計算可能性理論特論１・講義ノート *†

木原 貴行
名古屋大学 情報学部・情報学研究科

最終更新日: 2017 年 12 月 13 日

目次

1 **計算モデルと計算理論** 2
1.1 チューリング機械 2
1.2 文字列書換系とモノイドの表示 8
1.3 チューリング機械を用いた関数の実装 10
1.4 万能チューリング機械と停止問題 17

2 **決定問題：解ける問題と解けない問題** 21
2.1 多対一還元 21
2.2 モノイドの語の問題 23
2.3 ポストの対応問題 26
2.4 行列のモータリティ問題 28
2.5 その他の決定問題 * 30

3 **部分組合せ代数** 32
3.1 ストリーム計算と神託機械 32
3.2 計算可枚挙集合と枚挙還元 36
3.3 部分組合せ代数 42
3.4 ラムダ計算，不動点，再帰定理 46

* 本講義ノートは，2017 年度秋 1 期開講の名古屋大学大学院情報学研究科における講義「計算可能性理論特論１」の内容をまとめたものである．
† 講義のページ： http://www.math.mi.i.nagoya-u.ac.jp/~kihara/teach.html

1　計算モデルと計算理論

1.1　チューリング機械

文字列を書くことのできるテープと，その上の文字列の読み書きをするためのヘッドを持つ機械 TM を思い浮かべよう．この機械 TM は，有限種類の状態を取ることができる．

この機械 TM の現在の状況として，テープに $\mathtt{a}_0\mathtt{a}_1\mathtt{a}_2\ldots\mathtt{a}_n$ という文字列が書かれており，ヘッドは $\mathtt{a}_k$ の位置にあり，現在の状態は q であるとしよう．機械 TM の絵を毎回書くのは面倒なので，この状況を以下によって表すこととする．

$$[\ \sqcup\mathtt{a}_0\mathtt{a}_1\mathtt{a}_2\ldots\mathtt{a}_{k-1}\ \boxed{q}\Rrightarrow\mathtt{a}_k\mathtt{a}_{k+1}\ldots\mathtt{a}_n\sqcup\ ] \tag{1}$$

両端の記号 $\sqcup$ は，テープが両方向に伸びており，何も書かれていない空白セルがずっと続いていることを暗示している．つまり，機械の動作としては，(1) の両端には，実際には空白 $\sqcup$ が延々と伸びているものと思って取り扱う．

$$\ldots\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\mathtt{a}_0\mathtt{a}_1\mathtt{a}_2\ldots\mathtt{a}_{k-1}\ \boxed{q}\Rrightarrow\mathtt{a}_k\mathtt{a}_{k+1}\ldots\mathtt{a}_n\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\ldots$$

■初期状態:　まず，我々は TM の入力テープに文字列 $\mathtt{a}_0\mathtt{a}_1\mathtt{a}_2\ldots\mathtt{a}_n$ を書き，機械 TM を動作させる．機械 TM が最初に動作するときは，初期状態 q_{init} であり，ヘッドは文字列の一番左の位置，つまり $\mathtt{a}_0$ の位置に置かれている．

$$[\ \sqcup\ \boxed{q_{\mathrm{init}}}\Rrightarrow\mathtt{a}_0\mathtt{a}_1\mathtt{a}_2\ldots\mathtt{a}_n\sqcup\ ] \tag{2}$$

■状況の遷移:　機械の動作直後は，初期状態 q_{init} にありヘッドは $\mathtt{a}_0$ を読み込んでいる．1 ステップ後には，別の状態 q_0 に変化し，ヘッドは $\mathtt{a}_0$ を別の文字 $\mathtt{b}_0$ に書き換え，左右のどちらかに 1 セルだけ動くことができる．たとえば，今回はヘッドは右に 1 セル動いたとしよう．そうすると，現在の状況は以下によって表される．

$$[\ \sqcup\mathtt{b}_0\ \boxed{q_0}\Rrightarrow\mathtt{a}_1\mathtt{a}_2\ldots\mathtt{a}_n\sqcup\ ]$$

この状況の遷移を $\delta(q_{\mathrm{init}},\mathtt{a}_0)=(q_1,\mathtt{b}_0,\mathtt{right})$ と書く．一般に，機械 TM は，現在の状態 q とヘッドが現在注目している文字 $\mathtt{a}$ のみに依存して，$\delta(q,\mathtt{a})$ によって決定される新たな状況に遷移する．たとえば，現在の状況が (1) で表されており，$\delta(q,\mathtt{a}_k)=(q',\mathtt{b}_k,\mathtt{left})$ であれば，次の時刻で TM の状態は q から q' に変化し，ヘッドはテープ上の文字 $\mathtt{a}_k$ を文字 $\mathtt{b}_k$ に書き換えたあと，1 セル左に動く．つまり，次の時刻での機械 TM の状況は以下のようになる．

$$[\ \sqcup\mathtt{a}_0\mathtt{a}_1\mathtt{a}_2\ldots\mathtt{a}_{k-2}\ \boxed{q'}\Rrightarrow\mathtt{a}_{k-1}\mathtt{b}_k\mathtt{a}_{k+1}\ldots\mathtt{a}_n\sqcup\ ]$$

また，ヘッドは両端の空白記号 $\sqcup$ のセルまで行き，そこを別の文字に書き換えてもよい．たとえば，現在の状況として，ヘッドが空白記号 $\sqcup$ を読み込んでいるとしよう．

$$[\ \sqcup\mathtt{b}_0\mathtt{b}_1\mathtt{b}_2\ldots\mathtt{b}_m\ \boxed{q}\Rrightarrow\sqcup\ ] \tag{3}$$

最初に注意したように，右端の空白記号 $\sqcup$ の更に右にも本来は空白記号が $\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup$ と連なっている．したがって，一つ空白記号を別の文字に置き換えても，次の空白記号が新たに記述される．たとえば $\delta(q, \sqcup) = (q', \mathtt{c}, \mathtt{right})$ のような遷移を考えると，次の時刻での状況は以下のようになる．

$$[\ \sqcup \mathtt{b}_0\mathtt{b}_1\mathtt{b}_2\ldots\mathtt{b}_\mathtt{m}\mathtt{c}\ \boxed{q'} \Rrightarrow \sqcup\ ] \tag{4}$$

■受理状態: 機械 TM は，受理状態 q_{end} と呼ばれる状態を持つ．受理状態に辿り着いた段階で，機械は計算を停止し，現在のテープに書かれた文字列を出力とする．たとえば，

$$[\ \sqcup \mathtt{c}_0\mathtt{c}_1\mathtt{c}_2\ldots\mathtt{c}_{\mathtt{j}-1}\ \boxed{q_{\mathrm{end}}} \Rrightarrow \mathtt{c}_\mathtt{j}\mathtt{c}_{\mathtt{j}+1}\ldots\mathtt{c}_\mathtt{s}\sqcup\ ] \tag{5}$$

という状況になったとしたら，以後は遷移を行わず，テープ上の文字列 $\mathtt{c}_0\mathtt{c}_1\mathtt{c}_2\ldots\mathtt{c}_\mathtt{s}$ が出力される．つまり，動作開始時の状況 (2) から遷移を繰り返し，状況 (5) に辿り着いた場合，機械 TM に文字列 $\mathtt{a}_0\mathtt{a}_1\mathtt{a}_2\ldots\mathtt{a}_\mathtt{n}$ を入力したら文字列 $\mathtt{c}_0\mathtt{c}_1\mathtt{c}_2\ldots\mathtt{c}_\mathtt{s}$ が出力された，ということである．

受理状態に辿り着く場合，機械 TM は計算を**停止** (*halt*) する，と言う．

定義 1.1. チューリング機械 (*Turing machine*) とは，組 $M = (\Sigma, \sqcup; Q, q_{\mathrm{init}}, q_{\mathrm{end}}; \delta, \mathsf{left}, \mathsf{right})$ で以下の条件を満たすものである．

- Σ は有限集合であり，**アルファベット** (*alphabet*) と呼ばれる．また，$\sqcup \notin \Sigma$ であり，$\sqcup$ は空白記号と呼ばれる．
- Q は有限集合であり，q_{init}, q_{end} は Q の要素である．各 $q \in Q$ は**状態** (*state*) と呼ばれ，特に q_{init} は初期状態，q_{end} は受理状態と呼ばれる．
- $\delta : Q \times (\Sigma \cup \{\sqcup\}) \to Q \times \Sigma \times \{\mathsf{left}, \mathsf{right}\}$ は遷移関数と呼ばれる．

チューリング機械による計算の厳密な定義を与える必要がある．上に記述した機械 TM による計算の遷移を厳密な形で書き下せばよい．

このために，ふたたび現在の状況が (1) である場合を考えよう．遷移が $\delta(q, \mathtt{a}_\mathtt{k}) = (r, \mathtt{b}_\mathtt{k}, \mathtt{right})$ によって与えられるとする．このとき，チューリング機械の状態は r に変化し，記号 $\mathtt{a}_\mathtt{k}$ は $\mathtt{b}_\mathtt{k}$ に書き換わり，ヘッドは 1 つ右，つまり $\mathtt{a}_{\mathtt{k}+1}$ の位置に移動する．

$$[\ \sqcup \mathtt{a}_0\mathtt{a}_1\mathtt{a}_2\ldots\mathtt{a}_{\mathtt{k}-1}\mathtt{b}_\mathtt{k}\ \boxed{r} \Rrightarrow \mathtt{a}_{\mathtt{k}+1}\ldots\mathtt{a}_\mathtt{n}\sqcup\ ]$$

もし $\delta(r, \mathtt{b}_\mathtt{k}) = (s, \mathtt{c}_\mathtt{k}, \mathtt{left})$ であれば，次にチューリング機械の状態は s に変化し，記号 $\mathtt{b}_\mathtt{k}$ は $\mathtt{c}_\mathtt{k}$ に書き換わり，ヘッドは 1 つ左，つまり $\mathtt{c}_\mathtt{k}$ の位置に移動する．

$$[\ \sqcup \mathtt{a}_0\mathtt{a}_1\mathtt{a}_2\ldots\mathtt{a}_{\mathtt{k}-1}\ \boxed{s} \Rrightarrow \mathtt{c}_\mathtt{k}\mathtt{a}_{\mathtt{k}+1}\ldots\mathtt{a}_\mathtt{n}\sqcup\ ]$$

状況を表す図式をただの文字列とみなすと，一手先の状況は，状態が書かれている箇所とその両隣だけが変化し得る．たとえば，上の 2 回の遷移において，チューリング機械の状態が書かれてい

る箇所の両隣に注目すると，以下のような文字列書換が行われていることが分かる．

$$\begin{aligned} \mathtt{a}_{k-1}\ \boxed{q}\Rrightarrow \mathtt{a}_k \ &\longrightarrow\ \mathtt{a}_{k-1}\mathtt{b}_k\ \boxed{r}\Rrightarrow \\ \mathtt{b}_k\ \boxed{r}\Rrightarrow \mathtt{a}_{k+1} \ &\longrightarrow\ \boxed{s}\Rrightarrow \mathtt{c}_k\mathtt{a}_{k+1} \end{aligned}$$

そして，チューリング機械の状態が書かれている箇所とその両隣以外の部分の文字列には変化は一切ない．ただし，ヘッドが空白を読み込んでいる場合については注意が必要である．たとえば，状況 (3) から状況 (4) への遷移では，以下のような文字列書換が行われている．

$$\mathtt{b}_m\boxed{q}\Rrightarrow\sqcup\ ]\ \longrightarrow \mathtt{b}_m\mathtt{c}\ \boxed{q}\Rrightarrow\sqcup\ ]$$

定義 1.2. チューリング機械 M の書換規則 $\longrightarrow_M$ は以下によって定義される．状態 $q\in Q$ と文字 $\mathtt{a},\mathtt{b}\in\Sigma$ が与えられたとき，

$$\mathtt{a}\ \boxed{q}\Rrightarrow\mathtt{b}\longrightarrow_M \begin{cases} \boxed{r}\Rrightarrow\mathtt{ab}' & \text{if } \delta(q,\mathtt{b})=(r,\mathtt{b}',\mathtt{left}) \\ \mathtt{ab}'\ \boxed{r}\Rrightarrow & \text{if } \delta(q,\mathtt{b})=(r,\mathtt{b}',\mathtt{right}) \end{cases}$$

$$\mathtt{a}\ \boxed{q}\Rrightarrow\sqcup\ ]\longrightarrow_M \begin{cases} \boxed{r}\Rrightarrow\mathtt{ac}\sqcup\ ] & \text{if } \delta(q,\sqcup)=(r,\mathtt{c},\mathtt{left}) \\ \mathtt{ac}\ \boxed{r}\Rrightarrow\sqcup\ ] & \text{if } \delta(q,\sqcup)=(r,\mathtt{c},\mathtt{right}) \end{cases}$$

$$[\ \boxed{q}\Rrightarrow\sqcup\mathtt{b}\longrightarrow_M \begin{cases} [\ \boxed{r}\Rrightarrow\sqcup\mathtt{cb} & \text{if } \delta(q,\sqcup)=(r,\mathtt{c},\mathtt{left}) \\ [\ \sqcup\mathtt{c}\ \boxed{r}\Rrightarrow\mathtt{b} & \text{if } \delta(q,\sqcup)=(r,\mathtt{c},\mathtt{right}) \end{cases}$$

集合 A が与えられたとき，A の要素の有限列を A 上の**語** (*word*) と呼ぶ．A 上の語全体の集合を A^* と書く ($A^{<\omega}$ と書く流儀もある)．ところで，

$$A_M=\Sigma\cup\{\sqcup,[,]\}\cup\{\boxed{q}\Rrightarrow : q\in Q\}$$

と定義すると，チューリング機械 M の各時刻での状況は，A_M 上の語として書かれていることが分かる．したがって，書換規則 $\longrightarrow_M$ は A_M^* 上の 2 項関係，つまり $\longrightarrow_M\subseteq A_M^*\times A_M^*$ である．

- $u\longrightarrow_M v$ であるとき，任意の語 $s,t\in A_M^*$ について，$sut\longrightarrow_M^1 svt$ と書く．
- 各 $s\in\mathbb{N}$ について，語 σ を s 回の書換規則 $\longrightarrow_M$ の適用によって語 τ に書き換えられるとき，$\sigma\longrightarrow_M^s\tau$ と書く．正確には，$\sigma\longrightarrow_M^s\tau$ とは，

  $$\sigma=\sigma_0\longrightarrow_M^1\sigma_1\longrightarrow_M^1\ldots\longrightarrow_M^1\sigma_{s-1}=\tau$$

  となること $(\sigma_i)_{i<s}$ が存在することとして定義する．
- 語 σ を高々有限回の書換規則 $\longrightarrow_M$ の適用によって語 τ に書き換えられるとき，$\sigma\longrightarrow_M^*\tau$ と書く．つまり，$\sigma\longrightarrow_M^*\tau$ とは，ある $s\in\mathbb{N}$ が存在して，$\sigma\longrightarrow_M^s\tau$ となることである．

つまり，σ がある時刻での M の計算の状況であり，$\sigma\longrightarrow_M^s\tau$ であれば，s ステップ後の M の計算の状況が τ であることを意味する．そして，$\sigma\longrightarrow_M^*\tau$ とは，有限時間経過後に M の計算の状況が τ となることを意味する．混乱の恐れが無い場合は，$\longrightarrow_M^1$ のことも単に $\longrightarrow_M$ と書く．

定義 1.3. チューリング機械 M に文字列 $\sigma = \mathtt{a_0a_1}\ldots\mathtt{a_n}$ を入力したときの**時刻 s での計算状況** (*configuration at stage s*) とは，次を満たす語 $\tau \in A_M^*$ である．ある $t \leqslant s$ について，

$$[\ \sqcup \boxed{q_{\mathrm{init}}} \Rrightarrow \mathtt{a_0a_1}\ldots\mathtt{a_n}\sqcup\ ] \longrightarrow_M^t \tau$$

であり，もし $t < s$ ならば，τ は $\boxed{q_{\mathrm{end}}}\Rrightarrow$ を含んでいる．この場合，$M(\mathtt{a_0a_1}\ldots\mathtt{a_n})[s] = \tau$ と書く．

つまり，$M(\mathtt{a_0a_1}\ldots\mathtt{a_n})[s]$ とは，計算開始から s ステップ後の計算状況を表す．ただし，ある $t < s$ ステップで計算が終了している場合，時刻 t 時点での計算状況を表す，というのが後者の条件が意味するものである．ここで，$M(\mathtt{a_0a_1}\ldots\mathtt{a_n})[s] = \tau$ という記法は，各時刻での計算状況が一意に定まることから正当化される．つまり，定義 1.2 を眺めると容易に分かると思うが，$\sigma \in A_M^*$ がチューリング機械 M のある計算状況を表す文字列であれば，$\sigma \longrightarrow_M^1 \tau$ なる $\tau \in A_M^*$ は高々 1 つしか存在しない．よって，M は一価関数として定義されている：

$$(\forall \mathtt{a_0a_1}\ldots\mathtt{a_n} \in \Sigma^*)(\forall s \in \mathbb{N})(\exists! \tau \in A_M^*)\ M(\mathtt{a_0a_1}\ldots\mathtt{a_n})[s] = \tau.$$

定義 1.4. チューリング機械 M が入力 $\sigma = \mathtt{a_0a_1}\ldots\mathtt{a_n} \in \Sigma^*$ に対して計算を**停止** (*halt*) するとは，ある $s \in \mathbb{N}$ について，$M(\sigma)[s]$ が $\boxed{q_{\mathrm{end}}}\Rrightarrow$ を含むことを意味する．言い換えれば，$\boxed{q_{\mathrm{end}}}\Rrightarrow$ を含む語 $\tau \in A_M^*$ で，

$$[\ \sqcup \boxed{q_{\mathrm{init}}} \Rrightarrow \mathtt{a_0a_1}\ldots\mathtt{a_n}\sqcup\ ] \longrightarrow_M^* \tau$$

となるものが存在することである．この場合，$M(\sigma)\downarrow = \tau$ と書く．そのような τ が存在しない場合，$M(\sigma)\uparrow$ と書く．

さて，チューリング機械の計算状況の書換手続の結果，$M(\sigma)\downarrow = \tau$ となったとしよう．しかし，τ には状態記号 $\boxed{q}\Rrightarrow$, 左端の記号 [, 右端の記号], 空白記号 ␣ などの不要な情報が含まれている．これらの記号を**制御記号**と呼ぶこととしよう．また，我々は Σ の要素についても，どの記号が制御記号かを自由に指定できるとする．文字列 τ から，制御記号を全て取り除いた結果を，τ の**解釈**と呼び，$[\![\tau]\!]$ と書く．

例 1.5. 我々の定義では，チューリング機械はテープに空白記号 ␣ を書くことを許していないため，文字を削除することができない．これは，チューリング機械を定義 1.2 のような有限文字列の書換操作として表現するための都合上のものである．実際には，チューリング機械を文字列書換操作として表現する数学的理由はあまりないので，空白記号の書込みを許すように定義を訂正してもよい．しかし，チューリング機械を文字列書換操作として表現した場合の恩恵もある．このため，たとえば，アルファベット Σ の中に新しい文字 _ を加えて，これを制御記号として指定しよう．こうすれば，出力文字列 τ の解釈 $[\![\tau]\!]$ は，制御記号 _ が全て取り除かれたものとなる．つまり，記

号 _ は空白を意味するものとして解釈される．

定義 1.6. チューリング機械 $M = (\Sigma'; Q; \delta)$ が与えられているとする．任意の $\sigma \in \Sigma^*$ について，以下によって $[\![M]\!](\sigma)$ を定義する．

$$[\![M]\!](\sigma) = \begin{cases} [\![M(\sigma)]\!] & \text{if } M(\sigma)\downarrow \\ \text{未定義} & \text{if } M(\sigma)\uparrow \end{cases}$$

この $[\![M]\!]$ をチューリング機械 M の**解釈**と呼ぶ．

関数 $[\![M]\!]$ は Σ^* 全域で定義されるとは限らない．このような関数を Σ^* 上の部分関数と呼ぶ．より一般に，集合 X が与えられたとき，$D \subseteq X^k$ を定義域とする関数 $f : D \to Y$ を X から Y への**部分関数** (*partial function*) と呼ぶ．ただし，計算理論において，定義域 D を知ることは一般的には困難であり，アプリオリに D が与えられるものではない．特に，入力値の領域 X は知っているが，関数の本当の定義域 D（入力領域のうち出力が返ってくる部分）を知る術はない，ということが多々ある．このため，定義域 D に陽に言及することはせず，f が X から Y への部分関数であることを $f :\subseteq X \to Y$ と表す．たとえば，チューリング機械 $M = (\Sigma'; Q; \delta)$ は部分関数 $[\![M]\!] :\subseteq \Sigma^* \to \Sigma^*$ として解釈される．

定義 1.7. 部分関数 $f :\subseteq \Sigma^* \to \Sigma^*$ が**計算可能** (*computable*) であるとは，あるチューリング機械 M が存在して，任意の語 $\sigma \in \mathrm{dom}(f)$ に対して，$[\![M]\!](\sigma) = f(\sigma)$ となることである．

■自然数上の計算理論: ここまで文字列上の関数について議論してきたが，計算理論においては，自然数上の（多変数）部分関数も重要な考察対象である．自然数上の計算理論は，自然数を文字列でコードすることによって容易に取り扱うことができる．

自然数 $x \in \mathbb{N}$ を 2 進表記したものを $\mathtt{bin}(x)$ と表す．たとえば，$\mathtt{bin}(314) = \mathtt{100111010}$ である．自然数の有限列 $(x_1, x_2, x_3, \ldots, x_k) \in \mathbb{N}^k$ は以下のようにコードしよう．

$$\mathtt{bin}(x_1)\mathtt{,}\mathtt{bin}(x_2)\mathtt{,}\mathtt{bin}(x_3)\mathtt{,}\ldots\mathtt{,}\mathtt{bin}(x_k)$$

したがって，用いるアルファベットは $\Sigma = \{\mathtt{0}, \mathtt{1}, \mathtt{,}, \mathtt{>}, \mathtt{<}\}$ である．

