{N}_0$, $\boldsymbol{x} \in \mathbb{N}_0^m$ and $\boldsymbol{y} \in \mathbb{N}_0^n$,

$$\phi_e^{(m+n)}(\boldsymbol{x}, \boldsymbol{y}) = \phi^{(n)}_{s_{m,n}(e,\boldsymbol{x})}(\boldsymbol{y}).$$

Proof (Similar to dyadic case)

- Take an SGOTO program $P_e = s_0; s_1; \ldots; s_m$ calculating the function $\phi_e^{(m+n)}$, i.e.,

$$\|P_e\|_{m+n,1} = \phi_e^{(m+n)}.$$

- For each $\boldsymbol{x} \in \mathbb{N}_0^m$, extend P_e to SGOTO program $Q_{e,\boldsymbol{x}}$:

0	$\boldsymbol{y}$	0	0	0	$\ldots$	init
0	$\boldsymbol{0}$	$\boldsymbol{y}$	0	0	$\ldots$	reload $\boldsymbol{y}$
0	$\boldsymbol{x}$	$\boldsymbol{y}$	0	0	$\ldots$	generate parameter $\boldsymbol{x}$
0	$\phi_e^{(m+n)}(\boldsymbol{x},\boldsymbol{y})$	$.$	$.$	$.$	$\ldots$	end

- Then

$$\|Q_{e,\boldsymbol{x}}\|_{n,1}(\boldsymbol{y}) = \phi_e^{(m+n)}(\boldsymbol{x},\boldsymbol{y}), \quad \boldsymbol{x} \in \mathbb{N}_0^m, \boldsymbol{y} \in \mathbb{N}_0^n.$$

Proof (Cont'd)

- We have

$$\|Q_{e,\boldsymbol{x}}\|_{n,1}(\boldsymbol{y}) = \phi_e^{(m+n)}(\boldsymbol{x},\boldsymbol{y}), \quad \boldsymbol{x} \in \mathbb{N}_0^m, \boldsymbol{y} \in \mathbb{N}_0^n.$$

- Consider the function $s_{m,n} : \mathbb{N}_0^{m+1} \to \mathbb{N}_0$ defined by

$$s_{m,n}(e,\boldsymbol{x}) = \rho(Q_{e,\boldsymbol{x}}),$$

  i.e., $s_{m,n}(e,\boldsymbol{x})$ is the Gödel number of program $Q_{e,\boldsymbol{x}}$. By (243), this function is primitive recursive.

- Hence, $\phi^{(n)}_{s_{m,n}(e,\boldsymbol{x})} = \|Q_{e,\boldsymbol{x}}\|_{n,1}$. □

Universal Function

Another basic result of computability theory is the existence of a computable function called universal function that is capable of computing any other computable function (of the same arity).

Let $n \geq 1$. A *universal function* for n-adic partial recursive functions is an $n+1$-adic function $\psi_{\text{univ}}^{(n)} : \mathbb{N}_0^{n+1} \to \mathbb{N}_0$ such that for all $e \in \mathbb{N}_0$,

$$\psi_{\text{univ}}^{(n)}(e, \cdot) = \phi_e^{(n)}, \tag{249}$$

i.e., for all $e \in \mathbb{N}_0$ and $\boldsymbol{x} \in \mathbb{N}_0^n$,

$$\psi_{\text{univ}}^{(n)}(e, \boldsymbol{x}) = \phi_e^{(n)}(\boldsymbol{x}). \tag{250}$$

Universal function $\psi_{\text{univ}}^{(n)}$ takes as input Gödel number e and argument $\boldsymbol{x}$ in order to compute $\phi_e^{(n)}(\boldsymbol{x})$.

Main Result

For each arity $n \geq 1$, the universal function $\psi^{(n)}_{\text{univ}}$ exists and is partial recursive.

Let e be a Gödel number.

- Find the SGOTO program P_e by using the inverse of the bijection (243).
- Idea is to mimick the computation of P_e by universal function (similar to mimicking of P_e by partial recursive function in Part IV).
- Residual-step function R_{P_e} as in (135) mimicks the execution of P_e.
- Then by (136), $\psi^{(n)}_{\text{univ}}(e, \cdot) = \beta_1 \circ R_{P_e} \circ \alpha_n = \|P_e\|_{n,1}$ exists and is partial recursive.

Universal Function – Example

Consider the addition function $f_+ : \mathbb{N}_0^2 \to \mathbb{N}_0 : (x, y) \mapsto x + y$ and the corresponding SGOTO program P_+:

$$\begin{array}{lllll} s_0: & 0 & \texttt{if } x_2 = 0 & 3 & 1 \\ s_1: & 1 & x_1 \leftarrow x_1 + 1 & 2 & \\ s_2: & 2 & x_2 \leftarrow x_2 - 1 & 0 & \end{array}$$

- Associated Gödel number

$$\begin{aligned} e &= \rho(P_+) = \sigma(\sigma'(s_0), \sigma'(s_1), \sigma'(s_2)) \\ &= 1,269,420,373,033,475,160,642,134. \end{aligned}$$

Universal Function – Example (Cont'd)

- Universal function $\phi^{(2)}_{\text{univ}}(e, \cdot)$ mimicks the computation of $P_e = P_+$ by multi-step function (132).
- Runtime function (133) minimalizes the multi-step function (i.e., number of steps) until a label ℓ is found which is not a program label (here $\ell = 3$).
- Residual-step function R_{P_e} (135) then takes the number of steps as given by the runtime function and $(\beta_1 \circ R_{P_e} \circ \alpha_n)(\boldsymbol{x})$ outputs in register R_1 the result of the program execution.

Normal Form

- Kleene set T (1943) tells whether an SGOTO program will halt when run with an input.
- Kleene's normal form implies that any partial recursive function can be defined by using a single instance of (unbounded) minimalization applied to a primitive recursive function.
- In the context of programming, any program can be written using a single `while` loop.

Extended Kleene Set

Let $n \geq 1$. *Extended Kleene set* $S_n \subseteq \mathbb{N}_0^{n+3}$ is given as

$$(e, \boldsymbol{x}, z, t) \in S_n \quad :\Longleftrightarrow \tag{251}$$
$$\chi_{L(P_e)}(\pi_1^{(k)}(M(e, \omega_{\boldsymbol{x}}, t))) = 0 \quad \wedge \quad \pi_2^{(k)}(M(e, \omega_{\boldsymbol{x}}, t)) = z,$$

where

- $\omega_{\boldsymbol{x}} = (0, x_1, \ldots, x_n, 0, \ldots) \in \mathbb{N}_0^k$ initial configuration given by starting label $\ell = 0$ and input $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{N}_0^n$.
- $M(e, \omega_{\boldsymbol{x}}, t)$ is the multi-step function executing P_e with initial configuration $\omega_{\boldsymbol{x}}$ for t steps.
- $\pi_1^{(k)}(\ell, y_1, y_2, \ldots) = \ell$ is the label of the current configuration.
- $\pi_2^{(k)}(\ell, y_1, y_2, \ldots) = y_1$ is the value of (result) register R_1.
- $\chi_{L(P_e)}(\ell) = 1$ if $\ell \in L(P_e)$ is a label of P_e and 0 otherwise.

Extended Kleene Set

$(e, \boldsymbol{x}, z, t) \in S_n$ iff program P_e with input $\boldsymbol{x} \in \mathbb{N}_0^n$ terminates after t steps and the result of computation is z.

For each $n \geq 1$, the set S_n is primitive.

Proof.

All functions used to define the set S_n are primitive recursive. In particular, the characteristic function of $L(P_e)$ is primitive recursive since the label set $L(P_e)$ is finite (next chapter). □

Extended Kleene Set – Example

Consider the addition function $f_+ : \mathbb{N}_0^2 \to \mathbb{N}_0 : (x, y) \mapsto x + y$ and the corresponding SGOTO program P_+:

$$\begin{array}{llll} s_0: & 0 & \texttt{if } x_2 = 0 & 3 \quad 1 \\ s_1: & 1 & x_1 \leftarrow x_1 + 1 & 2 \\ s_2: & 2 & x_2 \leftarrow x_2 - 1 & 0 \end{array}$$

- Associated Gödel number

$$\begin{aligned} e &= \rho(P_+) = \sigma(\sigma'(s_0), \sigma'(s_1), \sigma'(s_2)) \\ &= 1,269,420,373,033,475,160,642,134. \end{aligned}$$

- Initial configuration $(k = 3)$

$$\omega = (0, x, y).$$

Extended Kleene Set – Example (Cont'd)

Extended Kleene set S_2 consists of all quadruples

$$(e, (x, y), x + y, 3y + 1), \quad x, y \in \mathbb{N}_0,$$

where

- e is the Gödel number,
- (x, y) is the input,
- $z = x + y$ is the result of computation, and
- $t = 3y + 1$ is the number of computational steps.

Kleene Set

Let $n \geq 1$. *Kleene set* $T_n \subseteq \mathbb{N}_0^{n+2}$ is given by

$$(e, \boldsymbol{x}, y) \in T_n \quad :\Longleftrightarrow \quad (e, \boldsymbol{x}, \kappa_2(y), \tau_2(y)) \in S_n. \tag{252}$$

- Cantor pairing function $\sigma_2 : \mathbb{N}_0^2 \to \mathbb{N}_0$ has inverse function pair $\kappa_2, \tau_2 : \mathbb{N}_0 \to \mathbb{N}_0$, where

$$\sigma_2^{-1}(y) = (\kappa_2(y), \tau_2(y)), \quad y \in \mathbb{N}_0, \tag{253}$$

  or equivalently,

$$\sigma_2(\kappa_2(y), \tau_2(y)) = y, \quad y \in \mathbb{N}_0, \tag{254}$$

- Component y encodes both, the result of computation $z = \kappa_2(y)$ and the number of steps $t = \tau_2(y)$.

Kleene Set

For each $n \geq 1$, the set T_n is primitive.

Proof.

We have

$$\chi_{T_n}(e, \boldsymbol{x}, y) = \chi_{S_n}(e, \boldsymbol{x}, \kappa_2(y), \tau_2(y)),$$

where the right-hand side is a composition of primitive recursive functions as χ_{S_n} is primitive recursive (since S_n is primitive) and κ_2, τ_2 are primitive recursive.
Hence, χ_{T_n} is primitive recursive, i.e., T_n is primitive. □

Kleene Set – Example

Consider the addition function $f_+ : \mathbb{N}_0^2 \to \mathbb{N}_0 : (x, y) \mapsto x + y$ and the corresponding SGOTO program P_+:

$$\begin{array}{llll} s_0: & 0 & \texttt{if } x_2 = 0 & 3 \quad 1 \\ s_1: & 1 & x_1 \leftarrow x_1 + 1 & 2 \\ s_2: & 2 & x_2 \leftarrow x_2 - 1 & 0 \end{array}$$

- Associated Gödel number

$$\begin{aligned} e &= \rho(P_+) = \sigma(\sigma'(s_0), \sigma'(s_1), \sigma'(s_2)) \\ &= 1,269,420,373,033,475,160,642,134. \end{aligned}$$

- Initial configuration $(k = 3)$

$$\omega = (0, x, y).$$

Kleene Set – Example (Cont'd)

Kleene set T_2 consists of all triples

$$(e, (x, y), w), \quad x, y \in \mathbb{N}_0,$$

where

- e is the Gödel number,
- (x, y) is the input,
- $\sigma_2^{-1}(w) = (z, t)$,
- $z = x + y$ is the result of computation, and
- $t = 3y + 1$ is the number of computational steps.

Normal Form Theorem

Let $n \geq 1$.
For all $e \in \mathbb{N}_0$ and $\boldsymbol{x} \in \mathbb{N}_0^n$,

$$\phi_e^{(n)}(\boldsymbol{x}) \quad = \quad \kappa_2(\mu y[\chi_{T_n}(e, \boldsymbol{x}, y) = 1]). \tag{255}$$

- $\chi_{T_n}(e, \boldsymbol{x}, y) = 1$ iff $(e, \boldsymbol{x}, y) \in T_n$, i.e., program P_e with input $\boldsymbol{x}$ yields output $\kappa_2(y)$ in $\tau_2(y)$ steps.
- Unbounded minimalization $\mu y[\chi_{T_n}(e, \boldsymbol{x}, y) = 1]$ finds the smallest $y \geq 0$ (if any) such that $(e, \boldsymbol{x}, y) \in T_n$.
- If so, $\kappa_2(y) = z$ yields the result of computation.

Consequence

By (255), any partial recursive function can be defined by using a single instance of (unbounded) minimalization applied to a primitive recursive function (since χ_{T_n} is primitive recursive).

Skills

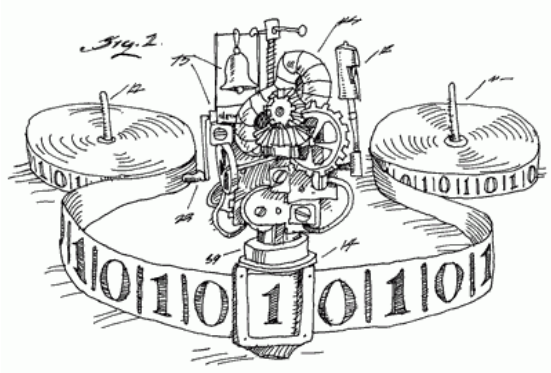
- Computation of functions σ_2, κ_2 and τ_2
- Computation of functions σ, κ and τ
- Computation of Gödel number of GOTO program
- Specification of GOTO program $Q_{e,x}$ (parametrization)
- Finding the Kleene sets S_n and T_n

Part VII

Turing Machine

Turing Machine – Motivation

- Turing machine as first univeral computation machine.
- Understanding the behavior of a Turing machine.
- Busy beaver problem and its relation to superpowers.



