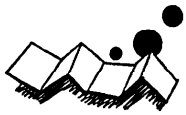
Vol. 28 No. 10　　情　報　処　理　　Oct. 1987

解　説

2. 方　式



2.3 帰納推論による自動プログラミング†

有川節夫††　石坂裕毅†††

1. はじめに

帰納推論 (inductive inference) とは，与えられた例から，それを説明する一般的な規則を推測する過程である．たとえば，次のような関数の入出力の例が与えられたとしよう．

$f(0)=1$, $f(1)=2$, $f(2)=3$.

これらの例から，

「$f(x)$ の値は x に1を加えたものである.」　(*)

という規則を導き出すような過程が帰納推論である．関数 f を計算可能関数 (recursive function) とし，上の規則の表現(*)を f を計算するプログラムとみるならば，この過程は例からプログラムを合成する過程であると考えられる．したがって，帰納推論を機械的に行うようなシステムを構築することは，直接，例によるプログラム自動合成 (programming by example) システムの開発につながる．

このように，帰納推論の機械化と例からのプログラム自動合成システムの開発は密接に関連している．実際，機械化を志向した帰納推論の研究は，順序機械 (sequential machine) の内部構造を，与えられた入出力の例から同定する (identify) という Moore[15] の研究に始まった．その後，帰納推論の理論的研究は計算可能性の理論を土台として発展し，Gold をはじめとする多くの研究者によって興味ある結果が得られている[2]．また，例によるプログラム自動合成に関しては，実際のプログラム（たとえば LISP のサブクラスなど）を対象に，具体的なシステムを開発するためのさまざまな手法が検討されてきた．本稿では，こうした研究の中から得られたいくつかの主要な結果について概観し，帰納推論のプログラム自動合成における可能性と問題点について考えてみる．

2. 帰納推論[3),4),22)]

ここでは，Gold にはじまる計算可能性の理論に基づいた帰納推論の理論について，特に重要と思われる概念や結果を中心にまとめておく．

先に挙げた順序機械の同定やプログラム自動合成においては，有限個の入出力例や，有限的に表現された仕様の記述を対象にして，推論（合成）は有限時間内に終了することが要求されている．しかし，一般に，プログラムの入出力の例としては無限個のものが考えられる．したがって，帰納推論の形式化においては，通常，**図-1** のようなモデルが考えられている．図において，各 e_i は推論対象の規則に対する例（たとえば，$f(0)=1$ といったような有限記号列）である．推論機械 IM は時点 i で例 e_i を受け取り，ある有限記号列 H_i（たとえば，(*)のような規則に対する有限な表現）を出力するようなチューリング機械である．この出力のことを推測と呼ぶ．すなわち，推論は

「入力要求→推測の計算→出力」

という過程を（無限に）繰り返すことによって行われることになる．

このように，帰納推論は一般に無限に続く過程として捉えられる．それでは，この枠組みのもとで，「推論機械が規則を帰納的に推論する」という概念はどのように定義されるのであろうか．また，そのような推論機械は具体的にどのような計算を行うのだろうか．次節で関数の帰納推論を例に挙げて，Gold[9] によって与えられた代表的な定義と推論手法，すなわち，極限における同定と枚挙手法について解説する．

2.1 関数の帰納推論[3),4)]

冒頭で述べたように，計算可能関数の帰納推論は，そのまま例によるプログラム自動合成とみなすことができる．以下では推論対象の規則として，自然数の集

† Program Synthesis by Inductive Inference by Setsuo ARIKAWA (Research Institute of Fundamental Information Science, Kyushu University) and Hiroki ISHIZAKA (International Institute for Advanced Study of Social Information Science, Fujitsu Ltd.).

†† 九州大学理学部基礎情報学研究施設
††† 富士通(株)国際情報社会科学研究所

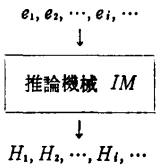


図-1

合 N から N への計算可能関数を考える.

f を N から N への計算可能関数とする. 関数 f の例 e_i としては, 任意の自然数 x_i と x_i における f の値の対 $(x_i, f(x_i))$ を考える. f の例の無限列で, 任意の自然数 x に対して, 対 $(x, f(x))$ が列のどこかに現れるものを, 関数 f の**枚挙** (enumeration) と呼ぶ. 推論機械 IM の出力 H_i としては関数を計算するプログラム(正確には関数の指数)を考える. すなわち, IM は自然数の対を入力として受け取り, ある関数の指数となっているような自然数を出力するものとする.