定義 1.8. 部分関数 $f :\subseteq \mathbb{N}^k \to \mathbb{N}$ が**計算可能** (*computable*) であるとは，あるチューリング機械 M が存在して，任意の $(x_1, x_2, \ldots, x_k) \in \mathbb{N}^k$ に対して，$x \in \mathrm{dom}(f)$ かつ $f(x) = y$ ならば，ある $\sigma \in \Sigma^*$ について，

$$[\![M]\!](\mathtt{bin}(x_1)\mathtt{,}\mathtt{bin}(x_2)\mathtt{,}\ldots\mathtt{,}\mathtt{bin}(x_k))\downarrow = \mathtt{bin}(y)$$

となることである．

混乱の恐れがない場合は，上のような状況のときも $[\![M]\!](x_1, x_2, \ldots, x_k) = y$ と書く．

例 1.9. $f(x) = x + 1$ は計算可能である．チューリング機械 M の状態は $Q = \{q, q', q_{\mathrm{init}}, q_{\mathrm{end}}\}$ からなる．遷移関数は以下によって与えられる．

$$\begin{aligned}
\delta(q_{\mathrm{init}}, \mathtt{a}) &= (q_0, \mathtt{a}, \mathtt{left}) \\
\delta(q_r, \mathtt{a}) &= \begin{cases} (q_r, \mathtt{a}, \mathtt{right}) & \text{if } \mathtt{a} = \mathtt{0} \text{ or } \mathtt{a} = \mathtt{1} \\ (q_\ell, \mathtt{_}, \mathtt{left}) & \text{if } \mathtt{a} = \sqcup \end{cases} \\
\delta(q_\ell, \mathtt{a}) &= \begin{cases} (q_{\mathrm{end}}, \mathtt{1}, \mathtt{left}) & \text{if } \mathtt{a} = \mathtt{0} \\ (q_\ell, \mathtt{0}, \mathtt{left}) & \text{if } \mathtt{a} = \mathtt{1} \\ (q_0, \mathtt{1}, \mathtt{left}) & \text{if } \mathtt{a} = \sqcup \end{cases} \\
\delta(q_0, \mathtt{a}) &= (q_{\mathrm{end}}, \mathtt{_}, \mathtt{left}).
\end{aligned}$$

これ以外の遷移は起きないので任意でよい．実際に，計算の遷移がどのように起きるか見てみよう．たとえば $x = 19$ とすると，$\mathtt{bin}(x) = \mathtt{10011}$ である．このとき，計算は以下のように遷移する．

$$\begin{aligned}
&[\ \sqcup\ \boxed{q_{\mathrm{init}}}\Rrightarrow \mathtt{10011}\sqcup\ ] \longrightarrow_M [\ \sqcup \mathtt{1}\ \boxed{q_r}\Rrightarrow \mathtt{0011}\sqcup\ ] \longrightarrow_M [\ \sqcup \mathtt{11}\ \boxed{q_r}\Rrightarrow \mathtt{011}\sqcup\ ] \\
\longrightarrow_M^* &[\ \sqcup \mathtt{10011}\ \boxed{q_r}\Rrightarrow \sqcup\ ] \longrightarrow_M [\ \sqcup \mathtt{1001}\ \boxed{q_\ell}\Rrightarrow \mathtt{1_}\sqcup\ ] \longrightarrow_M [\ \sqcup \mathtt{100}\ \boxed{q_\ell}\Rrightarrow \mathtt{10_}\sqcup\ ] \\
\longrightarrow_M &[\ \sqcup \mathtt{10}\ \boxed{q_\ell}\Rrightarrow \mathtt{000_}\sqcup\ ] \longrightarrow_M [\ \sqcup \mathtt{1}\ \boxed{q_{\mathrm{end}}}\Rrightarrow \mathtt{0100_}\sqcup\ ]
\end{aligned}$$

$\mathtt{bin}(20) = \mathtt{10100}$ であるから，$[\![M]\!](19) \downarrow = 20$ である．$x = 2^n - 1$ の形の場合だけ，少し計算の遷移が異なるので，たとえば $x = 15$ としよう．このとき，$\mathtt{bin}(x) = \mathtt{1111}$ であり，計算は以下のように遷移する．

$$\begin{aligned}
&[\ \sqcup\ \boxed{q_{\mathrm{init}}}\Rrightarrow \mathtt{1111}\sqcup\ ] \longrightarrow_M [\ \sqcup \mathtt{1}\ \boxed{q_r}\Rrightarrow \mathtt{111}\sqcup\ ] \longrightarrow_M [\ \sqcup \mathtt{11}\ \boxed{q_r}\Rrightarrow \mathtt{11}\sqcup\ ] \\
\longrightarrow_M^* &[\ \sqcup \mathtt{1111}\ \boxed{q_r}\Rrightarrow \sqcup\ ] \longrightarrow_M [\ \sqcup \mathtt{111}\ \boxed{q_\ell}\Rrightarrow \mathtt{1_}\sqcup\ ] \longrightarrow_M [\ \sqcup \mathtt{11}\ \boxed{q_\ell}\Rrightarrow \mathtt{10_}\sqcup\ ] \\
\longrightarrow_M &[\ \sqcup \mathtt{1}\ \boxed{q_\ell}\Rrightarrow \mathtt{100_}\sqcup\ ] \longrightarrow_M [\ \sqcup\ \boxed{q_\ell}\Rrightarrow \mathtt{1000<}\sqcup\ ] \longrightarrow_M [\ \boxed{q_\ell}\Rrightarrow \sqcup \mathtt{0000_}\sqcup\ ] \\
\longrightarrow_M &[\ \boxed{q_0}\Rrightarrow \sqcup \mathtt{10000_}\sqcup\ ] \longrightarrow_M [\ \boxed{q_{\mathrm{end}}}\Rrightarrow \sqcup\mathtt{_10000_}\sqcup\ ]
\end{aligned}$$

$\mathtt{bin}(16) = \mathtt{10000}$ であるから，$[\![M]\!](15) \downarrow = 16$ である．一般に $[\![M]\!](x) = x + 1$ となっていることを確認することは難しくない．

演習問題 1.10. 自然数上の部分的減法 $\dot{-}$ を $x \dot{-} y = \max\{0, x - y\}$ によって定義する．つまり，

$$x \dot{-} y = \begin{cases} x - y & \textit{if } x \geqslant y \\ 0 & \textit{otherwise.} \end{cases}$$

このとき，$f(x, y) = x \dot{-} y$ が計算可能であることを示せ．

1.2 文字列書換系とモノイドの表示

定義 1.11. 文字列書換系 (*string rewriting system*) とは，集合 A と語上の関係 $\to \subseteq A^* \times A^*$ の対 $(A, \to)$ である．もし A と $\to$ が共に有限であるならば，$(A, \to)$ を有限文字列書換系と呼ぶ．

文字列書換系における $\to$ は，**書換規則** (*rewrite rules*) をリストアップしたものと考える．各規則 $u \to v$ は，与えられた語が u を部分語として含んでいるならば，u の部分を v に書き換えることができることを意味する．

例 1.12. チューリング機械 M について，定義 1.2 の $(A_M, \longrightarrow_M)$ は有限文字列書換系である．

文字列書換系は，**半群の表示** (*presentation of semigroup*)，より正確にはモノイドの表示と密接に関係している．

定義 1.13. 半群 (*semigroup*) とは，結合的な 2 変数関数 $* : S^2 \to S$ の定義された集合 S である．つまり，$*$ は以下の結合法則を満たす．

$$\forall a, b, c \in S \ ((a * b) * c = a * (b * c)).$$

例 1.14. 集合 A 上の語全体 A^* について，積 $*$ を語の結合，つまり $u * v = uv$ によって定義すると，これは半群をなす．

定義 1.15. 半群 S 上の 2 項関係 $R \subseteq S \times S$ に関する以下の性質を考える．

$$\begin{array}{rl} \text{反射律} & (\forall s \in S)\ sRs \\ \text{推移律} & (\forall s, t, u \in S)\ sRt \wedge tRu \to sRu \\ \text{積の保存} & (\forall s, t, u \in S)\ sRt \to (s * u)R(t * u) \\ & (\forall s, t, u \in S)\ sRt \to (u * s)R(u * t) \end{array}$$

半群 S 上の与えられた 2 項関係 R について，R を含み，反射律，推移律，積の保存を満たす最小の 2 項関係を R^* と書く．

例 1.16. チューリング機械 M について，$(A_M, \longrightarrow_M)$ を定義 1.2 の文字列書換系とすると，例 1.14 のように，A_M^* は半群をなし，$\longrightarrow_M$ は半群 A_M^* 上の 2 項関係となる．このとき，第 1.1 節で定義した $\longrightarrow_M^*$ と定義 1.15 の $\longrightarrow_M^*$ は一致する．

例 1.17. より一般に，$(A, \to)$ が文字列書換系であれば，以下によって定義される $\to^*$ は，定義 1.15 の $\to^*$ と一致する．

- $u \to v$ であるとき，任意の語 $s, t \in A^*$ について，$sut \to^1 svt$ と書く．この遷移 $\to^1$ は**一手書換** (*one-step rewriting*) と呼ばれる．

- 語 s を高々有限回の書換規則の適用によって語 t に書き換えられるとき，$s \to^* t$ と書く．正確には，$s \to^* t$ とは，$s = t$ または $s = s_0 \to^1 s_1 \to^1 \cdots \to^1 s_n = t$ となることとして定義する．つまり，$\to^*$ は一手書換 $\to^1$ の推移閉包である．

定義 1.18. 半群 S 上の 2 項関係 $R \subseteq S \times S$ が**合同関係** (*congruence relation*) であるとは，R が反射律，推移律，積の保存，および以下の対称律を満たすことである．

$$(\forall s, t \in S)\ sRt \ \to\ tRs.$$

例 1.19. 半群 S 上の与えられた 2 項関係 R について，sRt または tRs であるとき $sR_{\mathrm{sym}}t$ と定義する．また，例 1.17 のように R^*_{sym} を R_{sym} の推移閉包として定義する．このとき，$x \equiv_R y$ を $xR^*_{\mathrm{sym}}y$ によって定義すれば，$\equiv_R$ は R を含む最小の合同関係である．

定義 1.20. モノイド (*monoid*) とは，単位元 $\varepsilon \in S$ を持つ半群 S である．つまり，ε は以下の性質を満たす．

$$\forall a \in S\ (a * \varepsilon = \varepsilon * a = a)$$

例 1.21. 自然数上の加法 $+$ と零 0 を考えると，$(\mathbb{N}, +, 0)$ はモノイドをなす．$n \times n$ 整数行列全体の集合を $M_n(\mathbb{Z})$ と書き，$M_n(\mathbb{Z})$ 上の積 $\cdot$ と単位行列 E を考えると，$(M_n(\mathbb{Z}), \cdot, E)$ はモノイドをなす．集合 X 上の関数 $f : X \to X$ の合成 $\circ$ と恒等写像 id を考えると，$(X^X, \circ, \mathrm{id})$ はモノイドをなす．

例 1.22. 集合 A 上の語全体 A^* について，積 $*$ を語の結合，ε を空語とすれば，これはモノイドをなす．これを A 上の**自由モノイド** (*free monoid*) と呼ぶ．

定義 1.23. A^* を A 上の自由モノイドとする．関係 $R \subseteq A^* \times A^*$ が与えられたとき，$\equiv_R$ を例 1.19 の合同関係とする．このとき，$\langle A \mid R \rangle$ を商モノイド $M = A^*/\equiv_R$ の**表示** (*presentation*) と呼ぶ．もし A と R が共に有限であれば，$\langle A \mid R \rangle$ を M の**有限表示** (*finite presentation*) と呼ぶ．

文字列書換系とモノイドの表示は数学的に同一であり，同様に，有限文字列書換系はモノイドの有限表示と同一である．$(A, \to)$ を文字列書換系とすると，$\to$ は自由モノイド A^* 上の 2 項関係と思える．このとき，$\equiv$ を例 1.19 のような $\to$ を含む最小の合同関係とする．商モノイド $M = A^*/\equiv$ を考えると，各 $[s], [t] \in M$ について次が成立する．

$$[s] = [t] \iff s \equiv t \iff (\exists (s_i)_{i \leqslant k})(\exists (\sim_i)_{i \leqslant k})\ s = s_0 \sim_1 s_1 \sim_2 \cdots \sim_k s_k = t.$$

ここで $\sim_i \in \{\leftarrow, \to\}$ である．つまり，$[s] = [t]$ とは，s と t が $\leftarrow$ または $\to$ の有限ステップで繋がっているということである．同様に，群の表示もある種の文字列書換系と思うことができる．

定義 1.24. 群 (*group*) とは，任意の $x \in G$ が逆元 $x^{-1} \in G$ を持つようなモノイド G である．集合 S が生成する**自由群** (*free group*) F_S は以下によって定義される．まず，各 $s \in S$ 毎に新たな記

号 s^{-1} を用意し，$S^{-1} = \{s^{-1} : s \in S\}$ と定義する．このとき，各 $s \in S$ 毎に以下の書換規則を定義する．

$$ss^{-1} \to_S \varepsilon \qquad s^{-1}s \to_S \varepsilon$$

ここで ε は空語を表す．このとき，文字列書換系 $\langle S \cup S^{-1} \mid \to_S \rangle$ から定義される商モノイドのことを S が生成する自由群と呼び，F_S と書く．

定義 1.25. 集合 S と語上の関係 $R \subseteq (S \cup S^{-1})^*$ が与えられたとき，N_R を R が生成する F_S の正規部分群とする．このとき，$\langle S \mid R \rangle$ を商群 $G = F_S/N_R$ の**表示** (*presentation*) と呼ぶ．もし S と R が共に有限であれば，$\langle S \mid R \rangle$ を G の有限表示 (*finite presentation*) と呼ぶ．

上の定義が述べていることは，各 $s \in R$ 毎に以下の書換規則を考えることに相当する．

$$s \to_R \varepsilon$$

このとき，文字列書換系 $\langle S \cup S^{-1} \mid \to_S \cup \to_R \rangle$ が定義する商モノイドが，$\langle S \mid R \rangle$ によって表示される群 $G = F_S/N_R$ と考えてよい．

1.3 チューリング機械を用いた関数の実装

チューリング機械を使ったプログラムを書くのは極めて面倒であるし，どのような関数がチューリング機械によって計算できるか一見では明らかではない．まず，チューリング機械でのプログラミングの難しさは逐次アクセスメモリ（つまり，端から順にしかデータにアクセスできないメモリ）を取り扱っている部分による所が大きい．多くの現代人はランダムアクセスメモリ（任意の場所のデータにアクセスできる）の方に慣れ親しんでいることが，その難しさに拍車を掛ける．

ここでは，チューリング機械による計算に近く，また，ランダムアクセス機械でのプログラミングにも比較的近い，多テープ・チューリング機械を用いた疑似プログラミング言語を導入する．

■多テープ・チューリング機械: **k-テープ・チューリング機械**とは，k 個のテープを持つチューリング機械である．遷移関数が $\delta : Q \times (\Sigma \cup \{\sqcup\})^k \to Q \times (\Sigma \times \{\mathsf{left}, \mathsf{right}\})^k$ によって与えられるという点以外は，チューリング機械と変わらない．つまり，各テープ毎にヘッドがあり，現在の状態と各ヘッドが読み込んでいる文字たちの組み合わせによって，次の時点での状況が決定される．ただし，入力は第 0 テープに記述され，第 0 テープに記述したものが出力されるものとする．他のテープは計算過程で用いるのみである．

定理 1.26. 任意の自然数 $k \geqslant 1$ について，1 テープ・チューリング機械と k テープ・チューリング機械の計算能力は等しい．

演習問題 1.27. 定理 *1.26* を証明せよ．

■疑似プログラミング言語 TURING　これから，多テープ・チューリング機械を疑似プログラミング言語として取り扱う．まず，各テープは，文字列を格納する変数，すなわち String 型の変数だと考える．第 k 番目のテープが表す変数を tape[k] と書く．実際の実装には，まずプログラムを与えた後，プログラムに現れる変数の種類の数だけテープを用意してチューリング機械を構築する，という形となる．特に，プログラムを 1 つ書く毎に，チューリング機械が 1 つ必要であることに注意する．

チューリング機械の状態は，プログラムの行番号に対応する．したがって，チューリング機械の状態の集合 Q のサイズとはプログラムの行数である．ただし，我々は今後マクロを定義し，プログラムの記述にマクロを用いるため，表面上のプログラムの行数と，機械の処理上の本来のプログラムの行数にずれが生じる．実際には，チューリング機械の状態は，マクロを全て基本コマンドに置き換えてプログラムを書き直したときの行番号に対応する．

■基本コマンド:　まず，疑似プログラミング言語 TURING における基本コマンドを導入する．以下のコマンドがチューリング機械によって実装できることは明らかであろう．

```
move_left[i]; % 第 i ヘッドを 1 セル左に動かす
move_right[i]; % 第 i ヘッドを 1 セル右に動かす
tape[i](head):=a; % 第 i テープのヘッド位置の記号を a に書き換える．
```

このような基本コマンドを用いて，プログラムを記述するが，プログラムには行番号が割り当てられる．特に言及が無い場合，プログラムの上の行から順に実行していく．これは，言及が無ければ，第 n 状態 q_n の直後の時刻では第 $n+1$ 状態 q_{n+1} となっていることを意味する．しかし，チューリング機械の基本動作としては，状態 q_n の直後に q_{n+1} 以外の状態になることも許される．これは，プログラム中に goto 文が使えることを意味する．

```
goto k; % プログラムの第 k 行へ移動する．
```

また，チューリング機械の基本動作は，ある種の条件分岐文を実行するものである．まず，tape[i](head)=a を**原子式**と呼ぶ．これは，第 i テープのヘッド位置に書かれている記号が a であることを意味する．原子式 L0,L1,...,Li と基本コマンド C と D が与えられたとき，以下のコマンドも用いることができる．

```
if L0 and L1 and ... and Li then C else D;
% もし L0,L1,...,Li が全て真ならば，コマンド C を実行する．
% もし L0,L1,...,Li のどれかが偽ならば，コマンド D を実行する．
```

これは遷移関数が k 個のヘッドが読み込んでいる値の組み合わせに依存して，次の時刻での遷移を決定できることに対応する．記号の種類が有限個であるから，基本条件分岐コマンドを入れ子にして用いることによって，原子式 L の代わりに原子式の有限ブール結合を用いてもよいことも分かる．同様に，C と D の部分には，複数のコマンドを配置してもよい．つまり，C と D の部分は単独

のコマンドだけでなく，言語 TURING の任意のプログラムを記述できる．

以上に述べた 4 種類のコマンドを**言語 TURING の基本コマンド**と呼ぶ．言語 TURING における計算とは，これらの 4 種の基本コマンドを組み合わせて書かれたプログラムによる計算である．ここで，プログラムが何も書かれていない行を読み込んだとき，計算は終了する．

■応用コマンド: 十分大きな k について，コマンド goto k は計算を停止する（受理状態に移行する）命令だと思える．また，プログラムの第 k 行に goto k+1 と記述すれば，これは何もしない命令と同様である．これは，以下のコマンドが言語 TURING で実装できることを意味する．

```
end; % 計算を停止する．
nothing; % 何もしない．
```

この言語 TURING の計算能力を理解するために，まず，条件分岐と goto 文を組み合わせれば WHILE ループが作れることを確認しよう．つまり，言語 TURING において，以下のコマンドを利用できる．

```
while tape[i](head)=a { P };
% 第 i テープのヘッドの位置の記号が a である間はずっとプログラム P を繰り返す．
```

このコマンドは以下のプログラムによって実装できる．

```
00  if tape[i](head)=a then P else end;
01  goto 00;
```

同様の方法でループコマンド while tape[i](head)!=a { P } も実装できる．これは，第 i テープのヘッドの位置の記号が a になるまで，ずっとプログラム P を繰り返す命令である．

■実装の簡易化のための約束事: これから，様々な関数を言語 TURING の基本コマンドを用いて実装していこう．実装の簡便性のために，各コマンドの実行開始時点では，テープ上には常に開始記号 > と終了記号 < が書かれているとする．つまり，各テープに書かれている文字は以下のようなものである．

$$\ldots ␣␣␣ a_0 a_1 \ldots a_i > b_0 b_1 \ldots b_j < c_0 c_1 \ldots c_k ␣␣␣ \ldots$$

ここで，$a_0, \ldots a_i, b_0, \ldots, b_j$ の中に記号 $>, <, ␣$ は含まれない．このとき，このテープには $b_0 b_1 \ldots b_j$ という文字列が格納されているものと思って取り扱う．

また，各コマンドの実行終了時点でも，テープ上には必ず開始記号 > と終了記号 < が書かれており，ヘッドは開始記号 > と終了記号 < の間のどこかに位置しているような実装を行う．

■ヘッド位置初期化コマンド init_head[i]; ヘッドを開始記号 > の位置に自動的に戻すコマンドがあれば便利である．これは，開始記号 > を読み込むまでヘッドを左に動かすチューリング機械の遷移によって容易に実現できる．

```
00  while tape[i](head)!=> {
01      move_left[i];
02  }
```

■条件分岐コマンド `if tape[i]=0 then { P } else { Q };` 変数 tape[i] に格納された文字列が >0< であれば，プログラム P を実行し，さもなくばプログラム Q を実行する命令である．第 i ヘッドを開始位置 > に戻し，右 2 つを読み込んで 0< であれば P を実行し，さもなくば Q を実行することによって実装できる．

```
00  init_head[i];
01  move_right[i];
02  if tape[i](head)=0 then move_right[i] else goto 04;
03  if tape[i](head)=< then { P; end; } else goto 04;
04  Q;
```

■2 進部抽出コマンド `tape[i]:=binary(tape[j]);` 変数 tape[j] に格納された文字列から 2 進部の情報 binary(tape[j]) を読み出し，その値を変数 tape[i] に代入する命令である．たとえば，第 j テープに書かれた文字列が >001101abb< であれば，第 i テープの文字列を >001101< と書き換える．

実装方法としては，第 i と第 j ヘッドを開始位置に戻した後，第 j ヘッドが 0 と 1 以外の記号を読み込むまで両方のヘッドを右に動かす．0 と 1 以外の記号を読み込んだら，第 i ヘッドはその位置に終了記号 < を書き込み，両方のヘッドを 1 つずつ左に動かしながら，第 j テープの内容を第 i テープにコピーしていけばよい．

```
00  init_head[i];  init_head[j];
01  move_right[i];  move_right[j];
02  while tape[j](head)=0 or tape[j](head)=1 {
03      move_right[i];  move_right[j];
04  }
05  tape[i](head):=<;
06  move_left[i];  move_left[j];
07  while tape[j](head)=0 or tape[j](head)=1 {
08      tape[i](head):=tape[j](head);
09      move_left[i];  move_left[j];
10  }
```

