再帰的関数
Recursive Function

y.*

2018 年 6 月 1 日

最終更新日: 2018 年 6 月 1 日

概要

原始再帰的関数 (primitive recursive function) とは，定数関数 $\mathtt{zero}$，後者関数 $\mathtt{succ}$，射影関数 $\mathtt{proj}_i^n$ と呼ばれる 3 つの初期関数から合成と原始再帰法 (いわゆる “漸化式”) によって得られる関数のクラスである．**再帰的関数** (recursive function) とは，原始再帰的関数に加えて最小化演算子 μ を用いて得られる部分関数のクラスである．本稿では再帰的関数が Turing 機械と同等の計算能力を持つことを示し，再帰的関数に関する Kleene の正規形定理を証明する．

Keywords: 原始再帰的関数 (primitive recursive function), 再帰的関数 (recursive function).

本稿では $\mathbb{N}$ は非負整数全体を表すものとする．すなわち $0 \in \mathbb{N}$ とする．

1 再帰的関数と原始再帰的関数

以下，自然数の有限列 $(x_1, \ldots, x_n) \in \mathbb{N}^n$ をベクトルのように $\vec{x}$ で表す．

定義 1.1 (再帰的関数，原始再帰的関数)． **再帰的関数** (recursive function) とは以下のようにして帰納的に構成される部分関数のクラスである．

1. 以下の 3 つの**初期関数** (initial function) は全て再帰的関数である．
   - 0 変数零関数 (定数)　$\mathtt{zero}\colon \mathbb{N}^0 \to \mathbb{N}$　$\mathtt{zero}() = 0$
   - **射影** (projection)　$\mathtt{proj}_i^n\colon \mathbb{N}^n \to \mathbb{N}$　$\mathtt{proj}_i^n(x_1, \ldots, x_n) = x_i \quad (1 \leq i \leq n)$
   - **後者関数** (successor)　$\mathtt{succ}\colon \mathbb{N} \to \mathbb{N}$　$\mathtt{succ}(x) = x + 1$
2. $g\colon \mathbb{N}^n \rightharpoonup \mathbb{N}\,(n > 0), h_1, \ldots, h_n\colon \mathbb{N}^m \rightharpoonup \mathbb{N}\,(m \geq 0)$ が再帰的関数ならば，それらの**合成** (composition) $f\colon \mathbb{N}^m \rightharpoonup \mathbb{N}; f(\vec{x}) :\simeq g(h_1(\vec{x}), \ldots, h_n(\vec{x}))$ も再帰的関数である．ここで $f(\vec{x})$ は $h_1(\vec{x})\downarrow = y_1, \ldots, h_n(\vec{x})\downarrow = y_n$ かつ $g(y_1, \ldots, y_n)\downarrow = z$ のとき，またそのときに限り $f(\vec{x})\downarrow = z$ となるものとする．つまり，$h_1, \ldots, h_n, g$ のうちでひとつでも値が未定義なものがあれば f も未定義となる．この f を $g \circ (h_1, \ldots, h_n)$ と書く．
3. $g\colon \mathbb{N}^n \rightharpoonup \mathbb{N}, h\colon \mathbb{N}^{n+2} \rightharpoonup \mathbb{N}\,(n \geq 0)$ が再帰的関数ならば，**原始再帰法** (primitive recursion) によって

$$f(0, \vec{y}) :\simeq g(\vec{y}),$$
$$f(x+1, \vec{y}) :\simeq h(x, \vec{y}, f(x, \vec{y}))$$

* http://iso.2022.jp/

で定義される $f\colon \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$ も再帰的関数である．ここでも先程と同様に，$f(x+1,\vec{y})$ が定義されるのは，$g(\vec{y})\,(= f(0,\vec{y})), h(0,\vec{y},f(0,\vec{y})), f(1,\vec{y}), h(1,\vec{y},f(1,\vec{y})), \ldots, f(x,\vec{y}), h(x,\vec{y},f(x,\vec{y}))$ が全て定義されるときに限る．

