特　集　「知的プログラミングの基礎」

帰納推論と類推によるプログラムの合成

Program Synthesis by Inductive Inference and Analogical Reasoning

有川　節夫*
Setsuo Arikawa

*　九州大学理学部基礎情報学研究施設
Research Institute of Fundamental Information Science, Kyushu Univ.

1987年5月11日　受理

Keywords : inductive inference, analogical reasoning, program synthesis, programming by examples.

1. はじめに

ソフトウェアの生産性向上のために，さまざまな知的手法が試みられている．その中に，例によるプログラムの自動合成の問題がある．これは，入力データと出力データの対を例として，次々に与え，意図したプログラムを自動的に合成するもので，広くとらえると帰納推論の問題である．こうした例によるプログラミングは，それを現実的なシステムとして機械化するには，解決すべき多くの課題が残されている．しかし，人間によるプログラム開発においては，極く自然に使われている手法であるために，比較的古くからいくつかの試みがなされてきた．最近，論理プログラムを対象にして，理論としては見通しの良い研究成果も得られている．

また，知的なプログラミングのお手本が，人間による知的なプログラミングにあるとすれば，新しい手法として類推の活用が考えられる．類推は，プログラム開発だけでなく，人間の知的な問題解決において，その初期の段階から活用される重要な推論である．この類推について，最近，論理プログラムを対象にした理論的成果が得られたので，それをプログラムの開発，特に既存のプログラムの再利用による開発の問題に応用することを考える．この研究は，まだ着手されたばかりであるが，知的プログラミングの基礎として重要な意味をもつと考えられる．

本稿では，上記のような視点に立って，まず帰納推論について説明し，その応用としての例によるプログラムの自動合成に関する代表的な研究の概要を紹介する．次に類推について説明し，論理プログラムを対象にした類推の機能を，帰納推論と併用する形で，プログラムの再利用による開発に適用することを考える．

2. 帰納推論[(2)-(4)]

帰納推論 (inductive inference) とは，与えられたデータから，それを説明する一般的な規則を導き出す推論である．こうした帰納推論を正確に定義するには，次の5つの項目を規定する必要がある．

① 推論の対象になる規則の集合
② 規則の表現法
③ 例の提示（例示）の仕方
④ 推論の方法
⑤ 推論を正しいとみる基準

推論の対象となる規則としては，主として，計算可能関数や形式言語，プログラムなどが考えられてきた．また，規則の表現としては，関数を計算するTuring機械や言語を規定する文法，Prologなどが考えられてきた．Turing機械は，計算機の数学的モデルであり，それはプログラムと見なすことができる．形式言語を対象とする場合には，規則として言語，その表現として文法を扱うので，通常文法推論と呼ばれている．

関数fに対する例示には，入出力の対$(x, f(x))$の例や，fを計算するTuring機械の動作系列などが考えられる．推論機械に関数fの入出力の対$(x, f(x))$を与えることは，プログラムの自動合成システムに，入力xとfを計算するプログラムの入力xに対する出力$f(x)$とを与えることに対応する．この意味で，例によるプログラムの自動合成は関数の帰納推論と見なすことができる．形式言語は語の集合であるから，言語Lの例には，Lに属す語と属さない語の2種類が考え

られる．前者を正データ，後者を負データと呼ぶ．つまり，形式文法の例示には正データ，負データによる例示の2通りがある．対象がプログラムの場合には，Lispについては，関数の場合と同様に考えられるが，Prologについては，事実(fact)の形で例示する．たとえば，<3≦4, true >, <2≦1, false >といった形である．このうち，trueと付値されたデータが言語でいう正データに，falseと付値されたデータが負データに対応する．

帰納推論は，

入力要求→仮説の生成・検証→出力

という基本過程を無限に繰り返すことによって行うものとする．つまり帰納推論は，次々に例を入力として受け取り，その時点での仮説を生成し検証して，その結果を出力する．各時点における仮説は，それまでに受け取った例の有限列に基づいていると考えられるので，帰納推論の方法としては，こうした例の有限列から仮説を計算する，すなわち，生成し検証するTuring機械を用いるのが普通である．このTuring機械を推論機械(inference machine)と呼ぶ．

帰納推論の結果を正しいと見なす基準としては，帰納推論が無限に続く過程であるということに立脚した，極限における同定(identification in the limit)という概念が一般的である．これはGold[11][12]によるもので，それ以後のほとんどの帰納推論の理論で採用されている基準である．推論機械Mが規則Rを極限において同定するとは，Rの例示が与えられたときに，Mの生成する出力の例がある表現τに収束し，すなわち，ある時点以降のMの出力がすべてτであり，しかもτがRの正しい表現になっていることをいう．また，規則の集合Γは，ある推論機械Mが存在して，そのMがΓの任意の規則Rを極限において同定するとき，帰納推論可能であるという．