■自然数部抽出コマンド `tape[i]:=int(tape[j]);` 変数 tape[j] に格納された文字列から自然数情報 int(tape[j]) を読み出し，その値を変数 tape[i] に代入する命令である．たとえば，第

j テープに書かれた文字列が >001101abb< であれば，これは 001101 が表す自然数 13 と考え，bin(13) = 1101 なので，第 i テープの文字列を >1101< と書き換える．

■インクリメント tape[i]++; 変数 tape[i] に格納されている自然数値を 1 加算する命令である．

```
00  init_head[i];
01  tape[i]:=int(tape[i]);
02  while tape[i](head)!=< {
03      move_right[i];
04  }
05  move_left[i];
06  while tape[i](head)!=> {
07      if tape[i](head)=1 then {
08          tape[i](head):=0;
09          move_left[i];
10      }
11      else if tape[i](head)=0 then {
12          tape[i](head):=1; end;
13      }
14      else if tape[i](head)=> then {
15          tape[i](head):=1;
16          move_left[i];
17          tape[i](head):=>; end;
18      }
19  }
```

■デクリメント tape[i]--; 変数 tape[i] に格納されている自然数値を 1 減算する命令である．ヘッドを一旦，終了記号 < の位置まで動かした後，徐々に左に戻していく．もし記号 0 が現れたら 1 に書き換えて左に動き，記号 1 が現れた時点で 0 に書き換えて計算を停止する．ただし，現れた 1 の左が開始記号 > である場合，この >1 を #> に書き換えて計算を停止する．

```
00  init_head[i];
01  tape[i]:=int(head[i]);
02  if tape[i]=0 then end; else nothing;
03  while tape[i](head)!=< {
04      move_right[i];
05  }
```

```
06  move_left[i];
07  while tape[i](head)!=> {
08      if tape[i](head)=0 then {
09          tape[i](head):=1;
10          move_left[i];
11      } else nothing;
12      if tape[i](head)=1 then {
13          move_left[i];
14          if tape[i](head)=> then {
15              tape[i](head):=#;
16              move_right[i];
17              tape[i](head):=>; end;
18          } else {
19              move_right[i];
20              tape[i](head):=0; end;
21          }
22      }
23  }
```

■ループコマンド `for tape[i] { P }`　これは変数 `tape[i]` に格納されている文字列が表す自然数の回数だけ，プログラム `P` を実行する命令である．

```
00  tape[z]:=int(tape[i]);
01  while tape[z]!=0 {
02      P;
03      tape[z]--;
04  }
```

■書換コマンド `tape[i](tape[j]):=a;`　これは変数 `tape[i]` の `int(tape[j])` 文字目の値を `a` に書き換える命令である．これは以下のように実装する．

```
00  init_head[i];
01  for tape[j] {
02      move_right[i];
03  }
04  tape[i](head):=a;
```

同様に tape[i](tape[j]):=tape[k](tape[l]); のような書換コマンドも実装できる．これは，これは変数 tape[i] の int(tape[j]) 文字目の値を変数 tape[k] の int(tape[l]) 文字目の値に書き換える命令である．

■文字列検索コマンド tape[i]:=IsSubstring(tape[j],tape[k]); 変数 tape[j] に格納された文字列が変数 tape[k] に格納された文字列の部分列であれば，tape[j] が tape[k] の何文字目から出現するかを tape[i] に書き込み，さもなくば空列を tape[i] に書き込む命令である．たとえば，tape[j] と tape[k] に格納されている文字列がそれぞれ >ab< と >cbabc< であれば，$\mathtt{bin}(2) = 10$ であるから，tape[i] には >10< と書き込む．tape[j] と tape[k] に格納されている文字列がそれぞれ >abc< と >cbab< であれば，条件を満たさないから，tape[i] には >< と書き込む．

実装のために，補助的なテープを 2 つ用意する．まず，補助変数 tape[z] には，まず tape[k] に書かれている文字列の長さを書き込む．

```
00  tape[z]:=0;
01  while tape[k](head)!=< {
02      tape[z]++;
03  }
```

各 $\ell <$ tape[z] について，$\mathtt{tape[j]}(n) = \mathtt{tape[k]}(\ell + n)$ の照合をしていけばよい．現在確認中の ℓ を格納するための変数として，tape[y] を用いる．$\mathtt{tape[y]} = \ell$ のとき，まず第 k ヘッドを ℓ セル右に動かす．その後，第 j ヘッドと第 k ヘッドが読み込んでいる値が一致するかどうかを 1 セルずつ右に動かしながら確認していく．途中で間違った値が見つかったら，変数 tape[i] に >0< を書き込み，計算を終了する．さもなくば，第 j ヘッドが終了記号 < を読み込んだ時点で，変数 tape[i] に >bin(ℓ)< を書き込み，計算を終了する．具体的な実装は次によって与えられる．

```
04  tape[y]:=0;
05  for tape[z] {
06      init_head[k];
07      for tape[y] { move_right[k]; }
08      init_head[j];
09      while tape[j](head)!=< {
10          if tape[j](head)=tape[k](head) then {
11              move_right[j]; move_right[k];
12          } else {
13              tape[i]:=0; goto 17;
14          }
15      }
16      tape[i]:=tape[y]; end;
17      tape[y]++;
```

```
18  }
```

1.4 万能チューリング機械と停止問題

チューリング機械という計算モデルは，物理的に見れば，計算する関数毎に機械を作っている．たとえば，第 1.3 節の疑似プログラミング言語 TURING について，一つ一つのプログラムがそれぞれ多テープ・チューリング機械なのである．したがって，100 個のプログラムを書くということは，100 個のチューリング機械を用意するということである．

しかし，我々の世代のコンピュータは，「たった 1 つの機械」の中で，プログラムを書くことによって，全ての計算可能関数を実装できる[*1]．このようなコンピュータは，数学的には**万能チューリング機械** (*universal Turing machine*) と呼ばれるものである．1930 年代，そのような機械がまだ存在しなかった時代，チューリングは万能機械の概念を定式化し，理論的にその存在を示した．このようにして，我々の世代が用いているようなコンピュータの到来をチューリングは予言したのである．

チューリング機械 U が**万能** (*universal*) とは，任意のチューリング機械 M に対して，

$$(\exists e \in \Sigma^*)(\forall \sigma \in \Sigma^*)\ [\![U]\!](e,\sigma) = [\![M]\!](\sigma).$$

現代的に言えば，U が我々の眼前にあるコンピュータであり，e というものは，機械 M の計算をシミュレートするプログラムである．我々のコンピュータ U の中でプログラム e に文字列 σ を入力すれば，$[\![M]\!](\sigma)$ という計算結果が得られる．現代文明の恩恵を享けている全ての人は，次の定理を体で知っているが，ここでは頭で理解することを目指そう．

定理 1.28. 万能チューリング機械が存在する．

Proof. 有限文字列書換系 $(\Sigma, \to)$ について，$\to$ は有限集合であるから，$\{u_i \to v_i : i \leqslant n\}$ のようにリストアップされていると仮定する．このとき，書換規則 $\to$ を以下の文字列でコードする．

$$u_0\texttt{@}v_0\,\texttt{|}\,u_1\texttt{@}v_1\,\texttt{|}\,u_2\texttt{@}v_2\,\texttt{|}\,\ldots\,\texttt{|}\,u_n\texttt{@}v_n$$

この文字列を $\texttt{code}(\to)$ と書く．次のような多テープ・チューリング機械 U を構成する．任意のチューリング機械 M に対して，$\longrightarrow_M$ を定義 1.2 の書換規則とすると，

$$[\![U]\!](\texttt{code}(\longrightarrow_M),\mathtt{a_0a_1\ldots a_n}) = [\![M]\!](\mathtt{a_0a_1\ldots a_n}).$$

チューリング機械 U のおおまかな構成は以下による．まず，変数 tape[0] には，最初は初期状況を表す文字列 $[\ \sqcup \boxed{q_{\text{init}}} \Rrightarrow \mathtt{a_0a_1\ldots a_n}\sqcup\ ]$ を格納しておく．各 $u \to v$ について，変数 tape[0] に格

[*1] あるいは，「たった 1 つの機械」にソフトウェアをインストールすることによって，無数のアプリケーションを実行できる．

納された文字列 w について，u が w の部分列になっているか判定する．そうであれば，w の u の部分を v に変えた列が表す数を変数 tape[0] に代入する．この手続を繰り返し，受理状態 $\boxed{q_{\mathrm{end}}}\Rightarrow$ を含む文字列が得られたら，制御記号の部分を消した残りの文字列を出力して，計算を終了する．

具体的な実装としては，以下の手続きを実行する TURING プログラムのコードを書けばよい．

1. まず，入力文字列の # より左側の文字列を変数 tape[r] に代入し，tape[0] からは消す．
2. tape[0] の文字列を [$\sqcup\boxed{q_{\mathrm{init}}}\Rightarrow \mathtt{a}_0\mathtt{a}_1\ldots\mathtt{a}_n\sqcup$] に書き換える．
3. tape[r] の @ の数をカウントして，変数 tape[z] に代入する．
4. 以下の手続きを tape[z] に格納された自然数の回数だけ繰り返す．
5. tape[y] = 0 からスタートする．
6. 第 r ヘッドを tape[y] の個数分 | を読むまで動かす．
7. 次の @ が現れるまで文字列を tape[x] に複製する．そこから，| が現れるまでの文字列を tape[w] に複製する．
8. IsSubstring(tape[x],tape[0]) を実行し，tape[x] が tape[0] の部分列になっているか調べる．
9. tape[x] が tape[0] の部分列なら，その部分を tape[w] に書き換える．3 に戻る．これを文字列内に $\boxed{q_{\mathrm{end}}}\Rightarrow$ が現れるまで繰り返す．
10. tape[x] が tape[0] の部分列でないなら，tape[y]++ して，5 に戻る．
11. tape[y] が tape[z] を越えたら，計算を終了する．
12. $\boxed{q_{\mathrm{end}}}\Rightarrow$ が現れたら，文字列部分だけ抽出して，計算を終了する．

定理 1.26 より，多テープ・チューリング機械の計算をシミュレートする 1-テープ・チューリング機械が存在するから，定理は示された． □

定理 1.28 の万能チューリング機械 U について，以後，各 $e,n\in\Sigma^*$ に対して，$[\![e]\!](n)=U(e,n)$ と定義する．$e\in\Sigma^*$ があるチューリング機械 M のコードである場合，すなわち $e=\mathtt{code}(\longrightarrow_M)$ である場合，チューリング機械 M が計算する部分関数 $[\![M]\!]:\subseteq\Sigma^*\to\Sigma^*$ (定義 1.6 を参照せよ) と部分関数 $[\![e]\!]:\subseteq\Sigma^*\to\Sigma^*$ は一致する．

万能チューリング機械の存在定理 1.28 の帰結の 1 つとして，以下の**パラメータ定理** (*parameter theorem*) または smn 定理と呼ばれるものを証明できる．

定理 1.29. 次のような計算可能関数 $s:\Sigma^*\to\Sigma^*$ が存在する．任意の $e,u,v\in\Sigma^*$ に対して，

$$[\![s(e,u)]\!](v)=[\![e]\!](u,v).$$

Proof. U を定理 1.28 の万能チューリング機械とすると，$[\![U]\!](e,u,v)=[\![e]\!](u,v)$ である．このとき，$M_{e,u}$ を入力 v に対して，$U(e,u,v)$ の計算をシミュレートするチューリング機械とする．つまり，$[\![M_{e,u}]\!](v)=[\![U]\!](e,u,v)$ である．与えられた e,u から $M_{e,u}$ のコードは容易に計算可能

であるから，s をそれを計算する関数とする．このとき，

$$[\![s(e,u)]\!](v) = [\![M_{e,u}]\!](v) = [\![U]\!](e,u,v) = [\![e]\!](u,v)$$

を得る．よって，定理は示された． □

パラメータ定理 1.29 は計算機科学における**カリー化** (*currying*) と呼ばれる操作に対応する．これについては，第??節で詳細を述べる．

■決定問題: 計算理論における重要な考察対象の 1 つは，決定問題というタイプの問題が計算可能に解けるかどうか，という点である．たとえば，与えられた 2 進文字列が素数を表しているかどうかを尋ねる問題 Prime を考えよう．もし，$\sigma \in \{0,1\}^*$ が素数を表しているなら 1, さもなくば 0 を返すチューリング機械が作れるならば，素数判定問題 Prime は計算可能に解けると言えるだろう．

このように，**決定問題** (*decision problem*) とは，各文字列 σ に対して真偽が指定された問題である．数学的には，これは集合 $Q \subseteq \Sigma^*$ またはその特性関数 $\chi_Q : \Sigma^* \to \{0,1\}$ と同一視される．たとえば，上の素数判定問題 Prime は，集合 $P = \{\mathtt{bin}(p) : p \text{ は素数}\}$ または，その特性関数

$$\chi_P(\mathtt{bin}(p)) = \begin{cases} 1 & \text{if } p \text{ は素数} \\ 0 & \text{otherwise} \end{cases}$$

として扱われる．以後は，特性関数の方もわざわざ χ_Q とは書かず，単に Q と表す．つまり，同じ記号 Q が，集合 $Q \subseteq \Sigma^*$ とその特性関数 $Q : \Sigma^* \to \{0,1\}$ の両方を表す．このような同一視をするので，「決定問題 Q が計算可能な方法で解ける」と言う代わりに，単に「決定問題 Q が計算可能である」と言う．

ところで，自然数を扱う際に毎回わざわざ $\mathtt{bin}(n)$ のように書くのは面倒である．自然数と有限アルファベット上の語を同一視してしまっても計算論的には全く影響はないので，以後は特に言及せずに $n \in \mathbb{N}$ と $\mathtt{bin}(n) \in \{0,1\}^*$ を同一視して取り扱うこととする．

この 2 つの約束事は以後ずっと用いるので，忘れないようにしよう．

- (集合と特性関数の同一視) $n \in Q$ を $Q(n) = 1$ と表し，$n \notin Q$ を $Q(n) = 0$ と表す．
- (自然数と語の同一視) 自然数 $n \in \mathbb{N}$ とその 2 進表記 $\mathtt{bin}(n) \in \{0,1\}^*$ は同一視する．

演習問題 1.30. 素数判定問題 Prime が計算可能であることを示せ．

定義 1.4 において，計算が停止する（受理状態に到達する）ことを下矢印 $\downarrow$ で表し，そうでない場合，上矢印 $\uparrow$ で表していたことを思い出そう．機械 M の**停止問題** (*halting problem*) とは，入力 n に対して，入力 n に対する M の計算が停止するか否かを判定する問題である．

$$\mathsf{Halt}_M(n) = \begin{cases} 1 & \text{if } [\![M]\!](n) \downarrow \\ 0 & \text{if } [\![M]\!](n) \uparrow . \end{cases}$$

定理 1.28 の万能チューリング機械 U について，$\mathsf{Halt} = \mathsf{Halt}_U$ と定義する．つまり，停止問題 Halt とは，

> 与えられたチューリング機械 M のコード e と入力 n に対して，入力 n に対する M の計算が停止するか否かを判定せよ

という問題である．この停止問題 $\mathsf{Halt} = \mathsf{Halt}_U$ こそが，人類が発見した最初の計算不可能な問題であり，そして計算可能性理論の始まりを告げる定理である．

定理 1.31. 停止問題 Halt は計算不可能である．

Proof. M を任意のチューリング機械とする．このとき，次のようなチューリング機械 N を考える．入力 $\sigma \in \Sigma^*$ が与えられたら，$M(\sigma,\sigma)$ の計算をシミュレートする．もし，$M(\sigma,\sigma)$ の計算が停止し，その出力が 0 を意味しているときに限り，$N(\sigma)$ は計算を停止し，適当な文字列を出力する．それ以外の場合は，計算を停止しない．つまり，$N :\subseteq \Sigma^* \to \Sigma^*$ は次の部分関数を計算する．

$$[\![N]\!](\sigma) = \begin{cases} \downarrow & \text{if } [\![M]\!](\sigma,\sigma)\downarrow = 0 \\ \uparrow & \text{otherwise.} \end{cases}$$

いま，$e = \mathsf{code}(\longrightarrow_N)$ とする．このとき，

$$\begin{aligned} [\![M]\!](e,e) = 0 &\iff [\![N]\!](e)\downarrow \iff [\![e]\!](e)\downarrow \iff \mathsf{Halt}(e,e) = 1 \\ [\![M]\!](e,e) \neq 0 &\iff [\![N]\!](e)\uparrow \iff [\![e]\!](e)\uparrow \iff \mathsf{Halt}(e,e) = 0. \end{aligned}$$

どちらにせよ，$[\![M]\!](e,e) \neq \mathsf{Halt}(e,e)$ であるから，チューリング機械 M は停止問題を計算しない．いま，M は任意であったから，停止問題が計算不可能であることが導かれる． □

ここで，定理 1.31 は単に計算不可能な問題の存在を示しているという以上の情報を含んでいる．実際，計算可能な関数の存在を示すというだけであれば，万能チューリング機械の議論などは全くもって不必要である．たとえば，高々可算個のチューリング機械しか存在しない一方，関数 $Q : \Sigma^* \to \{0,1\}$ は非可算個存在するという点に注目すればよい．それでは，定理 1.31 から得られるが，濃度に関する議論からは決して得られない情報とは何であるだろうか．それは，停止問題 Halt が「半分，計算可能」であるという点にある．

定義 1.32. 集合 $A \subseteq \mathbb{N}$ が**半計算可能** (*semicomputable*) とは，次を満たすチューリング機械 M が存在することである．

$$(\forall n \in \mathbb{N})\,[n \in A \iff [\![M]\!](n)\downarrow].$$

停止問題 Halt の半計算可能性は定義より自明である．よって，定理 1.31 の結論として，以下を得る．

系 1.33. 計算不可能だが半計算可能な集合が存在する.

豆知識. 実のところ，定義 1.32 の概念について，計算可能性理論において，「半計算可能」と呼ばれることは滅多に無い．理由として，計算可能性理論のもう少し発展的な話題では，半計算可能または**半再帰的** (*semirecursive*) という用語は，これとは全く別の概念に用いられるから，混乱の恐れがあるためであろう．しかし，本ノートでは，そのような発展的話題には触れないので，しばらくはこの用語を用いることとする．

もう少し現代的観点に立つと，定義 1.32 は，**計算可能開集合** (*computable open set*) と言った方がよい代物である．計算論を位相空間論的に理解する方法があり，その立場では，定義 1.32 は，位相空間の開集合に相当するものだからである．後のもう少し一般的な設定の下での定義 3.7 では，その言葉を採用する．

2 決定問題：解ける問題と解けない問題

定理 1.31 において，停止問題が計算不可能であることを示した．すなわち，チューリング機械 M のコードと入力 n が与えられたときに，M に n を入力した結果が停止するかどうかを計算するアルゴリズムは存在しない．

数学の世界には他にも無数の計算不可能問題が溢れている．これについては，たとえば Poonen によって書かれた “Undecidable problems: a sampler[*2]” というサーヴェイがある．著者の Poonen は，数学の広範な分野（論理学，組合せ論，行列の半群，群論，トポロジー，数論，解析学，力学系，確率論，代数幾何，代数学，ゲーム理論）からそれぞれ幾つかの具体的な計算不可能問題の例をピックアップして紹介している．

しかし，具体的な問題の計算不可能性の証明の多くは，その分野の多少の予備知識を前提とする．本節では，予備知識が殆ど不要で，計算不可能性の証明が短いものだけ幾つか紹介しよう．

2.1 多対一還元

ある問題 Q について，その計算可能な解法は存在しない，ということを示すにはどうすればよいだろうか．そのための 1 つの方法は，その問題 Q が停止問題以上に難しい，と示すことである．定理 1.31 より，停止問題は計算不可能であると知っているから，停止問題以上に難しい問題 Q も計算不可能なはずである．では，どうすれば，ある問題が別の問題より難しいと言えるだろうか．その答えの 1 つは，還元という概念によって与えられる．

定義 2.1. 集合 $A, B \subseteq \mathbb{N}$ に対して，A が B に**多対一還元** (*many-one reducible*) されるとは，ある計算可能関数 $f : \mathbb{N} \to \mathbb{N}$ が存在して，

$$n \in A \iff f(n) \in B$$

[*2] https://arxiv.org/abs/1204.0299 より入手可能.