4. $g\colon \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}\,(n \geq 0)$ が再帰的関数ならば，**最小化演算子** (minimization operator) (μ **演算子** (μ-operator) とも呼ばれる) μ によって

$$f(\vec{x}) :\simeq \mu y[g(\vec{x},y)] := \min\{\, y \in \mathbb{N} \mid g(\vec{x},y)\!\downarrow = 0 \,\}$$

で定義される $f\colon \mathbb{N}^n \to \mathbb{N}$ も再帰的関数である．ここで $f(\vec{x})$ は $g(\vec{x},y)\!\downarrow = 0$ となるような $y \in \mathbb{N}$ が存在しないときは定義されない．

5. 以上によって定義されるもののみが再帰的関数である．

再帰的関数の定義から最小化演算子を除いて，初期関数から始めて合成と原始再帰法のみで作られる関数のクラスを**原始再帰的関数** (primitive recursive function) と呼ぶ．原始再帰的関数は全て全域関数であることを注意しておく．つまり，再帰的関数の値が未定義になりうるのは μ 演算子が用いられているときだけである．

例 1.2. 1 変数零関数 $\mathtt{zero}_1(x) = 0$，n 変数零関数 $\mathtt{zero}_n(x_1,\ldots,x_n) = 0$，$n$ 変数 $(n \geq 0)$ 定数関数 $\mathtt{const}^n_c(x_1,\ldots,x_n) = c$，前者関数 $\mathtt{pred}(x) = \max\{x-1,0\}$，加法 $\mathtt{add}(x,y) = x+y$，減法 $\mathtt{sub}(x,y) = x \dot{-} y := \max\{x-y,0\}$,[*1] 乗法 $\mathtt{mult}(x,y) = x \cdot y$ は全て原始再帰的関数である．実際，次のように定義すればよい:

$$\begin{aligned}
\mathtt{zero}_1(0) &= \mathtt{zero}(),\\
\mathtt{zero}_1(x+1) &= \mathtt{proj}^2_2(x,\mathtt{zero}_1(x)),\\
\mathtt{zero}_n(\vec{x}) &= \mathtt{zero}_1 \circ \mathtt{proj}^n_1(\vec{x}),\\
\mathtt{const}^n_c(\vec{x}) &= \underbrace{\mathtt{succ}\circ\cdots\circ\mathtt{succ}}_{c} \circ \mathtt{zero}_n(\vec{x}),\\
\mathtt{pred}(0) &= \mathtt{zero}(),\\
\mathtt{pred}(x+1) &= \mathtt{proj}^2_1(x,\mathtt{pred}(x)),\\
\mathtt{add}(0,y) &= \mathtt{proj}^1_1(y),\\
\mathtt{add}(x+1,y) &= \mathtt{succ}\circ\mathtt{proj}^3_3(x,y,\mathtt{add}(x,y))),\\
\mathtt{sub}'(0,x) &= \mathtt{proj}^1_1(x),\\
\mathtt{sub}'(y+1,x) &= \mathtt{pred}\circ\mathtt{proj}^3_3(y,x,\mathtt{sub}'(y,x)),\\
\mathtt{sub}(x,y) &= \mathtt{sub}'\circ(\mathtt{proj}^2_2,\mathtt{proj}^2_1)(x,y),\\
\mathtt{mult}(0,y) &= \mathtt{zero}_1(y),\\
\mathtt{mult}(x+1,y) &= \mathtt{add}\circ(\mathtt{proj}^3_3,\mathtt{proj}^3_2)(x,y,\mathtt{mult}(x,y)).
\end{aligned}$$

最後の行のように $\mathtt{add}\circ(\mathtt{proj}^3_3,\mathtt{proj}^3_2)(x,y,\mathtt{mult}(x,y))$ などといちいち正確に全て書くのは煩わしいので，以降は射影や四則演算を省略して単に $x \cdot y + y$ と書く．

定義 1.3. true または false の二値のみをとるような自然数上の部分関数 $p\colon \mathbb{N}^n \rightharpoonup \{\mathsf{true},\mathsf{false}\}$ を (n 変数の) **述語** (predicate) という．true, false をそれぞれ $0,1$ と同一視することにより，値域が $\{0,1\}$ に含まれるよ

[*1] このように計算結果が自然数の範囲に収まるように修正された減算 $\dot{-}$ のことを **modified subtraction** と呼ぶ.