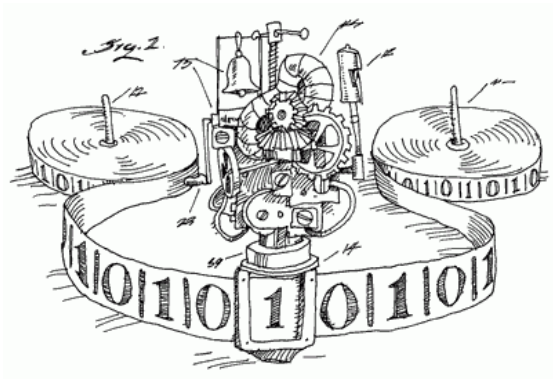
Turing Machine – Contents

- Machinery
- *Post-Turing machine
- *Turing computable functions
- Busy beaver

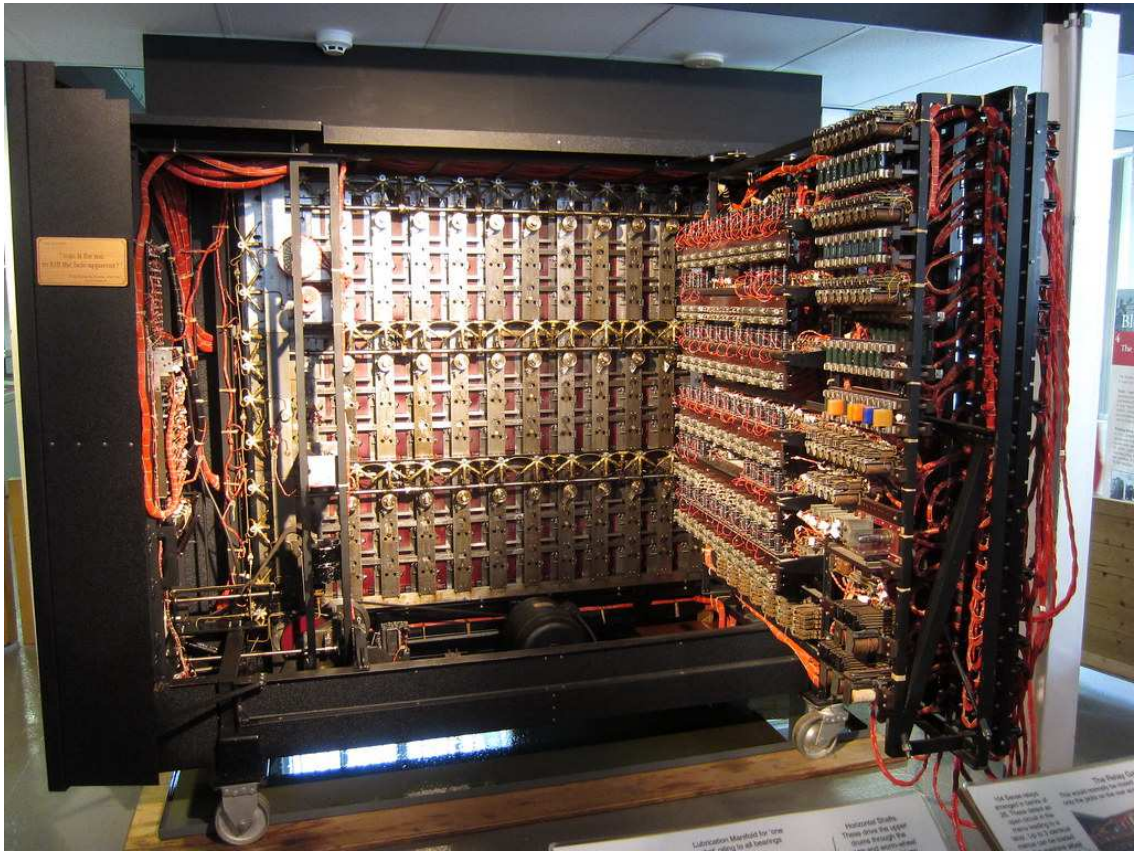
Reading: Zimmermann, Chapter 6

Turing Machine – Inside

- Turing machine has been described as a thought experiment representing a computing machine (Alan Turing, 1912-1954).
- Turing machine is a theoretical device that manipulates symbols on a strip of tape according to a set of rules.
- Turing machine is a device of universal computation.
- Busy beaver problem is a prominent example of recreational mathematics.

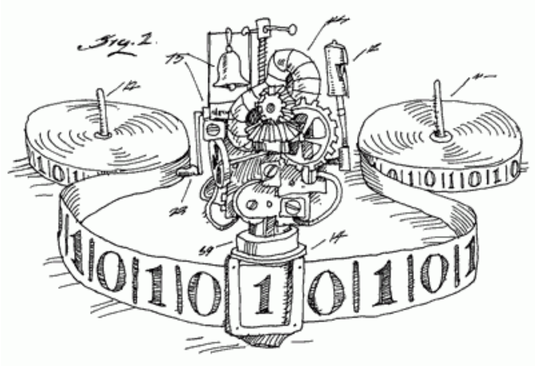


Turing machine at Bletchley Park (WW II)



"The Imitation Game" is a 2014 American historical drama based on the biography of Alan Turing.

Machinery



Machinery – Overall Behaviour

- Turing machine consists of infinite tape and associated read/write (R/W) head connected to a control mechanism.
- Tape is divided into denumerably many cells, each of which containing a symbol from the tape alphabet.
- Tape alphabet contains special symbol "b" signifying that a cell is blank or empty.
- Cells are scanned, one at a time, by R/W head which is able to move in both directions or remain fixed.
- At any time instant, the machine will be in one of a finite number of states.
- Behaviour of R/W head and change of machine's state are governed by present state and symbol in cell under scan.

Machinery – Formal Definition

Turing machine is a quintuple $M = (\Sigma, Q, \delta, q_0, q_F)$,

- finite alphabet Σ, called *tape alphabet*, contains distinguished *blank* symbol b, subset $\Sigma_I = \Sigma \setminus \{b\}$ is called *input alphabet*,
- finite set Q of *states*,
- partial function $\delta : Q \times \Sigma \to Q \times \Sigma \times \{L, R, \Lambda\}$ called *transition function*,
- *start state* $q_0 \in Q$,
- *halt state* $q_F \in Q$ such that $\delta(q_F, \sigma)$ is undefined for all σ in Σ.

Machinery – Addendum

- Symbols L, R, and Λ are interpreted as *left move*, *right move*, and *no move*, respectively.
- Tape cells can be numbered by the set of integers $\mathbb{Z}$.
- Tape content can be considered as mapping $\tau : \mathbb{Z} \to \Sigma$, where $\tau(z)$ is the content of cell $z \in \mathbb{Z}$.
- Tape contains blanks almost everywhere, i.e., only a finite portion of the tape contains symbols from the input alphabet.

$\ldots$	-3	-2	-1	0	1	2	3	$\ldots$
	$\tau(-3)$	$\tau(-2)$	$\tau(-1)$	$\tau(0)$	$\tau(1)$	$\tau(2)$	$\tau(3)$	
	b	1	1	b	1	1	b	

Machinery – Configurations

- *Configuration* of Turing machine M consists of
  - content of the tape,
  - cell controlled by R/W head,
  - state of the machine.
- Picture

$$\vdash \quad a_{i_1} \quad a_{i_2} \quad \ldots \quad \underset{\uparrow\, q}{a_{i_j}} \quad \ldots \quad a_{i_l} \quad \vdash$$

  $q \in Q$ (current state), arrow marking cell controlled by R/W head. All cells to the left of a_{i_1} and to the right of a_{i_l} contain the blank symbol.
- Shorthand notation of configuration

$$(a_{i_1} \ldots a_{i_{j-1}}, q, a_{i_j} \ldots a_{i_l}).$$

Machinery – Transitions

Transition $\delta(q, a) = (q', a', D)$ means

- machine is in state $q \in Q$ and reads the symbol $a \in \Sigma$ (in cell under R/W head),
- replaces $a \in \Sigma$ by $a' \in \Sigma$,
- moves left ($D = L$), moves right ($D = R$), or doesn't move ($D = \Lambda$),
- enters the state $q' \in Q$.

The R/W head is shifted and not the tape of cells.

Machinery – Move Right

Configuration (uc, q, adv) with $a, c, d \in \Sigma$, $u, v \in \Sigma^*$, $q \in Q$, $q \neq q_F$:

$$\begin{array}{cccccccc} — & u & c & a & d & v & — \\ & & & \uparrow & & & \\ & & & q & & & \end{array}$$

Move right: $\delta(q, a) = (q', a', R)$

$$\begin{array}{cccccccc} — & u & c & a' & d & v & — \\ & & & & \uparrow & & \\ & & & & q' & & \end{array}$$

New configuration (uca', q', dv).

Machinery – Move Left

Configuration (uc, q, adv) with $a, c, d \in \Sigma$, $u, v \in \Sigma^*$, $q \in Q$, $q \neq q_F$:

$$\begin{array}{cccccccc} — & u & c & a & d & v & — \\ & & & \uparrow & & & \\ & & & q & & & \end{array}$$

Move left: $\delta(q, a) = (q', a', L)$

$$\begin{array}{cccccccc} — & u & c & a' & d & v & — \\ & & \uparrow & & & & \\ & & q' & & & & \end{array}$$

New configuration $(u, q', ca'dv)$.

Machinery – No Move

Configuration (uc, q, adv) with $a, c, d \in \Sigma$, $u, v \in \Sigma^*$, $q \in Q$, $q \neq q_F$:

$$\begin{array}{ccccccc} — & u & c & a & d & v & — \\ & & & \uparrow & & & \\ & & & q & & & \end{array}$$

No move: $\delta(q, a) = (q', a', \Lambda)$

$$\begin{array}{ccccccc} — & u & c & a' & d & v & — \\ & & & \uparrow & & & \\ & & & q' & & & \end{array}$$

New configuration $(uc, q', a'dv)$.

Machinery – One-State Transition

Configuration (uc, q, av) with $a, c \in \Sigma$, $u, v \in \Sigma^*$, and $q \in Q$, $q \neq q_F$.

$$(u', q', v') = \begin{cases} (uca', q', v) & \text{if } \delta(q, a) = (q', a', R), \\ (u, q', ca'v) & \text{if } \delta(q, a) = (q', a', L), \\ (uc, q', a'v) & \text{if } \delta(q, a) = (q', a', \Lambda). \end{cases} \tag{256}$$

is the configuration *reached in one step*. Write

$$(uc, q, av) \vdash (u', q', v').$$

Machinery – Computation

- Input word $\boldsymbol{x} = x_1 x_2 \ldots x_n \in \Sigma_I^*$.
- *Initial configuration*

$$\begin{array}{ccccccc} — & x_1 & x_2 & \ldots & x_n & — \\ & \uparrow & & & & \\ & q_0 & & & & \end{array}$$

- *Computation* is a sequence of configurations

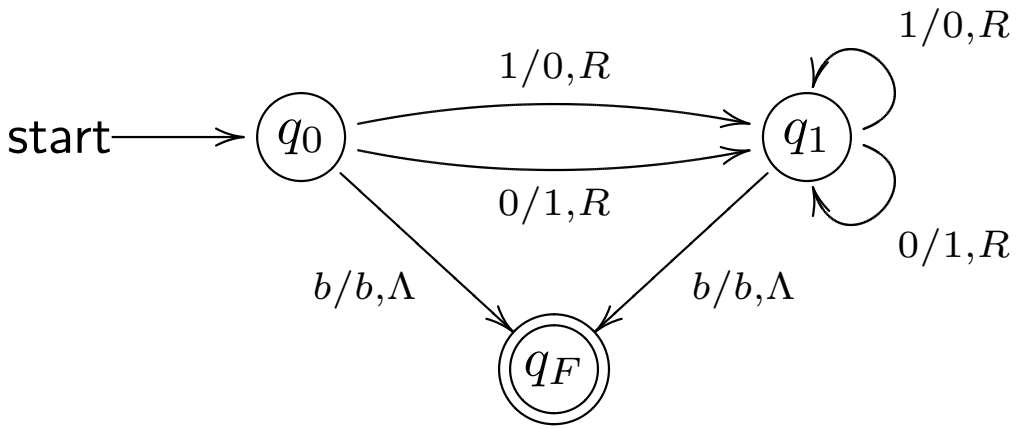
$$(b, q_0, \boldsymbol{x}) \vdash (u_1, q_1, v_1) \vdash (u_2, q_2, v_2) \vdash \ldots$$

started with initial configuration.

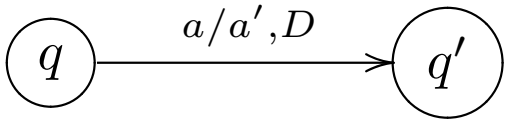
- Computation *terminates* if the machine reaches the stop state q_F; output is given by the collection of symbols on the tape.

Machinery – Example

Turing machine $M = (\Sigma, Q, \delta, q_0, q_F)$ with tape alphabet $\Sigma = \{0, 1, b\}$, state set $Q = \{q_0, q_1, q_F\}$, and transition function δ:



Transition $\delta(q, a) = (q', a', D)$ is indicated by

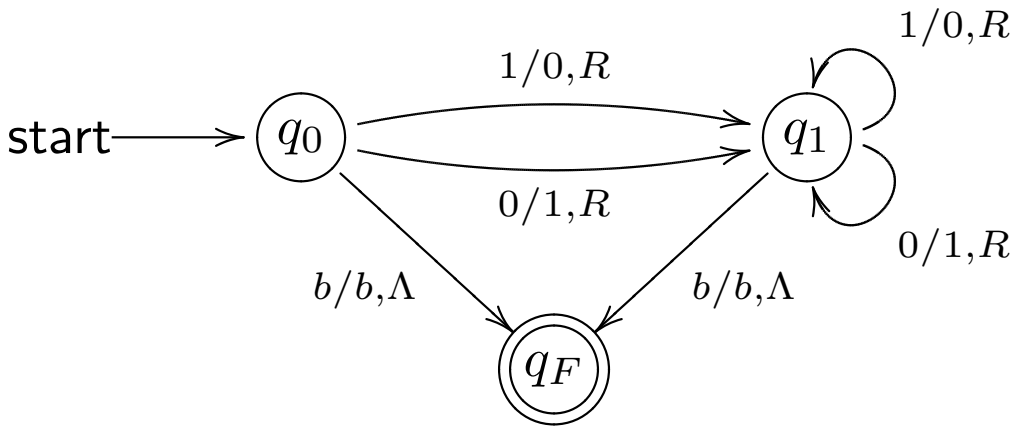


Machinery – Example

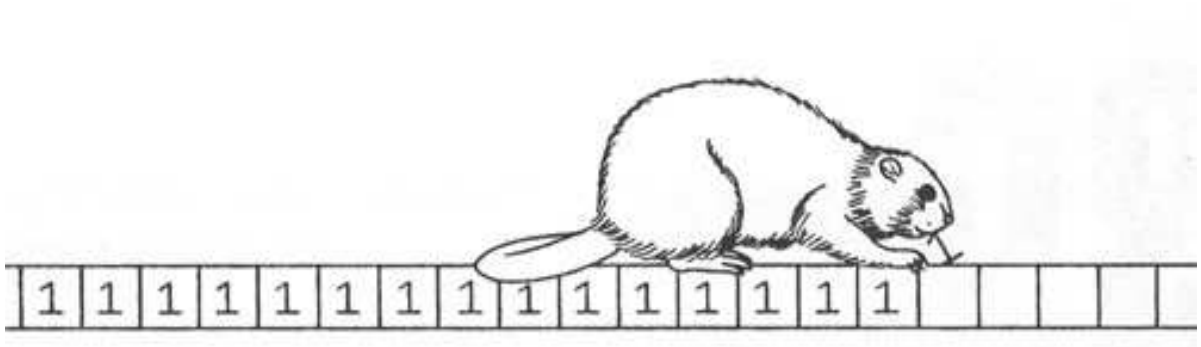
Computation provides bitwise complement of binary input word 0011:

$$\begin{aligned}(b, q_0, 0011) &\vdash (1, q_1, 011)\\ &\vdash (11, q_1, 11)\\ &\vdash (110, q_1, 1)\\ &\vdash (1100, q_1, b)\\ &\vdash (1100, q_F, b).\end{aligned}$$

Turing automaton graph:



Busy Beaver



Busy Beaver – Contents

- Problem statement
- Busy Beaver function
- Small Busy Beavers
- Correlation with superpowers
- Rado's result

Busy Beaver Problem

- *Busy beaver problem:*

  Construct a Turing machine with binary input alphabet $\Sigma = \{1, b\}$ which halts after writing the most 1's on the tape when started from the blank tape.