を満たすことである．このとき，$A \leqslant_m B$ と書く．

解説. $A \leqslant_m B$ の意味とは，$n \in A$ かどうかを知るためには，$f(n) \in B$ かどうかを知っていればよい，ということである．つまり，問題 A の方が問題 B より簡単である，と考えることができる．これにより，順序 $\leqslant_m$ は，問題の難しさの度合いを表すものと思える．

命題 2.2. $A, B \subseteq \mathbb{N}$ とする．

- $A \leqslant_m B$ であり B が計算可能ならば，A も計算可能である．
- $A \leqslant_m B$ であり B が半計算可能ならば，A も半計算可能である．

Proof. $A \leqslant_m B$ ならば，ある計算可能関数 $f : \mathbb{N} \to \mathbb{N}$ が存在して，$n \in A$ と $f(n) \in B$ は同値である．つまり，集合と特性関数を同一視すれば，

$$A(n) = 1 \iff B(f(n)) = 1$$

である．計算可能関数の合成は計算可能関数であるから，B が計算可能ならば $B \circ f$ は計算可能であり，よって，A は計算可能である．

もし B が半計算可能ならば，あるチューリング機械 M が存在して，$n \in B$ であることと $[\![M]\!](n) \downarrow$ であることが同値となる．したがって，

$$n \in A \iff f(n) \in B \iff [\![M]\!](f(n)) \downarrow$$

である．計算可能関数の合成は計算可能関数であるから，$[\![M]\!] \circ f$ は計算可能なので，A は半計算可能であることが従う． □

多対一還元の概念を用いると，停止問題が，半計算可能問題の中で最も難しい問題であるということが分かる．

命題 2.3. 集合 $A \subseteq \mathbb{N}$ について，次が成立する．

$$A \text{ は半計算可能である} \iff A \leqslant_m \mathsf{Halt}.$$

Proof. Halt は半計算可能なので，$A \leqslant_m \mathsf{Halt}$ ならば，命題 2.2 より，A は半計算可能である．逆に，A が半計算可能ならば，ある $e \in \mathbb{N}$ について，$n \in A$ であることと $[\![e]\!](n) \downarrow$ であることが同値となる．つまり，$n \in A$ と $(e,n) \in \mathsf{Halt}$ が同値であるから，計算可能関数 $n \mapsto (e,n)$ によって，$A \leqslant_m \mathsf{Halt}$ が保証される． □

この性質を持ってして，停止問題 Halt は Σ_1**-完全** (Σ_1-*complete*) であると呼ばれる．

2.2 モノイドの語の問題

それでは，停止問題と多対一還元を利用した計算不可能性証明の例を与えよう．チューリング機械は文字列書換系 (定義 1.11) の一種であるから，以下のように，文字列書換系における到達可能性問題が計算不可能であることを示すことができる．

定理 2.4 (文字列書換系における計算不可能性)**.** 次のような有限文字列書換系 $(A, \to)$ と語 $v \in A^*$ が存在する．与えられた $u \in A^*$ に対して，$u \to^* v$ かどうかを判定する計算可能な方法は存在しない．つまり，

$$\mathsf{Reach}_{\to,v} = \{u \in A^* : u \to^* v\}$$

は計算不可能集合である．

Proof. まず，任意のチューリング機械 M について，$\mathsf{Halt}_M \leqslant_m \mathsf{Reach}_{\rightsquigarrow_M,v}$ となる語 v を持つような文字列書換系 $(A, \rightsquigarrow_M)$ を作りたい．チューリング機械 M について，$(A_M, \longrightarrow_M)$ を定義 1.2 の有限文字列書換系とする．まず思い付くアイデアとしては，計算が受理状態に達した時点での状況を表す文字列を v とすることである．すると，u を初期状況とする計算が停止することと $u \longrightarrow_M^* v$ であることが同値になりそうである．しかし，受理状態の状況 v は出力文字列などの情報も含んでいるので，つまり，計算が受理状態に達した時点での状況を表す文字列 v というものは一種類ではない．この問題を解決するために，受理状態の如何なる状況に到達しても，特定の文字列に必ず収束するように新たな書換規則を付け加える．

アルファベット A_M に新しい記号 $\downarrow$ を加える．$\mathtt{a} \in \Sigma \cup \{\sqcup\}$ と $\mathtt{b} \in \Sigma \cup \{\sqcup, [\}$ について，以下の 3 つの書換規則 $\rightsquigarrow$ を考える．

$$(1)\quad \boxed{q_{\mathrm{end}}}\Rrightarrow \mathtt{a} \rightsquigarrow \boxed{q_{\mathrm{end}}}\Rrightarrow \qquad (2)\quad \boxed{q_{\mathrm{end}}}\Rrightarrow\ ] \rightsquigarrow \downarrow \qquad (3)\quad \mathtt{b}\downarrow \rightsquigarrow \downarrow$$

もしチューリング機械 M が受理状態に達しているとすると，その場合の状況は

$$[\ \sqcup \mathtt{b}_0\mathtt{b}_1 \dots \mathtt{b}_{k-1}\mathtt{b}_k \boxed{q_{\mathrm{end}}}\Rrightarrow \mathtt{b}_{k+1}\mathtt{b}_{k+2}\mathtt{b}_{k+3} \dots \mathtt{b}_t \sqcup\ ]$$

という形になっているはずである．この状況を表す文字列は，規則 $\rightsquigarrow$ によって，以下のように書き換えられる．

$$\begin{aligned}
&\rightsquigarrow [\ \sqcup \mathtt{b}_0\mathtt{b}_1 \dots \mathtt{b}_{k-1}\mathtt{b}_k \boxed{q_{\mathrm{end}}}\Rrightarrow \mathtt{b}_{k+2}\mathtt{b}_{k+3} \dots \mathtt{b}_t \sqcup\ ] \\
&\rightsquigarrow [\ \sqcup \mathtt{b}_0\mathtt{b}_1 \dots \mathtt{b}_{k-1}\mathtt{b}_k \boxed{q_{\mathrm{end}}}\Rrightarrow \mathtt{b}_{k+3} \dots \mathtt{b}_t \sqcup\ ] \\
&\rightsquigarrow^* [\ \sqcup \mathtt{b}_0\mathtt{b}_1 \dots \mathtt{b}_{k-1}\mathtt{b}_k \boxed{q_{\mathrm{end}}}\Rrightarrow \sqcup\ ] \\
&\rightsquigarrow [\ \sqcup \mathtt{b}_0\mathtt{b}_1 \dots \mathtt{b}_{k-1}\mathtt{b}_k \boxed{q_{\mathrm{end}}}\Rrightarrow\ ] \\
&\rightsquigarrow [\ \sqcup \mathtt{b}_0\mathtt{b}_1 \dots \mathtt{b}_{k-1}\mathtt{b}_k\downarrow \rightsquigarrow [\ \sqcup \mathtt{b}_0\mathtt{b}_1 \dots \mathtt{b}_{k-1}\downarrow \rightsquigarrow^* [\ \sqcup \mathtt{b}_0\downarrow \\
&\rightsquigarrow [\ \sqcup\downarrow \rightsquigarrow [\ \downarrow \rightsquigarrow \downarrow
\end{aligned}$$

したがって，受理状態における状況が何であれ，文字列は ↓ に書き換えられる．書換規則 $\rightsquigarrow_M$ をチューリング機械 M の遷移規則 $\longrightarrow_M$ に上記の規則 $\rightsquigarrow$ を加えたもの，つまり $\rightsquigarrow_M = \longrightarrow_M \cup \rightsquigarrow$ と定義する．すると，入力文字列 $\mathtt{a}_0\mathtt{a}_1\dots\mathtt{a}_n \in \Sigma^*$ に対して M の計算が停止するならば，次式が成立することが分かる．

$$[\ \sqcup\ \boxed{q_{\text{init}}}\Rrightarrow \mathtt{a}_0\mathtt{a}_1\dots\mathtt{a}_n\sqcup\ ]\ \rightsquigarrow_M^*\ \downarrow$$

逆に，$[\ \sqcup\ \boxed{q_{\text{init}}}\Rrightarrow \mathtt{a}_0\mathtt{a}_1\dots\mathtt{a}_n\sqcup\ ]$ から開始して，有限回の書換規則 $\rightsquigarrow_M$ の適用で ↓ に到達したとする．初期文字列に ↓ は含まれていないから，途中の書換規則の適用で ↓ が出現したということである．書換規則 $\rightsquigarrow$ の定義を見れば，↓ が新しく現れるためには，文字列が $\boxed{q_{\text{end}}}\Rrightarrow$ を含んでいる必要があることが分かる．書換規則 $\rightsquigarrow$ が新しく $\boxed{q_{\text{end}}}\Rrightarrow$ を出現させることはないので，$\boxed{q_{\text{end}}}\Rrightarrow$ の出現は $\rightsquigarrow_M$ によってなされる．しかし，これが意味することは，上の初期文字列を初期状況とする M の計算が受理状態 q_{end} に辿り着くということである．つまり，入力文字列 $\mathtt{a}_0\mathtt{a}_1\dots\mathtt{a}_n \in \Sigma^*$ に対して M の計算が停止する．以上をまとめると，

$$\mathtt{a}_0\mathtt{a}_1\dots\mathtt{a}_n \in \mathsf{Halt}_M \iff [\ \sqcup\ \boxed{q_{\text{init}}}\Rrightarrow \mathtt{a}_0\mathtt{a}_1\dots\mathtt{a}_n\sqcup\ ] \in \mathsf{Reach}_{\rightsquigarrow_M,\downarrow}$$

を得る．よって，計算可能関数 $f : \mathtt{a}_0\mathtt{a}_1\dots\mathtt{a}_n \mapsto [\ \sqcup\ \boxed{q_{\text{init}}}\Rrightarrow \mathtt{a}_0\mathtt{a}_1\dots\mathtt{a}_n\sqcup\ ]$ によって $\mathsf{Halt}_M \leqslant_m \mathsf{Reach}_{\rightsquigarrow_M,\downarrow}$ が保証される．このとき，U を万能チューリング機械とすれば，定理 1.31 より Halt_U は計算不可能であるから，命題 2.2 より $\mathsf{Reach}_{\rightsquigarrow_U,\downarrow}$ は計算不可能である． □

定義 1.23 とその下のパラグラフで説明したように，任意のモノイドは文字列書換系として表示することができる．特に，モノイドの各元は文字列として表され，モノイドの演算は文字列操作によって取り扱うことができる．しかし，問題となるのは，表示を 1 つ固定しても，モノイドの 1 つの要素に対して，それを表す文字列は複数存在し得るという点である．モノイドなどの代数構造を文字列操作として取り扱う場合，どの文字列の組が同じ要素を表すかくらいは判定できてほしい，と考えるのは妥当な要求であるように思える．

しかし，定理 2.4 の応用として分かることは，与えられた 2 つの文字列が，モノイドの同じ要素を表すかどうかについて，アルゴリズム的に判断することは残念ながら不可能である，ということである．これを数学的に正確に述べよう．A 上の与えられた 2 項関係 R に対して，$\equiv_R$ を例 1.19 のように R を含む自由モノイド A^* 上の最小の合同関係として定義する．すると，次のような計算不可能性を得られる．

定理 2.5 (モノイドの語の問題の計算不可能性)**.** 次のような有限表示モノイド $\langle A \mid R\rangle$ と語 $v \in A^*$ が存在する．与えられた $u \in A^*$ に対して，$u \equiv_R v$ かどうかについて，計算可能な判定方法は存在しない．特に，

$$\mathsf{WP}_{\langle A|R\rangle} = \{(u, v) \in A^* \times A^* : u \equiv_R v\}$$

は計算不可能集合である．

Proof. チューリング機械 M について，$(A_M \cup \{\downarrow\}, \rightsquigarrow_M)$ を定理 2.4 の有限文字列書換系とする．第 1.2 節で述べたように，有限文字列書換系とモノイドの有限表示は同一であるから，$\langle A \mid \rightsquigarrow_M \rangle$ は有限表示モノイドであると考えられる．$\equiv_M$ を $\rightsquigarrow_M$ を含む最小の合同関係とする．

このとき，u がチューリング機械 M の計算状況を表す語であるならば，$u \equiv_M \downarrow$ であることと $u \rightsquigarrow_M^* \downarrow$ であることが同値であることを示したい．もし，$u \equiv_M \downarrow$ であれば，ある語 $u_0, \dots u_n \in A^*$ が存在して，たとえば

$$u \leftsquigarrow_M u_0 \leftsquigarrow_M u_1 \rightsquigarrow_M \cdots \leftsquigarrow_M u_{n-1} \leftsquigarrow_M u_n \rightsquigarrow_M \downarrow$$

のようになるはずである．ここで，矢印の向きはもしかすると逆かもしれない．しかし，$\rightsquigarrow_M$ の定義より，$\downarrow \rightsquigarrow_M u$ となる語 $u \in A^*$ は存在しないから，$u_n \rightsquigarrow_M \downarrow$ は確定している．

$B = \{\boxed{q}\Rightarrow : q \in Q\} \cup \{\downarrow\}$ とする．さらに，W_1 を $A_M \cup \{\downarrow\}$ の語のうち，B の記号を正確に 1 つだけ含むもの全体の集合とする．たとえば，チューリング機械 M の計算状況を表す語は必ず W_1 に含まれる．さて，書換規則 $\rightsquigarrow_M$ の左右を見ると，書換前と書換後の語で，B に属する記号の数は変わらない．特に，$u \equiv_M v$ かつ $u \in W_1$ ならば $v \in W_1$ である．さらに，書換規則 $\rightsquigarrow_M$ をよく見ると，$u \in W_1$ ならば，$u \rightsquigarrow_M v$ なるような語 v は高々 1 つしか存在しないことがわかる．たとえば $\longrightarrow_M$ については，定義 1.3 の直後のパラグラフで言及した通りであり，定理 2.4 で新たに付加した $\rightsquigarrow$ の部分については明らかであろう．

さて，$u \in W_1$ であれば，上のような $u_0, \dots, u_n$ は全て W_1 に属す．$i \neq j$ のとき，$u_i \neq u_j$ であることは仮定できる．よって，どんな i を見ても，

$$u_{i-1} \leftsquigarrow_M u_i \rightsquigarrow_M u_{i+1}$$

となることは有り得ない．これより，$u_0, \dots u_n$ の矢印の向きは必ず，

$$u \rightsquigarrow_M u_0 \rightsquigarrow_M u_1 \rightsquigarrow_M \cdots \rightsquigarrow_M u_{n-1} \rightsquigarrow_M u_n \rightsquigarrow_M \downarrow$$

となるはずである．以上より，$u \in W_1$ であれば，$u \rightsquigarrow_M^* \downarrow$ であることと $u \equiv_M \downarrow$ であることが同値であることが示された．

定理 2.4 の証明中の計算可能関数 $f : \mathtt{a_0a_1 \dots a_n} \mapsto [\ \sqcup\ \boxed{q_{\mathrm{init}}}\Rightarrow \mathtt{a_0a_1 \dots a_n}\sqcup\ ]$ について，$\mathtt{a_0a_1 \dots a_n}$ とすると，必ず $f(\mathtt{a_0a_1 \dots a_n}) \in W_1$ である．よって，

$$\mathtt{a_0a_1 \dots a_n} \in \mathsf{Halt}_M \iff f(\mathtt{a_0a_1 \dots a_n}) \rightsquigarrow_M^* \downarrow \iff f(\mathtt{a_0a_1 \dots a_n}) \equiv_M \downarrow$$

1 つ目の同値性は，定理 2.4 の証明より成立し，2 つ目の同値性は，W_1 上での $\rightsquigarrow_M^*$ と $\equiv_M$ の同値性による．よって，$\mathsf{Halt}_M \leqslant_m \{u \in A_M \cup \{\downarrow\} : u \equiv_M \downarrow\}$ となる．以上より，U を万能チューリング機械とすれば，定理 1.31 より，Halt_U は計算不可能であるから，命題 2.2 より，$\{u \in A_M \cup \{\downarrow\} : u \equiv_M \downarrow\}$ は計算不可能である． □

定理 2.5 は，モノイド M の具体的な有限表示 $\langle A \mid R \rangle$ に対して，語の問題の計算不可能性を示すものである．しかし，実は，語の問題の計算不可能性は，表示には依存しない．つまり，M のある有限表示 $\langle A \mid R \rangle$ における語の問題が計算不可能であれば，如何なる有限表示 $\langle B \mid S \rangle$ における語の問題も計算不可能である．

命題 2.6. M をモノイドとし，$\langle A \mid R\rangle$ と $\langle B \mid S\rangle$ は共に M の有限表示であるとする．このとき，次が成立する．

$$\mathsf{WP}_{\langle A|R\rangle} \equiv_m \mathsf{WP}_{\langle B|S\rangle}.$$

Proof. ϕ を $\langle A \mid R\rangle$ と $\langle B \mid S\rangle$ の間の同型とする．このとき，各 $a \in A$ について，$\phi(a) \in B^*$ である．集合 A は有限なので，$\phi \upharpoonright A$ は有限関数であるから，計算可能である．A は n 元集合 $\{a_1, \ldots, a_n\}$ であると仮定する．いま，n 個の変数 $\overline{x} = (x_1, \ldots, x_n)$ を持つ項を以下のように帰納的に定義する．各 x_i は項であり，$s(\overline{x})$ と $t(\overline{x})$ が項ならば，$s(\overline{x})$ と $t(\overline{x})$ の結合 $s(\overline{x})t(\overline{x})$ も項である．各 $w \in A^*$ は，ある項 t_w に対して，$w = t_w(a_1, \ldots, a_n)$ という形である．このとき，$\phi(w) = t_w(\phi(a_1), \ldots, \phi(a_n)) \in B^*$ である．明らかに $w \mapsto t_w(\phi(a_1), \ldots, \phi(a_n))$ は計算可能である．よって，

$$u \equiv_R v \iff t_u(\phi(a_1), \ldots, \phi(a_n)) \equiv_S t_v(\phi(a_1), \ldots, \phi(a_n))$$

が成立するから，$\mathsf{WP}_{\langle A|R\rangle} \leqslant_m \mathsf{WP}_{\langle B|S\rangle}$ が導かれる． □

2.3 ポストの対応問題

つづいて，ドミノを思い浮かべよう．ドミノというものをドミノ倒しで知った筆者は，長らく意識したことが無かったのであるが，よく見るとドミノ牌には上下に賽の目のような記号が書かれている．賽の目を書くのは面倒なので，数字で表すこととすると，$\boxed{\begin{smallmatrix}5\\3\end{smallmatrix}}$ のような感じである．

ここでは，一般化されたドミノを考えよう．一般化されたドミノ牌では，上下には賽の目の代わりに語が書かれているとする．たとえば，

$$\boxed{\begin{smallmatrix}\texttt{a}\\\texttt{abc}\end{smallmatrix}}\ \boxed{\begin{smallmatrix}\texttt{b}\\\texttt{bc}\end{smallmatrix}}\ \boxed{\begin{smallmatrix}\texttt{ba}\\\texttt{a}\end{smallmatrix}}\ \boxed{\begin{smallmatrix}\texttt{bca}\\\texttt{ab}\end{smallmatrix}}\ \boxed{\begin{smallmatrix}\texttt{domino}\\\texttt{post}\end{smallmatrix}}$$

のような感じである．このとき，次の問題を考える．

決定問題. 与えられたドミノ牌を上手く配置することによって，上部と下部の語をマッチさせることができるだろうか．ただし，同じドミノ牌は複数回使ってよいし，使わないドミノ牌があってもよい．

たとえば，上の 5 つのドミノ牌であれば，

$$\boxed{\begin{smallmatrix}\texttt{a}\\\texttt{abc}\end{smallmatrix}}\ \boxed{\begin{smallmatrix}\texttt{bca}\\\texttt{ab}\end{smallmatrix}}\ \boxed{\begin{smallmatrix}\texttt{b}\\\texttt{bc}\end{smallmatrix}}\ \boxed{\begin{smallmatrix}\texttt{bca}\\\texttt{ab}\end{smallmatrix}}\ \boxed{\begin{smallmatrix}\texttt{ba}\\\texttt{a}\end{smallmatrix}}$$

と配置すれば，上部の語，下部の語は共に abcabbcaba となり，マッチしている．ここで，ドミノ牌 $\boxed{\begin{smallmatrix}\texttt{bca}\\\texttt{ab}\end{smallmatrix}}$ を 2 回使っており，$\boxed{\begin{smallmatrix}\texttt{domino}\\\texttt{post}\end{smallmatrix}}$ は 1 回も使っていないが，そのようなことは許されている．

もう少し数学的に正確な定式化をしよう．以後 k 未満の自然数の集合を $\mathbf{k} = \{0, 1, \ldots, k-1\}$ と表す．アルファベット A を固定したとき，与えられた有限個のドミノ牌 $(u, v) =$

$\left(\boxed{\begin{matrix}u(i)\\v(i)\end{matrix}} : i < k\right) \in (A^* \times A^*)^k$ について，空でない語 $a_0 a_1 \dots a_\ell \in \mathbf{k}^*$ で

$$u(a_0)u(a_1)\dots u(a_\ell) = v(a_0)v(a_1)\dots v(a_\ell)$$

となるものが存在するだろうか．これを判定せよ，という問題が**ポストの対応問題** (*Post's correspondence problem*) である．

定理 2.7. 十分大きな $k \in \mathbb{N}$ について，k 個のドミノ牌に関するポストの対応問題 PCP_k は計算不可能である．つまり，次の集合は計算不可能である．

$$\mathsf{PCP}_k = \{(u,v) \in (A^* \times A^*)^k : (\exists a_0 a_1 \dots a_\ell \in \mathbf{k}^* \backslash \{\varepsilon\}) \\ u(a_0)u(a_1)\dots u(a_\ell) = v(a_0)v(a_1)\dots v(a_\ell)\}$$

Proof. 定理 2.4 より，有限文字列書換系 $(A, \to)$ で，$\mathsf{Reach}_\to = \{(u,v) \in A^* \times A^* : u \to^* v\}$ が計算不可能であるものが存在する．$\mathsf{Reach}_\to \leqslant_m \mathsf{PCP}_k$ を示そう．ここで，$k = 2|A| + |\to| + 2$ とする．

このために，アルファベット $A \cup \{\overline{a} : a \in A\} \cup \{\#\}$ を固定する．k 個のドミノ牌は，以下によって与えられる．与えられた語 $u, v \in A^*$ に対して，ドミノ牌 $\boxed{\begin{matrix}\#u\\\#\end{matrix}}$ と $\boxed{\begin{matrix}\#\\v\#\end{matrix}}$ を用意する．また，各 $a \in A$ について，ドミノ牌 $\boxed{\begin{matrix}a\\\overline{a}\end{matrix}}$ と $\boxed{\begin{matrix}\overline{a}\\a\end{matrix}}$ を用意する．さらに，各書換規則 $r \to s$ について，$\boxed{\begin{matrix}\overline{s}\\r\end{matrix}}$ を用意する．ここで，$s = s_0 s_1 \dots s_n \in A^*$ とすると，$\overline{s} = \overline{s_0 s_1} \dots \overline{s_n}$ である．

まず，$u \to^* v$ ならば，$u \to w_1 \to w_2 \to \cdots \to w_i \to v$ となる語の列 $(w_j)_{j \leqslant i}$ が存在する．よって，ドミノ牌を

$$\boxed{\begin{matrix}\#u\\\#\end{matrix}}\ \boxed{\begin{matrix}\overline{w_1}\\u\end{matrix}}\ \boxed{\begin{matrix}w_1\\\overline{w_1}\end{matrix}}\ \boxed{\begin{matrix}\overline{w_2}\\w_1\end{matrix}}\ \boxed{\begin{matrix}w_2\\\overline{w_2}\end{matrix}} \cdots \boxed{\begin{matrix}w_i\\\overline{w_i}\end{matrix}}\ \boxed{\begin{matrix}\overline{v}\\w_i\end{matrix}}\ \boxed{\begin{matrix}v\\\overline{v}\end{matrix}}\ \boxed{\begin{matrix}\#\\v\#\end{matrix}}$$

