Turing 機械の定義と停止問題
Turing Machine and the Halting Problem

y.*

2018 年 1 月 1 日

最終更新日: 2018 年 1 月 5 日

概要

決定問題とは与えられた入力に対して Yes/No で答える種類の計算課題である．本稿では Turing 機械と呼ばれる計算モデルを導入し，どんなアルゴリズムをもってしても解くことができない決定問題，すなわち決定不能問題が存在することを示す．また決定不能問題の最も重要な例として停止問題を取り上げる．

Keywords: 決定問題 (decision problem), 決定不能問題 (undecidable problem), Turing 機械 (Turing machine), Church-Turing の提唱 (Church-Turing thesis), 停止問題 (halting problem).

1 決定問題

決定問題とは与えられた入力が所与の条件を満たすかどうかを Yes/No で答える種類の問題である．決定問題の正確な定義は後で述べることにして，ここでは先に例を見ることにしよう．意味のある決定問題の中で最も簡単なものは次の素数判定問題であろう．

問題 1.1 (素数判定問題 (primality testing; PRIMES**)).**

Input: 正の整数 n

Question: n は素数か？

この問題は簡単に “解ける”:

解答 1.2. $n = 1$ なら No，$n = 2$ なら Yes，$n > 2$ のときは $2, 3, \ldots, n-1$ で順番に割ってみて，ひとつでも割り切れるものがあれば No，そうでないときは Yes と答えればよい．

解答 1.2 を観察してみると，素数判定問題 PRIMES が「20180101 は素数か？」といった問題とは異なった種類の問題であることがわかる．「20180101 は素数か？」という問題に対する解答は Yes または No のどちらかでなければならないが (実際には No である)，一方で解答 1.2 では与えられた正整数が素数か否かを正しく判定するひとつの「方法」を与えている．すなわち，決定問題を解くということは，全ての入力に対して正しい答えを返すような方法を与えることに他ならない．ここでいう「方法」は，現代的には**アルゴリズム** (algorithm) ないしは**プログラム** (program) と言い換えることができる．

決定問題 A の全ての入力に対して正しく答えるアルゴリズムが存在するとき A は**決定可能** (decidable) で

* http://iso.2022.jp/

あるといい，そうでないとき，すなわち A の全ての入力に対して正しく答えるようなアルゴリズムが存在しないとき A は**決定不能** (undecidable) であるという．

「アルゴリズムとは何か」ということについては次節で述べることにして，ここでは決定問題の感じをつかんでもらうためにもう少し例を挙げることにする．

問題 1.3 (部分和問題 (subset sum problem)).

Input: 自然数の有限集合 $S \subseteq \mathbb{N}$ と自然数 $x \in \mathbb{N}$ の組[*1]

Question: $\sum_{n \in T} n = x$ なる部分集合 $T \subseteq S$ が存在するか？

例 1.4. 入力が $S = \{1, 3, 9, 27\}, x = 30$ のとき，$T = \{3, 27\}$ とすれば $\sum_{n \in T} n = 3 + 27 = 30 = x$ となるので答えは Yes である．一方，入力が $S = \{2, 5\}, x = 3$ のとき，S の全ての部分集合 $T = \emptyset, \{2\}, \{5\}, \{2, 5\}$ に対してその和はそれぞれ $0, 2, 5, 7$ であり，どれも x と異なるので答えは No である．

問題 1.5 (Hamilton 閉路問題 (Hamilton circuit problem)).

Input: 有限無向グラフ G

Question: G は Hamilton 閉路 (全ての頂点を一度だけ通るような閉路) を持つか？

例 1.6. 例えば，入力されたグラフ G が図 1a のものであるとすると，図 1b のような Hamilton 閉路を持つので答えは Yes である．

(a) 入力のグラフ G　　(b) G の Hamilton 閉路

図 1: Hamilton 閉路問題の入力例

上記 2 つの決定問題が決定可能であることを確認することは易しい．実際，部分和問題 1.3 では S の $2^{|S|}$ 個全ての部分集合それぞれについて和を計算して x と等しくなるかどうかを確かめればよいし，Hamilton 閉路問題 1.5 では G の頂点の全体の集合 V の $|V|!$ 通り全ての並べ方について，実際に閉路になっているかどうかを確かめればよい．