うな部分関数 $f\colon \mathbb{N}^n \rightharpoonup \mathbb{N}$ のことも述語と呼ぶ.[*2]再帰的関数，原始再帰的関数が述語であるとき，それぞれ再帰的述語，原始再帰的述語と呼ぶ.

補題 1.4. $x=0$ を表す述語 $\mathtt{isZero}(x)$，$x=y$ を表す $\mathtt{equal}(x,y)$，$x\leq y$ を表す $\mathtt{lessThanOrEqual}(x,y)$，$x<y$ を表す $\mathtt{less}(x,y)$ は全て原始再帰的述語である．また，論理演算 $\mathtt{not}(x)=\neg x, \mathtt{or}(x,y)=x\vee y, \mathtt{and}(x,y)=x\wedge y, \mathtt{imply}(x,y)=x\to y$ も全て原始再帰的述語である.

証明. 次のように定めればよい:

$$\begin{aligned}
\mathtt{isZero}(x) &= 1 \dot{-} (1 \dot{-} x),\text{[*3]}\\
\mathtt{equal}(x,y) &= \mathtt{isZero}((x \dot{-} y) + (y \dot{-} x)),\\
\mathtt{lessThanOrEqual}(x,y) &= \mathtt{isZero}(x \dot{-} y),\\
\mathtt{less}(x,y) &= \mathtt{lessThanOrEqual}(x+1, y),\\
\mathtt{not}(x) &= 1 \dot{-} x,\\
\mathtt{or}(x,y) &= x \cdot y,\\
\mathtt{and}(x,y) &= \mathtt{not}(\mathtt{or}(\mathtt{not}(x), \mathtt{not}(y))),\\
\mathtt{imply}(x,y) &= \mathtt{or}(\mathtt{not}(x), y).
\end{aligned}$$

□

以降，$\mathtt{lessThanOrEqual}(x,y)$ などと書くのは煩雑であるので通常どおり $x\leq y$ などの記法を縦横に用いる．また，述語を定義するときに “$=$” ではなく “$\Longleftrightarrow$” を用いることがある.

余談 1.5. 先程，原始再帰的関数は全て全域的であるということを述べた．この逆はどうであろうか？ つまり，全域的な再帰的関数は全て原始再帰的関数だろうか？ 実は，全域的な再帰的関数であって原始再帰的関数でないものの存在が知られている．その最も有名な例は Ackermann 関数であろう．詳細は藤田 [7] や新井 [4, 演習 2.6.3] を参照のこと.

2 有限列のコード化

本節では，有限列のコード化と呼ばれる，自然数の有限の長さの組 $(x_0,\dots,x_{n-1})\in\mathbb{N}^n$ を一つの自然数 $\langle x_0,\dots,x_{n-1}\rangle\in\mathbb{N}$ にまとめるテクニックを紹介する．有限列のコード化には色々なやり方があるが，ここでは一番わかりやすい素因数分解を利用した方法を採用する.[*4]

定義 2.1. 自然数の有限列を一つの自然数にエンコードする関数 $\bigcup_{n\geq 0}\mathbb{N}^n \to \mathbb{N}; (x_0,\dots,x_{n-1})\mapsto \langle x_0,\dots,x_{n-1}\rangle$ を

$$\langle x_0,\dots,x_{n-1}\rangle := p_0^{1+x_0} p_1^{1+x_1}\cdots p_{n-1}^{1+x_{n-1}} \tag{1}$$

と定める.[*5]ただし，p_k は k 番目の素数を表す $(p_0=2, p_1=3, p_2=5,\dots)$.