と配置すれば上部と下部をマッチさせられる．

逆に，上で用意したドミノ牌を上手く配置すると，上部と下部をマッチさせられると仮定する．左端が一致するドミノ牌は $\boxed{\begin{matrix}\#u\\\#\end{matrix}}$ しかないので，これが最初に配置される．このとき，上部にはバーなし文字の列 u が書かれているため，上下がマッチしているということは，下部にバーなし文字が書かれたドミノ牌が次に配置されているはずである．したがって，次に配置されている可能性のあるドミノは，$\boxed{\begin{matrix}\overline{a}\\a\end{matrix}}$ か $\boxed{\begin{matrix}\overline{s}\\r\end{matrix}}$ の形のものである．その後，下部の文字列が u になるまで，この形のドミノ牌が配置され続けているはずである．

下部の文字列で u が作られた段階で，上部に文字列 $\overline{w_1}$ が出来ていると仮定しよう．つまり，ドミノ牌を連結すれば $\boxed{\begin{matrix}\#u\overline{w_1}\\\#u\end{matrix}}$ となっている．この $\boxed{\begin{matrix}\overline{w_1}\\u\end{matrix}}$ の部分は $\boxed{\begin{matrix}\overline{a}\\a\end{matrix}}$ か $\boxed{\begin{matrix}\overline{s}\\r\end{matrix}}$ の形のドミノ牌を連結して作られたものである．このとき，$a \to^* a$ かつ $r \to^* s$ であるから，$u \to^* w_1$ が成立する．

いま，上部にはバー付き文字の列 $\overline{w_1}$ が余分に現れているので，上下がマッチしているということは，下部にバー付き文字が書かれたドミノ牌が次に配置されているはずである．したがって，続いて配置され得るドミノは，$\boxed{\begin{matrix}a\\\overline{a}\end{matrix}}$ の形のものしか有り得ない．先ほどと同様に，$\overline{w_1}$ を下部に配置

し終えるまで，この形のドミノが配置され続ける．その結果，ドミノを連結すると $\boxed{\begin{matrix}\#u\overline{w_1}w_1\\ \#u\overline{w_1}\end{matrix}}$ という形になる．そうすると，またバーなし文字の列 w_1 が余分に現れるから，次は下部にバーなし文字が書かれたドミノ牌が次に配置される．

この手続きを繰り返していくと，$w_1 \to^* w_2 \to^* \cdots \to^* w_i$ という列を次々に得て，ドミノを連結すると $\boxed{\begin{matrix}\#u\overline{w_1}w_1\overline{w_2}w_2\ldots\overline{w_i}w_i\\ \#u\overline{w_1}w_1\overline{w_2}w_2\ldots\overline{w_i}\end{matrix}}$ という形になる．さて，右端が一致するドミノは $\boxed{\begin{matrix}\#\\ v\#\end{matrix}}$ しかないので，上部と下部がマッチするためには，このドミノが最後に配置される必要がある．つまり，上部と下部がマッチする状況というのは，$\boxed{\begin{matrix}\#u\overline{w_1}w_1\overline{w_2}w_2\ldots\overline{w_i}w_i\#\\ \#u\overline{w_1}w_1\overline{w_2}w_2\ldots\overline{w_i}v\#\end{matrix}}$ という場合しか有り得ない．以上の議論より，ある語の列 $(w_j)_{j\leqslant i}$ が存在して，

$$u \to^* w_1 \to^* w_2 \to^* \cdots \to^* w_i = v$$

となるから，$u \to^* v$ となることが示された．

与えられた (u, v) に対して，上のような k 種類のドミノの列 D を作る手続きは計算可能であり，また，$u \to^* v$ であることと $D \in \mathsf{PCP}_k$ であることは同値だと示したから，$\mathsf{Reach}_{\to} \leqslant_m \mathsf{PCP}_k$ を得る．$\mathsf{Reach}_{\to}$ は計算不可能であったから，命題 2.2 より，PCP_k は計算不可能である． □

演習問題 2.8. 定理 2.7 において，アルファベット A は 2 つの記号からなるものでよいことを示せ．

2.4 行列のモータリティ問題

つづいて，行列に関する決定問題を考えよう．有限個の正方行列 $M_0, M_1, \ldots, M_k$ が与えられたとき，これらの行列を用いた有限積で零行列を作れるかどうかを判定したい．ここで，ポストの対応問題の場合と同様に，同じ行列を複数回使ってもよいし，使わない行列があってもよいものとする．簡単な例を挙げると，たとえば

$$\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \quad \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}, \quad \begin{pmatrix} 2 & 2 \\ 3 & 4 \end{pmatrix}$$

という 3 つの行列が与えられているとする．すると，以下のように零行列を作ることができる．

$$\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ -1 & 1 \end{pmatrix}^2 = O$$

一方，たとえば次のような行列が与えられているとする．

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}, \quad \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}, \quad \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}.$$

これらの行列を用いた積では決して零行列を作ることはできない．なぜなら，上の 3 つの行列の行列式はいずれも 0 でないが，行列式が 0 でない行列をいくら掛けても，決して行列式が 0 になることはないからである．

行列のモータリティの問題 (*matrix mortality problem*) とは，$d \times d$ 整数行列の有限集合 $\mathcal{M} = (M_0, \ldots, M_{m-1}) \in M_d(\mathbb{Z})^m$ が与えられたとき，$\mathcal{M}$ の中の行列の有限個の積で零行列が作

れるかどうかを判定する問題である．より正確には，$\mathbf{m} = \{0, \ldots, m-1\}$ 上の語 $w = a_0 a_1 \ldots a_s$ に対して，

$$\mathcal{M}_w = M_{a_0} M_{a_1} \cdots M_{a_s}$$

と書くとすれば，$\mathcal{M}_w = O$ となる語 $w \in \mathbf{m}^*$ が存在するかどうかを判定せよ，という問題である．ここで O は零行列を表すものとする．

より数学的に言い換えれば，与えられた有限集合 $S \subseteq M_d(\mathbb{Z})$ に対して，S が生成する乗法半群が零行列を含むか否かを判定する問題である．

定理 2.9. 十分大きな $m \in \mathbb{N}$ について，m 個の 3×3 整数行列のモータリティ問題は計算不可能である．つまり，以下の集合は計算不可能である．

$$\mathsf{MM}_m = \{\mathcal{M} \in M_3(\mathbb{Z})^m : (\exists w \in \mathbf{m}^*)\ \mathcal{M}_w = O\}.$$

Proof. 4 進アルファベット $\{\mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}\}$ 上の語 u が与えられたとき，4 進展開が u によって表される自然数を $(u)_4$ と書く．語 $u, v \in \{\mathtt{1}, \mathtt{2}, \mathtt{3}\}^*$ が与えられたとき，行列 $M_{u,v} \in M_3(\mathbb{Z})$ を以下によって定義する．

$$M_{u,v} = \begin{pmatrix} 4^{|u|} & 0 & 0 \\ 0 & 4^{|v|} & 0 \\ (u)_4 & (v)_4 & 1 \end{pmatrix}$$

この $(u, v) \mapsto M_{u,v}$ は単射モノイド準同型である．まず，単射性は明らかである．準同型性，つまり，$M_{u,v} M_{s,t} = M_{us,vt}$ であることは以下によって示される．

$$M_{u,v} M_{s,t} = \begin{pmatrix} 4^{|u|} \cdot 4^{|s|} & 0 & 0 \\ 0 & 4^{|v|} \cdot 4^{|t|} & 0 \\ 4^{|s|}(u)_4 + (s)_4 & 4^{|t|}(v)_4 + (t)_4 & 1 \end{pmatrix} = \begin{pmatrix} 4^{|us|} & 0 & 0 \\ 0 & 4^{|vt|} & 0 \\ (us)_4 & (vt)_4 & 1 \end{pmatrix} = M_{us,vt}$$

2 つめの等号については，$|u|+|s| = |us|$ であり，数の 4 進展開を考えれば，$4^{|s|}(u)_4 + (s)_4 = (us)_4$ であることから従う．つづいて，次の行列 B を考える．

$$B = \begin{pmatrix} 1 & 0 & 1 \\ -1 & 0 & -1 \\ 0 & 0 & 0 \end{pmatrix}$$

このとき，$B^2 = B$ であることは容易に確認できる．また，

$$B \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ c & d & 1 \end{pmatrix} B = \begin{pmatrix} a+b & d & 1 \\ -a-b & -d & -1 \\ 0 & 0 & 0 \end{pmatrix} B = \begin{pmatrix} a+b-d & 0 & a+b-d \\ -a-b+d & 0 & -a-b+d \\ 0 & 0 & 0 \end{pmatrix}$$

であるから，特に $BM_{u,v}B = (4^{|u|} + (u)_4 - (w)_4)B = ((\mathtt{1}u)_4 - (w)_4)B$ となる．このとき，

$$BM_{u,v}B = O \iff (\mathtt{1}u)_4 = (v)_4 \iff \mathtt{1}u = v.$$

これらの性質を利用して，$\mathsf{PCP}_k \leqslant_m \mathsf{MM}_{2k+1}$ を示す．ここで，定理 2.7 より，k はポストの対応問題 PCP_k が計算不可能であるような十分大きな k とする．ドミノに用いられるアルファベットは $\{2,3\}$ であるとする．与えられた k 個のドミノ $(u,v) = \left(\boxed{\begin{smallmatrix} u(i) \\ v(i) \end{smallmatrix}} : i < k \right)$ に対して，$2k+1$ 個の行列 $(M_i)_{i<k}$, $(M'_i)_{i<k}$ および B を考える．ここで，

$$M_i := M_{u(i),v(i)}, \quad M'_i := M_{u(i),1v(i)}.$$

まず，$(u,v) \in \mathsf{PCP}_k$ を仮定する．このとき，ある語 $w = a_0a_1\dots a_n \in \mathbf{k}^*$ でドミノの対応が作れる．これは $u(a_0)u(a_1)\dots u(a_n) = v(a_0)v(a_1)\dots v(a_n)$ を意味するので，特に $1u(a_0)u(a_1)\dots u(a_n) = 1v(a_0)v(a_1)\dots v(a_n)$ である．上で示したように，これは

$$BM_{u(a_0)\dots u(a_n),1v(a_0)\dots v(a_n)}B = BM'_{u(a_0),v(a_0)}M_{u(a_1),v(a_1)}\cdots M_{u(a_n),v(a_n)}B = O$$

を意味するので，つまり，上の $2k+1$ 個の行列の組合せで零行列を作ることができた．よって，$((M_i)_{i<k}, (M'_i)_{i<k}, B) \in \mathsf{MM}_{2k+1}$ が示された．

逆に上の $2k+1$ 個の行列のある組合せで零行列を作れると仮定しよう．この組合せは，$B^2 = B$ かつ $M_{u,v}M_{s,t} = M_{us,vt}$ であることを利用すれば，

$$BM_{s_0,t_0}BM_{s_1,t_1}BM_{s_2,t_2}B\dots BM_{s_\ell,t_\ell}B$$

という形であると仮定できる．しかし，$BM_{s,t}B$ は B のスカラー倍であるから，ある i について $1s_i = t_i$ であるときのみしか，上の組合せは零行列になることは有り得ない．上での $2k+1$ 個の行列の選び方より，s_i は $(u(j))_{j<k}$ の組合せで作られているので，$s_i \in \{2,3^*\}$ である．一方，$1s_i = t_i$ より，t_i の最初の文字は 1 である．よって，

$$M_{s_i,t_i} = M'_{r_0}M_{r_1}\cdots M_{r_j}$$

という形になっている．これより，

$$1u(r_0)u(r_1)\dots u(r_j) = 1s_i = t_i = 1v(r_0)v(r_1)\dots v(r_j)$$

であるから，語 $r_0r_1\dots r_j$ はドミノの対応を与える．

以上より，$(u,v) \in \mathsf{PCP}_k$ であることと $((M_i)_{i<k}, (M'_i)_{i<k}, B) \in \mathsf{MM}_{2k+1}$ であることの同値性が示された．還元 $(u,v) \mapsto ((M_i)_{i<k}, (M'_i)_{i<k}, B) \in \mathsf{MM}_{2k+1}$ は明らかに計算可能であるから，これより，$\mathsf{PCP}_k \leqslant_m \mathsf{MM}_{2k+1}$ を得る．よって，ポストの対応問題 PCP_k の計算不可能性と命題 2.2 より，行列のモータリティ問題 MM_{2k+1} も計算不可能であることが示された． □

2.5 その他の決定問題 *

計算不可能な問題は，数学の様々な分野で現れる．そのうちの幾つかに触れておこう．あらかじめ述べておくと，この節の内容には証明を与えない．

■語の問題とトポロジーにおける決定問題: 定理 2.5 では，モノイドの語の問題の計算不可能性を示した．これは，半群の語の問題の計算不可能性の証明にもなっている．予想は付くと思うが，群の語の問題もまた計算不可能である．しかし，その証明は意外にも非自明である．そんなに難しいというほどでもないが，本稿の主題から外れる内容となるので，証明は省略する．さて，群の語の問題は，もともとトポロジーの文脈で興味を持たれてきた．まず，有限表示群 $\langle A \mid R \rangle$ が与えられたとき，基本群が $\langle A \mid R \rangle$ と同型であるような 2 次元単体複体 $C_{A,R}$ を作ることができる．

$$\pi_1(C_{A,R}) \simeq \langle A \mid R \rangle.$$

これを利用すると，群の語の問題はトポロジーにおける次の問題に翻訳される．

定理 2.10. 与えられた 2 次元単体複体が単連結かどうかを判定する計算可能な方法は存在しない．

このような方針で，トポロジーにおいて，大体 1950 年代末頃から，たとえば次のような問題が計算不可能であることが知られるようになった．

- (マルコフ) $d \geqslant 4$ について，与えられた 2 つの d 次元多様体が同相かどうかを判定する計算可能な方法は存在しない．
- (ノヴィコフ) $d \geqslant 5$ とする．任意の d 次元多様体 M について，与えられた d 次元多様体が P と微分同相かどうかを判定する計算可能な方法は存在しない．

さて，ここまで語の問題が計算不可能であるような群について議論してきた．しかし，世の中には語の問題が計算可能であるような群もかなり多く，応用上はそのような群に注意を向けた方が生産的かもしれない．計算機科学の観点からは，**オートマトン群** (*automatic group*) と呼ばれるタイプの群が重要である．たとえば，有限生成のコセクター群やブレイド群（組み紐群）などがオートマトン群の代表例である．オートマトン群であれば，語の問題は計算可能となることが知られている．

■ヒルベルトの第 10 問題: 群の語から離れると，数学史において最も重要な決定問題は，**ヒルベルトの第 10 問題** (*Hilbert's tenth problem*) に関するものであろう．1900 年の国際数学者会議でヒルベルトが提示した問題は，ヒルベルトの 23 の問題として知られる．その中の第 10 の問題が，与えられた整係数ディオファントス方程式が整数解を持つかどうかを決定するアルゴリズムを見つけよ，というものであった．つまり，有限個の未知数を持つ整数係数多項式 $P(x_1, x_2, \ldots, x_n) = 0$ が与えられたとき，これが解を持つかどうか判定する計算可能な方法は存在するかどうか，ということを知りたい．この問題は 1970 年になってようやく，マチャセビッチによって解決が与えられた．理論的には次の概念が重要である．

定義 2.11. 集合 $A \subseteq \mathbb{N}$ が**ディオファントス的** (*Diophantin*) であるとは，ある整数係数多項式 P

が存在して，次の性質を満たすことである．

$$n \in A \iff (\exists x_1, x_2, \ldots, x_k \in \mathbb{N})\ P(n, x_1, x_2, \ldots, x_k) = 0.$$

以下の MRDP 定理が，ヒルベルトの第 10 問題の解決の鍵である．ここで MRDP は，マチャセビッチ (Yuri Matiyasevich), ロビンソン (Julia Robinson), デーヴィス (Martin Davis), パトナム (Hilary Putnam) のファミリーネームの頭文字を並べたものである．

定理 2.12 (MRDP 定理)**.** 集合 $A \subseteq \mathbb{N}$ が半計算可能であることとディオファントス的であることは同値である．

系 1.33 における停止問題をはじめとして，今や我々は沢山の計算不可能な半計算可能集合を知っている．したがって，MRDP 定理 2.12 より，計算不可能なディオファントス集合が存在する．つまり，うまく整数係数多項式 P を作ると，与えられた n について，$P(n, x_1, x_2, \ldots, x_k) = 0$ が整数解を持つかどうかを判定する計算可能な方法が存在しないことが分かる．このようにして，ヒルベルトの第 10 問題の否定的解決が与えられた．

豆知識. 数学の定理がどれくらい弱い公理系で証明可能かを決定することは，しばしば重要な応用を持つ．たとえば，もしペアノ算術を弱めた $I\Delta_0 + \Omega_1$ という公理系で MRDP 定理 2.12 を証明できることを示すことができるならば，NP=co-NP が導かれることが知られている．(……といっても計算理論への数理論理学によるアプローチというものは，計算理論の最初期の時代のポピュラーな手法のひとつであり，古い時代に大勢の手によって限界までやり尽くされている．今の時代に数理論理学的アプローチで計算量理論の大未解決問題を解こうと試みるのはあまりオススメしないかもしれない.)

3 部分組合せ代数

3.1 ストリーム計算と神託機械

ここまでに取り扱った計算モデルは，計算を終了すると計算を「停止」していた．しかし，我々の世代の慣れ親しんだコンピュータが「停止」するのは，電源を切るかフリーズしたときである．数学的には，フリーズというのは，むしろ「停止しない」ことに相当するものであり，電源が入って正常に動いている状態は，「停止」でも「非停止」でもないと言えそうである．このように「停止」も「非停止」も起きずに，コンピュータとその利用者のやり取りが延々続く，あるいはネットワークなどを介したコンピュータ間のやり取りが延々続く計算はどのようにモデル化されるだろうか．

計算可能性理論の次のステップに進むにあたって，まず重要となる概念は，**ストリーム** (*stream*) というデータ型に関する計算である．ストリームとは，次々に与えられるデータの奔流である．数学的には，

$$\mathtt{a}_0\mathtt{a}_1\mathtt{a}_2\mathtt{a}_3\ldots\mathtt{a}_n\mathtt{a}_{n+1}\ldots$$

という無限に続く文字列と同一視できるが，実際には，時間経過に従って徐々に $\mathtt{a}_0, \mathtt{a}_1, \ldots$ という

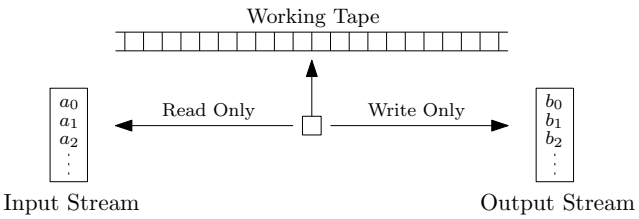

図 1　ストリーム上の関数の計算モデル

データを流し込んでくるものと思う方が都合が良い．つまり，各時刻では，ストリームは，まだ $\mathtt{a}_0\mathtt{a}_1 \ldots \mathtt{a}_k$ というような有限データしか配信していない．

ストリームは基本的には外部情報であるが，我々はストリーム型のデータを利用して計算を行うことが可能である．もう少し正確に議論するために，再びチューリング機械のようなモデルを考えよう．外部からのストリームを流し込むための専用テープを**神託テープ** (*oracle tape*) と呼ぶ．このテープは読込専用で，我々は書込不可能である．これを神託テープと呼ぶよりも，単純に，ストリーム型のデータの入力テープだと考えた方が都合がよいことも多い．

そうすると，我々もストリーム型のデータの出力を行いたい．この場合，我々はチューリング機械の計算を停止させずに，延々と稼働させつづけ，文字列を

$$\mathtt{b}_0\mathtt{b}_1\mathtt{b}_2\mathtt{b}_3 \ldots \mathtt{b}_n\mathtt{b}_{n+1} \ldots$$

というように，時間経過に沿って，次々に出力させていくこととなる．このためには，ストリーム型のデータの出力テープがあると便利である．ただし，一度配信したデータは，別の人物が既に受信しているかもしれないから，もう無かったことにはできない．この状況は，ストリーム出力テープは書込専用だが上書き不可能である，という特性によって定式化できる．

このストリーム入出力テープに加え，読込，書込，上書きが可能な**作業用テープ** (*working tape*) を合わせた計算モデルを考えよう（図 1）．これは，通常，**神託チューリング機械** (*oracle Turing machine*) と呼ばれるものに相当する．さて，この計算モデルにおいて，どのようなストリーム上の関数が計算可能であるだろうか．

この計算モデルを直接取り扱ってもよいが，ストリーム計算のモデルは，非常に単純明快な数学的記述を持つ．これを説明するために，集合 $\mathbb{A}$ を固定する．ただし，$\mathbb{A}$ として，たとえば $\{0,1\}$, Σ, $\mathbb{N}$, Σ^* などを想定している．以後，$\sigma \in \mathbb{A}^*$ が $\tau \in \mathbb{A}^*$ の**始切片** (*initial segment*) であるとき，$\sigma \preceq \tau$ と書く．つまり，$\sigma \preceq \tau$ であるとは，ある $\eta \in \mathbb{A}^*$ について，$\sigma\eta = \tau$ となることである．

定義 3.1. 部分関数 $\varphi :\subseteq \mathbb{A}^* \to \mathbb{A}^*$ が**単調** (*monotone*) とは，以下の条件を満たすことである．

$$(\forall \sigma, \tau \in \mathrm{dom}(\varphi))\ \sigma \preceq \tau \implies \varphi(\sigma) \preceq \varphi(\tau).$$

部分単調関数 $\varphi :\subseteq \mathbb{A}^* \to \mathbb{A}^*$ が与えられたとき，部分関数 $\widehat{\varphi} :\subseteq \mathbb{A}^{\mathbb{N}} \to \mathbb{A}^{\mathbb{N}}$ を次によって定義する．

$$\widehat{\varphi}(X)(n) = m \iff (\exists \sigma \prec X)\ \varphi(\sigma)(n) = m.$$

つまり，集合としては $\widehat{\varphi}(X) = \bigcup_{\sigma \prec X} \varphi(\sigma)$ によって定義する．部分関数 $\Phi :\subseteq \mathbb{A}^{\mathbb{N}} \to \mathbb{A}^{\mathbb{N}}$ が**連続** (*continuous*) とは，ある部分単調関数 φ に対して，$\Phi = \widehat{\varphi}$ が成り立つことである．