- *Busy beaver* is a Turing machine which solves the Busy beaver problem.
- *Busy beaver function* $\Upsilon : \mathbb{N} \to \mathbb{N}$, where $\Upsilon(n)$ is the maximum number of 1's finally on the tape among all halting two-symbol n-state Turing machines when started with the blank tape.
- *Maximum shifts function* $S : \mathbb{N} \to \mathbb{N}$, where $S(n)$ is the maximum number of moves (L or R) that can be made by any halting two-symbol n-state Turing machine.

Busy Beaver

For small values of n, the busy beaver function is known:

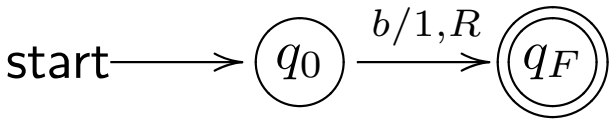
$$\begin{aligned} \Upsilon(1) &= 1, \\ \Upsilon(2) &= 4, \\ \Upsilon(3) &= 6, \\ \Upsilon(4) &= 13. \end{aligned}$$

In all other cases only lower bounds have been established.

Busy Beaver with One State

Busy beaver $M_1 = (\{1, b\}, \{q_0, q_F\}, \delta, q_0, q_F)$ computes

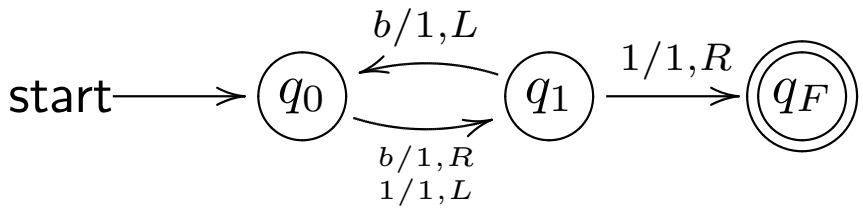
$$(b, q_0, b) \vdash (b1, q_F, b).$$



Busy Beaver with Two States

The busy beaver $M_2 = (\{1, b\}, \{q_0, q_1, q_F\}, \delta, q_0, q_F)$ computes

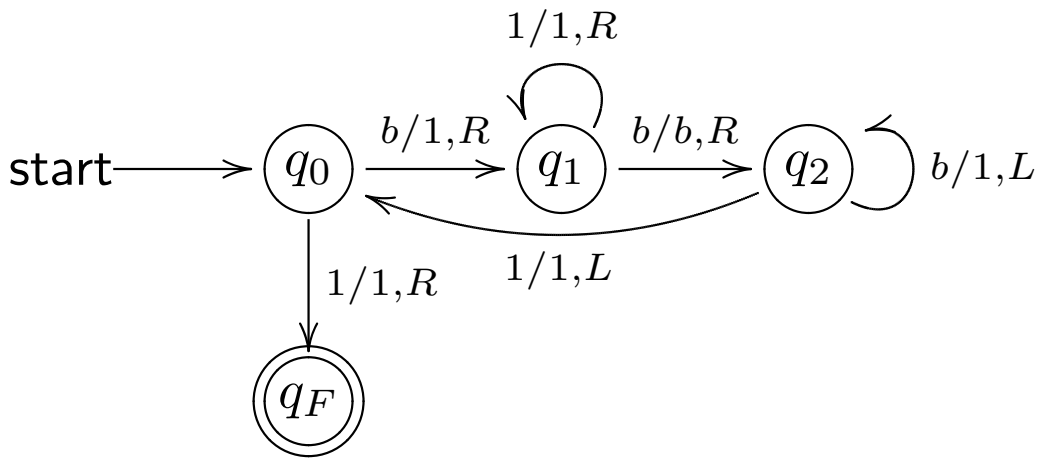
$$\begin{aligned}(b, q_0, b) &\vdash (b1, q_1, b) \vdash (b, q_0, 11b)\\ &\vdash (b, q_1, b11b) \vdash (b, q_0, b111b)\\ &\vdash (b1, q_1, 111b) \vdash (b11, q_F, 11b).\end{aligned}$$



Busy Beaver with Three States

Busy beaver $M_3 = (\{1, b\}, \{q_0, q_1, q_2, q_F\}, \delta, q_0, q_F)$ computes

$$\begin{aligned}
(b, q_0, b) &\vdash (b1, q_1, b) \vdash (b1b, q_2, b) \vdash (b1, q_2, b1b) \\
&\vdash (b, q_2, 111b) \vdash (b, q_2, b111b) \vdash (b, q_0, b1111b) \\
&\vdash (b1, q_1, 111b) \vdash (b11, q_1, 11b) \vdash (b111, q_1, 1b) \\
&\vdash (b1111, q_1, b) \vdash (b1111b, q_2, b) \vdash (b1111, q_2, b1b) \\
&\vdash (b111, q_2, 111b) \vdash (b11, q_0, 1111b) \vdash (b111, q_F, 111b).
\end{aligned}$$



Busy Beaver Function – Superpowers

(Milton Green, 1964) For each number $k \geq 2$,

$$\Upsilon(2k) > 3 \uparrow^{k-2} 3. \tag{257}$$

In particular,

$$\begin{aligned}
\Upsilon(4) &> 3 \uparrow^{0} 3 = 3 \cdot 3 = 9, \\
\Upsilon(6) &> 3 \uparrow^{1} 3 = 3^3 = 27, \\
\Upsilon(8) &> 3 \uparrow^{2} 3 = 3 \uparrow (3 \uparrow 3) = 3^{3^3} = 7\,625\,597\,484\,987, \\
\Upsilon(10) &> 3 \uparrow^{3} 3 = 3 \uparrow^{2} (3 \uparrow^{2} 3) = 3 \uparrow^{2} 3^{3^3} \quad \text{(tritri)} \\
\Upsilon(12) &> 3 \uparrow^{4} 3 = 3 \uparrow^{3} (3 \uparrow^{3} 3) \quad \text{(1st Graham number)}.
\end{aligned}$$

Busy Beaver Function – Rado's Result

The busy beaver function Υ is not partial recursive (Tibor Rado, 1962).

Proof Idea

For any computable function $f : \mathbb{N}_0 \to \mathbb{N}_0$, $\Upsilon(n) > f(n)$ for all sufficiently large n and hence Υ is not computable.

The maximum shifts function S is not partial recursive.

Proof.

There is a two-symbol n-state Turing machine M that writes $\Upsilon(n)$ 1's on the tape and then halts. This Turing machine makes at least $\Upsilon(n)$ moves. Thus $S(n) \geq \Upsilon(n)$. Hence S is not computable. □

Skills

- Writing and interpreting a Turing program

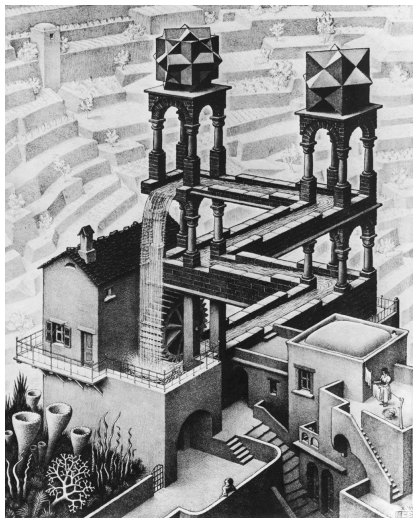
Part VIII

Undecidability

Undecidability – Motivation

- In computability theory, decision problems are yes-or-no questions on an input set.
- Undecidable problem does not allow to construct a general algorithm that always leads to a correct yes-or-no answer.
- Most prominent examples: halting problem, word problems in formal language theory and algebra, Hilbert's tenth problem.

Self-reference (M.C. Escher, 1898-1972)



Undecidability – Contents

- Undecidable sets
- Semidecidable sets
- Recursively enumerable sets
- Theorems of Rice and Rice-Shapiro
- Recursion theory

Reading: Zimmermann, Chapter 7

Decidable and Undecidable Sets

- Decidable sets
- Prototypical undecidable set
- Halting problem

Decidable and Undecidable Sets

Let $A \subseteq \mathbb{N}_0^n$.

- A is *decidable* if characteristic function χ_A is recursive, where for all $\boldsymbol{x} \in \mathbb{N}_0^n$,

$$\chi_A(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \in A, \\ 0 & \text{otherwise,} \end{cases} \tag{258}$$

  i.e., characteristic function χ_A is total and computable.
- If A is decidable, the question "$\boldsymbol{x} \in A$" can be decided by computing $\chi_A(\boldsymbol{x})$.
- Algorithm for computing χ_A is a *decision procedure* for A.
- A is *undecidable* if A is not decidable (i.e., χ_A is not computable).

Decidable Sets

If $A, B \subseteq \mathbb{N}_0^n$ are decidable, then decidable are

$$\bar{A},\ A \cup B,\ A \cap B, \text{ and } A \setminus B = A \cap \bar{B}. \tag{259}$$

Proof.

For each $x \in \mathbb{N}_0^n$,

$$\begin{aligned}
\chi_{\bar{A}}(\boldsymbol{x}) &= \operatorname{csg} \circ \chi_A(\boldsymbol{x}), && (260)\\
\chi_{A \cup B}(\boldsymbol{x}) &= \operatorname{sgn} \circ (\chi_A(\boldsymbol{x}) + \chi_B(\boldsymbol{x})), && (261)\\
\chi_{A \cap B}(\boldsymbol{x}) &= \chi_A(\boldsymbol{x}) \cdot \chi_B(\boldsymbol{x}), && (262)\\
\chi_{A \setminus B}(\boldsymbol{x}) &= \chi_A \cdot \chi_{\bar{B}}(\boldsymbol{x}). && (263)
\end{aligned}$$

The functions on the right-hand sides are compositions of recursive functions and so are also recursive. □

Decidable Sets

- Each finite set $A \subseteq \mathbb{N}_0$ is decidable, since

$$\chi_A(x) = \operatorname{sgn}\left(\sum_{a \in A} \chi_=(a, x)\right), \quad x \in \mathbb{N}_0, \tag{264}$$

  is a composition of primitive recursive functions and thus is also primitive recursive.

  E.g., if $A = \{1, 3, 5\}$, then $\chi_A(x) = \operatorname{sgn}\left(\chi_=(1, x) + \chi_=(3, x) + \chi_=(5, x)\right)$ for each $x \in \mathbb{N}_0$.

- Empty set $\emptyset$ is finite and so is decidable.
- $\mathbb{N}_0$ is decidable, since $\mathbb{N}_0 = \bar{\emptyset}$, $\emptyset$ is decidable and decidable sets are closed under complement (259).
- Set of prime numbers P is decidable (chapter 2).

Prototypical Undecidable Set

- Consider the sequence of all monadic partial recursive functions

$$\phi_0^{(1)}, \phi_1^{(1)}, \phi_2^{(1)}, \dots \tag{265}$$

where $\phi_e^{(1)} : \mathbb{N}_0 \to \mathbb{N}_0$ denotes the partial recursive function computed by GOTO program P_e, $e \in \mathbb{N}_0$.

- The set

$$K = \left\{ x \in \mathbb{N}_0 \mid x \in \operatorname{dom} \phi_x^{(1)} \right\} \tag{266}$$

is undecidable, where $x \in \operatorname{dom} \phi_x^{(1)}$ means that program P_x with input x halts.

Proof

Suppose K would be decidable; i.e., χ_K would be recursive.

- Function $f : \mathbb{N}_0 \to \mathbb{N}_0$ given by

$$f(x) = \begin{cases} 0 & \text{if } x \notin K, \\ \uparrow & \text{otherwise,} \end{cases} \tag{267}$$

  is partial recursive.

  Indeed, if P is an URM program for χ_K, then

$$P; (A1)1$$

  is an URM program for f; recall the progam $(A1)1$.

- Thus f is partial recursive and so $\phi_e^{(1)} = f$ for some $e \in \mathbb{N}_0$.
- *Diagonalization* argument:

$$e \in K \Leftrightarrow e \in \operatorname{dom} \phi_e^{(1)} \Leftrightarrow e \in \operatorname{dom}(f) \Leftrightarrow e \notin K.$$

  Contradiction. □

Computable Function I

Function $f : \mathbb{N}_0 \to \mathbb{N}_0$ in (267) is not partial recursive.

- Function $h : \mathbb{N}_0 \to \mathbb{N}_0$ defined by exchanging the cases in f:

$$h(x) = \begin{cases} 0 & \text{if } x \in K, \\ \uparrow & \text{otherwise,} \end{cases} \tag{268}$$

is partial recursive, since for each $x \in \mathbb{N}_0$,

$$\begin{aligned} h(x) &= 0 \cdot \phi_x^{(1)}(x) \\ &= 0 \cdot \psi_{\text{univ}}^{(1)}(x, x). \end{aligned} \tag{269}$$

Computable Function II

Function $h : \mathbb{N}_0 \to \mathbb{N}_0$ in (268) is partial recursive.