関数 f の枚挙が推論機械 IM に与えられたとき, IM が生成する出力の無限列がある自然数 i に収束し, i が表す関数 Φ_i が任意の自然数 x に対して $\Phi_i(x)=f(x)$ を満たすとき, IM は f を**極限において同定する**という. 計算可能関数の集合 S に対し, 推論機械 IM が S のすべての要素を極限において同定するとき, IM は S を**極限において同定する**という. 関数の集合 S が**推論可能**であるとは, S を極限において同定するような推論機械が存在することをいう.

たとえば, 原始帰納的関数の全体 S は上の定義のもとで推論可能である. S を同定する推論機械 IM としては次のようなものを考えればよい. IM は**図-2** にあるように, 大きく分けて三つの部分から成る. 仮説枚挙部 Es は, 推測の候補として S のすべての要素を $\Phi_{i_1}, \Phi_{i_2}, \cdots$ というようにもれなく生成する (すなわち, 枚挙する). また, 各時点で受け取ったすべての例は例記憶部 ME に記憶される. 仮説検証部 CH は新しい例 $(x_n, f(x_n))$ を受け取るたびに, Es によって

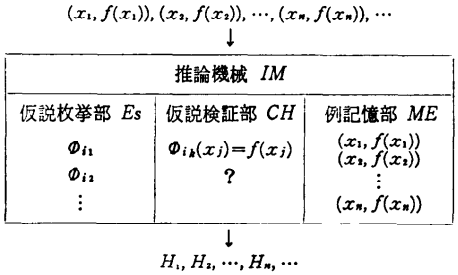


図-2

枚挙された現在の仮説 Φ_{i_k} に対し, $\Phi_{i_k}(x_1)=f(x_1)$, $\Phi_{i_k}(x_2)=f(x_2), \cdots, \Phi_{i_k}(x_n)=f(x_n)$ かどうかのチェックを行う. Φ_{i_k} がこのチェックに合格した場合には i_k を推測として出力し, 不合格の場合には Es に対して次の仮説 $\Phi_{i_{k+1}}$ を要求して同様の操作を繰り返す.

この推論方法は枚挙手法 (enumerative method) と呼ばれるものである. 実に単純な方法ではあるが, この枚挙手法により, 計算可能関数の枚挙可能な集合の部分集合はすべて推論可能であることが示される.

上で述べたような単純な枚挙手法に基づく推論機械の能力は, 必ずしも強力とはいえないので, 枚挙の仕方を工夫して能力の向上を図る場合もある. しかし, いずれにしても推論機械の本質が, この一見愚鈍とも思える枚挙にあることに変わりはない. 実用的な推論機械やプログラム自動合成システムを構築する際には, こうした枚挙の能率化が行われたり, 枚挙に変わる手法が用いられたりする. そのような推論方法については **4.** でふれることにする.

計算可能関数の枚挙可能な部分集合全体というものは, 推論可能な関数のクラスの一つの特徴付けである. このような推論可能な関数のクラスの特徴付けに関しては, この他にも計算の複雑さ (computational complexity) による特徴付けをはじめとして, 豊富な成果が得られている[8]. また, 東独の研究者による推論方式 (推論機械の推測作成方法に関する条件) に基づいた詳細な階層化も行われている[14].

2.2 言語の帰納推論[4),22)]

推論対象の規則として, 関数とともに盛んに研究が行われたものに形式言語がある. 言語の帰納推論が関数を対象とする場合と異なる点は, 推論機械に与えられる言語の例として, その言語における正しい文例 (正データ) と正しくない文例 (負データ) という二種類のタイプが考えられることにある.

Σ を有限アルファベットとし, L を Σ 上の形式言語とする. Σ 上のすべての語 w_i と, w_i が L に属しているかどうかを表す 0 または 1 ($t_1, t_2, \cdots$ で表わす) の対の無限列 $(w_1, t_1), (w_2, t_2), \cdots$ を L の完全提示と呼ぶ. 言語 L の正提示とは, L に属する語がすべて現れる無限列 $w_1, w_2, \cdots$ である. 推論機械の出力としては, 文法 (L の特徴関数の指数) を考える. 言語族 $\Gamma=\{L_1, L_2, \cdots\}$ が完全データ (正データ) から推論可能であるとは, ある推論機械 IM が存在して, Γ の任意の言語 L_i の任意の完全提示 (正提示) が与えられたとき, IM の生成する推測の無限列が,

L_i の文法に収束することをいう．

さて，推論可能性に影響を及ぼす要因として，例によって与えられる情報の量が考えられるが，次に挙げる結果[10]はそのことを如実に表している．

・Σ 上の文脈依存言語の族は完全データから推論可能である．

・Σ 上のすべての有限な言語と少なくとも一つの無限な言語を含むような言語族は正データから推論可能でない．

すなわち，正負両データを与える場合には文脈依存言語でも推論できるが，正データだけからでは正則言語さえも推論できないということである．例によるプログラム自動合成の場合には，例を与えるのは合成システムのユーザである．上の二つの結果は，そのユーザの能力と推論可能性の間に密接な関係があることを示唆している．実際，**4.** で紹介するような実用性を意識した推論機械の研究においては，例を与える物(者，通常オラクルと呼ばれる) に対してより優れた能力を仮定する場合もある．