豆知識． 集合 $\mathbb{A}$ に離散位相を入れ，$\mathbb{A}^{\mathbb{N}}$ にはその積位相を入れる．このとき，$\Phi :\subseteq \mathbb{A}^{\mathbb{N}} \to \mathbb{A}^{\mathbb{N}}$ が上の意味で連続であることと，位相空間論の意味で連続であることは同値である．

注意． 定義 3.1 の単調関数は常に全域関数であると仮定してよい．なぜなら，部分単調関数 $\varphi :\subseteq \mathbb{A}^* \to \mathbb{A}^*$ が与えられたとき，$\psi(\sigma) = \bigcup\{\varphi(\tau) : \tau \preceq \sigma \text{ and } \varphi(\tau)\!\downarrow\}$ で定義すれば ψ は全域になるが，φ と ψ が定義する連続関数は同一である．

つまり，連続関数とは，有限文字列上の単調関数 $\varphi :\subseteq \mathbb{A}^* \to \mathbb{A}^*$ によって制御されるストリーム上の関数のことである．ストリームの計算処理を行う機械（神託チューリング機械）は，この有限文字列上の単調関数の部分の計算を行うのみであり，動作としては通常のチューリング機械と全く等しい．ストリーム上の関数が計算可能とは，このような機械によって計算されることである．数学的には，以下のように定式化される．

定義 3.2. 部分関数 $\Phi :\subseteq \mathbb{A}^{\mathbb{N}} \to \mathbb{A}^{\mathbb{N}}$ が**計算可能** (*computable*) または**計算可能連続** (*computably continuous*) とは，ある部分計算可能単調関数 $\varphi :\subseteq \mathbb{A}^* \to \mathbb{A}^*$ に対して，$\Phi = \widehat{\varphi}$ が成り立つことである．

注意． 定義 3.2 の単調関数も常に全域関数であると仮定できる．ただし，定義 3.1 の後の注意のような全域関数 ψ は計算可能とは限らないので，以下の修正が必要である．部分計算可能単調関数 $\varphi :\subseteq \mathbb{A}^* \to \mathbb{A}^*$ が与えられたとき，$\eta(\sigma) = \bigcup\{\varphi(\tau) : \tau \preceq \sigma \text{ and } \varphi(\tau)[|\sigma|]\!\downarrow\}$ で定義する．ここで $\varphi(\tau)[|\sigma|]\!\downarrow$ は，$\varphi(\tau)$ を計算するチューリング機械の動作が高々 $|\sigma|$ ステップで停止することを意味する．このように定義すれば，η は全域計算可能単調関数であり，$\widehat{\eta} = \widehat{\varphi}$ となる．

■神託としてのストリーム: 最初に述べたように，ストリーム計算は，現実のコンピュータのような延々やり取りの続く計算モデルを表すものと考えることもできる．しかし，歴史上は，これは神託を用いた計算という，超越的な計算を表すものとして導入された．

$\mathbb{A} = \mathbb{N}$ の場合を考えよう．入力ストリーム $g : \mathbb{N} \to \mathbb{N}$ は，何らかの**神託** (*oracle*) として得られたとしよう．すると，出力ストリーム $f = \Phi(g) : \mathbb{N} \to \mathbb{N}$ は自然数上の関数である．この関数 $f : \mathbb{N} \to \mathbb{N}$ 自体は計算不可能かもしれないが，$g : \mathbb{N} \to \mathbb{N}$ の情報を使うことによって計算できた，ということである．この状況を f は **g-相対的に計算可能** (*g-relatively computable*) または単に **g-計算可能** (*g-computable*) と呼ぶ．

定義 3.3. 関数 $f, g : \mathbb{N} \to \mathbb{N}$ が与えられたとき，f が g に**チューリング還元** (*Turing reducible*) されるとは，f が g-相対的計算可能である，すなわち $f = \Phi(g)$ なる部分計算可能関数 $\Phi :\subseteq$

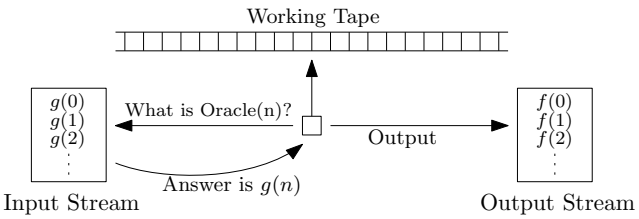

図 2　チューリング還元 $f \leqslant_T g$ の動作

$\mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ が存在することである．このとき，$f \leqslant_T g$ と書く．

神託チューリング機械を疑似プログラミング言語として表現することもできる．これは疑似プログラミング言語 TURING に「変数 y に神託テープの第 n 番目のセルに書かれた文字を入力せよ」という命令 `y:=Oracle(n)` を新たに加えたものである．これ以外の変更は一切ない．神託チューリング機械のプログラム自体は現実的に記述することができる．単純に神託テープの内容がブラックボックスなだけである．

演習問題 3.4. チューリング還元可能性関係 $\leqslant_T$ が $\mathbb{N}^{\mathbb{N}}$ 上の**前順序** (*preorder*) であることを示せ．つまり，$\leqslant_T$ が反射的 (reflexive) かつ推移的 (transitive) であることを証明せよ．

チューリング還元の簡単な具体例として，以下のようなものを証明してみよう．

命題 3.5. どんな計算可能関数よりも急増大する関数 $f \leqslant_T \mathsf{Halt}$ が存在する．つまり，ある Halt-計算可能関数 $f : \mathbb{N} \to \mathbb{N}$ が存在して，任意の計算可能関数 $g : \mathbb{N} \to \mathbb{N}$ に対して，次が成立する．

$$(\exists s \in \mathbb{N})(\forall n \geqslant s)\ g(n) < f(n).$$

Proof. 関数 $f : \mathbb{N} \to \mathbb{N}$ を次のように定義する．

$$f(n) = 1 + \max\{[\![e]\!](n) : e \leqslant n \text{ and } [\![e]\!](n)\downarrow\}.$$

ここで $\max \varnothing = 0$ とする．入力 n に対して，各 $e \leqslant n$ について $\mathtt{Oracle}(\mathtt{e},\mathtt{n}) = 1$ かどうかを尋ね，そうであるような全ての e について，$[\![e]\!][n]$ をシミュレートし，これら全ての計算が停止するのを待ち，その計算結果の中での最大値 $+1$ を出力する．神託テープに記述されているものが Halt であれば，この手続きは正しく f を計算する．よって，f は Halt-計算可能である．

また，$g : \mathbb{N} \to \mathbb{N}$ が計算可能であれば，ある e について $g = [\![e]\!]$ であるから，f の定義より，任意の $n \geqslant e$ について，$f(n) > g(n)$ となる．　□

命題 3.5 の性質を持つ関数の具体例として，**ビジービーバー関数** (*Busy Beaver function*) と呼ばれるものが知られている．

神託，つまり入力ストリームが g である場合，コード e のチューリング機械によって計算され

る関数（あるいはストリーム）を $[\![e]\!]^g$ によって表す．たとえば，単調関数 φ がコード e のチューリング機械で計算されているとき，定義 3.1 のようにして $\varphi = [\![e]\!]$ から作られる関数 $\hat{\varphi} = \widehat{[\![e]\!]}$ を考える．このとき，$\widehat{[\![e]\!]}$ に g を入力した結果 $\widehat{[\![e]\!]}(g)$ を $[\![e]\!]^g$ と表すということである．この $[\![e]\!]^g$ をコード e の g-相対的計算可能関数と思うことができる．すると，たとえば，**g-相対的停止問題** (*halting problem relative to* g) などを次のように定義できる．

$$\mathsf{Halt}^g = \{(e, n) : [\![e]\!]^g(n) \downarrow\}.$$

以下，$f <_T g$ によって，$f \leqslant_T g$ かつ $g \not\leqslant_T f$ であることを表す．

定理 3.6. 任意の $g : \mathbb{N} \to \mathbb{N}$ に対して，$g <_T \mathsf{Halt}^g$ が成り立つ．

Proof. 定理 1.31 の停止問題の計算不可能性証明を相対化すればよい． □

以後，Halt^g のことを g の**チューリング・ジャンプ** (*Turing jump*) と呼び，g' と書く．たとえば，$\varnothing''$ は停止問題に相対的な停止問題 $\mathsf{Halt}^{\mathsf{Halt}}$ を表す．定理 3.6 より，以下のようなチューリング順序関係 $\leqslant_T$ の無限増大列を得られる．

$$\varnothing <_T \mathsf{Halt} \equiv_T \varnothing' <_T \varnothing'' <_T \varnothing''' <_T \cdots.$$

最後に，チューリング還元と他の還元を比較しよう．定義 2.1 において，多対一還元可能性の概念を定義した．さて，多対一還元はチューリング還元にいくつかの制約を加えたものと思える．まず，次が成立することは容易に分かる．

$$A \leqslant_m B \implies A \leqslant_T B.$$

しかし，多対一還元では，各入力 n に対して，神託 B の一箇所 $f(n)$ の部分にしか質問をすることができない．つまり，$f(n)$ の値 k を計算するプログラムを記述した後に，命令 $\mathtt{y} := \mathtt{Oracle}(\mathtt{k})$ が一度だけ現れる．さらに，その解答 y がそのまま出力となる．一方，チューリング還元の場合は，プログラム中に $\mathtt{y} := \mathtt{Oracle}(\mathtt{n})$ の形の命令が何度も現れてよいし，クエリの解答をそのまま出力とする必要はない．実際，$A \leqslant_T B$ であっても $A \leqslant_m B$ とはならないことは，後に，系 3.15 として証明する．

3.2 計算可枚挙集合と枚挙還元

自然数の部分集合 $A \subseteq \mathbb{N}$ の枚挙をストリームの一種として考えることができる．つまり，枚挙とは，以下のように自然数を並べていくストリームである．

$$2, 3, 7, \bullet, 11, 5, 7, 13, \bullet, 19, 17, \bullet, \bullet, 23, 3, 29, \ldots$$

ここで，記号 $\bullet$ は何も並べないことを意味する．このストリームは素数の集合の枚挙のつもりである．ここで注意することは，枚挙を考える際，並べる順番は気にしないし，同じものを何度も並

べてもよいとする．したがって，たとえば素数の集合を枚挙するストリームは無限パターン存在する．

数学的な定式化を与えれば，**枚挙** (*enumeration*) というものは，関数 $p : \mathbb{N} \to \mathbf{1} + \mathbb{N}$ である．ここで $\mathbf{1} = \{\bullet\}$ であり，$\mathbf{1} + \mathbb{N}$ は $\mathbf{1}$ と $\mathbb{N}$ の和集合を表しているが，和集合の記号 $\cup$ を使わなかったのには理由があり，それは後の節で明らかになる．しかし，今回のケースでは単に

$$\mathbf{1} + \mathbb{N} = \mathbf{1} \cup \mathbb{N} = \{\bullet, 0, 1, 2, \dots\}$$

だと思っても差し支えはない．集合 $A \subseteq \mathbb{N}$ について，$\mathbf{1} + A$ も同様に，$\mathbf{1}$ と A の和集合を表すものとする．

定義 3.7. 集合 $A \subseteq \mathbb{N}$ が**計算可枚挙** (*computably enumerable*) とは，計算可能な全射 $p : \mathbb{N} \to \mathbf{1} + A$ が存在することを意味する．

豆知識. 計算可枚挙集合は，長らく**再帰的可算** (*recursively enumerable*) 集合と呼ばれてきたものである．頭文字を取って，計算量クラス RE と書かれることもあった．しかし，計算論の基礎概念について，英語圏での大規模な名称変更が前世紀末から始まり，少なくとも計算可能性理論周辺分野では，名称変更がかなり浸透している．これらの概念の和訳問題について語ることは極めて多いが，長くなるので省略する．

ところで，第 2 節では様々な決定問題を見てきた．停止問題 Halt をはじめとして，定理 2.4 の文字列書換系の到達可能性問題 $\mathsf{Reach}_{\to,v}$，定理 2.5 のモノイドの語の問題 $\mathsf{WP}_{\langle A|R\rangle}$，定理 2.7 のポストの対応問題 PCP_k，定理 2.9 の行列のモータリティ問題 MM_m はいずれも何らかの「存在」を問う．たとえば，到達可能性問題は「与えられた u に対して，u から v に辿り着くルート $u \to u_1 \to \cdots \to v$ は**存在**するか」という問題と言い換えられるし，行列のモータリティ問題は「与えられた 3×3 行列 $A_1, A_2, \dots, A_m$ を用いて，積が零行列になる組合せは**存在**するか」という問題と言い換えられる．このような「存在」を尋ねる問題は Σ_1 であると呼ばれる．

定義 3.8. 集合 $A \subseteq \mathbb{N}$ が Σ_1 とは，ある計算可能集合 $B \subseteq \mathbb{N} \times \mathbb{N}$ が存在して，次が成立することである．

$$A = \{n \in \mathbb{N} : (\exists s \in \mathbb{N})\ (n, s) \in B\}.$$

Σ_1 という性質と，半計算可能，計算可枚挙性の関連は明らかであろう．「白いカラスが存在するか」と尋ねられたとき，我々は白いカラスを見つけた段階で捜索を停止するが，白いカラスを見つけるまでは捜索は無限に終わらない．数 n を枚挙するという動作は，枚挙ストリームのどこかに n が存在していることである．念のため，厳密に証明を与えよう．

命題 3.9. 集合 $A \subseteq \mathbb{N}$ について，以下の条件は全て同値となる．

1. A は Σ_1 集合である．

2. A は半計算可能集合である．
3. A は計算可枚挙集合である．

Proof. (1)⇒(2): A が Σ_1 ならば，対応する計算可能集合 $B \subseteq \mathbb{N} \times \mathbb{N}$ を持つ．このとき，入力 n に対して，チューリング機械 M は順に

$$(n,0) \in B? \quad (n,1) \in B? \quad (n,2) \in B? \quad \dots$$

を判定していく．$(n,s) \in B$ なる $s \in \mathbb{N}$ が見つかったときに限り $M(n)$ は停止する．このとき，

$$M(n)\downarrow \iff (\exists s \in \mathbb{N})\ (n,s) \in B \iff n \in A$$

であるから，A が半計算可能であることが示された．

(2)⇒(3): 続いて，A を半計算可能であるとすると，あるチューリング機械 M が存在して，$n \in A$ と $M(n)\downarrow$ が同値になる．定義 1.3 において，$M(n)[s]$ を $M(n)$ の計算の第 s ステップでの状況を表していたことを思い出す．このとき，

$$p(n,s) = \begin{cases} n & \text{if } M(n)[s]\downarrow \\ \bullet & \text{otherwise} \end{cases}$$

と定義する．関数 $(n,s) \mapsto M(n)[s]$ は計算可能であるから，p は計算可能である．この p が A の枚挙であることは，以下により示される．

$$n \in A \iff M(n)\downarrow \iff (\exists s)\ M(n)[s]\downarrow \iff (\exists s)\ p(n,s) = n \iff (\exists a,b)\ p(a,b) = n.$$

以上より，A が計算可枚挙集合であることが示された．

(3)⇒(1): もし $A \subseteq \mathbb{N}$ が計算可枚挙集合ならば，計算可能な全射 $p : \mathbb{N} \to \mathbf{1} + A$ が存在する．このとき，$B = \{(n,s) \in \mathbb{N} \times \mathbb{N} : p(s) = n \neq \bullet\}$ とする．明らかに

$$n \in A \iff (\exists s \in \mathbb{N})\ (n,s) \in B$$

であるから，A は Σ_1 である． □

第 2.5 節で紹介した MRDP 定理 2.12 を用いれば，Σ_1 集合というものは，

言語 $\{+,\cdot,=,0,1\}$ 上の一階述語論理式で全称量化 $\forall$ を使わずに定義できる $\mathbb{N}$ の部分集合

として特徴づけられる．言い換えれば，「算術の言語を用いて，存在型の式で定義できる集合」が Σ_1 集合である．実際，Σ_1 集合は通常，言語 $\{+,\cdot,\leqslant,0,1\}$ 上の述語論理式で非有界全称量化 $\forall$ を使わずに定義できる $\mathbb{N}$ の部分集合として定義されることが多い．つまり，Σ_1 集合とは，アプリオリには計算可能性概念を伴わないものであるが，それが計算可能性理論の根幹をなす概念となるのである．

■枚挙還元:　関数 $p : \mathbb{N} \to \mathbf{1} + \mathbb{N}$ が枚挙する集合を $\mathrm{Enum}(p) = \{p(n) : n \in \mathbb{N}\} \cap \mathbb{N}$ によって定義する．以後，$\mathcal{P}\mathbb{N}$ によって，$\mathbb{N}$ の冪集合，つまり自然数の部分集合全体の集合を表す．

定義 3.10. 関数 $F : \mathcal{P}\mathbb{N} \to \mathcal{P}\mathbb{N}$ が**連続** (*continuous*) であるとは，次のような連続関数 $f : (\mathbf{1} + \mathbb{N})^{\mathbb{N}} \to (\mathbf{1} + \mathbb{N})^{\mathbb{N}}$ が存在することである．

$p \in (\mathbf{1} + \mathbb{N})^{\mathbb{N}}$ が $P \subseteq \mathbb{N}$ の枚挙ならば，$f(p) \in (\mathbf{1} + \mathbb{N})^{\mathbb{N}}$ は $F(P) \subseteq \mathbb{N}$ の枚挙である．

もし，この f が計算可能関数であれば，F は**枚挙作用素** (*enumeration operator*) と呼ばれる．

言い換えると，以下の図式が可換になっている．

$$\begin{array}{ccc} \mathcal{P}\mathbb{N} & \xrightarrow{F} & \mathcal{P}\mathbb{N} \\ {\scriptstyle \mathrm{Enum}}\uparrow & & \uparrow{\scriptstyle \mathrm{Enum}} \\ (\mathbf{1} + \mathbb{N})^{\mathbb{N}} & \xrightarrow[f]{} & (\mathbf{1} + \mathbb{N})^{\mathbb{N}} \end{array}$$

豆知識. 位相空間論を既に学んでいる人のために述べておくと，上の意味での $\mathcal{P}\mathbb{N}$ 上の連続性は，$\mathcal{P}\mathbb{N}$ 上のカントール位相での連続性でなく，$\mathcal{P}\mathbb{N}$ 上のスコット位相での連続性であることに注意する．これは，$\mathcal{P}\mathbb{N}$ を離散空間 $\mathbf{2}$ の可算積 $\mathbf{2}^{\mathbb{N}}$ ではなくシエルピンスキ空間 $\mathbb{S}$ の可算積 $\mathbb{S}^{\mathbb{N}}$ と考えることと同値である．

$\mathbb{A}^{\mathbb{N}}$ 上の連続関数と同様に，$\mathcal{P}\mathbb{N}$ 上の枚挙作用素もまた有限的に取り扱うことができる．まず，$\mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ を自然数の有限部分集合全体を表すものとすると，$\mathbb{N}$ と $\mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ を一対一に対応付けられる．たとえば，有限集合 $A \subset \mathbb{N}$ と自然数 $\sum_{n \in A} 2^n$ を同一視すればよい．この自然数を有限集合 A の**標準コード** (*canonical code*) と呼ぶ．標準コード $e \in \mathbb{N}$ の有限集合は D_e と書かれる．

連続関数 $F : \mathcal{P}\mathbb{N} \to \mathcal{P}\mathbb{N}$ の**グラフコード**とは，次によって与えられる．

$$\Gamma(F) = \{\langle n, e\rangle : n \in F(\mathsf{D}_e)\}$$

逆に集合 $\Psi \subseteq \mathbb{N}$ が与えられれば，次のように連続関数 $F_\Psi : \mathcal{P}\mathbb{N} \to \mathcal{P}\mathbb{N}$ を定義できる．

$$n \in F_\Psi(A) \iff (\exists e)\,[\langle n, e\rangle \in \Psi \text{ and } \mathsf{D}_e \subseteq A].$$

命題 3.11. 枚挙作用素 F のグラフコード $\Gamma(F)$ は半計算可能である．逆に，$\Psi \subseteq \mathbb{N}$ が半計算可能ならば，F_Ψ は枚挙作用素である．

Proof. F を枚挙作用素とすると，定義 3.10 のような計算可能関数 $f : (\mathbf{1} + \mathbb{N})^{\mathbb{N}} \to (\mathbf{1} + \mathbb{N})^{\mathbb{N}}$ が存在する．このとき，定義 3.2 とその直後の注意より，ある全域計算可能単調関数 φ について $f = \hat{\varphi}$ となる．このとき，与えられた $e \in \mathbb{N}$ に対して，有限集合 D_e の枚挙 p_e を作る機械は容易に構成できる．たとえば，D_e を小さい順に並べ上げ，全て並べ終えたら後は $\bullet$ を出力し続ければよい．つ

まり，ある計算可能関数 $p : \mathbb{N} \times \mathbb{N} \to \mathbf{1} + \mathbb{N}$ で，$p_e = p(e, \cdot)$ は D_e の枚挙であるようなものが存在する．f の定義より，$f(p_e)$ は $F(\mathsf{D}_e)$ の枚挙である．よって，$n \in F(\mathsf{D}_e)$ ならば，ある s について，$f(p_e)(s) = n$ となる．以上より，

$$\langle n, e\rangle \in \Gamma(F) \iff (\exists s \in \mathbb{N})\ f(p_e)(s) = n \iff (\exists s, t \in \mathbb{N})\ \varphi(p(e,0)p(e,1)\dots p(e,t))(s) = n$$

となるから，$\Gamma(F)$ は Σ_1 である．

逆に Ψ を半計算可能であるとする．このとき，Ψ_s を Ψ の時刻 s 近似とする．つまり，命題 3.9 の証明のように，$n \in \Psi$ と $M(n)\downarrow$ が同値であるような M を取り，$\Psi_s = \{n : M(n)[s]\downarrow\}$ と定義する．与えられた有限列 $\sigma \in (\mathbf{1} + \mathbb{N})^*$ について，D_σ を σ が枚挙する有限集合，つまり $D_\sigma = \{n \in \mathbb{N} : (\exists s \in \mathbb{N})\ \sigma(s) = n \neq \bullet\}$ とする．このとき，$|\sigma|$ によって σ の長さを表すとし，

$$n \in E_\sigma \iff (\exists e)\ [\langle n, e\rangle \in \Psi_{|\sigma|} \text{ and } \mathsf{D}_e \subseteq D_\sigma]$$

