数理解析研究所講究録
第 75 巻 1969 年 43-70

各種 algorithm と recursive な手続の相互間翻訳について

九大 理学部 西澤輝泰

§1. 序

algorithm について調べる場合、互いに同等で、それぞれに別な長所をもついくつかの algorithm を準備しておいて、場合に応じて適当な algorithm の使いわけができると便利である。この時は勿論それら algorithm 相互の翻訳の仕方、また recursive な手続との間の翻訳の仕方などが、すべて与えられていることが望ましい。recursive な手続と同等な力を持ち、充分に単純な構造をもつ algorithm として、Markov の normal algorithm（以下 Markov algorithm と略称），Post の normal algorithm（以下 Post algorithm と略称），Turing machine による algorithm（以下 Turing algorithm と略称），Asser の考案した finite automaton の反復使用による algorithm（以下 Asser algorithm と略称）がある。勿論これらを拡張或は変形して、これらと同等な

algorithm は無数につくれる。ここでは、これら4つのalgorithmを、それぞれが単純なword functionのくり返し適用で導かれるという共通性に着目して統一的に定義し、これらとrecursiveな手続の相互間翻訳の1つの例として、Markov → Post → Asser → Turing → Markovの順での翻訳と、Asser → recursive，recursive → (Markov, Post, Asser)の翻訳を与える。

この記述は同時に、これら4つのalgorithmとrecursiveな手続とが同等であることの、一括した自己充足的な証明を与えることになる。

特に、ここで述べるAsser → recursiveの翻訳は、直接に、つまりalgorithmの算術化を通さずに行う。即ち、computableならばrecursiveの証明が、これにより、Gödel-numberingなどの算術化によらずにできたことになる。

§2. Notation

空でない集合X，Yに対し、X → Yの一意対応をpartial functionと称し、これの集合をp.f.(X, Y)で表す。$f \in$ p.f.(X, Y)は、その定義域をDom(f)で表し、Xの元xが、$x \in \mathrm{Dom}(f)$であれば$f(x)$はdefined，$x \notin \mathrm{Dom}(f)$であ

ればf(x)はundefinedであるという。$f \in p.f.(X, Y)$を、Xの空でない部分集合Z上に制限して得られる$Z \to Y$のpartial functionを$f|Z$で表す。$f, g \in p.f.(X, Y)$に対し、$f|Z = g|Z$を$f(x) = g(x)$ ($x \in Z$)で表す。

空でない集合Aに対し、Aのword全体の集合（Aで生成される単位元をもつ自由半群）をA^*で表し、empty word（A^*の単位元）をΛ_A（混乱のおそれのない時は単にΛ）で表す。A^*の元$a_1 \cdots\cdots a_n$ ($a_i \in A$)を必要に応じて$\langle a_1, \cdots\cdots, a_n \rangle$のように表す。$a_i$をこのwordのi番目の項と称する。

$X = A^* \times \cdots \times A^*$ （A^* n個の直積，$(A^*)^n$と略記）と$Y = A^*$に対し、$p.f.(X, Y)$の元をA上n変数のword function（'n変数の'は場合に応じて省略する）と称する。$f \in p.f.(X, Y)$と$B \subset A$, $B \neq \phi$なるBをとる。$f|(B^*)^n$の値域がB^*に含まれているとき、$f|(B^*)^n$をfのB上への制限と称する。$\alpha \in A - B$なる元をとる（$A - B \neq \phi$と仮定して）。A上1変数のword function fが任意の$x_1, \cdots\cdots, x_m \in B^*$（mは与えられた正整数）に対し、$f(x_1 \alpha \cdots\cdots \alpha x_m)$がdefinedであれば$\in B^*$なる性質をもつとき、$\varphi(x_1, \cdots\cdots, x_m) = f(x_1 \alpha \cdots\cdots \alpha x_m)$によりB上m変数のword function φが定義される。このφを、fとαにより導かれるB上m変数のword function と称する。

§3. Iteration

Aを空でない集合とする。$\Gamma \in p.f.(A^*-\{\Lambda\}, A)$, $C \subset A^*$をとる。$a \in A$に対し、$\gamma(a) \subset A^*$で次のようなものを与える。

(1) $a \in \gamma(a)$

(2) $x \in \gamma(a) \cap C$なるxに対し、$x \in Dom(\Gamma)$ならば $x\Gamma(x) \in \gamma(a)$, $x \notin Dom(\Gamma)$ならば$\Lambda \in \gamma(a)$

(3) $\gamma(a)$は(1), (2)を満たすA^*の最小の部分集合。

$a \to \gamma(a)$なる写像$\gamma: A \to \mathcal{P}(A^*) = \{X; X \subset A^*\}$を、$C$に制御された$\Gamma$の iteration といい、$\gamma = itr(A, \Gamma, C)$で表す。$\omega \in p.f.(A, A)$を、$a \in A$に対し$\gamma(a)$の含む最も長い word の最後の項を対応させる partial function とする（そのような項が存在しなければ undefined）。$\omega' \in p.f.(A, A)$を$\Lambda \notin \gamma(a)$ならば$\omega'(a) = \omega(a)$, $\Lambda \in \gamma(a)$ならば$\omega'(a)$は undefined として定義する。このω, ω'をそれぞれγで導かれた type 1, type 2 の partial function と呼び、$[\gamma]_1$, $[\gamma]_2$で表す。

$\boxed{\text{定理1}}$ Aをalphabet（空でない有限集合），$X=(A^*)^n$ とする。$\Gamma \in p.f.(X^*-\{\Lambda_X\}, X)$ と $C \subset X^*$ が、

1) A上n変数のpartial recursive word function $g_1, \cdots, g_n$ により、$\Gamma(\langle x_1,\cdots,x_t\rangle) = (g_1(x_t),\cdots,g_n(x_t))$ （$x_i \in X$）

2) A上n変数のgeneral recursive word function f と、負でない整数mがあって、$\langle x_1,\cdots,x_t\rangle \in C \longleftrightarrow f(x_{t-m}) \neq \Lambda_A$ 又は $t \leqq m$

なる条件を満たすものとする。このとき、$\gamma = \mathrm{itr}(X,\Gamma,C)$ に対し、$[\gamma]_2$ の各成分（$[\gamma]_2(w_1,\cdots,w_n) = (w_1',\cdots,w_n')$ とするとき、$(w_1,\cdots,w_n)\longrightarrow w_i'$ なるA上n変数のpartial word function）はpartial recursiveである。