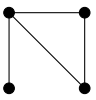
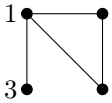
さて，決定問題の定義をいくつか見たところで，そろそろ決定問題の数学的な定義を与えたい．どのように定義すればよいだろうか？ 全ての入力に対して Yes または No という答えが決まるわけだから，決定問題とは入力全体の集合 I から $\{\text{Yes}, \text{No}\}$ への関数である，と言いたいところである．しかし，上で挙げた 3 つの問題を見れば明らかなように，入力の範囲 I は決定問題ごとに全然違う集合になる．入力は正整数であったり，自然数の有限集合であったり，そればかりかグラフであったりする．このように入力が“なんでもあり”になってしまうと，後でアルゴリズムを定義するときに少し困ったことになる．与えられた入力に対する“計算”を定義するのが非常に難しくなってしまうのである．したがって，多種多様な入力を表現できる柔軟性を持ちながら，なおかつ理論的な取り扱いが簡単な対象を見つけたい．このような性質を備えた数学的対象のひとつとして文字列がある．

定義 1.7 (文字列). Σ を有限集合とする．Σ の元の有限列を Σ 上の**文字列** (string) といい，Σ 上の文字列

[*1] 本稿では $\mathbb{N}$ は非負整数全体の集合を表すものとする．すなわち 0 を元として含む．

全体の集合を Σ^* で表す．例えば，$\Sigma = \{\mathtt{a}, \mathtt{b}\}$ のとき $\Sigma^* = \{\varepsilon, \mathtt{a}, \mathtt{b}, \mathtt{aa}, \mathtt{ab}, \mathtt{ba}, \mathtt{bb}, \mathtt{aaa}, \dots\}$ である (ここで ε は長さが 0 の文字列を表す)．また，Σ を**アルファベット** (alphabet) と呼ぶ．[*2]

例 1.8. 色々な数学的対象を文字列に変換する方法の例を以下に示す．これらはあくまでも一例であり，自然な方法であれば他のどのような方法を採用しても本質的には同じである．

自然数	314	$\mapsto$	`314`
有理数	$-\dfrac{22}{7}$	$\mapsto$	`-22/7`
有限集合	$\{1,2,3\}$	$\mapsto$	`{1,2,3}`
行列	$\begin{pmatrix} 0 & -1 \\ 1 & 1 \end{pmatrix}$	$\mapsto$	`((0,-1),(1,1))`
有限グラフ		$\mapsto$	`({1,2,3,4},{{1,2},{1,3},{1,4},{2,4}})`
			(次のように頂点に番号を付けた: )

ここで作った文字列はどれも，アルファベット $\Sigma := \left\{\mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}, \mathtt{4}, \mathtt{5}, \mathtt{6}, \mathtt{7}, \mathtt{8}, \mathtt{9}, \mathtt{-}, \mathtt{/}, \mathtt{,}, \mathtt{\{}, \mathtt{\}}, \mathtt{(}, \mathtt{)}\right\}$ 上の文字列になっている．

以上を踏まえて，本稿では次の原理を採用することにする．

原理 1.9. 計算の対象となるような有限的なデータは全て (適当なアルファベット上の) 文字列で表すことができる．

この原理 1.9 はあくまで経験則であるが，有限集合や有限列といったものが全て文字列で表現可能であることを考えると疑う余地はないだろう．

以降，何らかの数学的対象 X を変換してできる文字列を $\langle X \rangle$ で表すことにする．特に，文字列の組 (u, v) をひとつの文字列にしたものを $\langle u, v \rangle$ で表す．

これでようやく決定問題の数学的定義を述べることができる．

定義 1.10 (決定問題). **決定問題** (decision problem) とは，あるアルファベット Σ 上の文字列に対して Yes または No を返す関数 $f\colon \Sigma^* \to \{\text{Yes}, \text{No}\}$ のことをいう．あるいは，同じことだが，答えが Yes であるような文字列全体の集合 $\{\, w \in \Sigma^* \mid f(w) = \text{Yes} \,\}$ のことも決定問題という．[*3]

注意 1.11. 入力が有限グラフの場合など，グラフを表していない文字列 (例えば `,1},}}2{` のような) もあるのだから，決定問題を関数とみなしたとき定義域を文字列全体にするのはまずいのではないかと思うかもしれない．しかし例えば Hamilton 閉路問題は正確には

$$\{\, w \in \Sigma^* \mid w \text{ はあるグラフ } G \text{ を表しており，かつ } G \text{ は Hamilton 閉路を持つ} \,\}$$