- Function $h' : \mathbb{N}_0 \to \mathbb{N}_0$ related to h and given by

$$h'(x) = \begin{cases} x & \text{if } x \in K, \\ \uparrow & \text{otherwise,} \end{cases} \tag{270}$$

  is partial recursive, since for each $x \in \mathbb{N}_0$,

$$\begin{aligned} h'(x) &= x \cdot \operatorname{sgn}(\phi_x^{(1)}(x) + 1) \\ &= x \cdot \operatorname{sgn}(\phi_{\text{univ}}^{(1)}(x, x) + 1). \end{aligned} \tag{271}$$

- Domain and range of partial recursive function h' are undecidable sets, since

$$\operatorname{dom}(h') = K = \operatorname{ran}(h'). \tag{272}$$

Diagonalization and Reduction

- Now one undecidable set K is at hand (proved by diagonalization).
- Any other set can be proved to be undecidable by using reduction.
- Let $A \subseteq \mathbb{N}_0^k$ and $B \subseteq \mathbb{N}_0^l$.

  A is *reducible* to B if there is a recursive function $f : \mathbb{N}_0^k \to \mathbb{N}_0^l$ such that

$$\boldsymbol{x} \in A \quad \Longleftrightarrow \quad f(\boldsymbol{x}) \in B, \quad \boldsymbol{x} \in \mathbb{N}_0^k. \tag{273}$$

  Equivalently,

$$\chi_A(\boldsymbol{x}) = \chi_B(f(\boldsymbol{x})), \quad \boldsymbol{x} \in \mathbb{N}_0^k, \tag{274}$$

  i.e., if B is decidable, then A is decidable; or by contraposition, if A is undecidable, then B is undecidable.

Halting Problem

- *Halting problem* is one of the famous undecidabilty results.
- Interpretation: Given program P and input x to P, it cannot be decided generally whether P with input x terminates or continues to run forever.
- First proof for Turing machine by Alan Turing (1936).
- By Church's thesis, halting problem is undecidable not only for Turing machines but for any formalism capturing the notion of computability.

Halting Problem

The set

$$H = \left\{(x, y) \in \mathbb{N}_0^2 \mid y \in \operatorname{dom} \phi_x^{(1)}\right\} \tag{275}$$

is undecidable, where $y \in \operatorname{dom} \phi_x^{(1)}$ means that program P_x with input y halts.

Proof.

Set K can be reduced to set H by the recursive mapping

$$f : \mathbb{N}_0 \to \mathbb{N}_0^2 : x \mapsto (x, x).$$

Indeed, for each $x \in \mathbb{N}_0$,

$$\chi_K(x) = \chi_H(x, x) = \chi_H(f(x)).$$

Since K is undecidable, H is undecidable. □

Undecidable Sets

Let $a \in \mathbb{N}_0$. The following sets are undecidable:

$$C = \left\{ x \in \mathbb{N}_0 \mid \phi_x^{(1)} = c_0^{(1)} \right\}, \tag{276}$$

$$E = \left\{ (x, y) \in \mathbb{N}_0^2 \mid \phi_x^{(1)} = \phi_y^{(1)} \right\}, \tag{277}$$

$$T = \left\{ x \in \mathbb{N}_0 \mid \phi_x^{(1)} \text{ is total} \right\}, \tag{278}$$

$$I_a = \left\{ x \in \mathbb{N}_0 \mid a \in \operatorname{dom} \phi_x^{(1)} \right\}, \tag{279}$$

$$O_a = \left\{ x \in \mathbb{N}_0 \mid a \in \operatorname{ran} \phi_x^{(1)} \right\}, \tag{280}$$

where $c_0^{(1)}$ is the monadic zero function.

Proof by Theorem of Rice.

Undecidable Sets

Practical implications using the formalism of GOTO programs:

- The problem whether a GOTO program halts with a given input is undecidable (set H).
- The problem whether a GOTO program computes a specific function (here $c_0^{(1)}$) is undecidable (set C).
- The problem whether two GOTO programs are semantically equivalent, i.e., exhibit the same input-output behaviour, is undecidable (set E).
- The problem whether a GOTO program always halts is undecidable (set T).
- The problem whether a GOTO program halts for a specific input is undecidable (set I_a).
- The problem whether a GOTO program halts and computes a specific output is undecidable (set O_a).

Semidecidable Sets

- Semidecidability
- Partial decision procedure
- Existential quantification
- Function graph

Semidecidable Sets

Let $A \subseteq \mathbb{N}_0^n$.

- Informally, A is semidecidable if there is an algorithm such that the set of input numbers for which the algorithm halts is exactly the set of elements in A.
- A is *semidecidable* if the function $f : \mathbb{N}_0^n \to \mathbb{N}_0$ defined by

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \in A, \\ \uparrow & \text{otherwise,} \end{cases} \tag{281}$$

  is partial recursive; f is the *semicharacteristic function* of A.

Semidecidable Sets

Let $A \subseteq \mathbb{N}_0^n$. A is semidecidable iff there is an n-adic partial recursive function f such that

$$\operatorname{dom}(f) = A. \tag{282}$$

Proof

- Let A be semidecidable. Then the corresponding function f given in (281) has the property that $\operatorname{dom}(f) = A$.
- Let A be a subset of $\mathbb{N}_0^n$ for which there is a partial recursive function $h : \mathbb{N}_0^n \to \mathbb{N}_0$ such that $\operatorname{dom}(h) = A$. Then the function $f = \nu \circ c_0^{(1)} \circ h : \mathbb{N}_0^n \to \mathbb{N}_0$ satisfies for each $\boldsymbol{x} \in \mathbb{N}_0^n$:

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \in A, \\ \uparrow & \text{otherwise.} \end{cases} \tag{283}$$

f is partial recursive as it is the composition of partial recursive functions, and is the semicharacteristic function of A. Hence, A is semidecidable. □

Gödel Numbering of Domains

Let $n \geq 1$.

- Gödel numbering of SGOTO programs yields an enumeration of all n-adic partial recursive functions

$$\phi_0^{(n)}, \phi_1^{(n)}, \phi_2^{(n)}, \ldots, \tag{284}$$

where partial recursive function $\phi_e^{(n)} : \mathbb{N}_0^n \to \mathbb{N}_0$ is computed by SGOTO program P_e, i.e., $\phi_e^{(n)} = \|P_e\|_{n,1}$ for each $e \in \mathbb{N}_0$.

- By (282), we obtain an enumeration of all semidecidable subsets of $\mathbb{N}_0^n$:

$$D_0^{(n)}, D_1^{(n)}, D_2^{(n)}, \ldots, \tag{285}$$

where $D_e^{(n)} = \operatorname{dom} \phi_e^{(n)}$, $e \in \mathbb{N}_0$.

Semidecidable Sets – K

$K = \left\{ x \mid x \in \operatorname{dom} \phi_x^{(1)} \right\}$ is semidecidable.

Proof.

Function $h : \mathbb{N}_0 \to \mathbb{N}_0$ in (268) defined by

$$h(x) = \begin{cases} 0 & \text{if } x \in K, \\ \uparrow & \text{otherwise} \end{cases} \tag{286}$$

is partial recursive, since

$$h(x) = 0 \cdot \phi_x^{(1)}(x) = 0 \cdot \psi_{\text{univ}}^{(1)}(x, x).$$

Since $K = \operatorname{dom}(h)$, it follows from (282) that K is semidecidable. □

Semidecidable Sets – Halting Problem

$H = \left\{(x,y) \mid y \in \operatorname{dom} \phi_x^{(1)}\right\}$ is semidecidable.

Proof.

Universal function $\psi_{\text{univ}}^{(1)}$ has the property

$$\psi_{\text{univ}}^{(1)}(x,y) = \begin{cases} \phi_x^{(1)}(y) & \text{if } y \in \operatorname{dom} \phi_x^{(1)}, \\ \uparrow & \text{otherwise.} \end{cases} \tag{287}$$

Since $H = \operatorname{dom} \psi_{\text{univ}}^{(1)}$, it follows from (282) that H is semidecidable. □

Decidable Sets

Each decidable set is semidecidable.

Proof

Let $A \subseteq \mathbb{N}_0^n$ be decidable, i.e., its characteristic function χ_A is recursive.

- Function $g : \mathbb{N}_0^n \to \mathbb{N}_0$ defined by

$$g(\boldsymbol{x}) = (\mathrm{csg} \circ \chi_A)(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \boldsymbol{x} \in A, \\ 1 & \text{otherwise,} \end{cases}$$

is recursive, since it is the composition of recursive functions.

Proof (Cont'd)

- Let P be an URM program computing g, i.e., $g = \|P\|_{n,1}$. Then the URM program

$$P; (A1)1$$

computes the function (recall the program $(A1)1$)

$$h(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \boldsymbol{x} \in A, \\ \uparrow & \text{otherwise.} \end{cases}$$

- Function h is partial recursive, since it can be computed by an URM program.
- Since $A = \mathrm{dom}\,(h)$, it follows from (282) that A is semidecidable. □

Partial Decision Procedure

Let $A \subseteq \mathbb{N}_0^n$ be a semidecidable set.

- By (282), $A = \operatorname{dom}(f)$ for some partial recursive function $f : \mathbb{N}_0^n \to \mathbb{N}_0$.
- Let P be a GOTO program computing f, i.e., $f = \|P\|_{n,1}$.
- *Partial decision procedure* for A:
  - Start the program P with input $\boldsymbol{x} \in \mathbb{N}_0^n$.
  - If P halts, $\boldsymbol{x} \in A$. Otherwise, P will continue to run forever (undefined case).

Partial Decision Procedure – Consequence

Let $A \subseteq \mathbb{N}_0^n$.

$$A \text{ is decidable} \iff A \text{ and } \bar{A} \text{ semidecidable.} \tag{288}$$

Proof.

- Let A be decidable.

  Then by (259), $\bar{A}$ is also decidable. But each decidable set is semidecidable and so A and $\bar{A}$ are semidecidable.

- Let A and $\bar{A}$ be semidecidable.

  Run the partial decision procedures for A and $\bar{A}$ in parallel.

  For each $\boldsymbol{x} \in \mathbb{N}_0^n$, one of these procedures will halt in a finite number of steps giving an answer to the decision procedure for A.

Complement of Halting Problem

The complement of the halting problem H given by the set

$$\bar{H} = \left\{(x, y) \in \mathbb{N}_0^2 \mid y \notin \operatorname{dom} \phi_x^{(1)}\right\} \tag{289}$$

is not semidecidable.

Proof.

Suppose $\bar{H}$ would be semidecidable. Since H is semidecidable, it follows by (288) that H is decidable but it is not. □

Semidecidable Sets – Existential Quantification

A set A is semidecidable iff there is a decidable set B such that

$$A = \exists y[(\boldsymbol{x}, y) \in B] = \{\boldsymbol{x} \mid \exists y : \ (\boldsymbol{x}, y) \in B\}. \tag{290}$$

Proof

Let $B \subseteq \mathbb{N}_0^{n+1}$ be decidable and $A = \exists y[(\boldsymbol{x}, y) \in B]$.

- Consider the function $f : \mathbb{N}_0^n \to \mathbb{N}_0$ given by

$$\begin{aligned} f(\boldsymbol{x}) &= \mu(\operatorname{csg} \circ \chi_B)(\boldsymbol{x}) \\ &= \mu y[(\operatorname{csg} \circ \chi_B)(\boldsymbol{x}, y) = 0] \\ &= \begin{cases} 0 & \text{if } (\boldsymbol{x}, y) \in B \text{ for some } y \in \mathbb{N}_0, \\ \uparrow & \text{otherwise.} \end{cases} \end{aligned}$$

- This function is partial recursive with $\operatorname{dom}(f) = A$. By (282), A is semidecidable.

Proof (Cont'd)

Let $A \subseteq \mathbb{N}_0^n$ be semidecidable.

- Then $\operatorname{dom} \phi_e^{(n)} = A$ for some $e \in \mathbb{N}_0$.
- Consider the Kleene set T_n, i.e., $(e, \boldsymbol{x}, y) \in T_n$ iff SGOTO P_e with input $\boldsymbol{x}$ halts with output $\kappa_2(y)$ in $\tau_2(y)$ steps; alternatively consider $\phi_e^{(1)}(\boldsymbol{x})$ instead of P_e with input $\boldsymbol{x}$.
- T_n is primitive (decidable).
- For each $\boldsymbol{x} \in \mathbb{N}_0^n$, we have $\boldsymbol{x} \in A$ iff $\boldsymbol{x} \in \exists y[(e, \boldsymbol{x}, y) \in T_n]$, where e is kept fixed. Hence, $A = \exists y[(e, \boldsymbol{x}, y) \in T_n]$. □

Existential Quantification – Example

Consider the Kleene set T_1, i.e., $(e, x, y) \in T_1$ iff SGOTO P_e with input x halts with output $\kappa_2(y)$ in $\tau_2(y)$ steps; alternatively consider $\phi_e^{(1)}(x)$ instead of P_e with input x.

- T_1 is primitive (decidable).
- The semidecidable set $K = \left\{ x \mid x \in \operatorname{dom} \phi_x^{(1)} \right\}$ fulfills

$$K = \exists y[(x, x, y) \in T_1]. \tag{291}$$

- The semidecidable set $H = \left\{ (x, y) \mid y \in \operatorname{dom} \phi_x^{(1)} \right\}$ satisfies

$$H = \exists z[(x, y, z) \in T_1]. \tag{292}$$

Existential Quantification – Example

The set $Z_0 = \left\{ x \mid \phi_x^{(1)}(x) = 0 \right\}$ is semidecidable.

- Consider the extended Kleene set S_1, i.e., $(e, x, y, t) \in S_1$ iff SGOTO P_e with input x halts with output y in t steps; alternatively consider $\phi_e^{(1)}(x)$ instead of P_e with input x.
- S_1 is primitive (decidable).
- The set Z_0 fulfills

$$Z_0 = \exists t[(x, x, 0, t) \in S_1]$$

  and so is semi-decidable by (290).
- Z_0 is not decidable (Theorem of Rice).
- Complementary set $\bar{Z}_0 = \left\{ x \mid \phi_x^{(1)}(x) \neq 0 \right\}$ cannot be semidecidable by (288).

Existential Quantification – Closure Property

If B is semidecidable, then

$$A = \exists y[(\boldsymbol{x}, y) \in B] \tag{293}$$

is semidecidable.

