

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、場合に依じて適当な algorithm の使いわけができることと便利である。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、十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互通性に着目して統一的に定義し、これらと recursive な手続の相互翻訳の 1 つの例として、 $\text{Markov} \rightarrow \text{Post} \rightarrow \text{Aaser} \rightarrow \text{Turing} \rightarrow \text{Markov}$ の"順での翻訳と、 $\text{Aaser} \rightarrow \text{recursive}$, $\text{recursive} \rightarrow (\text{Markov}, \text{Post}, \text{Aaser})$ 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なすことになる。

特に、ここで述べる $\text{Aaser} \rightarrow \text{recursive}$ 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、 $X \rightarrow Y$ の一対対応を partial function と称し、この集合を $p.f.(X, Y)$ で表す。 $f \in p.f.(X, Y)$ は、その定義域を $\text{Dom}(f)$ で表し、 X の元 x が、 $x \in \text{Dom}(f)$ であれば $f(x)$ は defined , $x \notin \text{Dom}(f)$ 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 $Z \rightarrow Y$ 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 (混乱のかそれのない時は単に Λ) で表す。 A^* の元 $a_1, \dots, a_n \ (a_i \in A)$ を必要に応じて $\langle a_1, \dots, a_n \rangle$ のように表す。 a_i をこの word の i 番目の項と称する。

$X = A^* \times \dots \times A^* \ (A^* \text{ } n \text{ 個の直積, } (A^*)^n \text{ と略記})$ 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 $B \subset A, B \neq \emptyset$ 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 ($A - B \neq \emptyset$ と仮定して)。
 A 上 1 変数の word function f が任意の $x_1, \dots, x_m \in B^*$ (m は与えられた正整数) に対し、 $f(x_1, \alpha, \dots, \alpha, x_m)$ が *defined* であれば $\in B^*$ なる性質をもつとき、 $\varphi(x_1, \dots, x_m) = f(x_1, \alpha, \dots, \alpha, x_m)$ 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

d 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。

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

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

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

$a \rightarrow \gamma(a)$ なる写像 $\gamma: A \rightarrow \mathcal{P}(A^*) = \{X; X \subset A^*\}$ を、 C に制御された Γ の iteration といい、 $\gamma = \text{itr}(A, \Gamma, C)$ 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。

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 $g_1, \dots, g_n$ により、 $\Gamma(\langle x_1, \dots, x_t \rangle) = (g_1(x_t), \dots, g_n(x_t))$ ($x_i \in X$)

2) A 上 n 変数の general recursive word function f と、負でない整数 m があって、 $\langle x_1, \dots, x_t \rangle \in C \iff f(x_{t-m})$ キ Λ_A 又は $t \leq m$

なる条件を満たすものとする。このとき、 $\sigma = \text{str}(X, \Gamma, C)$ に対し、 $[\sigma]_2$ の各成分 $([\sigma]_2(w_1, \dots, w_n) = (w'_1, \dots, w'_n))$ とするとき、 $(w_1, \dots, w_n) \rightarrow w'_i$ 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 ($i=1, \dots, n$) を次のように構成する。

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

$$h_i(\gamma a, x^{(n)}) = g_i(h_1(\gamma, x^{(n)}), \dots, h_n(\gamma, x^{(n)}))$$

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

A の特定の元 a_0 を選び、 $\tau_{a_0} \in w \rightarrow w a_0$ ($w \in A^*$) なる successor function として、

$$\varphi(x^{(n)}) = (\tau_{a_0})^m(\mu_{\gamma}^{a_0}[f(h_1(\gamma, x^{(n)}), \dots, h_n(\gamma, x^{(n)})) = \Lambda])$$

なる partial recursive word function φ を作る。

$[r]_2(x^{(n)})$ の各成分は $h_i(\varphi(x^{(n)}), x^{(n)})$ であるから partial recursive である。特に、 $g_1, \dots, g_n, f$ が primitive recursive であれば、 $[r]_2$ の各成分は minimization 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 $*$ をとり、 $A \cup \{*\} = \tilde{A}$ 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 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) $f(\Lambda) = \Lambda$, $f(a_1 \dots a_k) = a_k \dots a_1$ なる f は primitive recursive

〔証〕 $f_a(\Lambda) = a$, $f_a(xb) = f_a(x)b$ なる f_a により、

$f(\Lambda) = \Lambda$, $f(xa) = f_a(f(x))$

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 によって, $\sigma(\Lambda) = \Lambda$, $\sigma(xa) = \sigma_a(\pi(xa), \sigma(x))$

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 によって,

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

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

[証] $J_2 = \rho \circ J_1 \circ \rho$ とおいて, ${}^m J_i = (J_2)^{i-1} \circ (J_1)^{m-i}$.