〔証明〕 A上$n+1$変数のpartial recursive word function h_i（$i=1,\cdots,n$）を次のように構成する。

$$h_i(\Lambda, x^{(n)}) = \pi_i^{(n)}(x^{(n)})$$

$$h_i(ya, x^{(n)}) = g_i(h_1(y,x^{(n)}),\cdots\cdots,h_n(y,x^{(n)}))$$

（$y\in A^*$，$a\in A$，$x^{(n)}\in X$，$\pi_i^{(n)}$は第i成分をとるprojection）

Aの特定の元a_0を選び、$+a_0$を$w\longrightarrow wa_0$（$w\in A^*$）なるsuccessor functionとして、

$$\varphi(x^{(n)}) = (+a_0)^m\left(\mu_y^{a_0}[f(h_1(y,x^{(n)}),\cdots,h_n(y,x^{(n)}))=\Lambda]\right)$$

なる partial recursive word function φ を作る。
$[\gamma]_2(x^{(n)})$ の第 i 成分は $h_i(\varphi(x^{(n)}), x^{(n)})$ であるから partial recursive である。特に、$g_1, \cdots, g_n, f$ が primitive recursive であれば、$[\gamma]_2$ の各成分は minimalization を1回だけ用いて作られる。

〔附記〕 primitive recursive word functions から m 重の primitive recursion を用いて構成される word function が再び primitive recursive であることの証明は次のようにして得られる。

alphabet A に対し、$* \notin A$ なる $*$ をとって、$A \cup \{*\} = \tilde{A}$ とおく。以下特に断りがない限り、a, b なる文字は $\tilde{A}$ の元を表し、x, y, z なる文字は $\tilde{A}^*$ の元を表し、$x^{(n)}$ なる表示は $(A^*)^n$ の元を表すものとする。

1) $J^{-1}(x, y) = x * y$ は primitive recursive

〔証〕 $J^{-1}(x, \Lambda) = x*$, $J^{-1}(x, ya) = J^{-1}(x, y)a$

2) $\rho(\Lambda) = \Lambda$, $\rho(a_1 \cdots\cdots a_k) = a_k \cdots\cdots a_1$ なる ρ は primitive recursive

〔証〕 $\rho_a(\Lambda) = a$, $\rho_a(xb) = \rho_a(x)b$ なる ρ_a により、
$\rho(\Lambda) = \Lambda$, $\rho(xa) = \rho_a(\rho(x))$

3) $\sigma(x) = \Lambda \ (x \in A^*)$, $\sigma(x * y) = * y \ (x \in A^*)$

なる σ は primitive recursive

〔証〕 $\pi(\Lambda)=\Lambda$, $\pi(xa)=\pi(x)$ $(a\neq *)$, $\pi(x*)=*$

なる π と、 $\sigma_a(\Lambda,x)=\Lambda$, $\sigma_a(yb,x)=xa$ なる σ_a

により、 $\sigma(\Lambda)=\Lambda$, $\sigma(xa)=\sigma_a(\pi(xa),\sigma(x))$

4) $J_1(x)=\Lambda$ $(x\in A^*)$, $J_1(y*x)=y$ $(x\in A^*)$ は primitive recursive

〔証〕 $\omega(\Lambda)=\Lambda$, $\omega(xa)=x$ なる ω により、

$J_1=\omega\circ\rho\circ\sigma\circ\rho$

5) m を正整数とするとき、${}^mJ_i(x_1*\cdots\cdots*x_m)=x_i$ $(x_j\in A^*)$ なる ${}^mJ_i$ $(1\leqq i\leqq m)$ と、${}^mJ^{-1}(x_1,\cdots,x_m)=x_1*\cdots\cdots*x_m$ なる ${}^mJ^{-1}$ は primitive recursive

〔証〕 $J_2=\rho\circ J_1\circ\rho$ とおいて、${}^mJ_i=(J_2)^{i-1}\circ(J_1)^{m-i}$.

また ${}^mJ^{-1}$ は ${}^2J^{-1}=J^{-1}$, ${}^{k+1}J^{-1}(x_1,\cdots,x_{k+1})=J^{-1}({}^kJ^{-1}(x_1,\cdots,x_k),x_{k+1})$ により得られる。

6) A 上 $m+n+1$ 変数の word function $h_i^{(a)}$ $(i=1,\cdots,m$; $a\in A)$ と A 上 n 変数の word function g_i $(i=1,\cdots,m)$ が primitive recursive であれば、

$f_i(\Lambda,x^{(n)})=g_i(x^{(n)})$

$f_i(ya,x^{(n)})=h_i^{(a)}(y,f_1(y,x^{(n)}),\cdots,f_m(y,x^{(n)}),x^{(n)})$

$(x^{(n)}\in(A^*)^n,\ y\in A^*,\ a\in A)$

で構成される A 上 $n+1$ 変数の word function f_i $(i=1,\cdots,$

m）は primitive recursive である。

〔証〕 $h_i^{(a)}$, g_i を $\widetilde{A}$ 上の primitive recursive word function に、値域が A^* に含まれるという性質を保ったまゝ拡張したものを $\widetilde{h}_i^{(a)}$, $\widetilde{g}_i$ とする。$\widetilde{h}_i^{(*)}$ を $\widetilde{h}_i^{(*)}(x^{(m+n+1)}) = \Lambda$ なるものとする。

$$H_a(y, z, x^{(n)}) = \widetilde{h}_1^{(a)}(y, {}^mJ_1(z), \cdots, {}^mJ_m(z), x^{(n)}) * \cdots\cdots * \widetilde{h}_m^{(a)}(y, {}^mJ_1(z), \cdots, {}^mJ_m(z), x^{(n)})$$

$$(a \in \widetilde{A},\ y \in \widetilde{A}^*,\ z \in (\widetilde{A}^*)^m,\ x^{(n)} \in (\widetilde{A}^*)^n)$$

なる H_a をとり、

$$F(\Lambda, x^{(n)}) = \widetilde{g}_1(x^{(n)}) * \cdots\cdots * \widetilde{g}_m(x^{(n)})$$

$$F(ya, x^{(n)}) = H_a(y, F(y, x^{(n)}), x^{(n)})$$

なる F を作れば、$\widetilde{f}_i = {}^mJ_i \circ F$ とおいて、明らかに

$f_i(y, x^{(n)}) = \widetilde{f}_i(y, x^{(n)})$ $(y \in A^*,\ x^{(n)} \in (A^*)^n)$ となり、f_i は primitive recursive である。

§4. Markov algorithm と Post algorithm

A を alphabet とする。$x, y \in A^*$ に対し、$x \longrightarrow y$ 又は $x \longrightarrow \cdot y$ なる形式を変換素子と呼び、$x \longrightarrow \cdot y$ なる変換素子は終結的であるという。$w_1 \longrightarrow (\cdot) w_2$ なる変換素子 τ は次のような p.f. (A^*, A^*) の２元 τ_m, τ_p を与える。

(1) $\tau_m(x)$: xに含まれる最も左側のw_1をw_2に書きかえたもの。xがw_1を含まなければundefined.

(2) $\tau_p(x)$: $x = w_1 x_1$ ($x_1 \in A^*$) となっている場合に限りdefinedで、このとき $\tau_p(x) = x_1 w_2$.