ここで，各素数の指数が $1+x_i$ の形になっているのは，もとの列を一意に復元できるようにするためである．もし $\langle x_0,\dots,x_{n-1}\rangle = p_0^{x_0}p_1^{x_1}\cdots p_{n-1}^{x_{n-1}}$ と定義してしまうと，$\langle 0\rangle = \langle 0,0\rangle = \langle 0,0,0\rangle = \cdots$ 等となってし

[*2] もちろん true を 1，false を 0 と同一視する流儀もあるが，本稿では $\mu y[p(\vec{x},y)]$ が「$p(\vec{x},y)$ が成り立つ最小の y」を表すようにしたかったのでこのようにした.

[*3] 言うまでもないが，この 1 は先程定義した定数関数である.

[*4] 素因数分解以外のコード化の方法として，例えば中国剰余定理を用いたものが知られている.

[*5] 空列 () のコードは $\langle\rangle = 1$ である.

まい，もとの列が一意に定まらなくなってしまう．

また，写像 (1) は全射ではない．すなわち，全ての自然数が何らかの有限列のコードになっているわけではない．例えば，$2^4 \cdot 5^2$ は (3 の指数が 0 なので) 有限列のコードではない．

x がある有限列のコードであるとき，x の i 番目の成分を $(x)_i$ で表すことにする (つまり，$(\langle x_0, \ldots, x_{n-1} \rangle)_i = x_i$ である)．このとき有限列のコードから各成分を取り出す (“デコード” する) 関数 $\mathbb{N} \times \mathbb{N} \to \mathbb{N}; (x, i) \mapsto (x)_i$ は

$$(x)_i = \max\{\, n \in \mathbb{N} \mid (p_i^n \mid x) \,\} \dot{-} 1 \tag{2}$$

と表される．

以降，本節ではこの有限列のエンコード (1)，デコード (2) がともに原始再帰的関数によって実現できることを示す．

補題 2.2. 原始再帰的関数 f，原始再帰的述語 p に対し，以下の関数・関係は全て原始再帰的である．

- 原始再帰的述語での場合分けによる定義，
- 累積足し算 $(x, \vec{y}) \mapsto \sum_{i<x} f(i, \vec{y})$，
- 累積掛け算 $(x, \vec{y}) \mapsto \prod_{i<x} f(i, \vec{y})$，
- 有界存在量化 $(x, \vec{y}) \mapsto \exists z < x[p(z, \vec{y})]$，
- 有界全称量化 $(x, \vec{y}) \mapsto \forall z < x[p(z, \vec{y})]$，
- 有界最小解関数 $(x, \vec{y}) \mapsto \min(\{\, z < x \mid p(z, \vec{y}) \,\} \cup \{x\})$，
- 有界最大解関数 $(x, \vec{y}) \mapsto \begin{cases} \max\{\, z < x \mid p(z, \vec{y}) \,\} & (\exists z < x[p(z, \vec{y})] \text{ のとき}), \\ x & (\text{それ以外のとき}), \end{cases}$
- 整除関係 $x \mid y$ (x は y を割り切る)，
- 「x は素数である」を表す $\texttt{isPrime}(x)$，
- 指数関数 x^y，
- 階乗 $x!$，
- x 番目の素数を返す関数 $x \mapsto p_x$，
- x が p_i で割れる回数を返す関数 $\texttt{exponent}(i, x)$，
- 各 $n \in \mathbb{N}$ について，n 変数のエンコード関数 $(x_0, \ldots, x_{n-1}) \mapsto \langle x_0, \ldots, x_{n-1} \rangle$，
- 有限列のコードの**長さ** (length) を返す関数 $\mathrm{lh}(x)$，
- 「x はある有限列のコードである」を表す $\texttt{isSequence}(x)$，
- デコード関数 $(x, i) \mapsto (x)_i$.