3．例によるプログラムの合成

帰納推論の理論をそのままの形で，例からのプログラムの合成に応用すると，非能率的であることが多い．そのため，具体的応用に関しては固有のアプローチがとられてきた．

3・1 LISPプログラムの自動合成[3]-[5]

例によるプログラムの自動合成(programming by examples)とは，入出力対や計算過程(トレース)の例から，プログラムを作り出す手続きのことをいう．この手続きに入力として与えられる例は，例仕様(example specification)ともいわれる．一般には，例は目的のプログラムの部分的情報しかもたないので，合成手続きが出力するプログラムは，目的のプログラムになるとは限らない．しかし，このような例の提示を限りなく続けると帰納推論を行っていることになる．

LISPプログラムの自動合成問題における例は，S式というデータ構造をもっている．本節では，例のこのような構造を活用したLISPプログラムの自動合成について，簡単な例を使って解説する．

S式の入出力の対$<X, Y>$を

$X \Rightarrow Y$

で表す．1つのS式の対，

(A B C D) ⇨ ((A) (B) (C) (D))

と，さらに出力が未知なS式の対，

(A B C D E) ⇨ ?

を与えて"?"を推論する問題が考えられる．S. Hardy[18]はこのような推論を行う計算機モデルGAP (Generalized Automatic Programmer)を試作している．帰納推論の場合と同様にGAPも，推論の前提である例の集合から仮説を生成する手続きと，仮説から論理的にその前提を導けるかどうかを検証する手続きの両者をもっている．

まず，与えられたS式の対から，S式やLISPプログラムに関する経験的な知識を使って仮説プログラムを生成する．次に，この仮説プログラムを演繹的なLISPシステムで検証することによって，仮説プログラムを完全なLISPのプログラムに変形する．先の例の場合には，結局，

```
(LABEL (SELF (LAMBDA (X)
          (COND ((ATOM X) NIL)
          (T (CONS (LIST (CAR X))
                   (SELF (CDR X))))
  )))
```

を得る．そして，このプログラムをもう1つのS式(A B C D E)に適用し，

? = ((A) (B) (C) (D) (E))

なる推論の結果を返す．このGAPシステムの能力は，基本的に使える経験的な知識に依存している．しかし，一般的な帰納推論の枠組みとは異なり，S式という個別的な対象領域に固有な知識を考えることができるという利点がある．

次に，複数個の例を与えた場合は，例の間の構造的な関係を解析することによって，いわゆるデータ駆動の手法でLISPプログラムの制御構造を推論することが自然に考えられる．このような戦略を用いて，Summers[28][29]は例によるLISPプログラム合成シ

ステムTHESYSを作成している．THESYSが対象とするLISPの基本演算子はcons, cdr, car, atomである．また，例の間の構造解析を簡潔に行うために，THESYSに与えられるS式の対の有限集合$E=\{\alpha_i \Rightarrow \beta_i ; 1 \leqq i \leqq n\}$に対し，入力S式からなる集合$E'=\{\alpha_i ; 1 \leqq i \leqq n\}$にある全順序を仮定する．THESYSは，このように順序づけられた例の集合から，各入力S式を識別するLISPの述語をまず生成する．例として，

$E=\{(\ )\Rightarrow(\ ), (A)\Rightarrow((A)), (BA)$
$\Rightarrow((B)(A))\}$

を考えてみる．この例の場合は，( )，(A)，(BA)に対してそれぞれ，

$p_1[x]=\text{atom}[x]$,
$p_2[x]=\text{atom}[\text{cdr}[x]]$,
$p_3[x]=\text{atom}[\text{cdr}[\text{cdr}[x]]]$

を生成する．次に，E中の各入出力対に対し，出力S式を入力S式で表現することを考え，たとえば(A)⇨((A))の場合，

$((A))=\text{cons}[(A) ; \text{nil}]$

であることに注目し，また(A)を変数化して

$f_2[x]=\text{cons}[x ; \text{nil}]$

なるM式を生成する．( )，(BA)についても同様に，それぞれ

$f_1[x]=\text{nil}$
$f_3[x]=\text{cons}[\text{cons}[\text{car}[x] ; \text{nil}] ;$
$\text{cons}[\text{cdr}[x] ; \text{nil}]]$