σを有限個の変換素子の系列とする。$x \in A^*$に対し、$\tau_m(x)$がdefinedとなるようなσの中の最初の変換素子τによる$\tau_m(x)$を$\sigma_m(x)$で表し、そのτが終結的であるとき、xはσ_m－終結的であるという。$\sigma_p(x)$, σ_p－終結的についても同様に定義する。

$X = A^*$に対し、$\Gamma \in p.f.(X^* - \{\Lambda_X\}, X)$, $C \subset X^*$を、$\Gamma(\langle x_1, \cdots, x_t \rangle) = \sigma_m(x_t)$, $\langle x_1, \cdots, x_t \rangle \notin C \longleftrightarrow$ (x_{t-1}がσ_m－終結的かつ$t \neq 1$) なるものとする。これ(X, Γ, C)をとって、これを$\mathcal{M}(\sigma)$とおく。添字mをpにかきかえて、$\mathcal{P}(\sigma)$を同様にして得る。$[\mathcal{M}(\sigma)]_1$, $[\mathcal{P}(\sigma)]_1$をそれぞれ$\mathcal{M}_\sigma$, $\mathcal{P}_\sigma$で表す。$\mathcal{M}_\sigma$, 又はそれを制限して得られる（或は$\alpha \in A$により導かれる）word functionをMarkov computableであるといい、$\mathcal{M}(\sigma)$をそのword functionのMarkov algorithm, σをそのMarkov algorithmの変換素子系列という。Post computable, Post algorithmについても同様に定義する。

$\boxed{\text{定理2}}$ alphabet A上のMarkov algorithm $\mathcal{M}(\sigma)$に対

し、Post algorithm $\mathcal{P}(\sigma')$ を構成して、$\mathfrak{M}_\sigma(x) = \mathcal{P}_{\sigma'}(x)$ $(x \in A^*)$ なるようにすることができる。特に、word function が Markov computable であれば同時に Post computable である。

［証明］ $A \cap \widetilde{A} = \phi$, $|A| = |\widetilde{A}|$ （$|\ |$は元の個数を示す）なる alphabet $\widetilde{A}$ と、$\varphi: A^* \longrightarrow \widetilde{A}^*$ なる isomorphism をとる。$x \in A^*$ に対し、$\varphi(x)$ を $\widetilde{x}$ で表す。変換素子系列 σ が n 個の変換素子により構成されているものとして、$2n+4$ 個の互いに異なる新しい symbol（A, $\widetilde{A}$ に含まれない symbol）$\#$, $\#_1$, $\cdots$, $\#_{n+1}$, $\widetilde{\#}_1$, $\cdots$, $\widetilde{\#}_{n+1}$, β をとる。A, $\widetilde{A}$ とこれらの元をすべてあわせて alphabet B とする。以下、変換素子の系列は記法を便利にするため、例えば $x_1 a y_1 \longrightarrow (\cdot)\, x_2 a y_2$ $(a \in A)$ のような形式は全ての $a \in A$ についてのこのような形の変換素子を任意の順序で並べた系列を表わすものとする。また変換素子の系列は上下に並べて書いたときは、この順で並んだものとする。

さて、与えられた $\mathfrak{M}(\sigma)$ の σ を $\tau_1, \cdots, \tau_n$; $\tau_i = x_i \longrightarrow (\cdot)\, y_i$ なるものとし、B 上の n 個の変換素子系列 σ'_i $(i = 1, \cdots, n)$ を次のように作る。

$$\sigma'_i : \left| \begin{array}{l} \#_i \widetilde{x}_i \longrightarrow \widetilde{y}_i \widetilde{\#}_i \\ \#_i \widetilde{a} \longrightarrow \widetilde{a}\, \#_i \quad (a \in A) \end{array} \right.$$

$$\left\{\begin{array}{l} \widetilde{\#}_i\,\widetilde{a} \longrightarrow \widetilde{a}\,\widetilde{\#}_i \quad (a \in A) \\ \#_i\,\beta \longrightarrow \beta\,\#_{i+1} \\ \widetilde{\#}_i\,\beta \longrightarrow \begin{cases} \beta\,\#_1 & (\tau_i \text{が終結的でない場合}) \\ \beta\,\#_{n+1} & (\tau_i \text{が終結的である場合}) \end{cases} \end{array}\right.$$

更に B 上の変換素子系列 σ' を次のように構成する。

$$\sigma' : \left\{\begin{array}{ll} \#_{n+1}\,\widetilde{a} \longrightarrow a\beta\,\#_{n+1} & (a \in A) \\ \beta a \longrightarrow \beta a & (a \in A) \\ a\beta \longrightarrow a & (a \in A) \\ \#_{n+1}\,\beta \longrightarrow \widetilde{\#}_{n+1} & \\ \widetilde{\#}_{n+1} \longrightarrow \cdot\Lambda & \\ \beta \longrightarrow \beta & \\ \widetilde{a} \longrightarrow \widetilde{a} & (a \in A) \\ \quad \sigma_1' & \\ \quad \vdots & \\ \quad \sigma_n' & \\ a \longrightarrow \#\#\,a & (a \in A) \\ \#\#\,a\,\# \longrightarrow \#\beta\#_1\,\widetilde{a} & (a \in A) \\ \#\,a\,\# \longrightarrow \widetilde{a} & \\ \#\#\,a \longrightarrow \beta\,\#_1\,\widetilde{a} & (a \in A) \\ \Lambda \longrightarrow \beta\,\#_1 & \end{array}\right.$$

この変換素子系列 σ' により生成される B 上の Post algorithm

$\mathcal{P}(\sigma')$ が、$\mathfrak{M}_\sigma(x) = \mathcal{P}_{\sigma'}(x)$ $(x \in A^*)$ をみたすことは容易に確かめることができる。