また，例の表現を工夫することにより，例の中にある種の付加的情報を込めることも考えられる．たとえば，LISP プログラムの合成の場合には，与えられる例はS式 (S-expression) である．したがって，例の中にデータ構造に関する情報が込められることになる．そのような情報を活用する推論方法については次章で紹介する．

言語の帰納推論に関しては，この他に，Angluin[1] によって正データだけから推論可能な言語族の特徴付けが与えられ，同時に興味ある言語族の具体例が示されている．また Shinohara[19),20)] は実用的推論機械の実現という観点から，推測と次の推測との間隔をそれまでに受け取った例の多項式時間内におさえるという多項式時間推論について考察を行い，多項式時間推論可能ないくつかの言語族の具体例を与えている．

3. 例による LISP プログラムの自動合成[3),5)]

前章で紹介したような帰納推論の計算理論的研究では，推論可能な規則のクラスを解明することに重きがおかれる．一方，例によるプログラム自動合成の研究では，実際のシステム作り，すなわち，推論機械 *IM* を具体的にどう実現するかに重点がおかれる．ここでは，代表的と思われる研究を概観することにする．

まず，Hardy[11] は1個の入出力例に対して，プログラマが何を自然に考えるかということから，GAP (Generalized Automatic Programmer) というシステムを試作した．

複数個の例を与える場合は例の間の関係を解析することによって，いわばデータ駆動型で制御構造を推論することが自然に考えられる．この方針に基づいて，Summers[23),24)] は THESYS というシステムを作成した．

THESYS では LISP を対象にして，基本演算を cons, car, cdr, atom に限定している．つまり構造を構造に変換する LISP 関数だけを対象にする．関数の例としては入力S式 α_i と出力S式 β_i の対 $\alpha_i \rightarrow \beta_i$ を考える．また，例の間の構造解析を簡潔に行うために，与えられる例の有限集合

$$E=\{\alpha_i \rightarrow \beta_i ;\ 1 \leqq i \leqq n\}$$

に対し，入力S式からなる集合

$$E'=\{\alpha_i ;\ 1 \leqq i \leqq n\}$$

は部分S式の順序づけくに関して全順序になることを仮定する．このような例の集合が与えられたとき，THESYS は，まず，$p_i[\alpha_i]=\mathrm{T}$ となる述語 $p_i[x]$ と $f_i[\alpha_i]=\beta_i$ なる関数 $f_i[x]$ を各 $i(1 \leqq i \leqq n)$ に対してそれぞれ作成する．

たとえば，E として次のようなものを考える．

$$E=\{(\ ) \rightarrow (\ ),\ (A) \rightarrow ((A)),\ (B, A) \rightarrow ((B)(A))\}$$

このとき，各 $p_i[x]$, $f_i[x]$ は次のようになる．

$p_1[x]=\mathrm{atom}[x]$,
$p_2[x]=\mathrm{atom}[\mathrm{cdr}[x]]$,
$p_3[x]=\mathrm{atom}[\mathrm{cdr}[\mathrm{cdr}[x]]]$
$f_1[x]=\mathrm{nil}$,
$f_2[x]=\mathrm{cons}[x ;\ \mathrm{nil}]$
$f_3[x]=\mathrm{cons}[\mathrm{cons}[\mathrm{car}[x] ;\ \mathrm{nil}] ;\ \mathrm{cons}[\mathrm{cdr}[x] ;\ \mathrm{nil}]]$

この $p_i[x]$ と $f_i[x]$ を用いると，与えられた例の集合 E を忠実に実行するプログラム

$$f[x]=[p_1[x] \rightarrow f_1[x] ;\ p_2[x] \rightarrow f_2[x] ;\ p_3[x] \rightarrow f_3[x]]$$

を得ることができる．

THESYS は，次に再帰関係を検出しようとする．上の例の場合には，

$p_3[x]=p_2[\mathrm{cdr}[x]]$,
$f_3[x]=\mathrm{cons}[\mathrm{cons}[\mathrm{car}[x] ;\ \mathrm{nil}] ;\ f_2[\mathrm{cdr}[x]]]$