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

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

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

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

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

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

m) は primitive recursive である。

[証] $h_i^{(a)}$, g_i を $\tilde{A}$ 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ま拡張したものを $\tilde{h}_i^{(a)}$, $\tilde{g}_i$ とする。 $\tilde{h}_i^{(*)}$ を $\tilde{h}_i^{(*)}(x^{(m+n+1)}) = \Lambda$ なるものとする。

$$H_a(\gamma, z, x^{(n)}) = \tilde{h}_1^{(a)}(\gamma, {}^mJ_1(z), \dots, {}^mJ_m(z), x^{(n)})$$

$$* \dots * \tilde{h}_m^{(a)}(\gamma, {}^mJ_1(z), \dots, {}^mJ_m(z), x^{(n)})$$

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

なる H_a とする。

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

$$F(\gamma a, x^{(n)}) = H_a(\gamma, F(\gamma, x^{(n)}), x^{(n)})$$

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

$$f_i(\gamma, x^{(n)}) = \tilde{f}_i(\gamma, x^{(n)}) \quad (\gamma \in A^*, x^{(n)} \in (A^*)^n) \text{ となり、}$$

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、 $x \rightarrow y$ 又は $x \rightarrow \cdot y$ なる形式を変換素子と呼び、 $x \rightarrow \cdot y$ なる変換素子は終結的であるという。 $w_1 \rightarrow (\cdot) w_2$ 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) $\tau_p(x)$: $x = w_1 x_1$ ($x_1 \in A^*$) となつてゐる場合に限り $defined$ 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、 $\Gamma \in p.f.(X^* - \{\Lambda_x\}, X)$, $C \subset X^*$ と、 $\Gamma(\langle x_1, \dots, x_t \rangle) = \sigma_m(x_t)$, $\langle x_1, \dots, x_t \rangle \notin C \iff (x_{t-1} \text{ が } \sigma_m\text{-終結的 かつ } t \neq 1)$ なるものとする。 $itr(X, \Gamma, C)$ をとって、これを $M(\sigma)$ とおく。添字 m を p にかきかえて、 $P(\sigma)$ を同様にして得る。 $[M(\sigma)]_1, [P(\sigma)]_1$ をそれぞれ M_σ, P_σ で表す。 M_σ , 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、 $M(\sigma)$ 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。

定理 2 alphabet A 上の Markov algorithm $M(\sigma)$ に対

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

[証明] $A \cap \tilde{A} = \emptyset$, $|A| = |\tilde{A}|$ ($| \cdot |$ は元の個数とする) なる alphabet $\tilde{A}$ と、 $\varphi: A^* \rightarrow \tilde{A}^*$ なる isomorphism をとる。 $x \in A^*$ に対し、 $\varphi(x)$ を $\tilde{x}$ 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 ($A, \tilde{A}$ に含まれない symbol) $\#, \#_1, \dots, \#_{n+1}, \tilde{\#}_1, \dots, \tilde{\#}_{n+1}, \beta$ をとる。 $A, \tilde{A}$ 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 $x_1 a y_1 \rightarrow (.) x_2 a y_2$ ($a \in A$) 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、この順序で並んだものとする。

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

$$\sigma'_i: \begin{cases} \#_i \tilde{x}_i \rightarrow \tilde{y}_i \tilde{\#}_i \\ \#_i \tilde{a} \rightarrow \tilde{a} \#_i \quad (a \in A) \end{cases}$$

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

更に B 上の変換素子系列 σ' と次のように構成する。

$$\sigma' : \left\{ \begin{array}{l} \#_{n+1} \widetilde{a} \rightarrow a \beta \#_{n+1} \quad (a \in A) \\ \beta a \rightarrow \beta a \quad (a \in A) \\ a \beta \rightarrow a \quad (a \in A) \\ \#_{n+1} \beta \rightarrow \widetilde{\#}_{n+1} \\ \widetilde{\#}_{n+1} \rightarrow \cdot \wedge \\ \beta \rightarrow \beta \\ \widetilde{a} \rightarrow \widetilde{a} \quad (a \in A) \\ \vdots \\ \sigma'_1 \\ \vdots \\ \sigma'_n \\ a \rightarrow \#\# a \quad (a \in A) \\ \#\# a \# \rightarrow \# \beta \#_1 \widetilde{a} \quad (a \in A) \\ \# a \# \rightarrow \widetilde{a} \\ \#\# a \rightarrow \beta \#_1 \widetilde{a} \quad (a \in A) \\ \wedge \rightarrow \beta \#_1 \end{array} \right.$$