§5. Asser algorithm

1．定義　alphabet A と $\alpha \notin A$ なる α をとり、$\tilde{A} = A \cup \{\alpha\}$ とおく。output をともなう automaton $\mathfrak{A} = \mathfrak{A}(\tilde{A}, S, \delta, \lambda, s^*, F)$ を与える。ここに $\tilde{A}$ は input-output alphabet, S は states alphabet, δ は state function $S \times \tilde{A} \longrightarrow S$, λ は output function $S \times \tilde{A} \longrightarrow \tilde{A}$, s^* は initial state, F は final states alphabet $(F \subset S)$ である。$s \in S$, $a \in \tilde{A}$, $x \in \tilde{A}^*$ に対し、
$$\lambda'(s, a) = \begin{cases} \lambda(s, a) & (\lambda(s, a) \neq \alpha) \\ \Lambda & (\lambda(s, a) = \alpha) \end{cases}$$
なる $\lambda' : S \times \tilde{A} \longrightarrow A^*$ と、$\tilde{\delta}(s, \Lambda) = s$, $\tilde{\delta}(s, xa) = \delta(\tilde{\delta}(s, x), a)$; $\tilde{\lambda}(s, \Lambda) = \Lambda$, $\tilde{\lambda}(s, xa) = \tilde{\lambda}(s, x) \cdot \lambda'(\tilde{\delta}(s, x), a)$ により、$\tilde{\delta} : S \times (\tilde{A})^* \longrightarrow S$, $\tilde{\lambda} : S \times (\tilde{A})^* \longrightarrow A^*$ を定義する。（これらは簡単に $S \cup \tilde{A}$ 上２変数の primitive recursive word function に拡張できる。）$S \times A^* = X$ とおき、$\Gamma_1 \in p.f.(X^* - \{\Lambda_X\}, X)$ と $C_1 \subset X^*$ を、
$$\Gamma_1(\langle (s_1, x_1), \cdots\cdots, (s_n, x_n) \rangle) = (\delta(s_n, \alpha), x_n \cdot \lambda'(s_n, \alpha))$$
$$\langle (s_1, x_1), \cdots\cdots, (s_n, x_n) \rangle \in C_1 \longleftrightarrow \lambda'(s_{n-1}, \alpha) \neq \Lambda \text{ 又は } n = 1$$

で定義し、$\gamma_1 = itr(X, \Gamma_1, C_1)$ に対し、$g(s, x) = [\gamma_1]_2(\tilde{\delta}(s, x), \tilde{\lambda}(s, x))$ なる $g \in p.f.(X, X)$ を作る。($g(s, x)$ の各成分は、undefined であるときに $=\Lambda$ にするという変更を加えれば、$S \cup A$ 上2変数の primitive recursive word function に拡張できる。何故なら、$T = \{s \in S ; \lambda(s, \alpha) = \gamma\}$ に対し、

$$\varphi_1(s, x) = \begin{cases} (\delta(s, \alpha), x \cdot \lambda(s, \alpha)) & (s \notin T) \\ (s, x) & (s \in T) \end{cases},$$

$$\varphi_2(s, x) = \begin{cases} (\Lambda, \Lambda) & (s \notin T) \\ (\delta(s, \alpha), x) & (s \in T) \end{cases}$$

とおいてやれば、$|S| = m$ として、$\varphi_2(\varphi_1^m(s, x))$ は $[\gamma_1]_2(s, x)$ が defined のときはその値に等しく、そうでないときは $= (\Lambda, \Lambda)$ となるからである。)

次に、$\Gamma_2 \in p.f.(X^* - \{\Lambda_X\}, X)$ と $C_2 \subset X^*$ を、

$$\Gamma_2(\langle (s_1, x_1), \cdots\cdots, (s_n, x_n) \rangle) = g(s_n, x_n)$$

$$\langle (s_1, x_1), \cdots\cdots, (s_n, x_n) \rangle \in C_2 \longleftrightarrow s_n \notin F$$

で定義し、$\gamma_2 = itr(X, \Gamma_2, C_2)$ をとる。$[\gamma_2]_2$ の各成分は明らかに $S \cup A$ 上2変数の partial recursive word function に拡張できて、しかもその構成は minimalization を1回しか用いないでできる。(定理1とその証明過程により明らか。) この $[\gamma_2]_2$ を $\mathcal{G}_{\sigma, \alpha}$ で、$\mathcal{G}_{\sigma, \alpha}(s^*, x)$ の第2成分を $g_{\sigma, \alpha}$ で表す。$g_{\sigma, \alpha}$ は A 上の word function である。

$g_{\mathfrak{A},\alpha}$ 又はこれを制限して（或は$\beta\in A$で導かれて）得られるword function を Asser computable であるといい、このときのγ_2をこのword function の automaton $\mathfrak{A}$ と Blank symbol α による Asser algorithm と称し、$\mathfrak{J}[\mathfrak{A},\alpha]$ で表す。Blank symbol α が了解ずみであると思われれば、αは略して単に$\mathfrak{J}_{\mathfrak{A}}$, $g_{\mathfrak{A}}$, $\mathfrak{J}[\mathfrak{A}]$のように記す.
上述したことから、次の定理が明らかである。

$\boxed{\text{定理3}}$ Asser algorithm $\mathfrak{J}[\mathfrak{A},\alpha]$ に対し、$g_{\mathfrak{A},\alpha}$ は partial recursive である。特に$g_{\mathfrak{A},\alpha}$ の構成は minimalization を1回用いるだけで得られる。

2. Asser algorithm の接続

Aをalphabet, $\alpha\notin A$ なるαに対し、$\tilde{A}=A\cup\{\alpha\}$ とおく。$\mathfrak{A}_1$, $\mathfrak{A}_2$ を $\mathfrak{A}_i=\mathfrak{A}_i(\tilde{A}, S_i, \lambda_i, \delta_i, s_i^*, F_i)$ $(i=1,2)$ なる finite automaton で、$s\in S_1\cap S_2$ なる state s に対しては、$\lambda_1(s,a)=\lambda_2(s,a)$, $\delta_1(s,a)=\delta_2(s,a)$ $(a\in\tilde{A})$ なるものとする。特定の$s^{**}\in F_1$ に対し、$T=\{s\in S_1;\ \delta_1(s,\alpha)=s^{**},\ \lambda_1(s,\alpha)=\alpha\}$ とおく。$S_1\cup S_2=S$ とし、$\lambda: S\times\tilde{A}\to\tilde{A}$, $\delta: S\times\tilde{A}\to S$ を $\lambda(s,a)=\lambda_i(s,a)$ $(s\in S_i)$, $\delta(s,a)=\delta_i(s,a)$ $(s\in S_i,\ a\neq\alpha)$, $s\in S_i-T$ に対し $\delta(s,\alpha)=\delta_i(s,\alpha)$, $s\in T$ に対し

$\delta(s, \alpha) = s_2^*$ で定義する。$\mathfrak{A} = \mathfrak{A}(\tilde{A}, S_1 \cup S_2, \lambda, \delta, s_1^*, F_1 \cup F_2)$ で生成される alphabet A 上の Asser algorithm $\mathfrak{F}[\mathfrak{A}, \alpha]$ を、$\mathfrak{F}[\mathfrak{A}_1, \alpha]$ に対する $\mathfrak{F}[\mathfrak{A}_2, \alpha]$ の s^{**} における接続と称する。

3. Post algorithm から Asser algorithm への翻訳

既に Asser computable であれば recursive であることを見たが、更に次の定理が成立する。

$\boxed{\text{定理 4}}$ alphabet A 上の Post algorithm $\mathfrak{P}(\sigma)$ に対し、Asser algorithm $\mathfrak{F}[\mathfrak{A}]$ を構成して、$\mathfrak{P}_\sigma(x) = \mathfrak{F}_{\mathfrak{A}}(x)$ $(x \in A^*)$ なるようにすることができる。特に Post computable ならば Asser computable である。