[*2] Σ の元ではなく，集合 Σ そのものをアルファベットと呼ぶ．

[*3] 集合 Σ^* の部分集合全体と Σ^* 上の**特性関数** (characteristic function) 全体との間には一対一の対応があったことを思い出そう．

という集合として表現すればよいのである．実際には「文字列がグラフを表しているかどうか」などは機械的に判定可能な条件なので本稿の範囲で問題が生じることはない．[*4]

本節では決定問題の数学的定式化を行うために文字列を導入した．次節では決定問題の決定可能性を正確に定義するためにアルゴリズムの数学的な定義を与える．

2 Turing 機械

我々が主として扱いたい対象は決定不能問題である．ある問題が決定不能であることを示すためには，いかなるアルゴリズムをもってしても計算不可能であることを示す必要がある．その証明を数学的に厳密なものとするためには，アルゴリズム，ないしはプログラムの数学的な定義が必要不可欠である．定義なしに厳密な証明をすることはできない．

プログラムは入力を受け取り，予め定められたプロセスに従って計算を進め，計算結果を出力する．だから，プログラムは入力を計算結果に対応させる関数とみなすことができる……ように思われる．ところが，プログラムはしばしば無限ループに陥り，計算が永遠に終わらず結果が出力されないことがある．このように値が定義されないかもしれない“関数”の概念を数学的に捉えるために，次の部分関数の定義を導入する．

定義 2.1 (部分関数). A, B を集合とする．A の部分集合から B への関数 f を A から B への**部分関数** (partial function) といいう．このことを本稿では $f\colon A \rightharpoonup B$ と書く．[*5] $x \in A$ に対し $f(x)$ が定義されていること (つまり x が f の定義域に属すこと) を $f(x)\downarrow$ ($f(x)$ converges と読む) で表し，$f(x)$ が定義されていないことを $f(x)\uparrow$ ($f(x)$ diverges と読む) で表す．さらに，$f(x)\downarrow$ かつ $f(x) = y$ であることを $f(x)\downarrow = y$ で表す．

部分関数 $f\colon A \rightharpoonup B$ の定義域が A 全体であるとき，すなわち任意の $x \in A$ に対し $f(x)\downarrow$ となるとき，f は**全域的** (total) であるという．全域的な部分関数を**全域関数** (total function) という．

2 つの部分関数 $f, g\colon A \rightharpoonup B$ と $x \in A$ に対し，両辺の値が定義されているかどうかも含めて等しいことを $f(x) \simeq g(x)$ で表す (すなわち，一方が未定義なら他方も未定義であり，一方が定義されているときは他方も定義されていて値が等しい)．

例 2.2. 小学校の算数では小さい数から大きい数を引くことはできないと教わったかもしれない．これは部分関数の視点では

$$f(x, y) = \begin{cases} x - y & \text{if } x \geq y \\ \text{未定義} & \text{if } x < y \end{cases}$$

という部分関数 $f\colon \mathbb{N} \times \mathbb{N} \rightharpoonup \mathbb{N}$ を計算することに他ならない．このとき例えば $f(5, 4)\downarrow = 1$ や $f(2, 3)\uparrow$ が成り立っている．

さて，用語の準備が整ったところでいよいよアルゴリズムの数学的な定義 (のひとつ) を与えよう．計算モデルとして C 言語や Python などの既に実在するプログラミング言語 (のサブセット) を採用するということも考えられるが，このようなモデルは決定不能性の証明のためには複雑すぎる．決定不能性を示すという目的

[*4] 実際には，入力が無曖昧な文脈自由文法全体である場合など，入力の範囲が決定不能になるような問題も存在する．しかしこの種の問題を正確に述べるためには **computably inseparable** (**recursively inseparable** ともいう) などの用語を導入する必要があり，本稿の範囲を超えてしまうためここでは扱わない．

[*5] 部分関数の記法は文献によってんでばらばらで，特に決まったものはないようである．

のためには可能なアルゴリズムのパターンが少なければ少ないほど都合がよい．そこで，本稿では 1936 年に Alan M. Turing が導入した Turing 機械という計算モデルを用いる．[*6]

Turing 機械は 1 本の無限に長い**テープ** (tape) と左右に動くことのできる**ヘッド** (head) からなる (図 2)．テープは正方形のマス目 (セル) に区切られており，1 つのマスには 1 つの文字が入る．ヘッドはテープの 1 つマスを見ており，左右に 1 マスずつ動くことができる．Turing 機械は内部状態を 1 つだけ保持することができ，機械は一定の規則 (遷移関数) に基づいてテープ上の文字を書き換えたりヘッドを左右に動かしたりして計算を進める．