Proof

Let $B \subseteq \mathbb{N}_0^{n+1}$ be semidecidable.

- By the above result, there is a decidable set $C \subseteq \mathbb{N}_0^{n+2}$ such that $B = \exists z[(\boldsymbol{x}, y, z) \in C]$.
- The search for a pair (y, z) with $(\boldsymbol{x}, y, z) \in C$ can be replaced by the search for a number u such that $(\boldsymbol{x}, \kappa_2(u), \tau_2(u)) \in C$ by using the inverse pair κ_2, τ_2 of the Cantor pairing σ_2.
- Thus $A = \exists u[(\boldsymbol{x}, \kappa_2(u), \tau_2(u)) \in C]$ and hence A is semidecidable by (290). □

Existential Quantification – Example

In view of Fermat's Last Theorem (around 1637), consider the set

$$F = \exists x\, \exists y\, \exists z\, [xyz \neq 0 \wedge x^n + y^n = z^n].$$

It can be written in the form

$$F = \exists t\ [\sigma_{31}(t)\sigma_{32}(t)\sigma_{33}(t) \neq 0 \wedge \sigma_{31}(t)^n + \sigma_{32}(t)^n = \sigma_{33}(t)^n],$$

where $\sigma_3 : \mathbb{N}_0^3 \to \mathbb{N}_0$ is bijective and $\sigma_{31}, \sigma_{32}, \sigma_{33} : \mathbb{N}_0 \to \mathbb{N}_0$ such that

$$\begin{aligned} \sigma_3(x,y,z) &= \sigma_2(\sigma_2(x,y),z), \\ \sigma_{31}(u) &= \kappa_2(\kappa_2(u)), \\ \sigma_{32}(u) &= \tau_2(\kappa_2(u)), \\ \sigma_{33}(u) &= \tau_2(u). \end{aligned}$$

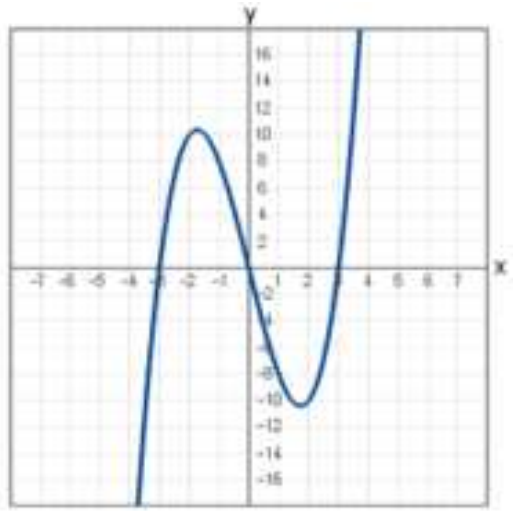
It is well-known (1994) that $F = \{1, 2\}$ and so F is decidable.

Function Graph

Each partial function $f : \mathbb{N}_0^n \to \mathbb{N}_0$ has a *graph* given by the $n+1$-adic relation

$$\mathrm{graph}(f) = \{(\boldsymbol{x}, f(\boldsymbol{x})) \in \mathbb{N}_0^{n+1} \mid \boldsymbol{x} \in \mathbb{N}_0^n\}. \tag{294}$$

If $f : \mathbb{N}_0^n \to \mathbb{N}_0$ is partial recursive, then $\mathrm{graph}(f)$ is semidecidable.



Proof

Let f be partially recursive.

- Then there is a Gödel number $e \in \mathbb{N}_0$ such that

$$f = \phi_e^{(n)}.$$

- By the extended Kleene set S_n, define the function

$$g(\boldsymbol{x}, z) = \mu t(\chi_{S_n}(e, \boldsymbol{x}, z, t) = 1), \quad \boldsymbol{x} \in \mathbb{N}_0^n, z \in \mathbb{N}_0,$$

  where $(e, \boldsymbol{x}, z, t) \in S_n$ iff program P_e with input $\boldsymbol{x}$ ends after t steps with output z, and e is kept fixed by parametrization (67).

- Function g is partial recursive as it is given by minimalization of the primitive recursive function χ_{S_n} with parameter e.

- Since $\mathrm{dom}\,(g) = \mathrm{graph}(f)$, it follows from (282) that $\mathrm{graph}(f)$ is semidecidable. □

Function Graph – Example

Let $e \in \mathbb{N}_0$. The function $\phi_e^{(1)} : \mathbb{N}_0 \to \mathbb{N}_0$ is partial recursive and so its function graph

$$\operatorname{graph}(\phi_e^{(1)}) = \left\{ (x, \phi_e^{(1)}(x)) \mid x \in \mathbb{N}_0 \right\}$$

is semidecidable.

Recursively Enumerable Sets

- Semidecidable sets can be equivalently described by r.e. sets.
- An r.e. set can be enumerated by a recursive function.
- Restriction to the case $\mathbb{N}_0^n$ with $n = 1$.

Encoding of $\mathbb{N}_0^n$

For each $n \geq 1$, define primitive recursive bijection $\sigma_n : \mathbb{N}_0^n \to \mathbb{N}_0$.

- For $n = 1$,

$$\sigma_1 = \mathrm{id}_{\mathbb{N}_0}. \tag{295}$$

- For $n > 1$,

$$\sigma_n(x_1, \ldots, x_n) = \sigma_2(\sigma_{n-1}(x_1, \ldots, x_{n-1}), x_n), \tag{296}$$

where $\sigma_2 : \mathbb{N}_0^2 \to \mathbb{N}_0$ is the Cantor pairing (216).

E.g., by (223),

$$\begin{aligned} \sigma_3(2,3,4) &= \sigma_2(\sigma_2(2,3),4) = \sigma_2(17,4) \\ &= \binom{17+4+1}{2} + 17 = 248. \end{aligned}$$

Semidecidable Sets

Let $A \subseteq \mathbb{N}_0^n$ and $\sigma_n : \mathbb{N}_0^n \to \mathbb{N}_0$ be a recursive bijection. Then A is semidecidable iff $\sigma_n(A)$ is semidecidable.

Proof

- Let $A \subseteq \mathbb{N}_0^n$ be semidecidable, i.e., there is a partial recursive function $f : \mathbb{N}_0^n \to \mathbb{N}_0$ such that $\operatorname{dom}(f) = A$. Then $g = f \circ \sigma_n^{-1} : \mathbb{N}_0 \to \mathbb{N}_0$ is partial recursive with $\operatorname{dom}(g) = \sigma_n(A)$:

$$\mathbb{N}_0 \xrightarrow{\sigma_n^{-1}} \mathbb{N}_0^n \xrightarrow{f} \mathbb{N}_0, \quad \operatorname{dom}(f) = A.$$

- Let $\sigma_n(A) \subseteq \mathbb{N}_0$ be semideciable, i.e., there is a partial recursive function $g : \mathbb{N}_0 \to \mathbb{N}_0$ such that $\operatorname{dom}(g) = \sigma_n(A)$. Then $f = g \circ \sigma_n : \mathbb{N}_0 \to \mathbb{N}_0$ is partial recursive with $\operatorname{dom}(f) = A$:

$$\mathbb{N}_0^n \xrightarrow{\sigma_n} \mathbb{N}_0 \xrightarrow{g} \mathbb{N}_0, \quad \operatorname{dom}(g) = \sigma_n(A).$$

Semidecidable Sets – Example

Halting problem $H = \left\{(x,y) \mid y \in \operatorname{dom} \phi_x^{(1)}\right\}$ is semidecidable with

$$H = \operatorname{dom} \psi_{\text{univ}}^{(1)}.$$

By the Cantor pairing $\sigma_2 : \mathbb{N}_0^2 \to \mathbb{N}_0$, the function

$$g = \psi_{\text{univ}}^{(1)} \circ \sigma_2^{-1} : \mathbb{N}_0 \to \mathbb{N}_0$$

is partial recursive with $\operatorname{dom}(g) = \sigma_2(H)$.

Recursive Enumeration

Let $A \subseteq \mathbb{N}_0$.

A is *recursively enumerable* (r.e.) if A is empty or there is a recursive function $f : \mathbb{N}_0 \to \mathbb{N}_0$ such that

$$A = \operatorname{ran}(f). \tag{297}$$

Such a function f is an *enumerator* of A, i.e.,

$$A = \{f(0), f(1), f(2), \ldots\}. \tag{298}$$

Recursive Enumeration

Let $A \subseteq \mathbb{N}_0$ be a nonempty r.e. set. There is an enumerator f of A such that

$$f(0) < f(1) < f(2) < \dots.$$

Proof

Define the monadic function f by minimalization and primitive recursion:

$$f(0) = \mu y[y \in A], \tag{299}$$

$$f(n+1) = \mu y[y \in A \wedge y > f(n)]. \tag{300}$$

Clearly, f is recursive, strictly monotonous and satisfies $\operatorname{ran}(f) = A$. □

Main Theorem

For each set $A \subseteq \mathbb{N}_0$ and $n \geq 1$, the following are equivalent:

- A is semidecidable.
- A is r.e.
- There is an n-adic partial recursive function g with $A = \operatorname{ran}(g)$.

Proof will be established by ring closure.

Proof

Let $A \neq \emptyset$ be semidecidable.

- By (285), $A = \operatorname{dom} \phi_e^{(1)}$ with Gödel number $e \in \mathbb{N}_0$.
- Fix $a \in A$ and use the Kleene set T_1 to define the monadic function

$$f : x \mapsto \begin{cases} \kappa_2(x) & \text{if } (e, \kappa_2(x), \tau_2(x)) \in T_1, \\ a & \text{otherwise,} \end{cases} \tag{301}$$

  where $(e, y, z) \in T_1$ iff program P_e with input y yields the output $\kappa_2(z)$ after $\tau_2(z)$ steps.
- This function is primitive recursive as it is defined by primitive case distinction and parametrization fixing e (67); κ_2, τ_2 and χ_{T_1} are primitive recursive.
- Since the program P_e computes $\phi_e^{(1)}$ and $A = \operatorname{dom} \phi_e^{(1)}$, it follows that $A = \operatorname{ran}(f)$.

Proof (Cont'd)

Let A be r.e., i.e., $A = \emptyset$ or $A = \operatorname{ran}(f)$ for some monadic recursive function f.

- Define the partial recursive function $g : \mathbb{N}_0^n \to \mathbb{N}_0$, where for all $\boldsymbol{x} \in \mathbb{N}_0^n$,

$$g(\boldsymbol{x}) = \begin{cases} \uparrow & \text{if } A = \emptyset, \\ f(\sigma_n(\boldsymbol{x})) & \text{otherwise.} \end{cases}$$

- It immediately follows that $\operatorname{ran}(g) = A$.

Proof (Cont'd)

Let g be an n-adic partial recursive function with $A = \operatorname{ran}(g)$.

- By (294), the graph of g,

$$G = \left\{(\boldsymbol{x}, g(\boldsymbol{x})) \in \mathbb{N}_0^{n+1} \mid \boldsymbol{x} \in \mathbb{N}_0^n\right\},$$

  is semidecidable.

- Since semidecidable sets are closed under existential quantification (293), the set

$$A = \exists x_1 \ldots \exists x_n [(x_1, \ldots, x_n, y) \in G]$$

  is semidecidable. □

Recursive Enumeration – K

- Semidecidable set $K = \left\{ x \mid x \in \operatorname{dom} \phi_x^{(1)} \right\}$ is the domain of the partial recursive function

$$h(x) = \psi_{\text{univ}}^{(1)}(x, x) = \begin{cases} \phi_x^{(1)}(x) & \text{if } x \in \operatorname{dom} \phi_x^{(1)}, \\ \uparrow & \text{otherwise.} \end{cases}$$

  There exists a Gödel number $e \in \mathbb{N}_0$ such that $h = \phi_e^{(1)}$.

- By (301), an enumerator of K is given by the recursive function

$$f : x \mapsto \begin{cases} \kappa_2(x) & \text{if } (e, \kappa_2(x), \tau_2(x)) \in T_1, \\ a & \text{otherwise,} \end{cases}$$

  where $a \in K$, i.e., program P_e computes for inputs $x = 0, 1, 2, \ldots$ the outputs $f(0), f(1), f(2), \ldots$.

Recursive Enumeration – Closure Properties

If $A, B \subseteq \mathbb{N}_0$ are r.e., then $A \cap B, A \cup B$ are r.e.

Proof

- Let f and g be monadic partial recursive functions with $A = \operatorname{dom}(f)$ and $B = \operatorname{dom}(g)$. Then $f \cdot g$ with $(f \cdot g)(x) = f(x) \cdot g(x)$ is partial recursive with $\operatorname{dom}(f \cdot g) = A \cap B$. By (282), $A \cap B$ is r.e.
- Let A and B be non-empty sets with enumerators f and g, resp.; i.e., $A = \{f(0), f(1), f(2), \ldots\}$ and $B = \{g(0), g(1), g(2), \ldots\}$. Then $A \cup B = \{h(0), h(1), h(2), \ldots\}$ with enumerator

$$h : x \mapsto \begin{cases} f(\lfloor x/2 \rfloor) & \text{if } x \text{ is even,} \\ g(\lfloor x/2 \rfloor) & \text{otherwise.} \end{cases}$$

Here $h(0) = f(0)$, $h(1) = g(0)$, $h(2) = f(1)$, $h(3) = g(1)$ and so on. Function h defined by cases is recursive and satisfies $\operatorname{ran}(h) = A \cup B$. □

Theorems of Rice and of Rice-Shapiro

- Theorem of Rice states that for any non-trivial property of partial recursive functions, there is no general and effective method to decide whether a partial recursive function has this property or not.
- A property of partial recursive functions is called *trivial* if it holds for all partial recursive functions or for none of them.
- Theorem of Rice-Shapiro posed by Henry Gordon Rice and proved by Norman Shapiro extends of the Theorem of Rice.

Theorem of Rice (1945)

Let $\mathcal{C}$ be a class of non-trivial monadic partial recursive functions, i.e.,

$$\emptyset \neq \mathcal{C} \subset \mathcal{R}^{(1)}. \tag{302}$$

The corresponding index set

$$\mathrm{I}(\mathcal{C}) = \left\{ x \in \mathbb{N}_0 \mid \phi_x^{(1)} \in \mathcal{C} \right\} \tag{303}$$

is undecidable.

Special case of Rice-Shapiro (proof below).