なる再帰関係が成立する．THESYS では $p_i[x]$, $f_i[x]$ $(k \leqq i \leqq n)$ に対して，再帰関係

$f_{i+1}[x]=F[x, f_i[x]]$,
$p_{i+1}[x]=P[x, p_i[x]]$

が成立する場合，その関係が n より大きな i に対しても成り立つと推論する．この原理に従って，再帰関係を関数 $u[x]$ で実現した

$f[x]$=[atom[x]→nil ; t→$u[x]$]
$u[x]$=[atom[cdr[x]]→cons[x ; nil] ;
t→cons[cons[car[x]] ; nil] ; u[cdr[x]]]

なる LISP プログラムを得るのである．

THESYS は，このように例の解釈法，再帰関係の発見法，推論の方式などがきわめて単純であるために，能力的に制限されたものになっている．

一方，Biermann[7] は Regular Lisp と呼ばれる LISP のサブクラスが，例からトレースを作り，そのトレースをマージすることによって再帰的なプログラムを作り出すという方法で，極限において同定できることを示している．この方法では，最小サイズのプログラムを求める計算量が指数関数のオーダになるという難点をもっている．

また，Bauer[6] は Biermann のマージの手法にヒントを得て，CDL (Computation Description Language) の言語設計とこの言語で書かれたトレースから，手続き的プログラムを合成するシステムを考えている．これは前の例に比べて実用性の高いものである．CDL では，トレースを記述するために，通常の手続きと同様に，手続き名や仮引数，局所変数，定数，配列，レコード，副手続き名なども許されている．ただし，利用者の意図を明確にするために，いくつかの制約条件が設けられている．合成はまったく字面だけで行われ，似たような種類の命令がまず分類され同値類を形成する．次に，先の制約条件に矛盾しないかどうかのチェックを同値類に対して行う．最後に，各同値類を一つの命令にマージしてループを含むプログラムを作り出す．ある命令が別の命令と似ているか否かのチェックはまったく字面だけで行われるので，トレース中に未定義の手続きを記入することができるわけである．このことによって，例による階層的なプログラムの自動合成が可能になる．

4. モデル推論：例による論理プログラムの自動合成

先に挙げたようなしらみつぶし的な枚挙手法では，無駄な仮説の生成と検証に莫大な時間を費やすことになる．そこで，実用的な推論機械を構築するためには，できるだけ無駄な仮説を生成しないような推論方法が望まれる．ここではそのような推論方法の一つとして，Shapiro[17),18)] の MIS (Model Inference System) で用いられた漸増的枚挙手法を紹介する．

4.1 漸増的枚挙手法

モデル推論では推論対象の規則として，ある固定された一階言語 L 上の Herbrand モデル（すなわち，L 上のグランドアトムの集合）M を考える．推論機械 IM に与えられる例は，L 上のグランドアトム α と，α の M における真理値との対 $\langle \alpha, V \rangle$（ただし，$\alpha \in M$ のとき $V=True$，それ以外のとき $V=False$）であり，M に関する事実（fact）と呼ばれる．$\langle \alpha, True \rangle$ なるグランドアトム α のことを正事実と呼び，$\langle \alpha, False \rangle$ なる α のことを負事実と呼ぶ．M に関する事実の無限列で，L 上のすべてのグランドアトムが列のどこかに出現するようなものをモデル M の枚挙と呼ぶ．推測は L 上の論理プログラムである．論理プログラム P の最小 Herbrand モデル（すなわち，P 上で成功するすべてのグランドアトムの集合）を $M(P)$ で表す．推論機械 IM にモデル M の枚挙が与えられたとき，IM が生成する推測の無限列が $M(P)=M$ なるプログラム P に収束するとき，IM は M を極限において同定するという．

MIS は基本的には枚挙手法により推論を行う．論理プログラムは単一の構造をもつプログラム節の有限集合であるから，プログラム節を枚挙し，それらの組合せを考えることにより，L 上のすべての論理プログラムを枚挙することができる．しかし，単にすべての組合せを考え，既知の事実の上でそれらすべてを検証していたのでは，単純な枚挙手法と同じである．MIS では論理プログラムの特性をうまく利用することによって，よけいな組合せを排除し，効率的なプログラムの枚挙を実現している．

図-3 に MIS の推論方法の概略を示す．節枚挙部 CE は L 上のすべてのプログラム節を $C_1, C_2, \cdots$ とい

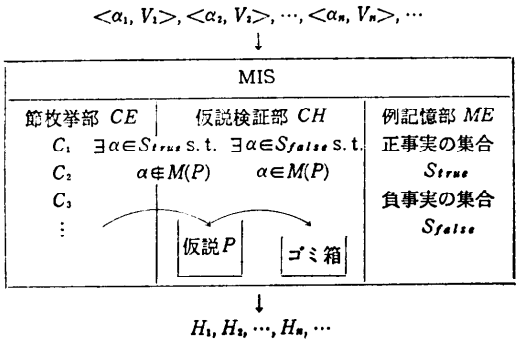


図-3

うようにある固定された順序で枚挙する．仮説検証部は現在の仮説 P が既知の事実と矛盾しないかどうかのチェックを行う．矛盾しない場合には仮説 P を推測として出力するが，矛盾する場合，すなわち，次のいずれかの場合には，それぞれに応じた仮説の修正がなされる．

（1） 仮説 P がモデル M に対して弱すぎる，すなわち，ある正事実 $\alpha(\in M)$ に対し，$\alpha \notin M(P)$.