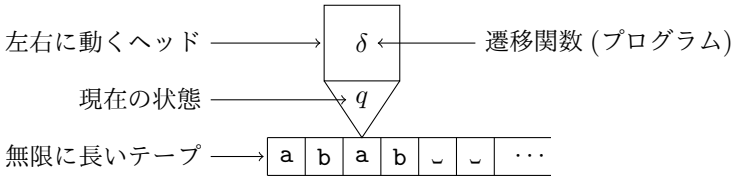


図 2: Turing 機械

Turing 機械は形式的には以下のように定義される．

定義 2.3 (Turing 機械). Turing 機械 (Turing machine) M とは以下のデータからなる 7 つ組 $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ である:

1. **状態** (state) の有限集合 Q,
2. **入力アルファベット** (input alphabet) Σ と呼ばれる，**空白記号** (blank symbol) ␣ を含まない有限集合，
3. **テープアルファベット** (tape alphabet) Γ と呼ばれる，空白記号 ␣ と Σ を含む有限集合，
4. **遷移関数** (transition function) $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$, [*7]
5. **開始状態** (initial state) $q_0 \in Q$,
6. **受理状態** (accepting state) $q_{\text{accept}} \in Q$,
7. **拒否状態** (rejecting state) $q_{\text{reject}} \in Q\,(q_{\text{reject}} \neq q_{\text{accept}})$.

Turing 機械 M は入力文字列 $w \in \Sigma^*$ に対して次のように計算を行う．これにより部分関数 $M\colon \Sigma^* \rightharpoonup \{\textsc{Yes}, \textsc{No}\}$ が定まる．[*8]

1. $w = w_1 w_2 \cdots w_n$ $(w_i \in \Sigma)$ とする．テープの左端から n マス目まで順に $w_1, w_2, \ldots, w_n$ を書き込み，それ以外の全てのマスに空白記号␣を書き込む．ヘッドはテープの左端のマスを見ている状態にし，機械の状態を開始状態 $q_0 \in Q$ に設定する．
2. 機械の現在の状態が $q \in Q$ でヘッドが見ている文字が $a \in \Gamma$ で，さらに $\delta(q, a) = (r, b, D)$ であるとする．このとき機械の現在の状態を q から r に変更し，ヘッドが見ているマスの文字を a から b に書き換え，$D = \mathrm{L}$ ならヘッドを左に，[*9] $D = \mathrm{R}$ なら右に 1 マスだけ移動させる．機械の現在の状態が $q_{\text{accept}}, q_{\text{reject}}$ でない限りこれを繰り返す．

[*6] 念のため言っておくと，Turing 本人は Turing 機械ではなく automatic machine (*a*-machine) と呼んでいた．

[*7] 正確には $\delta\colon (Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}) \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ などとするべきだが，表記の煩雑さを避けるためこのようにした．

[*8] 本稿では Turing 機械 M と，M から定まる部分関数 $\Sigma^* \rightharpoonup \{\textsc{Yes}, \textsc{No}\}$ を区別せずに同じ記号 M で表す．このことは本稿の範囲では問題を生じない．

[*9] ただし，ヘッドがテープの左端を見ていてかつ $D = \mathrm{L}$ のときはその場に留まるものとしておく．

3. 機械の現在の状態が q_{accept} に到達したら直ちに計算を停止して $M(w)\downarrow$ = 受理 = Yes = 1 と定める．[*10]
4. 機械の現在の状態が q_{reject} に到達したら直ちに計算を停止して $M(w)\downarrow$ = 拒否 = No = 0 と定める．
5. 機械が永遠に $q_{\text{accept}}, q_{\text{reject}}$ のどちらにも到達しない場合は $M(w)\uparrow$ と定める．