〔証明〕 与えられた Post algorithm $\mathfrak{P}(\sigma)$ の変換素子系列を $\sigma: \tau_1, \cdots\cdots, \tau_n$ とする。各 τ_i に対し automaton $\mathfrak{A}_i = \mathfrak{A}_i(\tilde{A}, S_i, \lambda_i, \delta_i, s_i^*, F_i)$ $(\tilde{A} = A \cup \{\alpha\}, \alpha \notin A)$, $F_i = \{s_i^{**}, t_i^{**}, s^{***}\}$ なるものを与えて（s^{***} 以外共通な state はないものとする）、$(\tau_i)_{\mathfrak{P}}(x)$ が undefined ならば $\mathfrak{F}_{\mathfrak{A}_i, \alpha}(s_i^*, x) = (s_i^{**}, x)$，$(\tau_i)_{\mathfrak{P}}(x) = y$ ならば τ_i が終結的であるとき $\mathfrak{F}_{\mathfrak{A}_i, \alpha}(s_i^*, x) = (s^{***}, y)$，$\tau_i$ が終結的でないとき $\mathfrak{F}_{\mathfrak{A}_i, \alpha}(s_i^*, x) = (t_i^{**}, y)$ となるようにできたとする。このとき、次のような $\mathfrak{F}[\mathfrak{A}_i, \alpha]$ $(i =$

$1, \cdots, n$) を構成する。

1) $\mathfrak{A}[\mathfrak{a}_1]$ を t_1^{**} において $\mathfrak{A}[\mathfrak{a}_1]$ 自身に接続したものを $\mathfrak{A}[\mathfrak{a}_1']$ とする。

2) $\mathfrak{A}[\mathfrak{a}_i']$ ($i < n$) までできたとして、s_i^{**} において $\mathfrak{A}[\mathfrak{a}_i']$ に $\mathfrak{A}[\mathfrak{a}_{i+1}]$ を接続、これに更に自分自身を t_{i+1}^{**} において接続して得られる algorithm を $\mathfrak{A}[\mathfrak{a}_{i+1}']$ とする。

このようにして得られた $\mathfrak{A}[\mathfrak{a}_n']$ が $P_\sigma(x) = g_{\mathfrak{a}_n'}(x)$ ($x \in A^*$) を満足することは明らかである。従って $\mathfrak{a}_i$ の構成が与えられれば証明は終る。$\mathfrak{a}_i$ の構成は τ_i が終結的である場合とそうでない場合とで殆んど変らないから τ_i が終結的で無い場合の構成のみ示す。以下 automaton の構成は diagram で示す。($s \xrightarrow{a/b} t$ は $\lambda(s,a) = b$, $\delta(s,a) = t$ であることを示す。$s \circlearrowleft a/a$ ($a \in \tilde{A}$) なる diagram は省略して記述しない。) 簡単のため、添字 i を省略し τ_i は $x \to y$ なるものとする。

最初に $\mathfrak{B} = \mathfrak{B}(\tilde{A}, S', \lambda', \delta', t^*, F')$, $F' = \{s^{***}, t^{**}\}$, $\mathfrak{A}_{\mathfrak{B}}(t^*, z) = (t^{**}, zy)$ ($z \in A^*$) なる automaton $\mathfrak{B}$ を構成する。

i) $y = \Lambda$ の場合

$$\begin{tikzcd} t^* \arrow[r, "\alpha/\alpha"] \arrow[loop below, "a \neq \alpha / a"] & t^{**} \end{tikzcd}, \quad S' = \{t^*\} \cup F'$$

ii) $y = b_1 \cdots\cdots b_l$ $\quad (b_j \in A)$ の場合

$$\begin{tikzcd} t^* \arrow[r, "\alpha/b_1"] \arrow[loop below, "\frac{a\neq\alpha}{a}"] & t_1 \end{tikzcd}, \quad \begin{tikzcd} t_j \arrow[r, "\alpha/b_{j+1}"] \arrow[loop below, "\frac{a\neq\alpha}{a}"] & t_{j+1} \end{tikzcd} \ (1 \leqq j < l), \quad \begin{tikzcd} t_l \arrow[r, "\alpha/b_l"] \arrow[loop below, "\frac{a\neq\alpha}{a}"] & t^{**} \end{tikzcd},$$

$\mathcal{S} = \{t^*, t_1, \cdots\cdots, t_l\} \cup F$

次に望む $\mathfrak{T}[\mathfrak{A}]$ の構成であるが、$x = \Lambda$ の場合は $\mathfrak{A} = \mathfrak{B}$ でよいから、$x \neq \Lambda$, $x = a_1 \cdots\cdots a_m$ $(a_j \in A)$ とする。
$\mathfrak{A}' = \mathfrak{A}'(\tilde{A}, \mathcal{S}'', \alpha'', \delta'', s^*, F'')$, $F'' = \{u^{**}, s^{**}\}$, $\mathcal{S}'' = \{s^*, s_1^1, \cdots\cdots, s_m^1, s_0^2, \cdots\cdots, s_m^2\} \cup F''$ なる $\mathfrak{A}'$ で state は全て $\mathfrak{B}$ の state と異なる、次のような diagram をもつ automaton $\mathfrak{A}'$ を構成する。

$$\begin{tikzcd} s^* \arrow[r, "a_1/a_1"] \arrow[d, "\frac{a\neq a_1}{a}"] & s_1^1 \\ s^{**} & \end{tikzcd}, \quad \begin{tikzcd} s_j^1 \arrow[r, "a_{j+1}/a_{j+1}"] \arrow[d, "\frac{a\neq a_{j+1}}{a}"] & s_{j+1}^1 \\ s^{**} & \end{tikzcd} \ (1 \leqq j < m), \quad \begin{tikzcd} s_m^1 \arrow[r, "\alpha/\alpha"] \arrow[loop below, "\frac{a\neq\alpha}{a}"] & s_0^2 \end{tikzcd},$$

$$s_j^2 \xrightarrow{a/\alpha} s_{j+1}^2 \quad (0 \leqq j < m), \quad \begin{tikzcd} s_m^2 \arrow[r, "\alpha/\alpha"] \arrow[loop left, "\frac{a\neq\alpha}{a}"] & u^{**} \end{tikzcd}$$

この $\mathfrak{T}[\mathfrak{A}']$ に $\mathfrak{T}[\mathfrak{B}]$ を u^{**} において接続すれば、$\mathfrak{T}[\mathfrak{A}]$ が得られる。

§6. Turing algorithm

alphabet A, $\mathcal{S}$ と $\alpha \notin A$ なる α をとり、$\tilde{A} = A \cup \{\alpha\}$ とおく
$a, b \in \tilde{A}$, $s, t \in \mathcal{S}$ に対し、$s\,a\,b\,t$, $s\,a\,R\,t$, $s\,a\,L\,t$ の