（2） 仮説 P がモデル M に対して強すぎる，すなわち，ある負事実 $\alpha(\notin M)$ に対し，$\alpha \in M(P)$.

(1)の場合には，CE に対して次のプログラム節 C を要求し，それを仮説 P に付加する．論理プログラムの単調性により，$M(P) \subseteq M(P \cup \{C\})$ が成り立つから，この修正は $M(P)$ を増加させることを意味する．逆に(2)の場合には，仮説 P 中のあるプログラム節を取り去ることにより $M(P)$ を減少させる．この際，どのプログラム節を除去するかが問題となるが，そのことについては後でふれることにして，まず，効率の面から単純な枚挙手法との比較をしてみる．

4.2 漸増的枚挙手法の効果

M を推論対象のモデルとし，P を $M(P)=M$ なる論理プログラムとする．単純な枚挙手法では，プログラム節の枚挙 $C_1, C_2, \cdots$ に対し，$\phi, \{C_1\}, \{C_2\}, \{C_1, C_2\}, \{C_3\}, \{C_1, C_3\}, \{C_2, C_3\}, \{C_1, C_2, C_3\}, \{C_4\}, \cdots$ というように，枚挙されたプログラム節のすべての組合せをとることにより，すべての論理プログラムを枚挙する．したがって，P を構成するプログラム節の中で添字が最大のものを C_n とすると，P が上の枚挙に現れるのは最悪の場合 2^n 番目ということになる．すなわち，推論機械は正しい推測を出力するまでに，2^n 個の仮説を生成し，それらすべてを既知の事実の上で検証しなければならない．

一方，図-3 にあるような漸増的枚挙手法では，P が仮説として現れるためには，C_1 から C_n までの n 個のプログラム節が仮説の中に取り込まれ，その中のいくつかが除去されればよい．$P \subseteq P'$ であるような仮説 P' に関しては，$M(P) \subseteq M(P')$ が成り立つから，P' が M に対して弱すぎることはなく，新たなプログラム節 C_{n+1} が P' に取り込まれることはない．したがって，プログラム節の削除が正しく行われるならば，P が仮説として出現するまでに高々 $2n$ 個の仮説しか生成されない．すなわち，漸増的枚挙手法は単純な枚挙手法に比べて指数オーダの高速化が期待できるのである．

4.3 矛盾点追跡アルゴリズム

上で述べたような高速化のためには，仮説 P が強すぎることが判明した場合に，P から取り除くべきプログラム節を決定するための手続きが必要となる．この手続きは矛盾点追跡アルゴリズム (contradiction backtracing algorithm) と呼ばれる．以下でこのアルゴリズムについて簡単にふれておく．

まず，削除すべきプログラム節の特徴を明らかにしておこう．C をプログラム節とする．C がモデル M において偽であることは，C のグランドインスタンス $\alpha \leftarrow \beta_1, \beta_2, \cdots, \beta_m$ で $\alpha \notin M$ かつ $\beta_i \in M$ $(1 \leqq i \leqq m, m=0$ のときは $\alpha \notin M)$ となるものが存在することをいう．モデル M に対して強すぎる仮説 P は M において偽であるようなプログラム節を少なくとも一つ含むことが保証される[18]．一方，$M(P)=M$ なるすべての論理プログラムは M において偽であるプログラム節を含むことはない．すなわち，削除すべきプログラム節とは，推論対象のモデルにおいて偽であるようなものである．矛盾点追跡アルゴリズムは，強すぎる仮説からそのようなプログラム節を探し出す．

たとえば，ある負事実 $\alpha(\notin M)$ が仮説 P 上で成功したとする．その計算木 (refutation tree) を図-4 の右側に，そこで用いられた P 中の節のグランドインスタンスを左側にあるようなものとする．まず，アルゴリズムは α の子 β, γ に対して，それらが M の要素かどうかのチェックを行う．β, γ がともに M の要素であるならば，この導出に用いられた節 $\alpha \leftarrow \beta, \gamma$ に対応する P 中のプログラム節が偽である．γ が M の要素でないならば $\gamma \leftarrow$ に対応する節が偽である．β が M の要素でないならば，α の場合と同様の操作を行う．成功する計算に対応する計算木は有限であり，葉の部分には単一節が対応しているから，この操作を繰り返すことによって必ず M において偽である節を探し出すことができる．