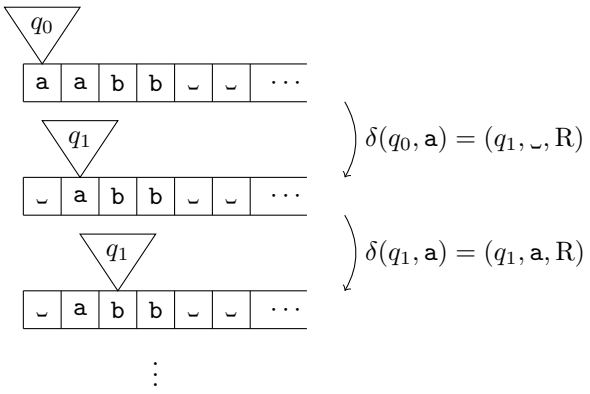
Turing 機械の計算の様子を簡単な例で確かめてみよう．

例 2.4 (Turing 機械の例). $Q := \{q_0, q_1, q_2, q_3, q_{\text{accept}}, q_{\text{reject}}\}, \Sigma := \{\mathtt{a}, \mathtt{b}\}, \Gamma := \{\mathtt{a}, \mathtt{b}, \mathtt{␣}\}$ とし，遷移関数 $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ を以下の表のように定義する．[*11]

	a			b			␣			
q_0	q_1	␣	R	q_{reject}	b	R	q_{accept}	␣	R	(左端の a を消す)
q_1	q_1	a	R	q_1	b	R	q_2	␣	L	(右端まで行く)
q_2	q_{reject}	a	R	q_3	␣	L	q_{reject}	␣	R	(右端の b を消す)
q_3	q_3	a	L	q_3	b	L	q_0	␣	R	(左端まで行く)

これにより定まる Turing 機械 M は決定問題 $\{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\} \subseteq \Sigma^*$ (ここで $\mathtt{a}^n := \underbrace{\mathtt{a}\cdots\mathtt{a}}_{n}$ である) を判定する機械であり，特に M は全域関数を計算する．

入力 aabb に対する M の動作を見てみよう．テープの左端から $\mathtt{a}, \mathtt{a}, \mathtt{b}, \mathtt{b}$ を書き込んだ状態から計算を開始し，遷移関数 δ に従って計算を進める．



しかしこれを続けると紙面を著しく消費するので (それに描くのが面倒だから)，計算の各時点での Turing 機械の状態，テープの内容，ヘッドの位置をひとまとめにした情報 (これを**計算状況** (configuration) と呼ぶ) を ($Q \cup \Gamma$ 上の) 文字列で表すことにする．状態の右隣に書いてある文字がヘッドが現在見ている文字である．

$$
\begin{aligned}
&q_0\mathtt{aabb}\\
&\mathtt{␣}q_1\mathtt{abb}\\
&\mathtt{␣a}q_1\mathtt{bb}\\
&\mathtt{␣ab}q_1\mathtt{b}\\
&\mathtt{␣abb}q_1\mathtt{␣}\\
&\mathtt{␣ab}q_2\mathtt{b␣}\\
&\mathtt{␣a}q_3\mathtt{b␣␣}
\end{aligned}
$$

[*10] 本稿では受理と拒否，Yes と No，1 と 0 をそれぞれ同一視する．
[*11] $q_{\text{accept}}, q_{\text{reject}}$ に対する値は使われることはなく，何であっても一緒なので載せていない．

$$\llcorner q_3\mathtt{ab}\lrcorner\lrcorner$$
$$q_3\llcorner\mathtt{ab}\lrcorner\lrcorner$$
$$\llcorner q_0\mathtt{ab}\lrcorner\lrcorner$$
$$\lrcorner\lrcorner q_1\mathtt{b}\lrcorner\lrcorner$$
$$\lrcorner\lrcorner\mathtt{b}q_1\lrcorner\lrcorner$$
$$\lrcorner\lrcorner q_2\mathtt{b}\lrcorner\lrcorner$$
$$\llcorner q_3\lrcorner\lrcorner\lrcorner\lrcorner$$
$$\lrcorner\lrcorner q_0\lrcorner\lrcorner\lrcorner$$
$$\lrcorner\lrcorner\lrcorner q_{\mathrm{accept}}\lrcorner\lrcorner$$

これより $M(\mathtt{aabb})\downarrow = 1$ である．同様にして計算することにより $\varepsilon, \mathtt{ab}, \mathtt{aaabbb}$ などは受理され，$\mathtt{a}, \mathtt{b}, \mathtt{aab}, \mathtt{abb}$ などは拒否されることがわかる．

Turing 機械の計算能力は一見してそれほど高くないように感じられるかもしれない．しかし，実際には現実のプログラミング言語で遂行できるようないかなる計算も実行できるだけの能力を備えている．このことを主張するのが次の **Church-Turing の提唱** (Church-Turing thesis) である．