を生成する．このような$p_i[x]$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

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

が得られる．次に，THESYSは，$f_j[x], f_i[x]$の間に再帰的関係を発見し，それをLISP関数の再帰的呼び出しの形でコード化する．再帰的関係の発見のために，自然数の差分と同様なテクニックとある種の単一化を用いる．こうして最終的には，

$f[x]=[[\text{atom}[x]\rightarrow\text{nil}] ;$
$t\rightarrow u[x]]$
$u[x]=[\text{atom}[\text{cdr}[x]]\rightarrow\text{cons}[x ; \text{nil}] ;$
$t\rightarrow\text{cons}[\text{cons}[\text{car}[x] ; \text{nil}] ;$
$u[\text{cdr}[x]]]]$

なるLISPプログラムを得る．

Hardy[18]やSummers[28][29]の推論システムでは，例の集合が増大した場合については言及していない．しかし，Biermann[9]は，LISPプログラムの例による自動合成の問題を，正則LISP (regular LISP)プログラムを対象にして，極限におけるプログラム同定としてとらえている．

正則LISPプログラムの例による合成は，LISPという実際のプログラミング言語を対象にして，かつ理論的な裏づけをもっているという意味で大きな特徴をもっている．しかし，最小サイズのプログラムを求めるための計算量が指数関数的になるという難点がある．

3・2 トレースの活用[5]

Bauer[8]は，Biermann[9]のマージングの手法にヒントを得て，トレース記述言語CDL (Computation Description Language)を設計し，そのCDLで書かれたトレースから手続き型言語の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，まず似たような種類の命令が分類され同値類を形成する．次に，各同値類を1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．

そこで扱われるトレースは，形式的には手続き型言語の文の有限列である．こうしたトレースを用いたプログラム合成の基本的なアイディアは，プログラム中に出現する変数や定数などの使い方に関するさまざまな制約(約束)を，プログラムを合成するための知識として合成システムが活用することにあり，それによって，プログラム合成システムの負担が軽減される．

3・3 論理プログラムの自動合成[3][4]

本節では，節形式の1階言語，特にホーン節の集合を対象にしたShapiro[22][23]による論理プログラムの例による自動合成に関する研究の要点を説明する．

有限個の述語記号と関数記号をもつホーン節の集合をLとし，Lの部分集合として，グランドアトムの全体Gを考える．任意の解釈のもとで，偽になる空文を□とし，$\square\in G$であるとする．LのモデルKに対して$\alpha\in G$がKで，真であれば真，偽であれば偽と答えてくれるような装置を仮定する．このようなαと真偽値

vの対(α, v)を事実と呼ぶ．モデルKで真になるGの文の全体をG^Kと書く．また，導出原理によってαがLの部分集合Tから証明できることを$T \vdash \alpha$と書く．Kで真であり，かつ任意の$\alpha \in G^K$に対して$T \vdash \alpha$となるような$T \subseteq L$を，KのG完全な公理と呼ぶ．与えられた事実の列から，このようなG完全な公理を見つける帰納推論の問題をモデル推論という．

Kについての事実の列F_0, F_1, …で各$\alpha \in G$に対して，$F_i = <\alpha, v>$となるiが存在するような無限列をKの枚挙として考える．そうすると，モデル推論機械MはGに対するモデルの枚挙を一度に1つずつ読み込み，ときどきLの文の有限集合を仮説として出力するアルゴリズムということになる．Kの任意の枚挙に対して，MがKのG完全な公理に極限において収束するとき，Mは極限においてモデルKを同定するという．

さて，T_0, T_1, T_2, ……をLのすべての有限部分集合の枚挙とするとき，最も単純な推論機械を図1のように作ることができる．これは単純であるが強力である．しかし，そのままの形で使うと，while ループから脱出できず，次の事実の検討に進めなくなることもあり，効率も悪い．

```
k ← 0
repeat
    次の事実を検討する
    while   T_k がそれまでの事実に関して
            強すぎるか弱すぎる
            do   k ← k + 1
    output  T_k
forever
```

図 1　枚挙に基づく推論機械

こうした2つの難点を克服するために，Shapiro はまず，計算可能な関数hを while ループ内の演繹の段数を制御するために導入した．文αがTからn段で演繹できることを$T \vdash_n \alpha$で表し，それができないときは$T \not\vdash_n \alpha$で表す．そうして，図1の推論機械を図2のように改良した．この段階でもまだ，候補者としてのT_kをやみくもに枚挙しているが，彼はさらに，この枚挙に代わる手法を開発した．まず，$T_0 \leftarrow \{\square\}$として，図2の意味で，$T_k$が強すぎるときは，$T_k$の中の少なくとも1つのホーン節が偽であるといえるから，これをある実験 (crucial experiment) によって検出して除去し，それをT_{k+1}とする（その操作を矛盾点追跡法という）．また弱すぎるときには，ホーン節の精密化によりT_{k+1}を作る．