いずれかの形式を quadruple，この sa の部分を左部分という。（R，L は考える対象となるどのような alphabet にも含まれない記号としておく。）A と S と，有限個の互いに左部分の異なる quadruples の集合 Φ と，S の特定の元 s^* と，α との組合せ $T = T(A, S, \Phi, s^*, \alpha)$ を Turing machine と称する。この Turing machine T に対し，$X = \widehat{A}^* \times S \times N$ $(N = \{1, 2, \cdots\})$ とおくと，Φ により次のような $\sigma_1, \sigma_2, \sigma_3 \in p.f.(X, X)$ が導かれる。ただしここで，$l(x)$ は word x の長さ，$P_n(x)$ は x の n 番目の項，$S_n^b(x)$ は x の n 番目の項を b でおきかえて得られる word を表わすものとする。

$$\sigma_1(x, s, n) = \begin{cases} (S_n^b(x), t, n) & n \leqq l(x) \wedge sP_n(x)bt \in \Phi \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\sigma_2(x, s, n) = \begin{cases} (x, t, n+1) & n < l(x) \wedge sP_n(x)Rt \in \Phi \\ (x\alpha, t, n+1) & n = l(x) \wedge sP_n(x)Rt \in \Phi \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\sigma_3(x, s, n) = \begin{cases} (x, t, n-1) & 1 < n \leqq l(x) \wedge sP_n(x)Lt \in \Phi \\ (\alpha x, t, 1) & n = 1 \wedge sP_1(x)Lt \in \Phi \\ \text{undefined} & \text{otherwise} \end{cases}$$

Φ の構造により各 σ_i の定義域は互いに disjoint である。$\sigma \in p.f.(X, X)$ を $\sigma(\varepsilon) = \sigma_i(\varepsilon)$ $(\varepsilon \in \mathrm{Dom}(\sigma_i))$，$\mathrm{Dom}(\sigma) = \bigcup_{i=1}^{3} \mathrm{Dom}(\sigma_i)$ で定義する。$\Gamma \in p.f.(X^* - \{1_X\}, X)$ を $\Gamma(\varepsilon_1 \cdots \varepsilon_n) = \sigma(\varepsilon_n)$ $(\varepsilon_i \in X)$ で定め，$\gamma =$

流 (X, Γ, X^*) をとる。この γ を、Turing machine T によるTuring algorithm と称する。$\varphi: \tilde{A}^* \longrightarrow A^*$ なる homomorphism φ で、$\varphi(\alpha) = \Lambda$, $\varphi(a) = a$ $(a \in A)$ で定められる φ をとり、$(x_1, \cdots\cdots, x_m) \in (A^*)^m$ に対し、$f(x_1, \cdots, x_m) = \varphi([\gamma]_1(x_1\alpha \cdots x_m\alpha, s^*, 1))$ とおけば、A上m変数の word function f が得られる。これをTuring machine T により定まるm変数のword function といい、$\Psi_T^{(m)}$ とかく。このword function 又はこれを制限して得られる(或はこれから導かれる) word function を Turing computable であるという。

$\boxed{\text{定理5}}$ alphabet A上の Asser algorithm $\mathfrak{A}[\mathfrak{O}, \alpha]$ に対して、Turing machine T を構成して、$g_{\mathfrak{O}}(x) = \Psi_T^{(1)}(x)$ $(x \in A^*)$ となるようにすることができる。特に Asser computable ならば Turing computable である。

〔証明〕 $\mathfrak{O} = \mathfrak{O}(\tilde{A}, S, \delta, \lambda, s^*, F)$, $\tilde{A} = A \cup \{\alpha\}$ とする。$s^* \in F$ ならば $g_{\mathfrak{O}}(x) = x$ $(x \in A^*)$ であるから Turing machine $T = T(A, S, \phi, s^*, \alpha)$ (ϕ は空集合) により $g_{\mathfrak{O}}(x) = \Psi_T^{(1)}(x)$ となる。次に $s^* \notin F$ とする。$\beta \notin A \cup \{\alpha\}$ なる β と、$\Delta, \tilde{\Delta} \notin S$ なる $\Delta, \tilde{\Delta}$ をとる。各 $s \in S$ に対し、互いに異なる新しい symbol $\bar{s}, \tilde{s}$ をとり、$\bar{S} = \{\bar{s} ; s \in S\}$, $\tilde{S} = \{\tilde{s} ; s \in S\}$ とおく。$\bar{\delta}: S \times (A \cup \{\alpha, \beta\}) \longrightarrow \bar{S} \cup \tilde{S}$, $\bar{\lambda}$:

$S \times (A \cup \{\alpha, \beta\}) \longrightarrow A \cup \{\beta\}$ なる写像を次のように定める。

$a \in \tilde{A}$ に対し、

$$\bar{\delta}(s, a) = \begin{cases} \overline{\delta(s,a)} & (a \neq \alpha) \\ \overline{\delta(s,\alpha)} & (a = \alpha) \end{cases}, \quad \bar{\lambda}(s, a) = \begin{cases} \lambda(s,a) & \lambda(s,a) \neq \alpha \\ \beta & \lambda(s,a) = \alpha \end{cases}$$

また、$\bar{\delta}(s, \beta) = \bar{s}$, $\bar{\lambda}(s, \beta) = \beta$。

Φ を次のような形の quadruple 全ての集合とする。

$s a \bar{\lambda}(s,a) \bar{\delta}(s,a)$ $(a \in \tilde{A} \cup \{\beta\}, s \in S)$, $\bar{s} a R s$ $(a \in A \cup \{\beta\}, s \in S)$, $\tilde{s} a L s$ $(a \in A \cup \{\beta\}, s \in S - F)$, $\tilde{s} \alpha R s$ $(s \in S - F)$, $\tilde{s} \beta \Delta \alpha$ $(s \in F)$, $\Delta a L \tilde{\Delta}$ $(a \in \tilde{A})$, $\tilde{\Delta} \beta \alpha \Delta$, $\tilde{\Delta} a a \Delta$ $(a \in A)$。 Turing machine $T = T(A \cup \{\beta\}, S \cup \bar{S} \cup \tilde{S} \cup \{\Delta, \tilde{\Delta}\}, \Phi, s^*, \alpha)$ により、$\Psi_T^{(1)}(x) = f_\sigma(x)$ $(x \in A^*)$ となることは容易に確かめることができる。

$\boxed{\text{定理6}}$ 与えられた Turing machine $T = T(A, S, \Phi, s^*, \alpha)$ に対し、Markov algorithm $\mathfrak{M}(\sigma)$ を与えて、$\Psi_T^{(m)}(x_1, \cdots, x_m) = \mathfrak{M}_\sigma(x_1 \alpha \cdots \alpha x_m)$ $(x_1, \cdots, x_m \in A^*)$ となるようにすることができる。特に word function が Turing computable であれば Markov computable である。