提唱 2.5 (Church-Turing の提唱). $f\colon \Sigma^* \rightharpoonup \{0,1\}$ を部分関数とするとき，

$$f \text{ が計算可能} \iff f \text{ を計算する Turing 機械が存在する.}$$

Church-Turing の提唱 2.5 の左辺は数学的な主張ではないことに注意せよ．Church-Turing の提唱はあらゆる計算が Turing 機械によって模倣できるという非形式的な主張であると思うこともできるが，本稿ではむしろ計算可能性を Turing 機械の存在で定義しているのだと思うことにする．

Church-Turing の提唱の妥当性を支持する根拠は色々ある．まず Turing 機械の概念が人間の紙とペンを使った計算の分析から得られたということ，それから Turing 機械が他の様々な計算モデル (ラムダ計算，再帰関数，レジスター機械など) と計算能力が等価であるということなどである．

Church-Turing の提唱の $\Longrightarrow$ を信じるならば，アルゴリズムを記述する際に Turing 機械の遷移関数を厳密に書き下す必要はないことになる．つまり，直観的に計算可能であることがわかれば，そのような計算を行う Turing 機械が存在するとしてよいわけである．本稿でもこの立場をとることにし，以降は遷移関数を直接記述することはしない．

Turing 機械 M は定義から有限のデータで構成されており，したがってそれを表す文字列 $\langle M\rangle$ を作ることができる．$\langle M\rangle$ を M の記述と呼ぶことにする．人間は Turing 機械 M の記述 $\langle M\rangle$ と入力文字列 w が与えられれば $M(w)$ の計算をシミュレートできるので，Church-Turing の提唱よりそのような Turing 機械が存在することがわかる．

定理 2.6 (万能 Turing 機械). 次のような Turing 機械 U が存在する: 全ての Turing 機械 M と入力文字列 w に対して $U(\langle\langle M\rangle, w\rangle) \simeq M(w)$ となる．このような U を**万能 Turing 機械** (universal Turing machine) と呼ぶ．

注意 2.7. U の入力アルファベットはひとつに固定されているのに対し，入力文字列 w は M ごとに範囲が異なることに注意せよ．したがって，文字列の組 $(\langle M\rangle, w)$ をひとつの文字列 $\langle\langle M\rangle, w\rangle$ に変換する際，U の入力アルファベットを用いて表現する必要がある．

3 停止問題

前節では Turing 機械を導入し，それにより決定問題の決定可能性の定義が数学的に厳密なものとなった．本節では具体的な決定不能問題の最も重要な例として停止問題を取り上げる．

問題 3.1 (停止問題 (halting problem; HALT)).
Input: Turing 機械 M の記述 $\langle M\rangle \in \Sigma^*$ と入力 $w \in \Sigma^*$ の組 $\langle\langle M\rangle, w\rangle \in \Sigma^*$
Question: $M(w)\downarrow = 1$ か？ (M は w を受理するか？) [*12]

定理 3.2. 停止問題 HALT は決定不能である．

証明. 背理法で示す．仮に HALT が決定可能であるとすると，HALT を判定する Turing 機械 H が存在する．つまり，任意の Turing 機械 M と文字列 w に対して

$$\begin{aligned} M(w)\downarrow = 1 &\implies H(\langle\langle M\rangle, w\rangle)\downarrow = 1, \\ M(w)\uparrow \text{ or } M(w)\downarrow = 0 &\implies H(\langle\langle M\rangle, w\rangle)\downarrow = 0 \end{aligned}$$

が成り立つ (実際には上の「$\implies$」は「$\iff$」である)．このとき，Turing 機械 D を任意の Turing 機械 M に対して

$$D(\langle M\rangle) = \begin{cases} 0 & \text{if } H(\langle\langle M\rangle, \langle M\rangle\rangle)\downarrow = 1, \\ 1 & \text{if } H(\langle\langle M\rangle, \langle M\rangle\rangle)\downarrow = 0 \end{cases}$$

となるように作ることができる．[*13]このとき入力 $\langle D\rangle$ に対する D の計算結果を考えると，D が全域的であることより $D(\langle D\rangle)\downarrow = 1$ または $D(\langle D\rangle)\downarrow = 0$ でなくてはならないが，

$$\begin{aligned} D(\langle D\rangle)\downarrow = 0 &\iff H(\langle\langle D\rangle, \langle D\rangle\rangle)\downarrow = 1 && (D \text{ の定義より}) \\ &\iff D(\langle D\rangle)\downarrow = 1 && (H \text{ の定義より}) \end{aligned}$$