と定義すると，E_σ は有限集合であり，$E = \{(n, \sigma) : n \in E_\sigma\}$ は計算可能集合である．

次によって単調関数 φ を定義する．与えられた σ に対し，$\varphi(\sigma)$ は $E_{\sigma\restriction 0}$ の要素を小さい順に枚挙し，次に $E_{\sigma\restriction 1}$ の要素を小さい順に枚挙し，$E_{\sigma\restriction 1}$ の要素を小さい順に枚挙し，... という動作を繰り返すものとする．ここで $\sigma \restriction n$ は σ の長さ n までの制限を表す．つまり，$E_\tau = \{a_0^\tau < a_1^\tau < \cdots < a_{i(\tau)}^\tau\}$ のように E_τ を下から順に並べたとき，

$$\varphi(\sigma) = \langle a_0^{\sigma\restriction 0}, a_1^{\sigma\restriction 0}, \dots, a_{i(\sigma\restriction 0)}^{\sigma\restriction 0}, a_0^{\sigma\restriction 1}, a_1^{\sigma\restriction 1}, \dots, a_{i(\sigma\restriction 1)}^{\sigma\restriction 1}, \dots, a_0^\sigma, a_1^\sigma, \dots, a_{i(\sigma)}^\sigma\rangle$$

と定義する．φ が単調関数であることは明らかである．φ の計算可能性は，E の計算可能性から従う．また，任意の A の枚挙 p に対して，$\widehat{\varphi}(p)$ が $F_\Psi(A)$ の枚挙となっていることは容易に確認できる．したがって，F_Ψ は枚挙作用素である． □

定義 3.12. 集合 $A, B \subseteq \mathbb{N}$ について，A が B に**枚挙還元** (*enumeration reducible*) されるとは，ある枚挙作用素 $\Psi : \mathcal{P}\mathbb{N} \to \mathcal{P}\mathbb{N}$ が存在して，$\Psi(B) = A$ となることである．このとき，$A \leqslant_e B$ と書く．

枚挙還元は，自然数の部分集合の持つ正の情報の複雑さを測る概念である．つまり，$A \leqslant_e B$ とは，B の正例（B を満たす例）が全て漏れずにストリームとして次々に与えられれば，A の正例を全て並べることができる，というものである．枚挙還元とチューリング還元の関連性を明確にしておこう．集合 $A, B \subseteq \mathbb{N}$ について，$A \oplus B = \{2n : n \in A\} \cup \{2n + 1 : n \in B\}$ と定義する．また，$\overline{A}$ によって A の補集合，つまり $\mathbb{N}\backslash A$ を表すものとする．

命題 3.13. 任意の集合 $A, B \subseteq \mathbb{N}$ について，次が成立する．

$$A \leqslant_T B \iff A \oplus \overline{A} \leqslant_e B \oplus \overline{B}.$$

Proof. まず，任意の $X \subseteq \mathbb{N}$ について，次の 2 つの性質が成り立つことを確認する．

1. p が $X \oplus \overline{X}$ の枚挙ならば，$X \leqslant_T p$ である．
2. $p \leqslant_T X$ となるような $X \oplus \overline{X}$ の枚挙が存在する．

1 つめについて，p を $X \oplus \overline{X}$ の枚挙とする．$m \in X$ かどうかを知るためには $2m$ と $2m+1$ のどちらが p によって並べられるかを確認すればよい．よって，$X \leqslant_T p$ である．2 つめについて，

$$p(n) = \begin{cases} 2n & \text{if } n \in X \\ 2n+1 & \text{if } n \notin X \end{cases}$$

と定義すれば，p が $X \oplus \overline{X}$ の枚挙であることは容易に分かる．また，明らかに $p \leqslant_T X$ である．それでは，命題の証明を開始しよう．

($\Rightarrow$) $A \leqslant_T B$ を仮定する．$B \oplus \overline{B}$ の任意の枚挙から相対的に A が計算できることを示す．$A \leqslant_T B$ を保証する神託チューリング機械 M とすると，この M は，$A(n)$ の値を求めるために，計算中で神託に有限個のクエリ $B(m_0), B(m_1), \ldots, B(m_k)$ を作るかもしれない．したがって，$B \oplus \overline{B}$ の枚挙 p が与えられれば，上の性質 (1) より $B \leqslant_T p$ であるから，B への任意のクエリへの解を計算することができる．よって，A を計算することができる．上の性質 (2) の方法によって，$A \oplus \overline{A}$ を枚挙できる．この手続きを枚挙作用素として表すことは容易にできる．以上より，$A \oplus \overline{A} \leqslant_e B \oplus \overline{B}$ であることが示された．

($\Leftarrow$) 逆に $A \oplus \overline{A} \leqslant_e B \oplus \overline{B}$ を仮定する．このとき，$B \oplus \overline{B}$ の任意の枚挙を神託として $A \oplus \overline{A}$ のある枚挙を出力するチューリング機械 M が存在する．性質 (2) より $p \leqslant_T B$ となる $B \oplus \overline{B}$ の枚挙 p が存在する．このとき，M にストリーム p を入力すれば，$A \oplus \overline{A}$ を枚挙するストリーム q が出力される．特に $q \leqslant_T p$ である．性質 (1) より $A \leqslant_T q$ を得る．以上より，$A \leqslant_T q \leqslant_T p \leqslant_T B$ であるから，$A \leqslant_T B$ が示された． □

系 3.14. 任意の集合 $A \subseteq \mathbb{N}$ について，以下は同値である．

1. A は計算可能である．
2. A と $\overline{A}$ は共に計算可枚挙である．

Proof. A が計算可能であるということと $A \leqslant_T \varnothing$ であることは同値である．よって，命題 3.13 より，これは $A \oplus \overline{A} \leqslant_e \varnothing \oplus \mathbb{N}$ と同値である．しかし，$\varnothing \oplus \mathbb{N}$ は明らかに計算可枚挙であるから，枚挙作用素の定義 3.10 を考えれば，これは $A \oplus \overline{A}$ が計算可枚挙であることを意味する．つまり，A と $\overline{A}$ は共に計算可枚挙である． □

この系 3.14 は，ポストの定理として知られるものの特殊な形である．命題 3.9 より，計算可枚挙性は Σ_1 であることと同値であったから，つまり，$A \subseteq \mathbb{N}$ が計算可能であるということは，A と $\overline{A}$ が共に Σ_1 であるということである．これは，集合の計算可能性の算術の言語における特徴付けを与える．

ポストの定理の一つの簡単な帰結を与えよう．

系 3.15. $A \leqslant_T B$ だが $A \not\leqslant_m B$ であるような $A, B \subseteq \mathbb{N}$ が存在する．

Proof. $\mathsf{Halt}^{\mathsf{c}}$ を停止問題の補集合，つまり，$\mathsf{Halt}^{\mathsf{c}} = \{(e,n) : [\![e]\!](n)\downarrow\}$ とする．明らかに $\mathsf{Halt}^{\mathsf{c}} \leqslant_T \mathsf{Halt}$ である．一方，もし $\mathsf{Halt}^{\mathsf{c}} \leqslant_m \mathsf{Halt}$ であったと仮定すると，命題 2.3 より，$\mathsf{Halt}^{\mathsf{c}}$ は半計算可能となる．したがって，命題 3.9 より，Halt と Halt^{c} は共に計算可枚挙となるが，系 3.14 より，これは停止問題 Halt が計算可能であることを導く．これは，定理 1.31 の停止問題の計算不可能性に矛盾するから，$\mathsf{Halt}^{\mathsf{c}} \not\leqslant_m \mathsf{Halt}$ であることが分かった． □

3.3 部分組合せ代数

20 世紀初期頃から，多くの研究者が「計算可能」という概念を数学的に定式化するために，様々な計算モデルを考案してきた．代表的なものは，ラムダ計算，再帰関数，チューリング機械，あるいは世に溢れる数多のプログラミング言語である．これらの計算モデルの計算能力は全て一致することが判明し，ならばこれが計算可能性の妥当な定義であろう，ということで決着が付いた[*3]．これが**チャーチ・チューリングの提唱** (*Church-Turing thesis*) である．

ところで，結局のところ，計算可能性の本質とは何であろうか．計算可能性の原理をより単純な数学的構造として取り扱えないだろうか．ここでは，計算の本質を代数構造として抽出することを考えよう．まず，以下のような最もプレーンな代数構造を導入する．

定義 3.16. 集合 A と部分 2 項演算 $\cdot :\subseteq A \times A \to A$ の対は，**部分マグマ** (*partial magma*) と呼ばれる．部分マグマ A の元 $x, y \in A$ について，$x \cdot y$ の値が定義されているとき，$x \cdot y\downarrow$ と書く．そうでないときは，$x \cdot y\uparrow$ と書く．

注意. マグマは亜群 (groupoid) と呼ばれることもあるが，亜群 (groupoid) の名はより重要な別概念（全ての射が可逆である圏，つまり “partial group” とも呼べる代数構造）に用いられて紛らわしいので，ここでは用いない．ちなみに partial を「部分」と訳すと，sub- と紛らわしいので，この種の partial のことも「偏」と訳す流儀があるようである．たとえば，部分マグマでなく偏マグマと訳すような感じである．確かに，部分 (partial) 組合せ代数の部分 (sub) 代数を後に頻繁に取り扱うという点を考慮に入れると，偏組合せ代数と訳したい気持ちもある．

例 3.17. 任意の半群は部分マグマである．実際，半群とは，結合律 $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ を満たす（全域）マグマのことである．

[*3] 一方で，20 世紀中期以降の再帰理論（計算可能性理論）のメインストリームは，計算可能性の外側の研究であり，計算可能性の外側にも計算可能性に近しい秩序と構造が広がっていることを明確にした．このような研究は 1960 年代〜80 年代にかけて，集合論やモデル理論と融合しつつ，数学的に極めて深い理論として発展し，その中で最も先鋭的な理論は，かつて**一般再帰理論** (*generalized recursion theory*) の名で隆盛を極めた．

しかし，これから取り扱う具体的な構造は，いずれも結合律を満たさない．実際，計算論的にある種の良い性質を持つ部分マグマは，決して結合律を満たし得ない，ということが後に分かる．それでは，結合律を満たさない部分マグマには，たとえばどのようなものがあるだろうか．

例 3.18. $(\mathbb{Z}, -)$ や $(\mathbb{R}, \div)$ は非結合的な部分マグマである．たとえば，$(4 \div 2) \div 2 = 2 \div 2 = 1$ であるが，$4 \div (2 \div 2) = 4 \div 1 = 4$ である．

とはいえ，今後扱う部分マグマは，例 3.18 のような非結合的マグマとは大きく異なる．今後の話で念頭に置いておくとよい部分マグマは，関数適用のなす部分マグマである．

例 3.19. Sets を集合全体のクラスとする．集合 $f \in \mathsf{Sets}$ の定義域 $\mathrm{dom}(f)$ とは，$(x, y) \in f$ となる $y \in \mathsf{Sets}$ が存在するような $x \in \mathsf{Sets}$ 全体のことである．集合 $f \in \mathsf{Sets}$ が関数であるとは，任意の $x \in \mathrm{dom}(f)$ について $(x, y) \in f$ となる $y \in \mathsf{Sets}$ が唯一であることであり，このとき $f(x) = y$ と書く．このとき，$f, x \in \mathsf{Sets}$ について，

$$f \cdot x \downarrow \iff f \text{ は関数である } \& \ x \in \mathrm{dom}(f)$$

とし，また，このときの値は $f \cdot x = f(x)$ により定義する．

部分マグマは，このような関数適用のなす代数系 $(\mathsf{Sets}, \cdot)$ を念頭に置いておくとイメージしやすい．ただし，厳密には Sets 自体は集合ではないので，気になる人は，$\mathsf{Sets} = V$ の代わりに，適当な基数 κ について V_κ を考えるとよい．

さて，計算概念の代数的抽象化の議論に戻ろう．まず，「計算モデル」というからには，基本的な関数は実装できてほしい．たとえば，各 $a \in A$ に対して，定数関数 $\mathtt{const}_a : B \to A$ を実装できるべきである．さらに言えば，$a \mapsto \mathtt{const}_a$ を実装できてほしい．つまり，$\mathtt{k}(a) = \mathtt{const}_a$ となる関数 $\mathtt{k} : A \to A^B$ を実装できる．これは以下のように表される．

$$\mathtt{k}(a)(b) = a.$$

続いて，関数適用も実装できるのがよいだろう．関数のリスト $f = (f_a : B \to C)_{a \in A}$ が与えられたとき，入力リスト $x = (x_a)_{a \in A}$, $x_a \in B$, に出力リスト $(f_a(x_a))_{a \in A}$ を対応させる関数を実装したい．これは，次の関数 $\mathtt{s} : [B \to C]^A \to [B^A \to C^A]$ が実装できるということである．

$$\mathtt{s}(f)(x)(a) = (f(a))(x(a)).$$

とりあえず，我々の「計算モデル」に求めるものは以上であるとする．

定義 3.20. 部分組合せ代数 (*partial combinatory algebra*) とは，部分マグマ $(A, \cdot)$ に**コンビネータ** (*combinator*) と呼ばれる以下の特殊な元 $\mathtt{k}, \mathtt{s} \in A$ が備わったものである．

$$\begin{aligned} (\forall a, b \in A) \quad & \mathtt{k} \cdot a \downarrow \text{ and } (\mathtt{k} \cdot a) \cdot b = a \\ (\forall f, x, a \in A) \quad & (\mathtt{s} \cdot f) \cdot x \downarrow \text{ and } ((\mathtt{s} \cdot f) \cdot x) \cdot a \simeq (f \cdot a) \cdot (x \cdot a). \end{aligned}$$

ここで，$\simeq$ は強い意味での同値性，すなわち $A \cup \{\uparrow\}$-値として一致するということである．これは，上の説明で言うところの f_a が部分関数であり，$f_a(x_a)$ が定義されない場合を想定している．さらに言えば，そもそも $a \mapsto f_a$ や $a \mapsto x_a$ が部分関数である状況も考慮に入れている．

部分組合せ代数の代数的性質を見ると，単なる部分マグマよりは幾分か秩序を持つ．たとえば，部分組合せ代数は，左単位的部分マグマである．

命題 3.21. 部分組合せ代数 A は必ず左単位元 $\mathtt{i} \in A$ を持つ．つまり，ある $\mathtt{i} \in A$ が存在して，任意の $a \in A$ に対して，$\mathtt{i} \cdot a \downarrow = a$ となる．

Proof. $\mathtt{i} = (\mathtt{s} \cdot \mathtt{k}) \cdot \mathtt{k}$ と定義する．このとき，

$$\mathtt{i} \cdot a = ((\mathtt{s} \cdot \mathtt{k}) \cdot \mathtt{k}) \cdot a \simeq (\mathtt{k} \cdot a) \cdot (\mathtt{k} \cdot a) = a$$

となるから，$\mathtt{i}$ は左単位元である． □

部分組合せ代数の重要な具体例は，チューリング機械の動作から得られる．第 1.4 節で，コード e のチューリング機械によって定義される部分関数を $[\![e]\!]$ と書いていたことを思い出そう．

命題 3.22 (クリーネの第一代数)**.** $\mathbb{N}$ 上の 2 項演算を $e \cdot n = [\![e]\!](n)$ によって定義する．このとき，$\mathbb{K}_1 = (\mathbb{N}, \cdot)$ は部分組合せ代数をなす．

Proof. $\mathbb{K}_1$ が部分組合せ代数であることを確かめるために，まず $\mathtt{k}$-コンビネータを構成しよう．まず，射影 $(u, v) \mapsto u$ は計算可能なので，$[\![i]\!](u, v) = u$ となるコード i を固定する．このとき，s をパラメータ定理 1.29 の条件を満たすものとすると，$[\![s(i, a)]\!](v) \simeq [\![i]\!](a, v) = a$ である．$a \mapsto s(i, a)$ は計算可能であるから，$[\![\mathtt{k}]\!](a) = s(i, a)$ となる $\mathtt{k} \in \mathbb{N}$ が存在する．よって，

$$[\![[\![\mathtt{k}]\!](a)]\!](b) = [\![s(i, a)]\!](b) \simeq [\![i]\!](a, b) = a.$$

つづいて，$\mathtt{s}$-コンビネータを構成しよう．まず，$(f, x, a) \mapsto [\![[\![f]\!](a)]\!]([\![x]\!](a))$ は計算可能なので，このコードを e とする．s をパラメータ定理 1.29 の条件を満たすものとすると，$[\![s(e, f, x)]\!](a) \simeq [\![e]\!](f, x, a)$ である．$(f, x) \mapsto s(e, f, x)$ は計算可能であるから，再びパラメータ定理 1.29 より，$[\![[\![\mathtt{s}]\!](f)]\!](x) \simeq s(e, f, x)$ なる $\mathtt{s} \in \mathbb{N}$ が存在する．よって，

$$[\![[\![[\![\mathtt{s}]\!](f)]\!](x)]\!](a) \simeq [\![s(e, f, x)]\!](a) \simeq [\![e]\!](f, x, a) \simeq [\![[\![f]\!](a)]\!]([\![x]\!](a)).$$

以上より，$\mathbb{K}_1$ が部分組合せ代数であることが示された． □

命題 3.22 の部分組合せ代数 $\mathbb{K}_1 = (\mathbb{N}, \cdot)$ は**クリーネの第一代数** (*Kleene's first algebra*) と呼ばれる．自然数と有限語は同一視できるので，$\mathbb{K}_1 = (\Sigma^*, \cdot)$ と思ってもよい．他にも部分組合せ代数の具体例は多数あるので，そのうちの重要な例をいくつか紹介する．

例 3.23 (クリーネの第二代数)**.** 各 $p : \mathbb{N}^* \times \mathbb{N} \to (\mathbf{1} + \mathbb{N})$ は単調関数 $\varphi_p : \mathbb{N}^* \to \mathbb{N}^*$ を以下のように生成する．ここで，$\mathbf{1} = \{\bullet\}$ である．$\varphi_p(\sigma)$ は帰納的に定義される．もし任意の $m < n$ について $\varphi_p(\sigma)(m)$ が定義されているならば，$\varphi_p(\sigma)(n)$ を以下のように定義する．

$$\varphi_p(\sigma)(n) = \begin{cases} p(\tau, n), & \text{ここで } \tau \text{ は } p(\tau,n) \neq \bullet \text{ なる最初の } \tau \preceq \sigma \text{ である．} \\ \text{未定義}, & \text{そのような } \tau \text{ が存在しない．} \end{cases}$$

$\mathbb{N} \simeq \mathbb{N}^* \times \mathbb{N} \simeq \mathbf{1} + \mathbb{N}$ という同一視をすれば，$p \in \mathbb{N}^{\mathbb{N}}$ と思える．つまり，任意の $p \in \mathbb{N}^{\mathbb{N}}$ は上のように部分単調関数 $\varphi_p :\subseteq \mathbb{N}^* \to \mathbb{N}^*$ を定義し，よって，定義 3.1 のように部分連続関数 $\widehat{\varphi_p} :\subseteq \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ を定義する．$\mathbb{N}^{\mathbb{N}}$ 上の 2 項演算を $p \cdot x = \widehat{\varphi_p}(x)$ によって定義する．このとき，$\mathbb{K}_2 = (\mathbb{N}^{\mathbb{N}}, \cdot)$ は**クリーネの第二代数** (*Kleene's second algebra*) と呼ばれ，部分組合せ代数をなす．

例 3.24 (スコットのグラフモデル)**.** 第 3.2 節で見たように，$A \subseteq \mathbb{N}$ は連続関数 $F_A : \mathcal{P}\mathbb{N} \to \mathcal{P}\mathbb{N}$ を定義する．$\mathcal{P}\mathbb{N}$ 上の 2 項演算を $A \cdot B = F_A(B)$ によって定義する．このとき，$\mathbb{P} = (\mathcal{P}\mathbb{N}, \cdot)$ は**スコットのグラフモデル** (*Scott's graph model*) と呼ばれ，部分組合せ代数をなす．グラフモデル $\mathbb{P}$ における 2 項演算は全域であるため，**全域組合せ代数** (*total combinatory algebra*) と呼ばれることもある．

ところで，クリーネの第一代数 $\mathbb{K}_1$ の演算は常に計算可能関数によって実装される．しかし，クリーネの第二代数 $\mathbb{K}_2$ とスコットのグラフモデル $\mathbb{P}$ では，各演算の実装が連続関数による．計算可能性理論の観点からは，演算適用はそれぞれ計算可能連続関数と枚挙作用素であってほしい．このため，相対部分組合せ代数の概念を導入する．

定義 3.25. 相対部分組合せ代数 (*relative partial combinatory algebra*) とは，2 項演算とコンビネータ $\mathtt{k}, \mathtt{s}$ を共有する部分組合せ代数の対 $\mathbf{A}^r = (\mathbf{A}; \mathbf{A}_\emptyset)$ で，$\mathtt{k}, \mathtt{s} \in \mathbf{A}_\emptyset \subseteq \mathbf{A}$ かつ，以下を満たすものである．

$$a, b \in \mathbf{A}_\emptyset \text{ and } a \cdot b \downarrow \in \mathbf{A} \implies a \cdot b \in \mathbf{A}_\emptyset.$$

このとき，$\mathbf{A}$ 上の部分関数 $f :\subseteq \mathbf{A} \to \mathbf{A}$ が $\mathbf{A}^r$**-実現可能** ($\mathbf{A}^r$*-realizable*) または単に $\mathbf{A}$-実現可能であるとは，以下を満たすことである．

$$(\exists e \in \mathbf{A})(\forall a \in \mathrm{dom}(f))\ [e \cdot a \downarrow, \text{ and } e \cdot a = f(a)].$$

$e \in \mathbf{A}_\emptyset$ として取れる場合，f は $\mathbf{A}^r$**-計算可能** ($\mathbf{A}^r$*-computable*) であるという．

相対部分組合せ代数の概念を説明すると，我々は計算可能なものと計算不可能なものが混在する世界に住んでいるものと考えよう．$\mathbf{A}$ がそのような世界を表し，$\mathbf{A}_\emptyset$ はそのうちの計算可能なものだけを取り出したものである．これはたとえば，記述集合論における**太字** (*boldface*) と**細字** (*lightface*) のポイントクラスと類似の発想である．

例 3.26. $\mathbb{K}_1^r = (\mathbb{K}_1, \mathbb{K}_1)$ は相対部分組合せ代数である．このとき，$\mathbb{K}_1^r$-実現可能関数および $\mathbb{K}_1^r$-計