証明. f_1, f_2 を原始再帰的関数とするとき，場合分けによって定義される関数

$$g(\vec{x}) := \begin{cases} f_1(\vec{x}) & (p(\vec{x}) \text{ が成り立っているとき (つまり } p(\vec{x}) = 0)) \\ f_2(\vec{x}) & (\text{それ以外のとき}) \end{cases}$$

は $g(\vec{x}) = f_1(\vec{x}) \cdot (\neg p(\vec{x})) + f_2(\vec{x}) \cdot p(\vec{x})$ と書ける．累積足し算と累積掛け算は原始再帰法によって

$$\begin{aligned}
\sum_{i<0} f(i,\vec{y}) &= 0,\\
\sum_{i<x+1} f(i,\vec{y}) &= \left(\sum_{i<x} f(i,\vec{y})\right) + f(x,\vec{y}),\\
\prod_{i<0} f(i,\vec{y}) &= 1,\\
\prod_{i<x+1} f(i,\vec{y}) &= \left(\prod_{i<x} f(i,\vec{y})\right) \cdot f(x,\vec{y})
\end{aligned}$$

と書ける．よって有界存在量化と有界全称量化は

$$\begin{aligned}
\exists z < x[p(z,\vec{y})] &= \mathtt{isZero}\left(\prod_{z<x} p(z,\vec{y})\right),\\
\forall z < x[p(z,\vec{y})] &\iff \neg(\exists z < x[\neg p(z,\vec{y})])
\end{aligned}$$

と書ける．有界最小解関数については，$p(i,\vec{y})$ が最初に成り立つところまで 1 を足し続ければよいので

$$\min(\{\, z < x \mid p(z,\vec{y}) \,\} \cup \{x\}) = \sum_{i<x} \big(\exists j \le i[p(j,\vec{y})]\big)$$

と書ける (ここで $\exists j \le i$ は $\exists j < (i+1)$ のことである)．有界最大解関数については，$x \dot{-} 1$ から 0 に向かって調べていけばよいので

$$\begin{cases}
x \dot{-} \min\{\, z < x \mid z \ge 1 \wedge p(x \dot{-} z,\vec{y}) \,\} & (\exists z < x[p(z,\vec{y})] \text{ のとき}),\\
x & (\text{それ以外のとき})
\end{cases}$$

と書ける．整除関係は

$$x \mid y \iff \exists z \le y[x \cdot z = y]$$

と書ける．「x は素数である」は

$$\mathtt{isPrime}(x) \iff x > 1 \wedge \forall y < x[y \mid x \to y = 1]$$

と書ける．指数関数と階乗は原始再帰法によって

$$\begin{aligned}
x^0 &= 1,\\
x^{y+1} &= x^y \cdot x,\\
0! &= 1,\\
(x+1)! &= x! \cdot (x+1)
\end{aligned}$$

と書ける．次に x 番目の素数 p_x は原始再帰法により

$$\begin{aligned}
p_0 &= 2,\\
p_{x+1} &= \min\{\, y \le p_x! + 1 \mid y > p_x \wedge \mathtt{isPrime}(y) \,\}
\end{aligned}$$

と書ける (なぜなら，Euclid による素数の無限性証明と同様にして，任意の n について $p_{n+1} \le p_n! + 1$ が成り立つことが示せるからである)．

以上により，残りの関数・述語は

$$\begin{aligned}
\texttt{exponent}(i,x) &= \max\{\, n \le x \mid (p_i^n \mid x)\,\},\\
\langle x_0,\dots,x_{n-1}\rangle &= p_0^{1+x_0}\cdot p_1^{1+x_1}\cdot\dots\cdot p_{n-1}^{1+x_{n-1}},\\
\mathrm{lh}(x) &= \min\{\, i \le x \mid \texttt{exponent}(i,x) = 0\,\},\\
\texttt{isSequence}(x) &\iff \forall i \le x[p_i \mid x \to i < \mathrm{lh}(x)],\\
(x)_i &= \texttt{exponent}(i,x) \dot{-} 1
\end{aligned}$$

と書ける．□

3 Turing 機械で計算可能ならば再帰的

本節では自然数値関数を計算する Turing 機械の定義を用いる．また，Turing 機械のテープアルファベットは $\Gamma = \{\textvisiblespace, \mathtt{1}\}$ に固定する．詳細は「Turing 機械の変種」[1] を参照のこと．

以下，ある数学的対象 E を自然数で表現したもの (E の “コード”) を $\ulcorner E \urcorner \in \mathbb{N}$ と表すことにする．