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

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

§ 5. Asper 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 $\mathcal{O} = \mathcal{O}(\tilde{A}, S, \delta, \lambda, s^*, F)$ を与える。ここに $\tilde{A}$ は input-output alphabet, S は states alphabet, δ は state function $S \times \tilde{A} \rightarrow S$, λ は output function $S \times \tilde{A} \rightarrow \tilde{A}$, s^* は initial state, F は final states alphabet ($F(S)$) 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 $\lambda': S \times \tilde{A} \rightarrow A^*$ と、 $\tilde{\delta}(s, \Lambda) = s$, $\tilde{\delta}(s, \alpha a) = \delta(\tilde{\delta}(s, x), a)$; $\tilde{\lambda}(s, \Lambda) = \Lambda$, $\tilde{\lambda}(s, \alpha a) = \tilde{\lambda}(s, x) \cdot \lambda'(\tilde{\delta}(s, x), a)$ により、 $\tilde{\delta}: S \times (\tilde{A})^* \rightarrow S$, $\tilde{\lambda}: S \times (\tilde{A})^* \rightarrow A^*$ を定義する。(これは簡単に $S \cup \tilde{A}$ 上 2 変数の primitive recursive word function に分解されてくる。) $S \times A^* = X$ とおき、 $\Gamma_1 \in p.f. (X^* - \{\Lambda_x\}, X)$ と $C_1 \subset X^*$ と、

$$\Gamma_1(\langle (s_1, x_1), \dots, (s_n, x_n) \rangle) = (\delta(s_n, \alpha), x_n \cdot \lambda'(s_n, \alpha))$$

$$\langle (s_1, x_1), \dots, (s_n, x_n) \rangle \in C_1 \iff \lambda'(s_{n-1}, \alpha) \neq \Lambda \text{ 又は } n=1$$

で定義し、 $\gamma_1 = \text{itr}(X, \Gamma_1, C_1)$ に対し、 $g(\alpha, x) = [\gamma_1]_2(\tilde{f}(\alpha, x), \tilde{x}(\alpha, x))$ なる $g \in p.f.(X, X)$ を作る。 ($g(\alpha, x)$ 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、 $T = \{\alpha \in S; \lambda(\alpha, \alpha) = x\}$ に対し、 $g_1(\alpha, x) = \begin{cases} (f(\alpha, \alpha), x \cdot \lambda(\alpha, \alpha)) & (\alpha \notin T) \\ (\alpha, x) & (\alpha \in T) \end{cases}$

$$g_2(\alpha, x) = \begin{cases} (\Lambda, \Lambda) & (\alpha \notin T) \\ (f(\alpha, \alpha), x) & (\alpha \in T) \end{cases} \quad \text{とおいてやれば、} |S|$$

$= m$ として、 $g_2(g_1^m(\alpha, x))$ は $[\gamma_1]_2(\alpha, x)$ が defined のときはその値に等しく、そうでないときは (Λ, Λ) 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、

$$\Gamma_2(\langle (\alpha_1, x_1), \dots, (\alpha_n, x_n) \rangle) = g(\alpha_n, x_n)$$

$$\langle (\alpha_1, x_1), \dots, (\alpha_n, x_n) \rangle \in C_2 \iff \alpha_n \notin F$$

で定義し、 $\gamma_2 = \text{itr}(X, \Gamma_2, C_2)$ 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 minimization を / 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 $g_{\alpha, \alpha}$ で、 $g_{\alpha, \alpha}(\alpha^*, x)$ の 2 成分を $f_{\alpha, \alpha}$ で表す。 $f_{\alpha, \alpha}$ は A 上の word function である。

$f_{\alpha, \alpha}$ 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 f_2 をこの word function の automaton α と Blank symbol α による Asser algorithm と称し、 $\mathcal{A}[\alpha, \alpha]$ で表す。Blank symbol α が「解ずみ」であると思われれば、 α は略して単に $\mathcal{A}\alpha$, f_{α} , $\mathcal{A}[\alpha]$ 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。