Theorem of Rice – Example

- Let $f : \mathbb{N}_0 \to \mathbb{N}_0 : x \mapsto 2x$ be the doubling function and

$$\mathcal{C} = \{g \mid g \text{ is a doubling function}\}.$$

Theorem of Rice says that $\mathrm{I}(\mathcal{C})$ is undecidable; it cannot be decided whether a given number is an index of a doubling function.

- Let P denote the set of primes and let

$$\mathcal{C} = \{f \mid \mathrm{dom}\,(f) = P\}.$$

Theorem of Rice implies that $\mathrm{I}(\mathcal{C})$ is undecidable; it cannot be decided whether a given number is an index of a partial recursive function whose domain is P.

Extension

- Let $f, g : \mathbb{N}_0 \to \mathbb{N}_0$ be functions.

  f is an *extension* of g, written $g \subseteq f$, if

  $$\mathrm{dom}\,(g) \subseteq \mathrm{dom}\,(f) \wedge g(x) = f(x) \text{ for all } x \in \mathrm{dom}\,(g). \quad (304)$$

- Relation $\subseteq$ is an ordering relation on the set of all monadic functions.
- Smallest element is the empty function $f_\uparrow$ with $\mathrm{dom}\,(f_\uparrow) = \emptyset$. Maximal elements are the total functions (domain $\mathbb{N}_0$).
- E.g., $f = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 2 & 3 \end{pmatrix}$ is an extension of $g = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 5 & 2 \end{pmatrix}$.

Finiteness

- Function $g : \mathbb{N}_0 \to \mathbb{N}_0$ is *finite* if its domain is finite.
- Each finite function g is partial recursive, since

$$g(x) = \sum_{a \in \mathrm{dom}\,(g)} \chi_=(x, a) \cdot g(a), \quad x \in \mathbb{N}_0. \tag{305}$$

  $\chi_=$ is primitive recursive with $\chi_=(x, y) = (x \dot{-} y) + (y \dot{-} x)$.

- E.g., if $g = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 2 & 3 \end{pmatrix}$, then

$$g(x) = \chi_=(x, 1) \cdot 2 + \chi_=(x, 2) \cdot 5 + \chi_=(x, 3) \cdot 2 + \chi_=(x, 4) \cdot 3.$$

Theorem of Rice-Shapiro (1956)

Let $\mathcal{C}$ be a class of monadic partial recursive functions whose corresponding index set

$$\mathrm{I}(\mathcal{C}) = \left\{ x \in \mathbb{N}_0 \mid \phi_x^{(1)} \in \mathcal{C} \right\} \tag{306}$$

is r.e.

Then for each monadic partial recursive function f,

$$f \in \mathcal{C} \iff \exists g[g \in \mathcal{C} \text{ finite} \wedge g \subseteq f], \tag{307}$$

i.e., each function $f \in \mathcal{C}$ is an extension of a finite function $g \in \mathcal{C}$.

Proof Idea

Use contraposition: If the condition (307) is not met, then $\mathrm{I}(\mathcal{C})$ will not be r.e.

*Proof

First, let $f \in \mathcal{C}$ and assume that no finite function g which is extended by f lies in $\mathcal{C}$. Take the r.e. set $K = \{x \mid x \in \operatorname{dom} \phi_x^{(1)}\}$, let e be an index for K, and let P_e be a GOTO program that computes $\phi_e^{(1)}$. Define the function

$$g : (z,t) \mapsto \begin{cases} \uparrow & \text{if } P_e \text{ computes } \phi_e^{(1)}(z) \text{ in } \leq t \text{ steps,} \\ f(t) & \text{otherwise.} \end{cases} \tag{308}$$

The function g is partial recursive. Thus by the smn theorem, there is a monadic recursive function s such that

$$g(z,t) = \phi_{s(z)}^{(1)}(t), \quad t, z \in \mathbb{N}_0. \tag{309}$$

Hence, $\phi_{s(z)}^{(1)} \subseteq f$ for each $z \in \mathbb{N}_0$. Consider two cases:

- If $z \in K$, the program $P_e(z)$ halts after, say t_0 steps. Then

$$\phi_{s(z)}^{(1)}(t) = \begin{cases} \uparrow & \text{if } t_0 \leq t, \\ f(t) & \text{otherwise.} \end{cases} \tag{310}$$

  Thus $\phi_{s(z)}^{(1)}$ is finite and hence, by hypothesis, $\phi_{s(z)}^{(1)}$ does not belong to $\mathcal{C}$.

- If $z \notin K$, the program $P_e(z)$ does not halt and so $\phi_{s(z)}^{(1)} = f$ which implies that $\phi_{s(z)}^{(1)} \in \mathcal{C}$.

It follows that the function s reduces the non-recursively enumerable set $\bar{K}$ to the set $\mathrm{I}(\mathcal{C})$ and hence the set $\mathrm{I}(\mathcal{C})$ is not r.e.

Proof (Cont'd)

Conversely, let f be a monadic partial recursive function that does not belong to $\mathcal{C}$ and let g be a finite function in $\mathcal{C}$ with $g \subseteq f$. Define the function

$$h : (z,t) \mapsto \begin{cases} f(t) & \text{if } t \in \operatorname{dom}(g) \text{ or } z \in K, \\ \uparrow & \text{otherwise.} \end{cases} \tag{311}$$

The function h is partial recursive. Thus by the smn theorem, there is a monadic recursive function s such that

$$h(z,t) = \phi^{(1)}_{s(z)}(t), \quad t, z \in \mathbb{N}_0. \tag{312}$$

Consider two cases:

- If $z \in K$, $\phi^{(1)}_{s(z)} = f$ and so $\phi^{(1)}_{s(z)} \notin \mathcal{C}$.
- If $z \notin K$, $\phi^{(1)}_{s(z)}(t) = g(t)$ for all $t \in \operatorname{dom}(g)$ and $\phi^{(1)}_{s(z)}$ is undefined elsewhere. Hence, $\phi^{(1)}_{s(z)} \in \mathcal{C}$.

It follows that the function s provides a reduction of the non-recursively enumerable set $\bar{K}$ to the set $\mathrm{I}(\mathcal{C})$ and hence the set $\mathrm{I}(\mathcal{C})$ is not r.e. □

Theorem of Rice-Shapiro – Example

Consider the set of total computable functions

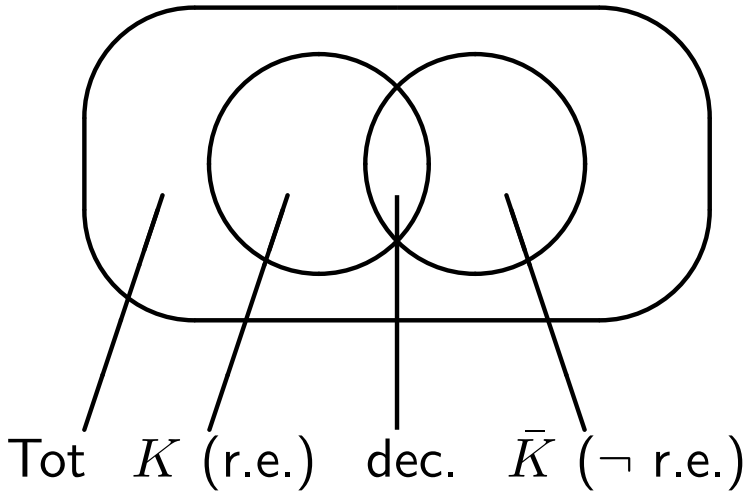
$$\mathcal{T}^{(1)} = \left\{ \phi_x^{(1)} \mid \phi_x^{(1)} \text{ total}, x \in \mathbb{N}_0 \right\}. \tag{313}$$

The set

$$\mathrm{Tot} = \mathrm{I}(\mathcal{T}^{(1)}) = \left\{ x \mid \phi_x^{(1)} \text{ total} \right\} \tag{314}$$

is not r.e., since each function $f \in \mathcal{T}^{(1)}$ is total and so cannot be an extension of a finite function $g \in \mathcal{T}^{(1)}$; finite functions are not total.

Decidable and Undecidable Sets



The decidable sets are at the intersection of r.e. and non-r.e. sets. Tot is not r.e. and its complement $\overline{\mathrm{Tot}}$ is also not r.e. (see next).
$\rightarrow$ Arithmetical hierarchy.

Total Computable Functions

$\overline{\mathrm{Tot}}$ is not r.e.

*Proof

Consider the dyadic function

$$h(x,y) = \begin{cases} x & \text{if } y \in K, \\ \uparrow & \text{otherwise.} \end{cases}$$

h is partial recursive, since

$$h(x,y) = x \cdot \operatorname{sgn}(\phi_y^{(1)}(y) + 1).$$

Thus there is an index $e \in \mathbb{N}_0$ such that $h = \phi_e^{(2)}$.
By the smn theorem (247), there is a primitive recursive function s such that

$$h(x,y) = \phi_e^{(2)}(x,y) = \phi_{s(e,y)}^{(1)}(x).$$

Proof (Cont'd)

We have

$$h(x,y) = \phi_e^{(2)}(x,y) = \phi_{s(e,y)}^{(1)}(x).$$

Since e is fixed, write

$$g_y(x) = \phi_{s(e,y)}^{(1)}(x).$$

If $y \in K$, then $g_y = \mathrm{id}_{\mathbb{N}_0}$ and if $y \notin K$, then $g_y = f_\uparrow$.
Thus if $y \in K$, then $g_y \in \mathrm{Tot}$ and if $y \notin K$, then $g_y \notin \mathrm{Tot}$.
Therefore, K is reduced to Tot and $\bar{K}$ is reduced to $\overline{\mathrm{Tot}}$.
Hence, $\overline{\mathrm{Tot}}$ is not r.e., since $\bar{K}$ is not. □

Theorem of Rice – Revisited

Let $\mathcal{C}$ be a class of monadic partial recursive functions.
If $\mathrm{I}(\mathcal{C})$ is decidable, then $\mathcal{C}$ is trivial, i.e., $\mathcal{C} = \emptyset$ or $\mathcal{C} = \mathcal{R}^{(1)}$.

Proof.

Let $\mathrm{I}(\mathcal{C})$ be decidable.

- Then by (288), $\mathrm{I}(\mathcal{C})$ and $\overline{\mathrm{I}(\mathcal{C})}$ are r.e.
- Thus we may assume that $\mathcal{C}$ contains the empty function $f_{\uparrow}$.
- Since each monadic partial recursive function f is an extension of $f_{\uparrow}$, it follows from Rice-Shapiro that $\mathcal{C} = \mathcal{R}^{(1)}$.

□

Theorem of Rice is a consequence of the Theorem of Rice-Shapiro.

Recursion Theory

- Operators on the list of partial recursive functions.
- Recursion theorems (Rodges, Kleene). For any recursive function manipulating GOTO programs there is a GOTO program which is semantically invariant under this manipulation.
- Quines as programming gimmick.
- Halting problem revisited.

Operators

- For each partial recursive function $f : \mathbb{N}_0 \to \mathbb{N}_0$, define the *operator*

$$\Phi_f : \left(\phi_e^{(1)}\right)_{e\in\mathbb{N}_0} \to \left(\phi_{f(e)}^{(1)}\right)_{e\in\mathbb{N}_0}, \tag{315}$$

  which maps the sequence of monadic partial recursive functions $\left(\phi_e^{(1)}\right)_{e\in\mathbb{N}_0}$ to the image sequence $\left(\phi_{f(e)}^{(1)}\right)_{e\in\mathbb{N}_0}$.

- If $f(e)$ is undefined, set $\phi_{f(e)}^{(1)} = f_\uparrow$ (empty function).

Recursion Theorem (Rodgers, 1967)

For each monadic recursive function f, there is an index $e \in \mathbb{N}_0$ such that

$$\phi_e^{(1)} = \phi_{f(e)}^{(1)}. \tag{316}$$

Index e in (316) is called a *fixed point* of f.

Corollary

For each monadic recursive function f, there is an index $e \in \mathbb{N}_0$ such that

$$D_e = D_{f(e)}, \tag{317}$$

where $D_k = \operatorname{dom} \phi_k^{(1)}$, $k \geq 0$.

Proof

- Consider the function $g : \mathbb{N}_0^2 \to \mathbb{N}_0$ defined by

$$g(x,y) = \begin{cases} \phi^{(1)}_{f(\phi^{(1)}_x(x))}(y) & \text{if } x \in K, \\ \uparrow & \text{otherwise.} \end{cases}$$

This function is partial recursive, since $\phi^{(1)}_{f(\phi^{(1)}_x(x))}$ is defined for each $x \in K = \left\{ x \mid x \in \operatorname{dom} \phi^{(1)}_x \right\}$ and

$$g(x,y) = \phi^{(1)}_{f(\phi^{(1)}_x(x))}(y) \cdot \operatorname{sgn}(\psi^{(1)}_{\text{univ}}(x,x) + 1).$$

- We can write for all $x \in \mathbb{N}_0$,

$$g(x,\cdot) = \phi^{(1)}_{f(\phi^{(1)}_x(x))}.$$

Proof (Cont'd)

- By the smn theorem (247), there is a primitive recursive function $s : \mathbb{N}_0 \to \mathbb{N}_0$ such that for all $x \in \mathbb{N}_0$,

$$g(x, \cdot) = \phi^{(1)}_{s(x)}.$$

- Thus

$$\phi^{(1)}_{f(\phi^{(1)}_x(x))} = g(x, \cdot) = \phi^{(1)}_{s(x)}.$$

  If $f(\phi^{(1)}_x(x))$ is undefined, $s(x)$ will be the index of the empty function.