となり矛盾． □

注意 3.3. 上の証明は実は対角線論法になっている．このことを確認してみよう．Turing 機械は可算個しかないので $M_1, M_2, \ldots$ と一列に並べることができる (例えば，記述 $\langle M\rangle$ の短い方から辞書順に並べればよい)．したがって $H(\langle\langle M_i\rangle, \langle M_j\rangle\rangle)$ の計算結果を以下の表 1 のように一覧にすることができる．

[*12] 「停止問題」という名前からすれば「$M(w)\downarrow$ か？」の方が適切である (問題 4.6 を見よ) が，本稿では問題 3.1 を停止問題と呼ぶことにする．

[*13] 「H の計算結果を見て，その逆の値を出力する」という計算が可能であることは直観的に明らかだから，Church-Turing の提唱よりこのような Turing 機械 D が存在する．

表 1: $H(\langle\langle M_i\rangle, \langle M_j\rangle\rangle)$ の計算結果 ($M_i(\langle M_j\rangle)\downarrow = 1$ かどうか) の一覧

	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\cdots$	$\langle D\rangle$	$\cdots$
M_1	$\boxed{1}$	0	1	0	$\cdots$	1	$\cdots$
M_2	1	$\boxed{1}$	1	1	$\cdots$	1	$\cdots$
M_3	0	0	$\boxed{0}$	0	$\cdots$	0	$\cdots$
M_4	1	1	0	$\boxed{0}$	$\cdots$	1	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	
D	$\boxed{0}$	$\boxed{0}$	$\boxed{1}$	$\boxed{1}$	$\cdots$	$\boxed{?}$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\ddots$

D 自身は Turing 機械なのでこの表のどこかに現れる，つまりある k について $D = M_k$ となるはずである．ところが D は対角線上の値と逆の計算結果を返すように設計されているため，$D(\langle M_k\rangle) \neq M_k(\langle M_k\rangle)$ となり矛盾する．

4 停止問題の変種

この節では，停止問題を少し変形した問題を 2 つ取り上げ，それらが決定不能であることを証明する．これらの証明は，停止問題の定義を少々変更しても本質的には何も変わらないということを示しているだけではなく，ある決定問題が決定不能であることを証明する典型的な手法の原型でもある．

問題 4.1 (停止問題の変種その 1; HALT′).
Input: Turing 機械 M の記述 $\langle M\rangle \in \Sigma^*$
Question: $M(\varepsilon)\downarrow = 1$ か？ (M は空文字列 ε を受理するか？)

定理 4.2. HALT′ は決定不能である．

証明. 背理法で示す．仮に HALT′ が決定可能であると仮定すると，HALT も決定可能となってしまうことを示す．これが言えれば，定理 3.2 より HALT は決定不能だったので矛盾を生じ，証明が終わる．

仮定より HALT′ は決定可能だから，HALT′ を判定する Turing 機械 H' が存在する．つまり，任意の Turing 機械 M に対して

$$\begin{aligned} M(\varepsilon)\downarrow = 1 &\implies H'(\langle M\rangle)\downarrow = 1, \\ M(\varepsilon)\uparrow \text{ or } M(\varepsilon)\downarrow = 0 &\implies H'(\langle M\rangle)\downarrow = 0 \end{aligned}$$

が成り立つ．この H' を利用して HALT を判定するアルゴリズムを作ればよい．

次のようなアルゴリズムを実行する Turing 機械 H を考える．

$H =$「入力 $\langle\langle M\rangle, w\rangle$ に対して:
　次のような Turing 機械 N を作る．

　　$N =$「入力 v に対して:
　　　v を無視し，$M(w)$ を計算し，その結果を出力する．」

　$H'(\langle N\rangle)$ を計算し，その結果を出力する．」

このとき，任意の Turing 機械 M と入力文字列 w に対して

$$\begin{aligned}
&M(w)\!\downarrow = 1 \implies N(\varepsilon)\!\downarrow = 1 \implies H'(\langle N\rangle)\!\downarrow = 1 \implies H(\langle\langle M\rangle, w\rangle)\!\downarrow = 1,\\
&M(w)\!\uparrow \quad\ \implies N(\varepsilon)\!\uparrow \quad\ \implies H'(\langle N\rangle)\!\downarrow = 0 \implies H(\langle\langle M\rangle, w\rangle)\!\downarrow = 0,\\
&M(w)\!\downarrow = 0 \implies N(\varepsilon)\!\downarrow = 0 \implies H'(\langle N\rangle)\!\downarrow = 0 \implies H(\langle\langle M\rangle, w\rangle)\!\downarrow = 0
\end{aligned}$$