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

2. Asser algorithm の接続

A は alphabet, $\alpha \notin A$ なる α に対し、 $\tilde{A} = A \cup \{\alpha\}$ とおく。 α_1, α_2 を $\alpha_i = \alpha_i(\tilde{A}, S_i, \lambda_i, \delta_i, \rho_i^*, F_i)$ ($i=1, 2$) なる finite automaton と、 $\rho \in S_1, S_2$ なる state ρ に対し $\lambda_1(\rho, a) = \lambda_2(\rho, a)$, $\delta_1(\rho, a) = \delta_2(\rho, a)$ ($a \in \tilde{A}$) なるものとする。特定の $\rho^{**} \in F_1$ に対し、 $T = \{\rho \in S_1; \delta_1(\rho, \alpha) = \rho^{**}, \lambda_1(\rho, \alpha) = \alpha\}$ とおく。 $S_1 \cup S_2 = S$ とし、 $\lambda: S \times \tilde{A} \rightarrow \tilde{A}$, $\delta: S \times \tilde{A} \rightarrow S$ を $\lambda(\rho, a) = \lambda_i(\rho, a)$ ($\rho \in S_i$), $\delta(\rho, a) = \delta_i(\rho, a)$ ($\rho \in S_i, a \neq \alpha$), $\rho \in S_i - T$ に対し $\delta(\rho, \alpha) = \delta_i(\rho, \alpha)$, $\rho \in T$ に対し

$\delta(\rho, \alpha) = \rho_2^*$ で定義する。 $\mathcal{O} = \mathcal{O}(\tilde{A}, \delta_1 \cup \delta_2, \lambda, \delta, \rho_1^*, F_1 \cup F_2)$ で生成される alphabet A 上の Asser algorithm $\mathcal{O}[\mathcal{O}, \alpha]$ と、 $\mathcal{O}[\mathcal{O}_1, \alpha]$ に対する $\mathcal{O}[\mathcal{O}_2, \alpha]$ の ρ^{**} 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" であることも見だが、更に次の定理が成立する。

定理 4 alphabet A 上の Post algorithm $P(\sigma)$ に対し、 Asser algorithm $\mathcal{O}[\mathcal{O}]$ と構成して、 $P_{\mathcal{O}}(x) = \mathcal{O}_{\mathcal{O}}(x)$ ($x \in A^*$) なるようにする ことが出来る。特に Post computable ならば Asser computable である。

[証明] 与えられた Post algorithm $P(\sigma)$ の変換素子系列を $\sigma: \tau_1, \dots, \tau_n$ とする。各 τ_i に対し automaton $\mathcal{O}_i = \mathcal{O}_i(\tilde{A}, \delta_i, \lambda_i, \delta_i, \rho_i^*, F_i)$ ($\tilde{A} = A \cup \{\alpha\}, \alpha \in A$), $F_i = \{\rho_i^*, t_i^{**}, \rho_i^{***}\}$ なるものを与えて (ρ_i^{***} 以外共通な state はないものとする)、 $(\tau_i)_P(x)$ が undefined ならば $\mathcal{O}_{\mathcal{O}_i, \alpha}(\rho_i^*, x) = (\rho_i^{**}, x)$, $(\tau_i)_P(x) = \gamma$ ならば $\mathcal{O}_{\mathcal{O}_i, \alpha}(\rho_i^*, x) = (\rho_i^{***}, \gamma)$, τ_i が終結的でないとき $\mathcal{O}_{\mathcal{O}_i, \alpha}(\rho_i^*, x) = (t_i^{**}, \gamma)$ とする。このとき、次のような $\mathcal{O}[\mathcal{O}_i, \alpha]$ ($i =$

$1, \dots, n$) を構成する。

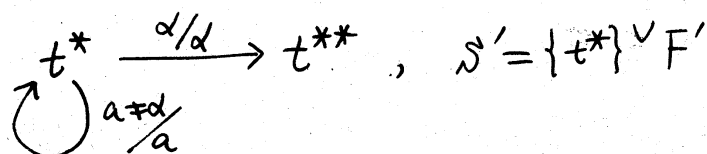
1) $q[\alpha_1]$ を t_1^{**} において $q[\alpha_1]$ 自身に接続したものを $q[\alpha'_1]$ とする。

2) $q[\alpha'_i]$ ($i < n$) までできたとして、 $\mathcal{A}_i^{**}$ において $q[\alpha'_i]$ に $q[\alpha_{i+1}]$ 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 $q[\alpha'_{i+1}]$ とする。