- Since s is computable, there is an index m for s; i.e., $s = \phi^{(1)}_m$. Putting $x = m$ and $e = \phi^{(1)}_m(m) = s(m)$, we obtain

$$\phi^{(1)}_{f(e)} \overset{\text{def. } e}{=} \phi^{(1)}_{f(\phi^{(1)}_m(m))} = \phi^{(1)}_{s(m)} = \phi^{(1)}_e.$$

□

Recursion Theorem – Example

- In view of the increment function $\nu : \mathbb{N}_0 \to \mathbb{N}_0 : x \mapsto x + 1$, there is a number $e \geq 0$ such that

$$\phi_e^{(1)} = \phi_{e+1}^{(1)}.$$

- In view of the squaring function $f : \mathbb{N}_0 \to \mathbb{N}_0 : x \mapsto x^2$, there is a number $e \geq 0$ such that

$$\phi_e^{(1)} = \phi_{e^2}^{(1)}.$$

- In view of tetration $A(4, \cdot) : \mathbb{N}_0 \to \mathbb{N}_0 : x \mapsto A(4, x)$, there is a number $e \geq 0$ such that

$$\phi_e^{(1)} = \phi_{A(4,e)}^{(1)}.$$

Recursion Theorem – Example

There exists an index $e \in \mathbb{N}_0$ such that

$$D_e = \{e\}. \tag{318}$$

- For each index $e \in \mathbb{N}_0$ construct an SGOTO program P_e which halts with input e but otherwise runs forever.
- Define the function $f : \mathbb{N}_0 \to \mathbb{N}_0$ with

  $$f(e) = \rho(P_e),$$

  where $\rho(P)$ is the Gödel number of SGOTO program P. Since ρ is primitive recursive, f is primitive recursive.
- By the Recursion theorem (317), there exists an index $e \in \mathbb{N}_0$ such that

  $$D_e = D_{f(e)} = \{e\}.$$

Recursion Theorem – Example

SGOTO program P_2 which halts with input $x_1 = 2$ but otherwise runs forever:

$$\begin{array}{ll} (0, \text{if } x_1 = 0, 5, 1) & \text{input } x_1 = 0, \text{ loop forever} \\ (1, x_1 \leftarrow x_1 - 1, 2) & \\ (2, \text{if } x_1 = 0, 5, 3) & \text{input } x_1 = 1, \text{ loop forever} \\ (3, x_1 \leftarrow x_1 - 1, 4) & \\ (4, \text{if } x_1 = 0, 6, 5) & \text{input } x_1 = 2, \text{ exit} \\ (5, x_1 \leftarrow x_1 + 1, 5) & \text{input } x_1 > 2, \text{ loop forever} \end{array} \tag{319}$$

Recursion Theorem (Kleene, 1938)

For each dyadic partial recursive function f, there is an index $e \in \mathbb{N}_0$ such that

$$\phi_e^{(1)} = f(e, \cdot). \tag{320}$$

Proof

Let f be a dyadic partial recursive function.

- By the smn theorem (247), there is a primitive recursive function $s : \mathbb{N}_0 \to \mathbb{N}_0$ such that for all $x \in \mathbb{N}_0$,

$$f(x, \cdot) = \phi_{s(x)}^{(1)}.$$

- By the recursion theorem (316), there is an index $e \in \mathbb{N}_0$ s.t.

$$\phi_e^{(1)} = \phi_{s(e)}^{(1)}.$$

- Hence, for all $y \in \mathbb{N}_0$, $f(e, y) = \phi_{s(e)}^{(1)}(y) = \phi_e^{(1)}(y)$. □

Recursion Theorem – Quines

Consider the projection function

$$\pi_1^{(2)} : \mathbb{N}_0^2 \to \mathbb{N}_0 : (x, y) \mapsto x.$$

By (320), there is an index $e \geq 0$ such that for all $y \in \mathbb{N}_0$,

$$\phi_e^{(1)}(y) = \pi_1^{(2)}(e, y) = e.$$

Thus the function $\phi_e^{(1)}$ outputs its own index when applied to any input value. Such indices are called *quines*.

In programming, a *quine* is a computer program which takes no input and produces a copy of its own source code as its only output (Douglas Hofstadter, 1979).

Quine in Language C

```
#include <stdio.h>
int main(){
char*s="#include <stdio.h>%cint main(){%cchar*s=%c%s%c;%cprintf(s,10,10,34,s,34,10);return 0;
printf(s,10,10,34,s,34,10);return 0;}
```

Right end:

```
...;%cprintf(s,10,10,34,s,34,10);return 0;}";
```

Recursion Theorem – Undecidability of H

Suppose the complement (289) of the halting problem

$$H = \left\{(x, y) \in \mathbb{N}_0^2 \mid y \in \operatorname{dom} \phi_x^{(1)}\right\}$$

is semidecidable.

- By (282), there is a dyadic partial recursive function f such that
$$\bar{H} = \operatorname{dom}(f).$$
- By (320), there is an index $e \geq 0$ such that
$$\phi_e^{(1)} = f(e, \cdot).$$
- For each $y \in \mathbb{N}_0$: $(e, y) \in \bar{H}$ iff $(e, y) \in \operatorname{dom}(f)$, or equivalently $f(e, y) = \phi_e^{(1)}(y)$ is defined, which means $(e, y) \in H$. Contradiction!

Thus $\bar{H}$ is not semidecidable and hence, by (288), H cannot be decidable.

Skills

- Computability of a function, e.g. (268).
- Decidability of a set, e.g. (276).
- Existential quantification of a set, e.g. Z_0.
- Recursive enumeration of a set, e.g. (313).
- Existence of fixed points, e.g. (318).

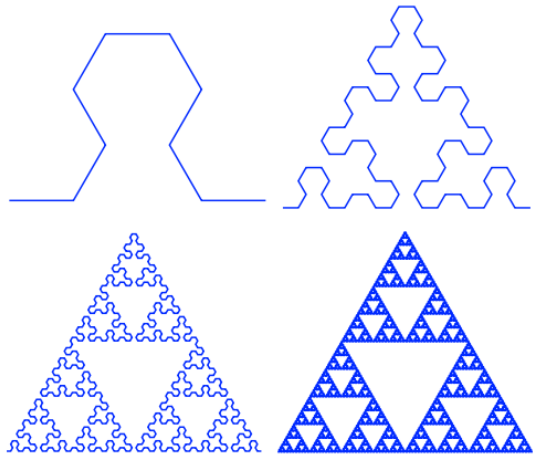
Part IX

Word Problems

Word Problems – Motivation

- Investigation of word problems in formal language theory.
- String rewriting systems have an undedicable word problem.
- Notion of string rewriting coincides with the presentation of semigroups.

String rewriting fractals (L-systems)



Word Problems – Contents

Reading: Zimmermann, Chapter 8

Semi-Thue Systems – Inside

- String rewriting systems are historically called semi-Thue systems.
- Introduced by Axel Thue (1914).
- Unsolvable word problem proved independently by Emil Post and A.A. Markov Jr. (1947).

Semi-Thue Systems

Semi-Thue system (STS) is a pair (Σ, R), where

- Σ is an alphabet (i.e., non-empty finite set) and
- $R \subseteq \Sigma^+ \times \Sigma^+$ is a dyadic relation on the non-empty strings over Σ,

$$\Sigma^+ = \Sigma^* \setminus \{\epsilon\}. \tag{321}$$

Each element $(u, v) \in R$ is called a *rewriting rule* and is written as

$$u \to_R v. \tag{322}$$

STS – One-Step Rewriting

Given STS (Σ, R).

One-step rewriting relation is the dyadic relation

$$\Rightarrow_R \subseteq \Sigma^+ \times \Sigma^+$$

such that for all s, t in Σ^+,

$$s \Rightarrow_R t \quad :\Longleftrightarrow \quad s = xuy,\ t = xvy,\ u \to_R v \text{ for some } x, y \in \Sigma^*,\ u, v \in \Sigma^+, \tag{323}$$

i.e., string s is rewritten by string t if substring u of s is replaced by substring v of t by the rewriting rule $u \to_R v$.

STS – Multi-Step Rewriting

Given STS (Σ, R).

Multi-steps rewriting or *derivation* is captured by the reflexive transitive closure of $\Rightarrow_R$ denoted by $\overset{*}{\Rightarrow}_R$.

For any strings $s, t \in \Sigma^+$, $s \overset{*}{\Rightarrow}_R t$ iff

- $s = t$ or
- there is a finite sequence $s_0, s_1, \ldots, s_m$ of elements in Σ^+ such that
  - $s = s_0$,
  - $s_i \Rightarrow_R s_{i+1}$ for $0 \leq i \leq m-1$, and
  - $s_m = t$.

Let $R \subseteq A \times A$ be a relation. Put $R^0 = \mathrm{id}_A$ and $R^{n+1} = R \circ R^n$ for all $n \geq 0$. Then $R^* = \bigcup_{n \geq 0} R^n$ is the *reflexive transitive closure* of R.

STS – Example

Consider the semi-Thue system (Σ, R) with alphabet $\Sigma = \{a, b\}$ and rule set

$$R = \{(ab, a), (ba, b)\}.$$

The derivation

$$\begin{array}{llll} baab & \Rightarrow_R & bab, & ba \to_R b, \\ & \Rightarrow_R & ba, & ab \to_R a, \\ & \Rightarrow_R & b, & ba \to_R b, \end{array}$$

shows that

$$baab \stackrel{*}{\Rightarrow}_R b.$$

STS – Word Problem

Given semi-Thue system (Σ, R) and two strings $s, t \in \Sigma^+$.

Decide whether or not

$$s \overset{*}{\Rightarrow}_R t. \tag{324}$$

This problem is undecidable.

Proof

Reduce the halting problem for SGOTO-2 programs to this word problem.

STS – SGOTO-2 Programs

Given SGOTO-2 program consisting of n instructions,

$$P = s_0; s_1; \ldots; s_{n-1}$$

- By definition, instruction s_i has the label i, $0 \leq i \leq n-1$.
- Assume the label n, called *exit label*, which does not address an instruction, is the only label signifying termination.

Define STS (Σ, R_P) that simulates the behavior of SGOTO-2 program P.

STS – Configurations

- *Configuration* of the two-register machine is given by a triple

$$(j, x, y), \tag{325}$$

where $j \geq 0$ is the actual instruction number of SGOTO program, x is the content of R_1 and y is the content of R_2.

- Register content can be encoded in unary format,

$$\overline{x} = \overbrace{LL\ldots L}^{x} \quad \text{and} \quad \overline{y} = \overbrace{LL\ldots L}^{y}. \tag{326}$$

- Now each configuration of the two-register machine can be written as a string

$$a\overline{x}j\overline{y}b \tag{327}$$

over the alphabet $\Sigma = \{a, b, 0, 1, 2, \ldots, n-1, n, L\}$.

STS – Encoding of Instructions

Configuration of two-register machine:

$$aL \dots LjL \dots Lb.$$

SGOTO-2 instructions are assigned appropriate rewriting rules:

GOTO-2 instructions	Rewriting rules
$(j, x_1 \leftarrow x_1 + 1, k)$	(j, Lk)
$(j, x_2 \leftarrow x_2 + 1, k)$	(j, kL)
$(j, x_1 \leftarrow x_1 - 1, k)$	$(Lj, k),\ (aj, ak)$
$(j, x_2 \leftarrow x_2 - 1, k)$	$(jL, k),\ (jb, kb)$
$(j, \texttt{if}\ \ x_1 = 0, k, l)$	$(Lj, Ll),\ (aj, ak)$
$(j, \texttt{if}\ \ x_2 = 0, k, l)$	$(jL, lL),\ (jb, kb)$

(328)

Moreover, there are two clean-up rewriting rules:

$$(Ln, n) \quad \text{and} \quad (anL, an). \tag{329}$$

STS – Example

SGOTO-2 program P given by

$$\begin{aligned}&(0, \texttt{if} \quad x_2 = 0, 3, 1)\\ &(1, x_1 \leftarrow x_1 + 1, 2)\\ &(2, x_2 \leftarrow x_2 - 1, 0)\end{aligned}$$

computes the addition of two numbers,

$$\|P\|_{2,1}(x, y) = x + y, \quad x, y \in \mathbb{N}_0.$$

Associated STS over $\Sigma = \{a, b, 0, 1, 2, 3, L\}$ has rewriting rules:

Instructions	Rewriting rules
$(0, \texttt{if} \quad x_2 = 0, 3, 1)$	$(0L, 1L),\ (0b, 3b),$
$(1, x_1 \leftarrow x_1 + 1, 2)$	$(1, L2),$
$(2, x_2 \leftarrow x_2 - 1, 0)$	$(2L, 0),\ (2b, 0b),$
	$(L3, 3),\ (a3L, a3).$

STS – Example (Cont'd)

Sample computation with initial register values $x_1 = 3$ and $x_2 = 2$:

SGOTO-2 instruction	Configuration	Derivation
$(0, \texttt{if}\ \ x_2 = 0, 3, 1)$	$(0, 3, 2)$	$aLLL0LLb$
$(1, x_1 \leftarrow x_1 + 1, 2)$	$(1, 3, 2)$	$aLLL1LLb$
$(2, x_2 \leftarrow x_2 - 1, 0)$	$(2, 4, 2)$	$aLLLL2LLb$
$(0, \texttt{if}\ \ x_2 = 0, 3, 1)$	$(0, 4, 1)$	$aLLLL0Lb$
$(1, x_1 \leftarrow x_1 + 1, 2)$	$(1, 4, 1)$	$aLLLL1Lb$
$(2, x_2 \leftarrow x_2 - 1, 0)$	$(2, 5, 1)$	$aLLLLL2Lb$
$(0, \texttt{if}\ \ x_2 = 0, 3, 1)$	$(0, 5, 0)$	$aLLLLL0b$
	$(3, 5, 0)$	$aLLLLL3b$
		$aLLLL3b$
		$aLLL3b$
		$aLL3b$
		$aL3b$
		$a3b$

STS – One-Step Reduction

Configuration (j, x, y) of SGOTO-2 program P with $j < n$ leads to successor configuration (k, u, v), i.e. by (125)

$$(j, x, y) \vdash_P (k, u, v), \tag{330}$$

iff STS (Σ, R_P) provides the one-step derivation

$$a\overline{x}j\overline{y}b \Rightarrow_{R_P} a\overline{u}k\overline{v}b. \tag{331}$$

There is no other one-step rewriting rule in the STS applicable to $a\overline{x}j\overline{y}b$.

STS – Multi-Step Reduction

Configuration (j, x, y) of SGOTO-2 program P with $j < n$ leads to configuration (k, u, v), i.e. by (125)

$$(j, x, y) \vdash_P^* (k, u, v), \tag{332}$$

iff STS (Σ, R_P) yields the derivation

$$a\overline{x}j\overline{y}b \stackrel{*}{\Rightarrow}_{R_P} a\overline{u}k\overline{v}b. \tag{333}$$

STS – Termination

- When the SGOTO-2 program terminates, its final configuration is of the form

$$(n, x, y) \tag{334}$$

where n is the exit label.

- This corresponds in the STS to the word

$$a\overline{x}n\overline{y}b \tag{335}$$

which can be further rewritten according to the clean-up rules (329):

$$a\overline{x}n\overline{y}b \stackrel{*}{\Rightarrow}_{R_P} anb. \tag{336}$$

STS – Main Theorem

The word problem for semi-Thue systems is undecidable.