MIS ではこのほかに精密化演算子 (refinement operator) によるプログラム節の枚挙という手法が用いられる．これは，矛盾点追跡アルゴリズムによって反駁されたプログラム節の情報を，それ以後のプログラム節の枚挙にフィードバックすることにより，無駄

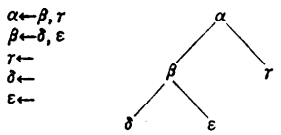


図-4

な枚挙を避けるためのものである.

4.4 モデル推論の問題点

以上述べたように, MIS は論理プログラムの特性をうまく利用することによって, 効率的にモデルの推論を行う. 実際, 先に述べた Summers や Biermann の方法に比べて, 格段に高速化された実験例が報告されている. しかし, 実際のプログラム自動合成システムとしてみた場合には, いくつかの問題点があることも指摘されており, それらを改善する試みもなされている[12),13)].

たとえば, 文献18) の中でインプリメントされている MIS は, 次のようなプログラムを推論することができない.

```
reverse([X|Xs], Y) ← reverse(Xs, Z), append(Z,
                                [X], Y).
reverse([ ], [ ]).
```

これは MIS の節枚挙部 CE が本体中のアトムの引数に複合項が出現するような節を枚挙できないためである. このような節を枚挙することは理論的には可能である[17)]が, 既存の計算機で実現するには, 時間及び領域計算量の大幅な増加を引き起こすために実用的でなくなる.

また, MIS の効率的な枚挙を支えているのは, 図-3 にある右側の矢印の操作, すなわち, ある負事実 α が仮説 P 上で成功した場合に, P 中の偽なプログラム節を発見し, P から削除するという操作である. 偽なプログラム節の発見は, 負事実 α の P 上での計算木をもとに行われる. したがって, 推論の効率化に大きく貢献しているのは, 正事実よりむしろ負事実によって与えられる情報であるといえる. しかし, Summers の合成手法にみられるように, 通常のプログラム自動合成においては, 正事実から得られる情報が合成に用いられる. したがって, 正事実のもつ情報をうまく推論に反映できれば, さらに効率的な推論システムの実現が期待できる.

最も厄介な問題は, モデル推論の枠組みでは, あらかじめプログラムを記述するための一階言語が与えられるということである. たとえば, 次のような挿入法による整列化プログラム

```
isort([X|Xs], Y)←isort(Xs, Z), insert(X, Z, Y).
isort([ ], [ ]).
```

を MIS で推論するためには, 述語 isort(__, __) と insert(__, __, __) がシステムに与えられなければならない. そのような情報を与えるのはユーザの作業であり, ユーザが目標のプログラムに関するかなりの知識を有することを仮定することになる. しかし, プログラム自動合成システムに要求されることは, ユーザが述語 isort(__, __) に関する情報だけを与えることにより, 適当な整列化プログラムを合成できることである. このモデル推論とプログラム自動合成の間のギャップが存在するかぎり, MIS を実用的なプログラム自動合成システムとはいいがたい.

5. 実用化の例

ここまでにみてきたように, 帰納推論ならびにプログラム自動合成に関する多くの研究が成され, 具体的推論手法が考案されている. 残念ながら, 現在のところ, 任意のプログラムを効率的に合成するような手法が得られるまでには至っていない. しかし, ある限定されたプログラムのクラスに関しては, 効率的な手法も考案されており, 実際のシステムとして実現されている. ここでは, そのような実用化の例として, 例による編集 (editing by example) とデータエントリ機能について紹介する.

5.1 例による編集

限定されたタイプのプログラム合成の応用として, 例による編集 (editing by example, 以後 EBE と略記する) がある[16)]. これは, プログラム自動合成の問題を人間と計算機の最も基本的な接点であるテキスト・エディタという形で実現したものとして注目される.

エディタの利用者は, 手紙や論文の作成といった比較的高度な作業だけでなく, ときには反復的で退屈な作業も余儀なくされる. こうした, 反復的で定形的な作業の能率化は, そのためのプログラムやマクロコマンドを用意して行うのが普通である. これに対して Nix が開発した EBE は, このようなテキスト処理プログラムを, 利用者が入力した例をもとにして自動合成するシステムであり, テキスト・エディタに内蔵されたエディタの一機能として働くものである.