このようにして得られた $q[\alpha'_n]$ が $p_\alpha(x) = q\alpha_n(x)$ ($x \in A^*$) を満足することは明らかである。従って α_i の構成が与えられるれば証明は終了。 α_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 が終結的でない場合の構成のみを示す。以下 automaton の構成は diagram で示す。 $(\mathcal{A} \xrightarrow{a} t$ は $\lambda(\mathcal{A}, a) = t$, $\delta(\mathcal{A}, a) = t$ であることを示す。 $\mathcal{A} \xrightarrow{a} \mathcal{A}$ ($a \in \tilde{A}$) 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 $x \rightarrow y$ なるものとする。

最初に $\mathcal{A} = \mathcal{A}(\tilde{A}, \mathcal{S}', X, \delta', t^*, F')$, $F' = \{t^{***}, t^{**}\}$, $q_{\mathcal{A}}(t^*, z) = (t^{**}, zy)$ ($z \in A^*$) なる automaton $\mathcal{A}$ を構成する。

i) $\gamma = \mathcal{A}$ の場合



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

$$t^* \xrightarrow{\alpha/b_1} t_1, \quad t_j \xrightarrow{\alpha/b_{j+1}} t_{j+1}, \quad t_l \xrightarrow{\alpha/b_l} t^{**},$$

$$\begin{array}{c} \curvearrowright \frac{a \neq \alpha}{a} \end{array} \quad \begin{array}{c} \curvearrowright \frac{a \neq \alpha}{a} \end{array} \quad (1 \leq j < l) \quad \begin{array}{c} \curvearrowright \frac{a \neq \alpha}{a} \end{array}$$

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

次に望む $\mathcal{O}[\mathcal{O}]$ の構成であるが、 $x = \Lambda$ の場合は $\mathcal{O} = \mathcal{L}$

でよいから、 $x \neq \Lambda$, $x = a_1 \dots a_m$ ($a_j \in A$) とする。

$\mathcal{O}' = \mathcal{O}'(\tilde{A}, S'', \alpha'', \delta'', \rho^*, F'')$, $F'' = \{u^{**}, \rho^{**}\}$, $S'' = \{\rho^*, \rho_1^1, \dots, \rho_m^1, \rho_0^2, \dots, \rho_m^2\} \cup F''$ なる $\mathcal{O}'$ で state は全て $\mathcal{L}$ の state と異なる、次のような diagram をもつ automaton $\mathcal{O}'$ を構成する。

$$\begin{array}{c} \rho^* \xrightarrow{a_1/a_1} \rho_1^1, \quad \rho_j^1 \xrightarrow{a_j + a_{j+1}} \rho_{j+1}^1, \quad \rho_m^1 \xrightarrow{\alpha/\alpha} \rho_0^2, \\ \downarrow \frac{a \neq a_1}{a} \quad \downarrow \frac{a \neq a_{j+1}}{a} \quad (1 \leq j < m) \quad \curvearrowright \frac{a \neq \alpha}{a} \\ \rho^{**} \quad \rho^{**} \quad \rho^{**} \end{array}$$

$$\rho_j^2 \xrightarrow{\alpha/\alpha} \rho_{j+1}^2 \quad (0 \leq j < m), \quad \begin{array}{c} \curvearrowright \frac{a \neq \alpha}{a} \end{array} \rho_m^2 \xrightarrow{\alpha/\alpha} u^{**}$$

この $\mathcal{O}[\mathcal{O}']$ に $\mathcal{O}[\mathcal{L}]$ を u^{**} において接続すれば、 $\mathcal{O}[\mathcal{O}]$ が得られる。

§ 6. Turing algorithm

alphabet A , $S \subset \alpha \notin A$ なる α をとり、 $\tilde{A} = A \cup \{\alpha\}$ とおく

$a, b \in \tilde{A}$, $s, t \in S$ に対し、 $sabt$, $sart$, $salt$ の

いずれかの形式を *quadruple* , この α 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かく。) 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、 $X = \bar{A}^* \times S \times N$ ($N = \{1, 2, \dots\}$) 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 を表すものとする。

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

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

$$\sigma_3(x, s, n) = \begin{cases} (x, t, n-1) & 1 < n \leq l(x) \wedge s P_n(x) L t \in \Phi \\ (\alpha x, t, 1) & n = 1 \wedge s P_1(x) L t \in \Phi \\ \text{undefined} & \text{otherwise} \end{cases}$$