となるので H は HALT を判定する． □

注意 4.3. 定理 4.2 の証明において最も大切なことは，**アルゴリズムの非存在を示す問題がアルゴリズムの存在を示す問題に帰着された**ということである．このように，ある問題が決定可能であると仮定して他の決定問題の決定可能性を示す手法を **Turing 還元** (Turing reduction) (または Turing 帰着) という．[*14]決定不能性を直接示すためにはいかなるアルゴリズムも正しい答えを返さないことを示す必要があるが，それよりはアルゴリズムをひとつ作る方が易しいというわけである．

問題 4.4 (停止問題の変種その 2; HALT″).
Input: Turing 機械 M の記述 $\langle M\rangle \in \Sigma^*$
Question: M に受理されるような文字列は存在するか？

定理 4.5. HALT″ は決定不能である．

証明. 定理 4.2 の証明と同様に，HALT″ が決定可能であることを仮定すると HALT も決定可能となってしまうことを示す．HALT″ が決定可能であることより HALT″ を判定する Turing 機械 H'' が存在する．Turing 機械 H を，定理 4.2 の証明で構成した H 中の H' を H'' に置き換えたものとする．このとき，N が受理する文字列全体の集合を $L(N) := \{\, v \in \Sigma^* \mid N(v)\!\downarrow = 1 \,\}$ とおくと $L(N)$ は $\emptyset$ か Σ^* にしかならないことに注意する．これより任意の Turing 機械 M と入力文字列 w に対して

$$\begin{aligned}
&M(w)\!\downarrow = 1 \qquad\qquad \implies L(N) = \Sigma^* \implies H'(\langle N\rangle)\!\downarrow = 1 \implies H(\langle\langle M\rangle, w\rangle)\!\downarrow = 1,\\
&M(w)\!\uparrow \text{ or } M(w)\!\downarrow = 0 \implies L(N) = \emptyset \quad \implies H'(\langle N\rangle)\!\downarrow = 0 \implies H(\langle\langle M\rangle, w\rangle)\!\downarrow = 0
\end{aligned}$$

となるので H は HALT を判定する． □

筆者の好みで問題 3.1 を停止問題と呼んだが，“正統な” 停止問題は次の問題 4.6 であろう．

問題 4.6 (停止問題の変種その 3; HALT‴).
Input: Turing 機械 M の記述 $\langle M\rangle \in \Sigma^*$ と入力 $w \in \Sigma^*$ の組 $\langle\langle M\rangle, w\rangle \in \Sigma^*$
Question: $M(w)\!\downarrow$ か？ (M は入力 w に対して停止するか？)

定理 4.7. HALT‴ は決定不能である．

[*14] 今回の場合は「HALT は HALT′ に Turing 還元可能である」と表現する．HALT を判定するという問題を HALT′ を判定する問題に帰着した，すなわち「HALT′ さえ解ければ HALT も解ける」ということを示したわけである．

証明. HALT''' が決定可能であることを仮定すると HALT も決定可能となってしまうことを示す．HALT''' が決定可能であることより HALT''' を判定する Turing 機械 H''' が存在する．H''' が全域的であることに注意して

$H =$「入力 $\langle\langle M\rangle, w\rangle$ に対して:
　$H'''(\langle\langle M\rangle, w\rangle)\downarrow = 0$ なら拒否する．
　$H'''(\langle\langle M\rangle, w\rangle)\downarrow = 1$ なら $M(w)$ を計算し，その結果を出力する．」

とおけば H は HALT を判定する． □

参考文献

[1] M. Sipser (太田和夫・田中圭介 監訳, 阿部正幸・植田広樹・藤岡淳・渡辺治 訳), 計算理論の基礎 [原著第 2 版] 2. 計算可能性の理論, 共立出版, 2008.

[2] 河村彰星, はじめての計算可能性, 数学基礎論サマースクール 2017, http://www.graco.c.u-tokyo.ac.jp/~kawamura/t/summer_H29/, 2017.

[3] y., 決定不能問題の話, 第 10 回すうがく徒のつどい, http://iso.2022.jp/math/tsudoi/10/slide.pdf, 2017.

[4] y., 決定不能問題から始める計算可能性理論入門, 都数 12 月総会, http://iso.2022.jp/math/tosuu-2017-12/resume.pdf, 2017.

変更履歴

2018/01/01　公開

2018/01/05　問題 4.6 を追加