このような自動合成システムでもって反復的な作業のすべてを対象とすることはもとより不可能であるので, EBE では次のようなある書式のテキストを他の書式に書き換える作業の処理を対象としている.

```
支店名ファイル
北海道支店  〒060  札幌市…(011)271-4311
旭川支店    〒070  旭川市…(0166)24-0337
釧路支店    〒085  釧路市…(0154)25-4200
                ⋮
```

を，データベースの入力形式

支店一覧［支店名：北海道支店；郵便番号：060；所在地：札幌市…；電話番号：011-271-4311］
支店一覧［支店名：旭川支店；郵便番号：070；所在地：旭川市…；電話番号：0166-24-0337］
支店一覧［支店名：釧路支店；郵便番号：085；所在地：釧路市…；電話番号：0154-25-4200］
⋮

に書き換える．

EBE が合成するプログラムは gap プログラムと呼ばれるもので，例として与えられた入出力テキストの対から，入力テキストを解析し，その解析に従って出力テキストを編集する機能をもつものである．上の例に対する gap プログラムは次のようなものである．

bol～1～□〒～2～□～3～□(～4～)～5～--～6～eol⇒bol 支店一覧［支店名：～1～；郵便番号：～2～；所在地：～3～；電話番号：～4～-～5～-～6～］eol.

ここに，bol, eol はそれぞれ行の始まりと終わりを表す記号で，□は1個の空白を表すものである．～1～,～2～ などは gap と呼ばれるもので，入出力テキスト対における定数部分の対応を表すものである．また⇒の左側が入力に，右側が出力に対応している．この gap プログラムは⇒の左側とマッチする部分を見つけて，その部分を⇒の右側の指示に従って書き換える働きをする．

EBE の gap プログラムの合成アルゴリズムは，利用者によって与えられる入出力テキスト対から極限において，上記のような gap プログラムを推論するわけである．少し詳しくいうと，これは以下のような性質をもつものである．

（1） 例が追加されたとき常にプログラムを合成するわけではない．

（2） 合成されたプログラムはそれまでの例もうまく処理する．

（3） 例を追加しても，矛盾の生じないかぎりそれまでのプログラムを変更しない．

（4） ある例が追加されたとき，それまでの例の長さと例の個数に関する多項式時間内に応答する．

これらの性質のうち，(1)～(3)は極限における推論の方式では，よく知られているものであるが，エディタなどにおいては応答時間の短さを保証した(4)の性質も重要である．このような推論アルゴリズムはいくつかの段階で NP 完全な問題に直面する．Nix はそのことを示したうえで，近似的な高速アルゴリズムを与えている．これは EBE の一つの特徴であるが，さらに実際に動いているシステムであるという点にも意義がある．

5.2 データ・エントリ機能

帰納推論の一分野である文法推論の興味深い問題としてパターン言語の推論[1]がある．パターン言語の族は，**2.2** でふれた正データから推論可能な言語の族であることが知られている．これを実際的な場面に応用するためには，上述 **5.1** の場合と同様に，応答時間が短いことが要求される．そこで Shinohara[19),20)]は，一つの妥当な目安として受け入れられている多項式時間応答を保証できるパターン言語族（拡張正則パターン言語族）を発見し，それを単純ではあるが，学習機能をもつデータエントリ・システムに応用している[21)]．

たとえば，文献データベースを作成するために，利用者が次のような形でタイプしているとする．

```
$
Author : Gold, E. M.
Title : Language Identification in the Limit
Journal : Inform. and Contr. 10
Year : 1967
$
Author : Blum, L. and Blum, M.
Title : Toward a Mathematical Theory of
        Inductive Inference
Journal : Inform. and Contr. 28
Year : 1975
$
⋮
```

いくつかのレコードが入力された後で，システムはレコードの構造を

Author : w Title : x Year : z

の形であると推論して，以後プロンプトとして，たとえば Author : を出し，利用者がフィールド w を入力するのを待つようになる．

6. おわりに

帰納推論とそれによるプログラムの自動合成について，代表的と思われる理論やシステムを中心にして簡

単に解説した．この方面の研究に関する全般的な印象として，Gold に始まる極限における同定という枠組みで帰納的関数や形式言語を扱った理論的成果は，すでに十分得られているように考えられる．しかし，論理プログラムを対象としたモデル推論に関しては，多くの利点や応用の可能性を秘めているものの，いまだ十分な展開をみていないのが現状である．