Φ の構造により各 σ_i の定義域は互いに disjoint である。

$\sigma \in p.f.(X, X)$ と $\sigma(\varepsilon) = \sigma_i(\varepsilon)$ ($\varepsilon \in \text{Dom}(\sigma_i)$) ,

$\text{Dom}(\sigma) = \bigcup_{i=1}^3 \text{Dom}(\sigma_i)$ で定義する。 $\Gamma \in p.f.(X^* - \{1_X\}, X)$ と $\Gamma(\varepsilon_1 \dots \varepsilon_n) = \sigma(\varepsilon_n)$ ($\varepsilon_i \in X$) で定め、 $\sigma =$

流 (X, Γ, X^*) をとる。この Γ を Turing machine T による Turing algorithm と称する。 $\varphi: \tilde{A}^* \rightarrow A^*$ を homomorphism φ とし、 $\varphi(\alpha) = \Lambda$, $\varphi(a) = a$ ($a \in A$) と定めよう。 φ をとり、 $(x_1, \dots, x_m) \in (A^*)^m$ に対し、 $f(x_1, \dots, x_m) = \varphi([\gamma], (x_1\alpha \dots x_m\alpha, \Lambda^*, 1))$ 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 Ψ はこれを制限して得られる (或はこの Ψ から導かれる) word function を Turing computable であるという。

定理 5 alphabet A 上の Asser algorithm $\mathcal{A}[\alpha, \alpha]$ に対し、 Turing machine T を構成して、 $g_{\alpha}(x) = \Psi_T^{(1)}(x)$ ($x \in A^*$) とおけるようにすることが出来る。特に Asser computable ならば Turing computable である。

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

$S \times (A^u \setminus \{\alpha, \beta\}) \rightarrow A^u \setminus \{\beta\}$ を字像 $\bar{\delta}$ のように定める。

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

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

Φ を次のような形の quadruple 全体の集合とする。

$s a \bar{\lambda}(s, a) \bar{\delta}(s, a)$ ($a \in A^u \setminus \{\beta\}$, $s \in S$), $\bar{\delta} a R s$ ($a \in A^u \setminus \{\beta\}$, $s \in S$), $\bar{\delta} a L s$ ($a \in A^u \setminus \{\beta\}$, $s \in S - F$), $\bar{\delta} \alpha R s$ ($s \in S - F$), $\bar{\delta} \beta \Delta \alpha$ ($s \in F$), $\Delta a L \bar{\delta}$ ($a \in \bar{A}$), $\bar{\delta} \beta \Delta \alpha$, $\bar{\delta} a a \Delta$ ($a \in A$)。 Turing machine $T = T(A^u \setminus \{\beta\}, S^u \bar{S}^u \bar{S}^u \setminus \{\alpha, \bar{\delta}\}, \Phi, s^*, \alpha)$ により、 $\bar{\Psi}_T^{(u)}(x) = f_\alpha(x)$ ($x \in A^*$) とする。これは容易に確かめることができる。

定理 6 T と与えられた Turing machine $T = T(A, S, \Phi, s^*, \alpha)$ に対し、Markov algorithm $M(\sigma)$ とする。 $\bar{\Psi}_T^{(u)}(x_1, \dots, x_m) = M_\sigma(x_1 \alpha \dots \alpha x_m)$ ($x_1, \dots, x_m \in A^*$) とする。特に word function σ Turing computable であれば Markov computable である。

[証明] Φ の quadruple $s a b t$ に対し、変換素子系列 $\{s a \rightarrow t b\}$ と対応させ、quadruple $s a R t$ に対し、変換素子系列 $\{s a b \rightarrow a s b \ (b \in \bar{A}), s a \rightarrow a s \alpha\}$ と対応させ、quadruple $s a L t$ に対し、変換素子系列 $\{b s a \rightarrow s b a \ (b \in \bar{A}), s a \rightarrow s \alpha a\}$ と対応させ、 Φ の quadruple の

左部分に現われない αa ($\alpha \in S$, $a \in \tilde{A}$) に対しては $\beta \notin S$ なる symbol β をとって、変換素子系列 $\{\alpha a \rightarrow \beta a\}$ と対応させる。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。