このように，段数監視用の関数hや矛盾点追跡法，精密化法を使って，モデル推論の効率化に成功した．

```
S_false ← {□} ; S_true ← { } ; k ← 0
repeat
    次の事実  F_n = < α, v > を読み込む
    S_v ← S_v ∪ { α }
    while   T_k ⊢_n α となる α ∈ S_false が存在するか
            T_k ⊬_h(i) α_i となる α_i ∈ S_true が存在する
      do    k ← k + 1
    output  T_k
forever
```

図 2　計算可能関数 h を用いた推論機械

これによって，帰納推論の実用化，すなわち，論理プログラムの例による自動合成が期待できるようになった．また，ホーン節の枠組みの中で，これまでの文法推論やプログラムの自動合成が統一的に扱え，さらに矛盾点追跡法の応用によって，論理プログラムを対象にした自動デバッガも構成でき，帰納推論の汎用性も高められた．

しかし，彼のモデル推論機械は，$T_0 \leftarrow \{\square\}$からスタートするために，正データ，すなわち$<\alpha$, true$>$なる事実が先行したときには，推論がまったく進まず，しかもT_kからT_{k+1}を決める際の探索空間は，まだ大きすぎるという欠点をもっていた．ごく最近，石坂[19]は，これらの問題点を最小汎化の概念を使って理論的に解決し，システムを実働化した．図3に，石坂の推論システムで扱った Prolog プログラムの合成の例を示しておく．

```
領域：リスト
言語：[ ]…空リスト
      [X | Y]…XとYを結合したリスト
      append(X, Y, Z)… ZはXにYを結合したリストである
      reverse(X, Y)… YはXの反転リストである
事実の例：
        <reverse([1, 2, 3], [3, 2, 1], true>
        <reverse([1], [2]), false>
        <append([1, 2], [3], [1, 2, 3]), true>
        <append([2, 3], [1], [2, 3]), false>
プログラム：reverse([ ], [ ]) ←
        reverse([X | Y], [Z | U]) ←reverse(Y, V),
                                append(V, [X], [Z | U])
        append([ ], X, X) ←
        append([Y|Z], X, [Y|U])←append(Z,X,U)
```

図 3　プログラムの帰納推論

3・4　例による編集[3]

具体的なプログラム自動合成システムとして，Nix[21]による例による編集 (editing by example, 以後 EBE と略記する）がある．これは，プログラムの自動合成の問題を人間と計算機の接点であるテキスト・エディタという形で実現したもので注目される．

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，時には反復的で退屈な作業も余儀なくされる．こうした，反復的で定型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の開発した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．

Nixの自動合成システムは，たとえば，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．すなわち，

大学名ファイル

北海道大学　〒060　札幌市…(011) 711−2111
東北大学　〒986　仙台市…(0222) 74−1111
東京大学　〒153　目黒区…(03) 485−3111
⋮

を，データベースの入力形式

大学一覧［大学名：北海道大学；郵便番号：060；所在地：札幌市…；電話番号：011−711−2111］
大学一覧［大学名：東北大学；郵便番号：986；所在地：仙台市…；電話番号：0222−74−1111］
大学一覧［大学名：東京大学；郵便番号：153；所在地：目黒区…；電話番号：03−485−3111］
⋮

に書き換える．

3・5　データエントリ機能[3]

帰納推論の一分野である文法推論を直接的に例によるプログラムの自動合成に応用した研究もある．Shinohara[24]−[27]は，Angluin[1]のパターン言語の帰納推論を推論時間も考慮したうえで，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．

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 : Angluin, D.
Title : Inductive Inference of Formal Languages from Positive Data
Journal : Inform. Contr. 45
Year : 1980
$
Author : Mayer, D.
Title : The Complexity of Some Problems on Subsequences and Supersequences
Journal : JACM 25
Year : 1978
$
⋮

何個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 : wTitle : xJournal : yYear : 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．

4．類　　推[3][6][7][13]−[17]

類推とは，与えられたいくつかの対象間に類似性（これを類比という）を検出し，その類比を用いて一方の対象で成立している事実や知識を，もう一方の対象に変換することにより，問題解決の手がかりを得たり，未知の事実などを予測・推定する推論方式のことである．