こうした帰納推論の応用としてのプログラムの自動合成については，これまでに報告されているシステムでは実用性が期待できそうにない．LISP プログラムの合成，あるいは例による編集やデータエントリなどの具体的なシステムでは本質的に扱えるプログラムのクラスが小さいし，汎用性のある MIS の応用には時間と領域量の面から問題点がある．このように理論とその応用との間に距離があり過ぎるのはなぜかという視点に立って，もう一度帰納推論の理論の枠組みを考え直してみる必要があろう．人間が自然に行っている帰納推論との間に違いがあり過ぎるのではないだろうか．たとえば，人間が例によってプログラムを合成できるのは，目標のプログラムを合成するのに十分な知識をすでに備えているからではないだろうか．そうした知識を自然な形で活用できるような枠組みを工夫することが必要であろう．また，自動合成の対象としてのプログラムのクラスを実際問題に即して，適当に狭めることも必要であろう．

このように，帰納推論の応用としての自動プログラミングは，興味深い重要な，しかも未解決の課題を多く残した研究分野である．なお，本稿では紙面の関係で十分な解説ができなかったところが多い．興味ある読者は参考文献を参照していただきたい．

参 考 文 献

1) Angluin, D.: Inductive Inference of Formal Languages from Positive Data, Inf. Control, Vol. 45, pp. 117-135 (1980).
2) Angluin, D. and Smith, C. H.: Inductive Inference: Theory and Methods, Computing Surveys, Vol. 15, No. 3, pp. 237-269 (1983). (大谷木重夫訳：帰納推論—理論と方法，bit 別冊，acm computing surveys '83, コンピュータ・サイエンス，共立出版，pp. 107-135 (1985).)
3) 有川節夫：帰納推論と類推—理論と応用，古川他編，知識の学習メカニズム，共立出版，pp. 23-51 (1986).
4) 有川節夫，篠原 武，宮原哲浩：帰納推論の理論，大須賀，佐伯編，知識の獲得と学習，オーム社，pp. 147-198 (1987).
5) 有川節夫，原口 誠：例によるプログラムの合成，大須賀，佐伯編，知識の獲得と学習，オーム社，pp. 197-219 (1987).
6) Bauer, M. A.: Programming by Examples, Artificial Intelligence, Vol. 12, pp. 1-12 (1979).
7) Biermann, A. W.: Regular LISP Programs and Their Automatic Synthesis from Examples, Duke Univ., CS-1976-12 (1976).
8) Blum, L. and Blum, M.: Toward a Mathematical Theory of Inductive Inference, Inf. Control, Vol. 28, pp. 125-155 (1975).
9) Gold, E. M.: Limiting Recursion, J. Symbolic Logic, Vol. 30, pp. 28-48 (1965).
10) Gold, E. M.: Language Identification in the Limit, Inf. Control, Vol. 10, pp. 447-474 (1967).
11) Hardy, S.: Synthesis of LISP Functions from Examples, Proc. of 4th IJCAI, pp. 240-245 (1975).
12) Ishizaka, H.: Model Inference Incorporating Generalization, Proc. SSE symp., Kyoto Univ. (1986).
13) 石坂裕毅：モデル推論による正則言語の帰納推論，ロジックプログラミング・コンファレンス '87 論文集 (1987).
14) Jantke, K. P. and Beick, H. R.: Combining Postulates of Naturalness in Inductive Inference, Electron. Informationsverarb. Kybern., Vol. 17, No. 8-9, pp. 465-484 (1981).
15) Moore, E. F.: Gedanken-Experiments on Sequential Machines, Automata Studies, Princeton U. P. (1956).
16) Nix, R. P.: Editing by Example, Res. Rep. 280, Yale Univ. Dept. Comp. Sci. (1983).
17) Shapiro, E. Y.: Inductive Inference of Theories From Facts, Res. Rep. 192, Yale Univ. Dept. Comp. Sci. (1981). (有川節夫訳：知識の帰納推論，共立出版 (1986).)
18) Shapiro, E. Y.: Algorithmic Program Debugging, The MIT Press (1983).
19) Shinohara, T.: Polynomial Time Inference of Pattern Languages and Its Application, Proc. 7th IBM Symp. Math. Found. of Comp. Sci., pp. 191-209 (1982).
20) Shinohara, T.: Polynomial Time Inference of Extended Regular Pattern Languages, Proc. RIMS Symp. Software Sci. & Eng. (1982), LNCS-147, Springer-Verlag, pp. 115-127 (1983).
21) Shinohara, T. and Arikawa, S.: Learning Data Entry System: An Application of Inductive Inference of Pattern Languages, Res. Rep. 102, Research Institute of Fundamental Information Science, Kyushu Univ. (1983).
22) 篠原 武：言語の帰納推論，古川他編，知識の学習メカニズム，共立出版，pp. 53-70 (1986).
23) Summers, P. D.: Program Construction from Examples, IBM Res. PC-5637 (1975).
24) Summers, P. D.: A Methodology for LISP Program Construction from Examples, JACM, Vol. 24, pp. 161-175 (1977).

(昭和 62 年 6 月 1 日受付)