定理 3.1. Turing 機械で計算可能な部分関数 $\mathbb{N}^n \rightharpoonup \mathbb{N}$ は再帰的関数である．

証明のアイデア. まず，前節で定義した有限列のコード化を用いて，M の情報 (遷移関数) を一つの自然数 e にエンコードする．$M(\vec{x})\!\downarrow$ のとき，またそのときに限って，有限の長さの計算の履歴が存在する．計算履歴も一つの自然数にコード化できるので，「y は e がコードする Turing 機械 M の入力 $\vec{x}$ に対する計算履歴である」を表す述語 $T_n(e,\vec{x},y)$ を作ることができる．よって $T_n(e,\vec{x},y)$ が成り立つ y を μ 演算子によって探索すればよい．□

証明. Turing 機械 $M = (Q, \Sigma = \{\mathtt{1}\}, \Gamma = \{\textvisiblespace, \mathtt{1}\}, \delta, q_0, q_{\mathrm{halt}})$ をとる．集合の元の名前は計算可能性に関係しないから，自然数で置き換えて $Q = \{0,1,\dots,|Q|-1\}, \Sigma = \{0\}, \Gamma = \{0,1\}, q_0 = 0, q_{\mathrm{halt}} = 1$ としてよい ($q_0 = q_{\mathrm{halt}}$ だったら $q_0 \neq q_{\mathrm{halt}}$ となるように改造しておけばよいので $q_0 \neq q_{\mathrm{halt}}$ と仮定しても一般性を失わない)．また L, R も $0, 1$ と同一視する．

まず M のコード $e = \ulcorner M \urcorner \in \mathbb{N}$ を構成する．遷移関数 $\delta\colon \{0,\dots,|Q|-1\} \times \{0,1\} \to \{0,\dots,|Q|-1\} \times \{0,1\} \times \{0,1\}$ のコード $\ulcorner \delta \urcorner \in \mathbb{N}$ を，値を一列に並べた

$$\ulcorner \delta \urcorner = \langle \ulcorner \delta(0,0) \urcorner, \ulcorner \delta(0,1) \urcorner, \ulcorner \delta(1,0) \urcorner, \ulcorner \delta(1,1) \urcorner, \dots, \ulcorner \delta(|Q|-1,0) \urcorner, \ulcorner \delta(|Q|-1,1) \urcorner \rangle$$

と定める ($\delta(q,a) = (r,b,D)\ (0 \le r < |Q|, b, D \in \{0,1\})$ のコードは $\ulcorner (r,b,D) \urcorner = \langle r,b,D \rangle$ で与えられる)．よって $e = \langle |Q|, \ulcorner \delta \urcorner \rangle$ とおけばよい．この e は M の動作をシミュレートするために必要な情報を全て含んでいる．

次に，計算の履歴を一つの自然数にコードする方法を与える．まず，M の**時点表示** (instantaneous description) とは，M の現在の状態 $q \in Q$，M のテープの内容 $\alpha = (a_0,\dots,a_{l-1})$，ヘッドの位置 $i < l$ の三つ組 $D = (q,\alpha,i)$ のことである．入力 $\vec{x}$ に対する M の計算履歴は時点表示の列 $(A_0,\dots,A_m)$ であって以下を満たすもののことである．

1. $A_0 = (0, (\underbrace{1,\dots,1}_{x_1+1}, 0, \underbrace{1,\dots,1}_{x_2+1}, 0, \dots, 0, \underbrace{1,\dots,1}_{x_n+1}), 0)$.
2. 各 $i < m$ について，A_{i+1} は A_i から δ によって 1 ステップだけ計算を進めた後の時点表示である．(ただし，A_{i+1} のテープの内容の長さは A_i のそれよりちょうど 1 だけ長いものとする．)