［証明］ Φ の quadruple $sabt$ に対し、変換素子系列 $\{sa \to tb\}$ を対応させ、quadruple $saRt$ に対し、変換素子系列 $\{sab \to asb\ (b \in \tilde{A}),\ sa \to as\alpha\}$ を対応させ、quadruple $saLt$ に対し、変換素子系列 $\{bsa \to sba\ (b \in \tilde{A}),\ sa \to s\alpha a\}$ を対応させ、Φ の quadruple の

左部分に現われない sa ($s \in S$, $a \in \tilde{A}$) に対しては $\beta \notin S \vee \tilde{A}$ なる symbol β をとって、変換素子系列 $\{sa \to \beta a\}$ を対応させる。このような対応によって生ずる全ての変換素子系列を任意の順序で並べて得られる変換素子系列を Π とする。今までのどの元とも異なる新しい symbol $\tilde{\beta}$ をとって、変換素子系列 σ を次のように構成する。

σ : $\beta a \to a\beta$ ($a \in A$), $\beta\alpha \to \beta$, $\beta \to \tilde{\beta}$, $a\tilde{\beta} \to \tilde{\beta}a$ ($a \in A$), $\alpha\tilde{\beta} \to \tilde{\beta}$, $\tilde{\beta} \to \cdot\Lambda$, Π, $s^* \to s^*\alpha$, $\Lambda \to s^*$

これにより、$\Psi_T^{(m)}(x_1, \cdots, x_m) = \mathfrak{M}_\sigma(x_1\alpha \cdots \alpha x_m)$ ($x_1, \cdots, x_m \in A^*$) は容易に確かめることができる。

ところで、与えられた recursive function を、Markov, Post, Asser 又は Turing の各 algorithm で表現することは簡単である。実際、recursive function を実現する Turing machine の構成はよく知られているし、また次の §7 でも、recursive function を algorithm で表現する1つの方法を示す。従って、定理2～定理6にこの結果をあわせて、次の定理を得る。

$\boxed{\text{定理7}}$ recursive な手続, Markov, Post, Asser, Turing の各 algorithm のどの2つも相互に翻訳可能である。特に、partial recursive であること、Markov, Post, Asser, Turingのいずれかの意味でcomputableであることは全て同等である。

§7. recursiveな手続からalgorithmへの翻訳

以下にrecursiveな手続を順を追ってalgorithmに翻訳していく。Aをalphabet, $A' = A \cup \{*\}$ $(* \notin A)$とする。

(1) $a \in A'$によるsuccessor $+_a$ $(+_a(x) = xa)$.

σを$\Lambda \to \cdot a$なる変換素子系列として、

$+_a(x) = P_\sigma(x)$ $(x \in (A')^*)$

(2) eraser $\phi^{(n)}$ $(\phi^{(n)}(x_1, \cdots, x_n) = \Lambda)$.

σを$a \to \Lambda$ $(a \in A')$なる変換素子系列とすれば、

$\phi^{(n)}(x_1, \cdots, x_n) = P_\sigma(x_1 * \cdots * x_n)$ $(x_i \in A^*)$

(3) projection $\pi^{(n)}_m$ $(\pi^{(n)}_m(x_1, \cdots, x_n) = x_m,\ 1 \leq m \leq n)$.

$\beta \notin A'$なるβをとって、変換素子系列σを次のように与える。σ: $*^{m-1} \to \beta$, $\beta a \to a\beta$ $(a \in A)$, $\beta * a \to \beta *$ $(a \in A')$, $\beta * \to \Lambda$, $\beta \to \Lambda$, $a * \to *$ $(a \in A)$

すると、$\pi^{(n)}_m(x_1, \cdots, x_n) = \mathfrak{M}_\sigma(x_1 * \cdots * x_n)$ $(x_i \in A^*)$

(4) composition

i) Asser algorithmの接続で考えれば、それぞれA上1変数, n変数のcomputableなword function $f^{(1)}$, $g^{(n)}$に対し

、$f^{(1)} \circ g^{(n)}$ を実現する algorithm が構成できることは明らかである。

ii) alphabet A' の各元 a に対し、新しい symbol $\bar{a}$ を対応させ（全ての $\bar{a}$ は互いに異るとして）、$\overline{A'} = \{ \bar{a} \,;\, a \in A' \}$ とおく。$\beta \notin A' \cup \overline{A'}$ なる symbol β をとって、

σ_1 : $a \to \bar{a}\, a \quad (a \in A')$

σ_2 : $\bar{a}\, a \to a\, \bar{a} \quad (a \in A')$, $\bar{a} \to \cdot \beta a \, (a \in A')$, $\Lambda \to \cdot \beta$

σ_3 : $\bar{a} \to a \quad (a \in A')$

なる変換素子系列 $\sigma_1, \sigma_2, \sigma_3$ を与える。$f = \mathfrak{M}_{\sigma_3} \circ \mathfrak{M}_{\sigma_2} \circ \mathfrak{P}_{\sigma_1}$ とおけば、$f(x) = x \beta x \quad (x \in (A')^*)$

iii) $\beta \notin A'$ なる β をとり、$\sigma_1 : \Lambda \to \cdot \beta$, $\sigma_2 : a \to a \, (a \in A')$, $\beta \to \cdot \Lambda$ により、$f = \mathfrak{P}_{\sigma_2} \circ \mathfrak{P}_{\sigma_1}$ とおけば、$f(x \beta y) = y \beta x \quad (x, y \in (A')^*)$

iv) $\mathfrak{A} = \mathfrak{A}(\widetilde{A'}, S, \delta, \lambda, s^*, F)$ （$\widetilde{A'} = A' \cup \{\alpha\}$）に対し、$f = g_{\mathfrak{A}, \alpha}$ とおく。すると、新しい automaton $\mathfrak{A}'$ をつくって、$g_{\mathfrak{A}', \alpha}(x \beta y) = x \beta f(y)$ （$x, y \in (A')^*$, β は $\beta \notin A'$ なる symbol）となるようにすることができるのは明らかである。

v) 上記 i) ～ iv) により、computable な A' 上 1 変数の word function f, g に対し、$h(x) = f(x) * g(x) \quad (x \in (A')^*)$ なる h を実現する algorithm が構成できる。

vi) A 上 m 変数の word function f と A 上 n 変数の word function $g_1, \cdots, g_m$ が computable であるとき、A' 上 n 変数の h_1 と A 上 n 変数の h で、

$$h_1(x_1, \cdots, x_n) = g_1(x_1, \cdots, x_n) * \cdots * g_m(x_1, \cdots, x_n)$$

$$h(x_1, \cdots, x_n) = f(g_1(x_1, \cdots, x_n), \cdots\cdots, g_m(x_1, \cdots, x_n))$$

$$(x_1, \cdots, x_n \in A^*)$$

なる h_1, h を与える algorithm が構成できることは、i), V) により明らかである。(h_1 を $g_1, \cdots, g_m$ の combination, h を $f, g_1, \cdots, g_m$ の composition という。)