$\sigma: \beta a \rightarrow a\beta$ ($a \in A$), $\beta\alpha \rightarrow \beta$, $\beta \rightarrow \tilde{\beta}$, $a\tilde{\beta} \rightarrow \tilde{\beta}a$ ($a \in A$), $\alpha\tilde{\beta} \rightarrow \tilde{\beta}$, $\tilde{\beta} \rightarrow \cdot \Lambda$, Π , $\alpha^* \rightarrow \alpha^*\alpha$, $\Lambda \rightarrow \alpha^*$
 これにより、 $\Xi_T^{(m)}(x_1, \dots, x_m) = M_\sigma(x_1\alpha \dots \alpha x_m)$ ($x_1, \dots, x_m \in A^*$) 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おり、また次の §7 でも、recursive function を algorithm で表現する / > 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。

定理 7 recursive を半統, Markov, Post, Asser, Turing の各 algorithm のどの二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 のいふものの意味で computable であることは全く同等である。

§7. recursive な系統から algorithm への翻訳

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

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

$\sigma \in \Lambda \rightarrow \cdot a$ なる変換素子系列として、

$$+a(x) = p_{\sigma}(x) \quad (x \in (A')^*)$$

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

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

$$\phi^{(n)}(x_1, \dots, x_n) = p_{\sigma}(x_1 * \dots * x_n) \quad (x_i \in A^*)$$

(3) projection $\pi_m^{(n)}$ ($\pi_m^{(n)}(x_1, \dots, x_n) = x_m, 1 \leq m \leq n$).

$\beta \notin A'$ なる β とって、変換素子系列 σ を次のように与える。

$$\sigma: *^{n-1} \rightarrow \beta, \beta a \rightarrow a\beta \quad (a \in A), \beta * a \rightarrow \beta * \quad (a \in A'),$$

$$\beta * \rightarrow \Lambda, \beta \rightarrow \Lambda, a * \rightarrow * \quad (a \in A)$$

$$\text{すると、} \pi_m^{(n)}(x_1, \dots, x_n) = m_{\sigma}(x_1 * \dots * x_n) \quad (x_i \in A^*)$$

(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

i) $f^{(1)} \circ g^{(n)}$ 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 $\bar{a}$ は互いに異なるとして)、 $\bar{A}' = \{\bar{a}; a \in A'\}$ とおく。 $\beta \notin A' \cup \bar{A}'$ なる symbol β をとり、

$$\sigma_1: a \rightarrow \bar{a}a \quad (a \in A')$$

$$\sigma_2: \bar{a}a \rightarrow a\bar{a} \quad (a \in A'), \quad \bar{a} \rightarrow \cdot\beta a \quad (a \in A'), \quad \Lambda \rightarrow \cdot\beta$$

$$\sigma_3: \bar{a} \rightarrow a \quad (a \in A')$$

なる変換要素系列 $\sigma_1, \sigma_2, \sigma_3$ をとる。 $f = m_{\sigma_3} \circ m_{\sigma_2} \circ p_{\sigma_1}$ とおけば、 $f(x) = x\beta x \quad (x \in (A')^*)$

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

iv) $\alpha = \alpha(\tilde{A}', \delta, \delta', \lambda, \delta'', F)$ ($\tilde{A}' = A' \cup \{\alpha\}$) に対し、 $f = g_{\alpha, \alpha}$ とおく。すると、新しい automaton α' をつくり、 $g_{\alpha', \alpha}(x\beta y) = x\beta f(y) \quad (x, y \in (A')^*, \beta \text{ は } \beta \notin A' \text{ なる symbol})$ とおけるようにするこゝができてゐ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 $g_1, \dots, g_m$ が computable であるとき、 A' 上 n 変数の h_1 と A 上 n 変数の h で、

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

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

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

なる h_1, h とする algorithm が構成できることは、i), v) により明らかである。(h_1 は $g_1, \dots, g_m$ の combination, h は $f, g_1, \dots, g_m$ の composition という。)

(5) minimalization $\mu_y^a [f(y, x_1, \dots, x_n) = 1]$.