3. ある有限列 β と i について $A_m = (1, \beta, i)$ となっている．

計算履歴のコードは ( ) を全て $\langle\ \rangle$ に置き換えたものとして定める．

原始再帰的述語 T_n と原始再帰的関数 U を以下のように構成する．

$$
\begin{aligned}
\texttt{isTuringMachine}(e) :&\iff \texttt{isSequence}(e) \wedge \mathrm{lh}(e) = 2 \wedge \mathrm{lh}((e)_1) = 2 \cdot (e)_0 \\
&\quad \wedge \forall i < \mathrm{lh}((e)_1)[\texttt{isSequence}(((e)_1)_i) \wedge \mathrm{lh}(((e)_1)_i) = 3 \\
&\qquad \wedge (((e)_1)_i)_0 < (e)_0 \wedge (((e)_1)_i)_1 \leq 1 \wedge (((e)_1)_i)_2 \leq 1] \\
&\iff e = \langle |Q|, \ulcorner \delta \urcorner \rangle \text{ はある Turing 機械 } M \text{ のコードである,} \\
\texttt{isTapeContent}(z) :&\iff \texttt{isSequence}(z) \wedge \forall i < \mathrm{lh}(z)[(z)_i \leq 1] \\
&\iff z \text{ はテープの内容のコード,} \\
\texttt{isDescription}(e, z) :&\iff \texttt{isSequence}(z) \wedge \mathrm{lh}(z) = 3 \wedge (z)_0 < (e)_0 \\
&\quad \wedge \texttt{isTapeContent}((z)_1) \wedge (z)_2 < \mathrm{lh}((z)_1) \\
&\iff z \text{ は } e \text{ (でコードされる Turing 機械) の時点表示である,} \\
\texttt{isInitialDescription}(\vec{x}, z) :&\iff (z)_0 = 0 \wedge \mathrm{lh}((z)_1) = x_1 + x_2 + \cdots + x_n + 2 \cdot n \dot{-} 1 \\
&\quad \wedge \forall i < \mathrm{lh}((z)_1) \left[((z)_1)_i = \begin{bmatrix} i = x_1 + 1 \vee i = x_1 + x_2 + 2 \vee \cdots \\ \vee i = x_1 + x_2 + \cdots + x_{n-1} + n \dot{-} 1 \end{bmatrix} \right] \\
&\quad \wedge (z)_2 = 0 \\
&\iff z \text{ は入力 } \vec{x} \text{ に対する計算開始時の時点表示である,} \\
\texttt{isHaltingDescription}(z) :&\iff (z)_0 = 1 \\
&\iff z \text{ は計算停止時の時点表示である,} \\
t(e, u) :&= ((e)_1)_{2 \cdot (u)_0 + ((u)_1)_{(u)_2}} \\
&= (u = \langle q, \alpha, i \rangle \text{ のときの } \delta(q, (\alpha)_i) \text{ のコード),} \\
\texttt{isValidTransition}(e, u, v) :&\iff \neg\, \texttt{isHaltingDescription}(u) \wedge (v)_0 = (t(e, u))_0 \\
&\quad \wedge \mathrm{lh}((v)_1) = \mathrm{lh}((u)_1) + 1 \wedge ((v)_1)_{\mathrm{lh}((v)_1) \dot{-} 1} = 0 \\
&\quad \wedge \forall i < \mathrm{lh}((u)_1) \begin{bmatrix} i = (u)_2 \to ((v)_1)_i = (t(e, u))_1 \\ \wedge \neg i = (u)_2 \to ((v)_1)_i = ((u)_1)_i \end{bmatrix} \\
&\quad \wedge ((t(e, u))_2 = 0 \to (v)_2 = (u)_2 \dot{-} 1) \\
&\quad \wedge ((t(e, u))_2 = 1 \to (v)_2 = (u)_2 + 1) \\
&\iff u \text{ を } e \text{ によって 1 ステップ計算すると } v \text{ になる,} \\
T_n(e, \vec{x}, y) :&\iff \texttt{isTuringMachine}(e) \\
&\quad \wedge \texttt{isSequence}(y) \wedge \forall i < \mathrm{lh}(y)[\texttt{isDescription}(e, (y)_i)] \\
&\quad \wedge \texttt{isInitialDescription}(\vec{x}, (y)_0) \\
&\quad \wedge \forall i < (\mathrm{lh}(y) \dot{-} 1)[\texttt{isValidTransition}(e, (y)_i, (y)_{i+1})] \\
&\quad \wedge \texttt{isHaltingDescription}((y)_{\mathrm{lh}(y) \dot{-} 1}) \\
&\iff y \text{ は } e \text{ がコードする Turing 機械の入力 } \vec{x} \text{ に対する計算履歴である,} \\
\texttt{numberOfOnes}(x) :&= \sum_{i < \mathrm{lh}(x)} (x)_i \\
&= (\text{テープ } x \text{ 上にある 1 の個数}), \\
U(y) :&= \texttt{numberOfOnes}(((y)_{\mathrm{lh}(y) \dot{-} 1})_1) \\
&= (\text{計算履歴 } y \text{ の計算結果 (出力値)}).
\end{aligned}
$$