(5) minimalization $\mu_y^a[f(y, x_1, \cdots, x_n) = \Lambda]$.

f を A 上 $n+1$ 変数の computable word function, a を A の特定の元とする。$x = x_1 * \cdots * x_n$ ($x_i \in A^*$) とする。必要に応じて x は $(x_1, \cdots, x_n)$ を表すものとも解釈する。$\beta \notin A'$ をとる。$a^m * x \longrightarrow f(a^m, x)\beta a^m * x$ なる変換を Asser algorithm で実現する automaton を $\mathfrak{A}_1$, その final state を s^{**} とする。(final state は 1 つと仮定してかまわない。) $\mathfrak{A}_1$ の states alphabet とは disjoint な states alphabet をもつ次のような automaton を $\mathfrak{A}_2$ とする。ただし、下の diagram で指示のない input に対しては、state は全て diagram にない state Δ にかわり、$\Delta \circlearrowright$ で、Δ は final state では無い、とする。final states alphabet $= \{t_1^{**}, t_2^{**}\}$ として、任意の $b \in A$ に対し、

$t^* \xrightarrow{b/\alpha} t_1 \xrightarrow{\beta/a} t_2 \xrightarrow{\alpha/\alpha} t_1^{**}$, $t_1 \circlearrowright b/\alpha$, $t_2 \circlearrowright \begin{matrix} b/b \\ */* \end{matrix}$, $t^* \xrightarrow{\beta/\alpha} t_3$, $t_3 \circlearrowright a/a$

, $t_3 \xrightarrow{*/\alpha} t_4 \xrightarrow{\alpha/\alpha} t_2^{**}$, $t_4 \circlearrowright \begin{matrix} b/\alpha \\ */\alpha \end{matrix}$

このような $\mathfrak{A}_2$ により、$z \in A^*$ に対し、$g_{\mathfrak{A}_2,\alpha}(z\beta a^m * x, t^*)$ は $z=\Lambda$ ならば (a^m, t_2^{**})、$z \neq \Lambda$ ならば $(a^{m+1} * x, t_1^{**})$ になる。$g[\mathfrak{A}_1,\alpha]$ に $g[\mathfrak{A}_2,\alpha]$ を t^{**} において接続し、さらにこれに自分自身を t_1^{**} において接続する。その結果得られる algorithm を $g[\mathfrak{A},\alpha]$ とすれば、$\mu_y^a[f(y,x)=\Lambda] = g_{\mathfrak{A}}(*x)$ となる。$x \to *x$ の変換は Markov algorithm $\mathfrak{M}(\sigma)$; $\sigma : \Lambda \to \cdot *$ により実現できるから、minimalization を実現する Asser algorithm が得られることがわかる。

(6) primitive recursion

A上 $n+2$ 変数の word function h_a $(a \in A)$ と、n 変数の word function g を computable であるとする。先のように、$x = x_1 * \cdots * x_n$ $(x_i \in A^*)$ とし、これは必要に応じて $(x_1, \cdots, x_n)$ とも解釈するものとする。g と h_a $(a \in A)$ による premitive recursion を algorithm で実現するには、$a_1, \cdots, a_m \in A$ として $*x \to g(x)$, $a_1 \cdots a_m * x \to a_1 \cdots a_m * g(x) * x \to a_2 \cdots a_m * y_1 * x \to a_3 \cdots a_m * y_2 * x \to \cdots\cdots \to *y_m * x \to y_m$ $(y_1 = h_{a_1}(\Lambda, g(x), x)$,

$y_{i+1} = h_{a_{i+1}}(a_1 \cdots a_i, y_i, x)$ ）なる変換が algorithm で実現できればよい。そのためには $\beta \notin A'$ として、

i）$z \in A^*$ に対し、$z * x \longrightarrow \beta z * y(x) * x$ なる変換を実現する algorithm

ii）$\mathfrak{g}_{\mathfrak{a},\alpha}(s^*, z_1 \beta z_2 * y * x)$ $(z_1, z_2, y \in A^*)$

$$= \begin{cases} (s_1^{**}, z_1 a \beta z_3 * h_a(z_1, y, x) * x) & (z_2 = a z_3) \\ (s_2^{**}, y) & (z_2 = \Lambda) \end{cases}$$

なる automaton $\mathfrak{a}$

が構成できればよい。i）の構成は自明である。ii）の構成は states alphabet が互いに disjoint な automata $\mathfrak{a}_1$, $\mathfrak{a}_a$ $(a \in A)$, $\mathfrak{a}_2$ で、

$$\mathfrak{g}_{\mathfrak{a}_1}(s_1^*, z_1 \beta z_2 * y * x) = \begin{cases} (t_a^{**}, z_1 \beta z_2 * y * x) & (z_2 = a z_3) \\ (t^{**}, z_1 \beta z_2 * y * x) & (z_2 = \Lambda) \end{cases}$$

$$\mathfrak{g}_{\mathfrak{a}_a}(t_a^*, z_1 \beta a z_3 * y * x) = (s_1^{**}, z_1 a \beta z_3 * h_a(z_1, y, x) * x)$$

$$\mathfrak{g}_{\mathfrak{a}_2}(t^*, z_1 \beta z_2 * y * x) = (s_2^{**}, y)$$

なるものを構成して、$\mathfrak{g}[\mathfrak{a}_1]$ に $\mathfrak{g}[\mathfrak{a}_a]$ を t_a^{**} において接続（全ての $a \in A$ について）、更にこれに t^{**} において $\mathfrak{g}[\mathfrak{a}_2]$ を接続したものを $\mathfrak{g}[\mathfrak{a}]$ とすればよい。$\mathfrak{a}_1$, $\mathfrak{a}_2$ の構成は簡単であるし、$\mathfrak{a}_a$ も、$z_1 \beta a z_3 * y * x \longrightarrow z_1 \beta a z_3$, $z_1 \beta a z_3 * y * x \longrightarrow z_1 * y * x$, $z_1 \beta a z_3 * y * x \longrightarrow x$ の各変換を実現する Asser algorithm が直ちに与えられることを考

えれば簡単に構成できる。以上により primitive recursion を実現する Asser algorithm が得られることがわかる。

参考文献

G. Asser, Über eine Darstellung der rekursiven Wortfunktionen in endlichen Automaten, Zeitschr. f. math. Logik und Grundlagen d. Math. 12 (1966) 1–12

G. Asser, Rekursiven Wortfunktionen, Zeitschr. f. math. Logik und Grundlagen d. Math. 6 (1960) 258–278

M. Davis, Computability and Unsolvability, McGraw-Hill, Inc. (1958)

A.A. Марков, Теория алгорифмов, Труд. Мат. инс. им. В.А. Стеклова, 42 (1954)