Proof.

- SGOTO-2 program P started in configuration $(0, x, y)$ halts iff in the corresponding STS

$$a\overline{x}0\overline{y}b \overset{*}{\Rightarrow}_{R_P} anb. \tag{337}$$

- This provides an effective reduction of the halting problem for SGOTO-2 programs to the word problem for semi-Thue systems (324).
- Halting problem for SGOTO-2 programs is undecidable implies word problem for STS is undecidable.

Thue Systems

- Thue systems form a subclass of semi-Thue systems.
- Word problem for Thue systems is still undecidable.
- Thue systems correspond to presentations of semigroups.

Thue Systems

Thue system (TS) is an STS (Σ, R) whose relation R is symmetric, i.e.,

$$u \to_R v \quad \Longrightarrow \quad v \to_R u. \tag{338}$$

In the TS (Σ, R), the reflexive transitive closure $\overset{*}{\Rightarrow}_R$ of the one-step rewriting relation $\Rightarrow_R$ is also symmetric and hence an *equivalence relation* on Σ^+.

In the TS (Σ, R), we have for all strings $s, t \in \Sigma^*$,

$$s \Rightarrow_R^* t \quad \Longleftrightarrow \quad t \Rightarrow_R^* s. \tag{339}$$

Thue Systems – Word Problem

Given Thue system (Σ, R) and two strings $s, t \in \Sigma^+$.

Decide whether or not

$$s \overset{*}{\Rightarrow}_R t. \tag{340}$$

This problem is undecidable.

Proof

Using Post's Lemma.

Thue Systems – Symmetric Closure

Given STS (Σ, R).

- *Symmetric closure* of rewriting relation R is the symmetric relation

$$R^{(s)} = R \cup R^{-1}, \tag{341}$$

  where

$$R^{-1} = \{(v, u) \mid (u, v) \in R\} \tag{342}$$

  is the *inverse relation* of R.

- $R^{(s)}$ is the smallest symmetric relation containing R and $(\Sigma, R^{(s)})$ is a Thue system.

- Relation $\overset{*}{\Rightarrow}_{R^{(s)}}$ is the reflexive transitive and symmetric closure of $\Rightarrow_R$ and hence an equivalence relation on Σ^+.

Thue Systems – Multi-Step Reduction

For any strings $s, t \in \Sigma^+$, $s \overset{*}{\Rightarrow}_{R^{(s)}} t$ iff

- $s = t$ or
- there is a finite sequence $s_0, s_1, \ldots, s_m$ of elements in Σ^+ such that
  - $s = s_0$,
  - $s_i \Rightarrow_R s_{i+1}$ or $s_{i+1} \Rightarrow_R s_i$ for $0 \leq i \leq m-1$, and
  - $s_m = t$.

Thue Systems – Multi-Step Reduction

Derivations in STS (Σ, R) and associated TS $(\Sigma, R^{(s)})$ are related as follows:

$$s \Rightarrow_R^* t \quad \Longrightarrow \quad s \Rightarrow_{R^{(s)}}^* t, \quad s, t \in \Sigma^+, \tag{343}$$

since $R \subseteq R^{(s)}$.

But if in TS $(\Sigma, R^{(s)})$

$$s \overset{*}{\Rightarrow}_{R^{(s)}} t,$$

holds, then

$$\text{neither } s \overset{*}{\Rightarrow}_R t \text{ nor } t \overset{*}{\Rightarrow}_R s$$

need to be valid in the STS (Σ, R).

Thue Systems – Example

Consider STS (Σ, R) with alphabet $\Sigma = \{a, b\}$ and rule set

$$R = \{(ab, a), (ba, b)\}.$$

- Associated TS $(\Sigma, R^{(s)})$ has rule set

$$R^{(s)} = R \cup \{(a, ab), (b, ba)\}.$$

- In the STS, rewriting is strictly antitone leading to smaller strings, i.e., if $u \overset{*}{\Rightarrow}_R v$ and $u \neq v$, then $|u| > |v|$, e.g.,

$$baab \Rightarrow_R bab \Rightarrow_R ba \Rightarrow_R b.$$

- In the TS, $baab \Rightarrow_{R^{(s)}} bab \Rightarrow_{R^{(s)}} baba$, but in the STS

$$\text{neither} \quad baab \overset{*}{\Rightarrow}_R baba \quad \text{nor} \quad baba \overset{*}{\Rightarrow}_R baab.$$

Post's Lemma

Given SGOTO-2 program P with n instructions and exit label n.

Let (Σ, R_P) be the corresponding semi-Thue system, and $(\Sigma, R_P^{(s)})$ be the associated Thue system.

For each configuration (j, x, y) of P,

$$a\overline{x}j\overline{y}b \stackrel{*}{\Rightarrow}_{R_P} anb \quad \Longleftrightarrow \quad a\overline{x}j\overline{y}b \stackrel{*}{\Rightarrow}_{R_P^{(s)}} anb. \tag{344}$$

Compare (344) with the general case (343).

Proof

- By (343), the direction from left-to-right is clear:

$$a\overline{x}j\overline{y}b \overset{*}{\Rightarrow}_{R_P} anb \quad \Longrightarrow \quad a\overline{x}j\overline{y}b \overset{*}{\Rightarrow}_{R_P^{(s)}} anb.$$

- Conversely, let (j, x, y) be a configuration of P and

$$a\overline{x}j\overline{y}b \overset{*}{\Rightarrow}_{R_P^{(s)}} anb.$$

Then there is a rewriting sequence in the TS:

$$w_0 = a\overline{x}j\overline{y}b \Rightarrow_{R_P^{(s)}} w_1 \Rightarrow_{R_P^{(s)}} \ldots \Rightarrow_{R_P^{(s)}} w_q = anb. \qquad (345)$$

Assume that the length q of this derivation is minimal.

Proof (Cont'd)

Suppose derivation (345) cannot be established by the STS.

- Then the sequence (345) contains a reverse rewriting step

$$w_p \Leftarrow_{R_P} w_{p+1}, \quad 0 \leq p \leq q-1.$$

  Choose the index p maximal with this property.

- Since there is no rewriting rule applicable to $w_q = anb$, we have $p+1 < q$. This leads to the following situation:

$$w_p \Leftarrow_{R_P} w_{p+1} \Rightarrow_{R_P} w_{p+2}. \tag{346}$$

  But by construction, there is at most one rule applicable to the string $w_{p+1} = a\overline{x}j\overline{y}b$ given by instruction s_j and configuration (j, x, y).

- Thus the words w_p and w_{p+2} must be identical and hence the derivation (345) can be shortened by deleting the string w_{p+1} contradicting the (minimality) assumption. □

Semigroups

- Word problems are also encountered in abstract algebra.
- Thue systems give rise to semigroups in a natural way.
- Word problem for semigroups (and also groups) is undecidable.

Semigroups

A *semigroup* is a pair $(S, \cdot)$, where

- S is a non-empty set and
- $\cdot : S \times S \to S$ is a dyadic operation such that the associative property holds, i.e., for all $a, b, c \in S$,

$$(a \cdot b) \cdot c = a \cdot (b \cdot c). \tag{347}$$

- A semigroup $(S, \cdot)$ is *commutative* if for all $a, b \in S$,

$$a \cdot b = b \cdot a. \tag{348}$$

Reading: Zimmermann, Appendix A

Example

Let Σ be an alphabet. The pair $(\Sigma^+, \cdot)$ is the semigroup of all non-empty words over Σ and $\cdot$ is the concatenation of words.

Equivalence Relation

Let $(\Sigma, R^{(s)})$ be a Thue system.

- Rewriting relation $\overset{*}{\Rightarrow}_{R^{(s)}}$ on Σ^+ is an equivalence relation.
- Two strings $s, t \in \Sigma^+$ are *equivalent* if

$$s \overset{*}{\Rightarrow}_{R^{(s)}} t. \tag{349}$$

- Equivalence class of string $s \in \Sigma^+$ is the set of all strings in Σ^+ that can be derived from s by a finite number of rewriting steps, i.e.,

$$[s] = \{t \in \Sigma^+ \mid s \overset{*}{\Rightarrow}_{R^{(s)}} t\}. \tag{350}$$

- Note that for all strings $s, t \in \Sigma^+$, we have

$$[s] = [t] \quad \Longleftrightarrow \quad s \overset{*}{\Rightarrow}_{R^{(s)}} t. \tag{351}$$

Quotient Set

Let $(\Sigma, R^{(s)})$ be a Thue system.

- Quotient set of the equivalence relation is given by the set of all equivalence classes,

$$S = S(\Sigma, R^{(s)}) = \{[s] \mid s \in \Sigma^+\}. \tag{352}$$

Semigroups

Let $(\Sigma, R^{(s)})$ be a Thue system.
Quotient set $S = S(\Sigma, R^{(s)})$ forms a semigroup under the operation

$$[s] \cdot [t] = [st], \quad s, t \in \Sigma^+. \tag{353}$$

Proof

- Operation is well-defined.

  Indeed, let $[s] = [s']$ and $[t] = [t']$, where $s, s', t, t' \in \Sigma^+$.

  Then by (351),

  $$s \overset{*}{\Rightarrow}_{R^{(s)}} s' \text{ and } t \overset{*}{\Rightarrow}_{R^{(s)}} t'.$$

  Thus

  $$st \overset{*}{\Rightarrow}_{R^{(s)}} s't \overset{*}{\Rightarrow}_{R^{(s)}} s't'$$

  and hence by (351),

  $$[st] = [s't'].$$

Proof (Cont'd)

- Operation is associative.

  Indeed, let $r, s, t \in \Sigma^+$. Then

$$\begin{aligned}
[r] \cdot ([s] \cdot [t]) &= [r] \cdot [st], \text{ by (353)} \\
&= [r(st)], \text{ by (353)} \\
&= [(rs)t], \ \Sigma^+ \text{ is associative} \\
&= [rs] \cdot [t], \text{ by (353)} \\
&= ([r] \cdot [s]) \cdot [t], \text{ by (353)}.
\end{aligned}$$

Semigroups – Presentation

Semigroup $S = S(\Sigma, R^{(s)})$ has a *presentation* in terms of generators and relations,

$$S = \langle a_1, \ldots, a_n \mid u_1 = v_1, \ldots, u_m = v_m \rangle, \tag{354}$$

where

$$\Sigma = \{a_1, \ldots, a_n\}$$

is the alphabet and

$$R = \{(u_1, v_1), \ldots, (u_m, v_m)\}$$

is the rule set.

Reading: Zimmermann, Appendix A

Presentation – Example

Given Thue system $(\Sigma, R^{(s)})$ with alphabet $\Sigma = \{a, b\}$ and rule set $R^{(s)}$, where

$$R = \{(ab, ba), (a, aa), (b, bbb)\}$$

and so

$$R^{(s)} = \{(ab, ba), (a, aa), (b, bbb), (ba, ab), (aa, a), (bbb, b)\}.$$

- Semigroup $S = S(\Sigma, R^{(s)})$ has the presentation

$$S = \langle a, b \mid ab = ba, a = aa, b = bbb \rangle.$$

- Each word of Σ^+ has the form

$$a^{i_1} b^{j_1} \dots a^{i_k} b^{j_k},$$

where $k \geq 1$ and at least one of the numbers $i_1, \dots, i_k, j_1, \dots, j_k \geq 0$ is non-zero.

Example (Cont'd)

As seen, each word of Σ^+ has the form

$$w = a^{i_1} b^{j_1} \dots a^{i_k} b^{j_k} \neq \epsilon.$$

- In view of the rule (ba, ab),

$$w = a^i b^j$$

where $i, j \geq 0$ and i, j are not both zero.

- In view of the rules (aa, a) and (bbb, b),

$$w = a^i b^j$$

where $0 \leq i \leq 1$, $0 \leq j \leq 2$ and i, j are not both zero.

Example (Cont'd)

As seen, each word of Σ^+ can be transformed to

$$w = a^i b^j$$

where $0 \leq i \leq 1$, $0 \leq j \leq 2$ and i, j are not both zero.

- This leads to the semigroup

$$S = \{[a], [ab], [abb], [b], [bb]\}.$$

- Multiplication table of S:

$\cdot$	$[a]$	$[ab]$	$[abb]$	$[b]$	$[bb]$
$[a]$	$[a]$	$[ab]$	$[abb]$	$[ab]$	$[abb]$
$[ab]$	$[ab]$	$[abb]$	$[ab]$	$[abb]$	$[ab]$
$[abb]$	$[abb]$	$[ab]$	$[abb]$	$[ab]$	$[abb]$
$[b]$	$[ab]$	$[abb]$	$[ab]$	$[bb]$	$[b]$
$[bb]$	$[abb]$	$[ab]$	$[abb]$	$[b]$	$[bb]$

Example (Cont'd)

Multiplication in semigroup S:

- $[ab] \cdot [ab] = [abb]$, since
  - $[ab] \cdot [ab] = [ab \cdot ab] = [abab]$ and
  - $[abab] = [abb]$ as

$$abab \Rightarrow_{R^{(s)}} aabb \Rightarrow_{R^{(s)}} abb.$$

- $[ab] \cdot [abb] = [ab]$, since
  - $[ab] \cdot [abb] = [ab \cdot abb] = [ababb]$ and
  - $[ababb] = [ab]$ as

$$ababb \Rightarrow_{R^{(s)}} aabbb \Rightarrow_{R^{(s)}} abbb \Rightarrow_{R^{(s)}} ab.$$

Multiplication is commutative because of the rules $ab \rightarrow_R ba$ and $ba \rightarrow_R ab$.

Semigroups – Word Problem

Given semigroup $(S, \cdot)$ and two elements $s, t \in S$.
Decide whether or not

$$s = t. \tag{355}$$

This problem is undecidable.

Let $(\Sigma, R^{(s)})$ be a Thue System, $S = S(\Sigma, R^{(s)})$ be the corresponding semigroup and $s, t \in \Sigma^+$.
Decide whether or not

$$[s] = [t]. \tag{356}$$

Proof.

By (351), the word problem for Thue systems can be reduced to word problem for semigroups. But the word problem for Thue systems is undecidable and so is the word problem for semigroups. □

Skills

- Multi-step rewriting in semi-Thue system.
- Representation of computation of SGOTO program by derivation of associated semi-Thue system.
- Multi-step rewriting in Thue system.
- Symmetrizing semi-Thue system to provide Thue system.
- Representation of Thue system as semigroup including multiplication table.