算可能関数とは，$\mathbb{N}$ 上の部分計算可能関数である．

例 3.27. クリーネの第二代数 $\mathbb{K}_2$ について，コンビネータ $\mathtt{k}$ と $\mathtt{s}$ は $\mathbb{N}^{\mathbb{N}}$ の要素であるが，これを $\mathbb{N}$ 上の関数と思うと，$\mathtt{k}$ と $\mathtt{s}$ を計算可能関数として選ぶことができる．$\Delta_1^0 \subseteq \mathbb{N}^{\mathbb{N}}$ で $\mathbb{N}$ 上の計算可能関数全体を表すとする．このとき，$\mathbb{K}_{2\emptyset} = (\Delta_1^0, \cdot, \mathtt{k}, \mathtt{s})$ を考えると，$\mathbb{K}_2^r = (\mathbb{K}_2, \mathbb{K}_{2\emptyset})$ は相対部分組合せ代数をなす．このとき，$\mathbb{K}_2^r$-実現可能関数および $\mathbb{K}_2^r$-計算可能関数とは，それぞれ $\mathbb{N}^{\mathbb{N}}$ 上の部分連続関数および部分計算可能関数である．

例 3.28. スコットのグラフモデル $\mathbb{P}$ について，コンビネータ $\mathtt{k}$ と $\mathtt{s}$ を計算可枚挙集合として選ぶことができる．$\Sigma_1^0 \subseteq \mathcal{P}\mathbb{N}$ で $\mathbb{N}$ 上の計算可枚挙集合全体を表すとする．このとき，$\mathbb{P}_\emptyset = (\Sigma_1^0, \cdot, \mathtt{k}, \mathtt{s})$ を考えると，$\mathbb{P}^r = (\mathbb{P}, \mathbb{P}_\emptyset)$ は相対部分組合せ代数をなす．このとき，$\mathbb{P}^r$-実現可能関数および $\mathbb{P}^r$-計算可能関数とは，それぞれ $\mathcal{P}\mathbb{N}$ 上の連続関数および枚挙作用素である．

相対部分組合せ代数は対であるが，単に $\mathbf{A}$ と表すことも多い．その場合，常に定義 3.25 のような $\mathbf{A}_\emptyset \subseteq \mathbf{A}$ を伴うものと考える．$\mathbf{A} = \mathbf{A}_\emptyset$ であるような相対部分組合せ代数は**フル** (*full*) であると呼ばれる．たとえば，$\mathbb{K}_1^r$ はフルだが，$\mathbb{K}_2^r$ と $\mathbb{P}^r$ はフルではない．

定義 3.29. 相対部分組合せ代数 $\mathbf{A}$ が与えられているとする．$a, b \in \mathbf{A}$ について，ある部分 $\mathbf{A}$-計算可能関数 $f :\subseteq \mathbf{A} \to \mathbf{A}$ が存在して，$f(b) = a$ となるとき，a は b から**相対的 $\mathbf{A}$-計算可能** (*relatively* $\mathbf{A}$*-computable*) であると言い，$a \leqslant_{\mathbf{A}} b$ と書く．つまり，

$$a \leqslant_{\mathbf{A}} b \iff (\exists e \in \mathbf{A}_\emptyset)\ e \cdot b \downarrow = a$$

によって定義する．

例 3.30. クリーネ第二代数 $\mathbb{K}_2^r$ における相対的計算可能性 $\leqslant_{\mathbb{K}_2^r}$ はチューリング還元 $\leqslant_T$ と同値である．スコットのグラフモデル $\mathbb{P}^r$ における相対的計算可能性 $\leqslant_{\mathbb{P}^r}$ は枚挙還元 $\leqslant_e$ と同値である．

3.4 ラムダ計算，不動点，再帰定理

これから，如何なる部分組合せ代数の中でも “計算” を展開できることを示す．つまり，あらゆる計算を，積の適用という代数的演算の組合せで表せる，ということを見る．以後，部分組合せ代数が与えられたときに，積 $\cdot$ の記号は省略し，また積は左結合的であると仮定する．つまり，$(a \cdot b) \cdot c$ は abc のように積と括弧は省略する．たとえば，

$$\mathtt{k}ab = a, \quad \mathtt{s}abc \simeq ba(ca), \quad \mathtt{i} = \mathtt{skk}$$

のように書かれる．

定義 3.31. $\mathbf{A}$ を部分組合せ代数とする．$\mathbf{A}$ 上の**項**とは以下のように帰納的に定められる．

1. 変数 $x, y, z, \ldots$ は項である．
2. 各元 $a \in \mathbf{A}$ は項である．
3. P と Q が項ならば PQ も項である．

$\mathbf{A}$ 上の項 P と変数 x が与えられたとき，項 $\lambda x.P$ を以下によって帰納的に定義する．

$$\begin{aligned} \lambda x.x &\equiv \mathtt{i} \\ \lambda x.y &\equiv \mathtt{k}y && (y \neq x \text{ が変数であるか } y \in \mathbf{A} \text{ のとき}) \\ \lambda x.(PQ) &\equiv \mathtt{s}(\lambda x.P)(\lambda x.Q) \end{aligned}$$

ここで $\mathtt{i}$ は命題 3.21 のような $\mathbf{A}$ の左単位元とする．

以後，$\lambda x.(\lambda y.(\lambda z.P))$ などは $\lambda xyz.P$ のように省略する．たとえば，

$$\lambda xy.x = \lambda x.(\lambda y.x) = \lambda x.(\mathtt{k}x) = \mathtt{s}(\lambda x.\mathtt{k})(\lambda x.x) = \mathtt{s}(\mathtt{kk})\mathtt{i}$$

となる．項 $\lambda x.P$ に含まれる変数は，項 P に含まれる変数から x を除いたものである．よって，P が x のみを変数として含むならば，$\lambda x.P$ は変数を含まない．定義 3.20 より，$\mathtt{k}a \downarrow$ および $\mathtt{s}ab \downarrow$ が成立しているから，このとき $\lambda x.P$ は $\mathbf{A}$ の元を定義することが示される．

項 P が与えられたとき，$P[Q/x]$ によって，P の変数 x に項 Q を代入した結果を表す．上の議論をより一般化すると，$y_0, \ldots, y_n$ を $\lambda x.P$ に含まれる自由変数のリストとすれば，任意の $a_0, \ldots, a_n \in \mathbf{A}$ について，$(\lambda x.P)[a_0, \ldots, a_n/y_0, \ldots, y_n]$ は $\mathbf{A}$ の元を表す．

以下，重要な性質として，$\Lambda x.P$ は本物のラムダ計算のような働きをするということを示す．つまり，部分組合せ代数が与えられれば，内部でラムダ計算っぽいものを展開できてしまうということである．

補題 3.32. P と Q を $\mathbf{A}$ 上の項とする．このとき，$(\lambda x.P)Q \simeq P[Q/x]$ が成り立つ．

Proof. P の構造に関する帰納法による．$P = x$ のとき，

$$(\lambda x.P)Q = \mathtt{i}Q = Q = P[Q/x].$$

続いて，$P = y$ が $y \neq x$ なる変数であるか，$y \in \mathbf{A}$ であるとき，

$$(\lambda x.P)Q = \mathtt{k}yQ = y = P[Q/x].$$

最後に，$P = RS$ である場合，帰納的に $(\lambda x.R)Q \simeq R[Q/x]$ かつ $(\lambda x.S)Q \simeq S[Q/x]$ が成り立っていると仮定する．このとき，

$$(\lambda x.P)Q = \mathtt{s}(\lambda x.R)(\lambda x.S)Q \simeq (\lambda x.R)Q((\lambda x.S)Q) \simeq R[Q/x]S[Q/x] = (RS)[Q/x] = P[Q/x].$$

以上より，帰納法によって，求める性質が得られる． □

命題 3.33. 任意の部分組合せ代数 $\mathbf{A}$ について，次を満たす $\mathtt{pair}, \pi_0, \pi_1 \in \mathbf{A}$ が存在する．

$$\mathtt{pair}\, ab \downarrow, \quad \pi_0(\mathtt{pair}\, ab) = a, \quad \pi_1(\mathtt{pair}\, ab) = b$$

Proof. $\mathtt{pair} = \Lambda xyz.zxy$, $\pi_0 = \Lambda z.z(\Lambda xy.x)$, $\pi_1 = \Lambda z.z(\Lambda xy.y)$ によって定義する．このとき，補題 3.32 を利用すると，Λ は代入と解釈できるから，$\mathtt{pair}\, ab = \Lambda z.zab$ となる．したがって，補題 3.32 より，

$$\pi_0(\mathtt{pair}\, ab) = (\Lambda z.z(\Lambda xy.x))(\Lambda z.zab) = (\Lambda z.zab)(\Lambda xy.x) = (\Lambda xy.x)ab = a,$$
$$\pi_1(\mathtt{pair}\, ab) = (\Lambda z.z(\Lambda xy.y))(\Lambda z.zab) = (\Lambda z.zab)(\Lambda xy.y) = (\Lambda xy.y)ab = b.$$

よって，主張は示された． □

以後，$\mathtt{pair}\, ab$ のことを $\langle a, b\rangle$ と書くことにする．このとき，$\mathbf{A}$ の任意の有限列 $(a_i)_{i\leqslant n}$ は，次のように 1 つの要素としてコードできる．

$$\langle a_0, a_1, a_2, \dots, a_n\rangle := \langle\langle\langle\langle a_0, a_1\rangle, a_2\rangle, \dots\rangle, a_n\rangle.$$

それでは，部分組合せ代数が与えられれば，自然数上の計算論が展開できることを見ていこう．このために，部分組合せ代数 $\mathbf{A}$ の中で自然数をコードする必要がある．これはたとえば次のようにコードできる．

定義 3.34. $\mathbf{A}$ を部分組合せ代数とする．$\mathtt{true}, \mathtt{false} \in \mathbf{A}$ および各自然数 $n \in \mathbb{N}$ について $\underline{n} \in \mathbf{A}$ を以下のように定義する．

$$\mathtt{true} = \Lambda xy.x \qquad \mathtt{false} = \Lambda xy.y$$
$$\underline{0} = \langle \mathtt{true}, \mathtt{i}\rangle \qquad \underline{n+1} = \langle \mathtt{false}, \underline{n}\rangle$$

この自然数のコード方法を理解しかねるという人もいるかもしれないが，取り扱いの容易さから，このコーディングを利用する．しかし，「自然数を実装できる」という点のみが重要なのであって，自然数の具体的な実装方法は何でもよいし，その意味を問うことはあまり生産的ではない．文字列によって自然数をコードする方法がいくらでもあるように，部分組合せ代数 $\mathbf{A}$ の中で自然数をコードする方法は唯一ではない．

命題 3.35. 次のような元 $\mathtt{succ}, \mathtt{pred}, \mathtt{iszero} \in \mathbf{A}$ が存在する．

$$\mathtt{succ}\, \underline{n} = \underline{n+1} \qquad \mathtt{pred}\, \underline{n} = \underline{n \dot{-} 1}$$
$$\mathtt{iszero}\, \underline{n} = \begin{cases} \mathtt{true} & \text{if } n = 0 \\ \mathtt{false} & \text{if } n \neq 0 \end{cases}$$

ここで $\dot{-}$ は $x \dot{-} y = \max\{0, x - y\}$ によって定義される部分的減法である．

Proof. まず，$\mathtt{succ} = \Lambda x.\langle \mathtt{false}, x\rangle$ によって定義する．このとき，補題 3.32 より，$\mathtt{succ}\,\underline{n} = \langle \mathtt{false}, \underline{n}\rangle = \underline{n+1}$ である．$\mathtt{iszero} = \pi_0$ が条件を満たすことは明らかである．最後に，$\mathtt{pred}$ を定義するために，補題 3.32 から以下の式を得られることに注意する．

$$\langle a, b\rangle \mathtt{true} = (\Lambda xyz.zxy)ab(\Lambda xy.x) = (\Lambda z.zab)(\Lambda xy.x) = (\Lambda xy.x)ab = a$$
$$\langle a, b\rangle \mathtt{false} = (\Lambda xyz.zxy)ab(\Lambda xy.y) = (\Lambda z.zab)(\Lambda xy.y) = (\Lambda xy.y)ab = b.$$

$\mathtt{pred} = \Lambda x.\langle \underline{0}, \pi_1 x\rangle(\mathtt{iszero}\, x)$ と定義する．このとき，上の式と補題 3.32 より，

$$\mathtt{pred}(\underline{0}) = \langle \underline{0}, \pi_1\, \underline{0}\rangle(\mathtt{iszero}\, \underline{0}) = \langle \underline{0}, \mathtt{i}\rangle \mathtt{true} = \underline{0}$$
$$\mathtt{pred}(\underline{n+1}) = \langle \underline{0}, \pi_1\, \underline{n+1}\rangle(\mathtt{iszero}\, \underline{n+1}) = \langle \underline{0}, \underline{n}\rangle \mathtt{false} = \underline{n}.$$

よって，主張は示された． □

次に，部分組合せ代数におけるある種の不動点定理を示そう．関数 $f : X \to X$ の**不動点** (*fixed point*) とは，$f(x) = x$ なる $x \in X$ のことである．しかし，ここでは X は関数空間 $[A \to B]$ であると考えると都合がよい．つまり，関数 $f : [A \to B] \to [A \to B]$ の不動点とは $g = f(g)$ を満たす関数 $g : A \to B$ のことである．f の不動点のことを $\mathtt{fix}\, f$ と書くとすると，$\mathtt{fix}\, f = f(\mathtt{fix}\, f)$ を満たす．つまり

$$(\forall a \in A)\ f(\mathtt{fix}\, f)(a) = (\mathtt{fix}\, f)(a)$$

を満たすということである．このような不動点 $\mathtt{fix}\, f$ のようなものが常に存在する，というのが次の定理である．

定理 3.36. 次のような元 $\mathtt{fix} \in \mathbf{A}$ が存在する．任意の $f, a \in \mathbf{A}$ に対して，

$$\mathtt{fix}\, f \downarrow \qquad \mathtt{fix}\, f a = f(\mathtt{fix}\, f)a.$$

Proof. $\mathtt{r} = \Lambda xyz.x(yy)z$ とし，$\mathtt{fix} = \Lambda g.\mathtt{r}g(\mathtt{r}g)$ と定義する．このとき，

$$\mathtt{fix}\, f = \mathtt{r}f(\mathtt{r}f) = (\Lambda xyz.x(yy)z)f(\mathtt{r}f) = (\Lambda yz.f(yy)z)(\mathtt{r}f) = \Lambda z.f(\mathtt{r}f(\mathtt{r}f))z$$

である．いま，$\mathtt{r}$ は自由変数を持たず，よって，$f(\mathtt{r}f(\mathtt{r}f))z$ は z のみを自由変数に含む．一方，定義 3.31 の直後に述べたように，もし P が z のみを変数に含むならば，$\Lambda z.P \downarrow$ である．よって，$\mathtt{fix}\, f \downarrow$ を得る．また，

$$\mathtt{fix}\, f a = (\mathtt{r}f(\mathtt{r}f))a = (\Lambda z.f(\mathtt{r}f(\mathtt{r}f))z)a = f(\mathtt{r}f(\mathtt{r}f))a = f(\mathtt{fix}\, f)a$$

となるから定理は示された． □

不動点定理 3.36 の重要な帰結の 1 つが，**クリーネの再帰定理** (*Kleene's recursion theorem*) である．まず，定理 3.36 から次の系が得られる．

系 3.37. $\mathbf{A}$ を部分組合せ代数とし，Q を e のみを自由変数とする $\mathbf{A}$ 上の項とする．このとき，次を満たす項 $r \in \mathbf{A}$ が存在する．

$$(\forall a \in \mathbf{A})\ ra \simeq Q[r/e]a.$$

Proof. $f = \Lambda e.Q$ とする．ここで，Q に含まれる自由変数は e のみであるから，定義 3.31 の直後の議論より，$f \downarrow \in \mathbf{A}$ と考えてよい．よって，不動点定理 3.36 の証明の $\mathtt{fix}$ について，$\mathtt{fix}\, f \downarrow = r$ となる $r \in \mathbf{A}$ を得る．いま，任意の $a \in \mathbf{A}$ について，

$$ra = fra = (\lambda e.Q)ra \simeq Q[r/e]a$$

であるから，主張は示された． □

ここで，$r = Q[r/e]$ となるとは限らないことに注意する．これは，クリーネの第一代数 $\mathbb{K}_1$ で考えたときに明確となる．$\mathbb{K}_1$ において，$p = q$ であるということは，この 2 つの p と q が一字一句違わない文字列あるいはプログラムであることを意味する．一方，全ての $a \in \mathbb{K}_1$ について $pa \simeq qa$ ということは，この 2 つの p と q がプログラムとして同一の働きをすることを意味している．後者の概念の方が本質的であり，前者を要求することはあまり重要でないということは予想が付くだろう．系 3.37 を $\mathbb{K}_1$ で解釈した結果は，クリーネの再帰定理として知られる．

定理 3.38 (クリーネの再帰定理)**.** 任意の計算可能関数 $q : \mathbb{N} \to \mathbb{N}$ について，次を満たす $r \in \mathbb{N}$ が存在する．

$$[\![r]\!] \simeq [\![q(r)]\!]$$

Proof. 部分組合せ代数として，命題 3.22 のクリーネの第一代数 $\mathbb{K}_1$ を取る．このとき，q は計算可能であるから，ある $d \in \mathbb{N}$ について，$q = [\![d]\!]$ となる．$Q = de$ とすると，系 3.37 より，任意の $a \in \mathbb{K}_1$ について，$ra \simeq Q[r/e] \simeq dra$ なる $r \in \mathbb{K}_1$ を得る．$\mathbb{K}_1$ における積演算の定義より，

$$(\forall a \in \mathbb{N})\ [\![r]\!](a) \simeq [\![[\![d]\!](r)]\!](a) \simeq [\![q(r)]\!](a)$$

となるから，定理は示された． □

クリーネの再帰定理の直感的な説明を与えよう．固定した未知変数 I を使いながら，コンピュータ・プログラム $Q(I)$ を書いている，というシチュエーションを想定しよう．我々はその段階では I が何であるかは知らないが，とにかく I は自由に使えるので，プログラム内部に「プログラム I に n を入力した計算を実行せよ」などの命令を書き込むことができる．

さて，クリーネの再帰定理 3.38 における r を取ってきて，$Q = Q(r)$ としよう．つまり，先ほど我々の書いたプログラムの中で I と書かれている部分を r で上書きしたものが新しいプログラム Q である．すると，再帰定理より，コード r のプログラムを実行したものと Q を実行したものの

計算結果は等しい．したがって，Q のプログラム内部に書かれている「プログラム I に n を入力した計算を実行せよ」という命令は「プログラム Q に n を入力した計算を実行せよ」という命令に等しい．

これが意味していることは何だろうか．我々はプログラム Q を書いている途中段階では，最終的な Q がどうなるかは知らないし，無限の自由度がある．それにも関わらず，我々が途中で書いた「プログラム I に n を入力した計算を実行せよ」と言う命令の意味は，常に「最終的な Q に n を入力した計算を実行せよ」を表す．つまり，我々はあたかも「最終的に書き上げる予定のプログラムが何であるか既に知っているかの如く」プログラムを記述することができるのである．

そういうわけで，以後，プログラム I すなわち『私』すなわち自己に言及したプログラムは自由に記述してよい．特に自身のソースコードを出力するプログラムは，コンピュータ・プログラミングの文脈では**クワイン** (*Quine*) としてよく知られており，様々なプログラミング言語での実装例を見つけることができるだろう．

クリーネの再帰定理は数学的には一見単純であるが，それ故に強力である．基礎的なレベルから研究の最先端に至るまで，極めて広範な応用を持つ．計算可能性理論の入門的内容における最も重要な定理と言っても過言ではないだろう．

ここでは，クリーネの再帰定理の簡単な応用として，原始再帰法の実装を行う．原始再帰法とは，関数 f, g から次のような関数 h を作る操作である．

$$h(0, x) = g(x)$$
$$h(n+1, x) = f(n, x, h(n, x))$$

この関数 h を $\mathtt{rec}\, gf$ として表そう．この原始再帰作用素 $\mathtt{rec}$ は，不動点として実装できる．

命題 3.39. 次のような元 $\mathtt{rec} \in \mathbf{A}$ が存在する．

$$\mathtt{rec}\, gf\underline{0} = g$$
$$\mathtt{rec}\, gf\underline{n+1} = f\underline{n}(\mathtt{rec}\, gf\underline{n})$$

Proof. まず，命題 3.35 の証明で見たように，

$$\langle a, b\rangle \mathtt{iszero}\, n = \begin{cases} a & \text{if } n = \underline{0} \\ b & \text{if } n \neq \underline{0} \end{cases}$$

が成り立つことに注意する．このため，$\langle a, b\rangle \mathtt{iszero}\, n$ を $\mathtt{if}\ n\ \mathtt{iszero}\ \mathtt{then}\ a\ \mathtt{else}\ b$ と表す．

証明のアイデアとしては，$n = 0$ ならば $hn = g$ であり，さもなくば $hn = f(\mathtt{pred}\, n)(h(\mathtt{pred}\, n))$ であるような h を作ればよい．ただし，h は f と g に依存するので，$h = egf$ という形である．具体的には，次の項 Q を考える．

$$\Lambda gfn.\mathtt{if}\ n\ \mathtt{iszero}\ \mathtt{then}\ g\ \mathtt{else}\ f(\mathtt{pred}\, n)(egf(\mathtt{pred}\, n)).$$

項 Q は e のみを自由変数に持つので，系 3.37 より，e を自己への言及と解釈するような $\mathtt{rec} \in \mathbf{A}$ が存在する．つまり，任意の $a \in \mathbf{A}$ について $\mathtt{rec}\, a = Q[\mathtt{rec}/e]a$ が成立する．このとき，

$$\mathtt{rec}\, gf\underline{n} = \mathtt{if}\ \underline{n}\ \mathtt{iszero}\ \mathtt{then}\ g\ \mathtt{else}\ f(\mathtt{pred}\,\underline{n})(\mathtt{rec}\, gf(\mathtt{pred}\,\underline{n}))$$

であるから，明らかに

$$\mathtt{rec}\, gf\underline{0} = g, \qquad \mathtt{rec}\, gf\underline{n+1} = f\underline{n}(\mathtt{rec}\, gf\underline{n})$$

が導かれる．よって求める性質が示された． □