f は A 上 $n+1$ 変数の computable word function, $a \in A$ の特定の元とする。 $x = x_1 * \dots * x_n$ ($x_i \in A^*$) とする。必要に応じて x は $(x_1, \dots, x_n)$ を表すものと解釈する。 $\beta \in A'$ とする。 $a^m * x \rightarrow f(a^m, x) \beta a^m * x$ なる変換を Asper algorithm で実現する automaton を Ω_1 , その final state を β^{**} とする。(final state は 1 つと仮定してかまわない。) Ω_1 の states alphabet とは disjoint な states alphabet をもつ次のような automaton を Ω_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 にならぬ state Δ にかわり、 $\Delta \xrightarrow{b} \Delta$ で、 Δ は final state とはしないとする。 final states alphabet $= \{t_1^{**}, t_2^{**}\}$ として、任意の $b \in A$ に対し、

$$t^* \xrightarrow{b/a} t_1 \xrightarrow{\beta/a} t_2 \xrightarrow{\alpha/a} t_1^{**}, \quad t^* \xrightarrow{\beta/a} t_3 \xrightarrow{a/a}$$

このように α_2 により、 $z \in A^*$ に対し、 $\mathcal{F}_{\alpha_2, \alpha}(z \beta a^m * x, t^*)$ 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。 $\mathcal{F}[\alpha_1, \alpha]$ に $\mathcal{F}[\alpha_2, \alpha]$ を β^{**} 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。この結果得られる algorithm を $\mathcal{F}[\alpha, \alpha]$ とすれば、 $\mu_y^a[f(\gamma, x) = \Lambda] = \mathcal{F}_\alpha(*x)$ となる。 $x \rightarrow *x$ の変換は Markov algorithm $\mathcal{M}(\sigma)$; $\sigma: \Lambda \rightarrow \cdot *$ 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 f が computable であるとする。先のように、 $x = x_1 * \dots * x_n$ ($x_i \in A^*$) とし、これは必要に応じて $(x_1, \dots, x_n)$ とも解釈するものとする。 f と $h_a (a \in A)$ による primitive recursion を algorithm で実現するには、 $a_1, \dots, a_m \in A$ として $*x \rightarrow f(x)$, $a_1 \dots a_m * x \rightarrow a_1 \dots a_m * f(x) * x \rightarrow a_2 \dots a_m * \gamma_1 * x \rightarrow a_3 \dots a_m * \gamma_2 * x \rightarrow \dots \rightarrow * \gamma_m * x \rightarrow \gamma_m$ ($\gamma_1 = h_{a_1}(\lambda, f(x), x)$),

$g_{i+1} = h_{a_{i+1}}(a_1 \dots a_i, y_i, x)$) 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、 $z * x \rightarrow \beta z * g(x) * x$ なる変換を実現する algorithm

$$\begin{aligned} \text{ii) } \sigma_{\alpha, \alpha}(\beta^*, z_1, \beta z_2 * \gamma * x) & \quad (z_1, z_2, \gamma \in A^*) \\ &= \begin{cases} (\beta_1^{**}, z_1, \alpha \beta z_3 * h_a(z_1, \gamma, x) * x) & (z_2 = a z_3) \\ (\beta_2^{**}, \gamma) & (z_2 = \Lambda) \end{cases} \end{aligned}$$

なる automaton α

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 $\alpha_1, \alpha_a (a \in A), \alpha_2$ で、

$$\sigma_{\alpha_1}(\beta^*, z_1, \beta z_2 * \gamma * x) = \begin{cases} (t_a^{**}, z_1, \beta z_2 * \gamma * x) & (z_2 = a z_3) \\ (t^{**}, z_1, \beta z_2 * \gamma * x) & (z_2 = \Lambda) \end{cases}$$

$$\sigma_{\alpha_a}(t_a^*, z_1, \beta a z_3 * \gamma * x) = (\beta_1^{**}, z_1, \alpha \beta z_3 * h_a(z_1, \gamma, x) * x)$$

$$\sigma_{\alpha_2}(t^*, z_1, \beta z_2 * \gamma * x) = (\beta_2^{**}, \gamma)$$

なるものを構成して、 $\sigma[\alpha_1]$ に $\sigma[\alpha_a]$ 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 $\sigma[\alpha_2]$ を接続したものを $\sigma[\alpha]$ とすればよい。 α_1, α_2 の構成は簡単である、 α_a も、 $z_1 \beta a z_3 * \gamma * x \rightarrow z_1 \beta a z_3$, $z_1 \beta a z_3 * \gamma * x \rightarrow z_1 * \gamma * x$, $z_1 \beta a z_3 * \gamma * x \rightarrow x$ の各変換を実現する Asser algorithm が直ちに与えられることを考

之れが簡単に構成できる。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, Mc
Graw-Hill, Inc. (1958)
- A. A. Марков, Теория алгоритмов, Труд. Мат.
инс. им. В. А. Стеклова, 42 (1954)