以上のように定めると

$$M(\vec{x}) \simeq U(\mu y[T_n(e, \vec{x}, y)])$$

となる．□

この定理の逆，すなわち「全ての再帰的関数は Turing 機械で計算可能」は (Church-Turing の提唱から) 直観的には当たり前である．きちんとした証明は「カウンター機械 (レジスター機械)」[2] を参照のこと．

定理 3.1 から，どんな再帰的関数も μ 演算子を一度だけ使用する形に書き直せることがわかる．

系 3.2 (再帰的関数の正規形定理 (normal form theorem)). 任意の Turing 機械 M に対し，ある自然数 $e \in \mathbb{N}$ が存在して

$$M(\vec{x}) \simeq U(\mu y[T_n(e, \vec{x}, y)])$$

となる．ここで U, T_n は M によらない原始再帰的関数・述語である．この e を M の**指標** (index)，T_n を **Kleene の T 述語** (Kleene's T predicate) という．

Turing 機械の停止問題が決定不能であることと正規形定理から，原始再帰的関数が特定の値をとるかどうかは決定不能であることがわかる．

問題 3.3 (原始再帰的関数の到達可能性問題 (reachability problem for primitive recursive functions)).

Input: 原始再帰的関数 $f\colon \mathbb{N} \to \mathbb{N}$ と自然数 $k \in \mathbb{N}$ [*6]

Question: $f(x) = k$ となるような自然数 $x \in \mathbb{N}$ は存在するか？

問題 3.4 (原始再帰的関数の等価性問題 (equality problem for primitive recursive functions)).

Input: 原始再帰的関数 $f, g\colon \mathbb{N} \to \mathbb{N}$

Question: 全ての $x \in \mathbb{N}$ に対して $f(x) = g(x)$ か？

系 3.5. 原始再帰的関数の到達可能性問題，等価性問題はともに決定不能である．

証明. 仮に到達可能性問題が決定可能だったとすると，Kleene の T 述語を関数と見た $y \mapsto T_n(e, \vec{x}, y)$ が 0 をとるかどうかも決定可能となる．このとき正規形定理から Turing 機械の停止問題が決定可能となってしまい矛盾する．

等価性問題については，$g(x)$ が特に定数関数 k の場合を考えると，これは到達可能性問題そのものであるのでやはり決定不能である．□

参考文献

[1] y., Turing 機械の変種 (2018), http://iso.2022.jp/math/undecidable-problems/files/variants-of-turing-machine.pdf.

[2] y., カウンター機械 (レジスター機械) (2018), http://iso.2022.jp/math/undecidable-problems/files/counter-machine.pdf.

[3] y., Cantor の対関数の全単射性の証明 (2016), http://iso.2022.jp/math/pairing_function.pdf.

[*6] f への入力の数を増やすほど問題は難しくなるので，f が 1 変数関数の場合に決定不能であることだけ示しておけば十分である．

[4] 新井敏康, 数学基礎論, 岩波書店, 2011.
[5] 高橋正子, 計算論 —計算可能性とラムダ計算—, 近代科学社, 1991.
[6] 高橋正子, コンピュータと数学, 朝倉書店, 2016.
[7] 藤田博司, 原始帰納的函数とアッカーマン函数 (2012), http://tenasaku.com/academia/notes/ackermann-20120302.pdf.

変更履歴

2018/06/01　公開