こうした類推については，最近まで個別的な問題や手法によって散発的に研究されただけで[20][30]−[32]，概念の定式化さえ十分でない状態であった．しかし，ごく最近になって原口，有川[13]−[17]により述語論理，特に確定節の枠組みの中で定式化された．

Winstonによると，類推とは，類似した前提が成立しているときに，類似した結論が成立するかもしれないと推論することである．彼は，前提と結論を因果関係における原因と結果としてとらえていたが，これを論理的含意関係における条件と結論と考え，確定節を対象にすれば，類推を述語論理の枠組みの中で一般的に定式化することができる．まず，類推の原理は，図4のように図式化することができる．図において，α_i，α_i'，β，β'は事実（本節ではグランドアトムのことを事実という）を表し，φは類比，すなわち類似性を示す対象間の関係である．そうすると，類推とは，$\alpha_i \varphi \alpha_i'$ $(1 \leq i \leq n)$であるとき，$\beta \varphi \beta'$なるS_2における事実β'を求めることであるといえる．

対象S_1：結論β←条件$\alpha_1, \cdots, \alpha_n$
φ類比
対象S_2：結論β'？←条件$\alpha_1', \cdots, \alpha_n'$

図4　類推の原理

このような類推を実現するためには，

①　類比φの定義

② 与えられた対象S_1とS_2からφを求める方法

③ $\beta\ \varphi\ \beta'$なるβ'を求める方法

が必要である．まず，φをS_1とS_2のグランド項間の対応関係として選ぶ．

αとα'が類比φによって同一視されることを$\alpha\ \varphi\ \alpha'$と書く．類推により，このような事実α'を求めるために，まずルール（確定節のこと）

$$R:\ \alpha \leftarrow \beta_1,\ \cdots,\ \beta_n$$

から，φによって変数を含まないルール

$$R':\ \alpha' \leftarrow \beta_1',\ \cdots,\ \beta_n'$$

を作り，これと既知の事実β_1'，…，β_n'からα'を三段論法で導き出すことを考える．RからR'を作る操作をルールの変換と呼び，このようにしてαを導き出す過程を基本図式，

$$\text{（具体化）}\quad \frac{\mathrm{A} \leftarrow \mathrm{B}_1, \cdots, \mathrm{B}_n}{}$$

$$\text{（ルールの変換）}\quad \frac{\alpha \leftarrow \beta_1, \cdots, \beta_n}{}$$

$$\text{（三段論法）}\quad \frac{\alpha' \leftarrow \beta_1', \cdots, \beta_n';\ \beta_1', \cdots, \beta_n'}{\alpha'}$$

で表す．具体化と三段論法は演繹における推論であるから，ルールの変換を演繹システム内で実現すれば，類推を演繹システムの中で統一的に扱うことが可能になる．

実際に，原口，有川[6][15]はそのような類推システムを実現している．簡単な例により類推の過程を示しておこう．

確定節の2つの集合，

$$\begin{aligned} S_1 = \{&f(b,c),\\ &m(a,b),\\ &gf(X,Z) \leftarrow p(X,Y), f(Y,Z),\\ &p(X,Y) \leftarrow f(X,Y),\\ &p(X,Y) \leftarrow m(X,Y)\} \end{aligned}$$

$$\begin{aligned} S_2 = \{&m(a',b'),\\ &f(b',c')\} \end{aligned}$$

が与えられたとする．ここに，f, m, p, gfはそれぞれ，父，母，親，祖父を表し，たとえば，$f(X, Y)$は，Xの父はYであることを表す．実現した類推システムでは，推論は逆向きに進むが，ここでは説明の都合上前向き推論を考える．また，類比φは，推論の過程で自動的に求まるようになっているが，ここでは，天下りに

$$\varphi = \{<a, a'>, <b, b'>, <c, c'>\}$$

を与えて話を進める．

さて，問題は，S_2において$gf(a', c')$を類推することである．類推システムは，まずS_2内で演繹を使って導出することを試みる．この場合，それに失敗するので，先に述べた類推の基本図式を適用する．まず，S_2において$p(b', c')$を類推する．すなわち，

$$\frac{p(X,Y) \leftarrow m(X,Y)}{p(a,b) \leftarrow m(a,b)}$$

$$\frac{p(a',b') \leftarrow m(a',b');\ m(a',b')}{p(a',b')}$$

次に，この$p(a', b')$を使って，

$$\frac{gf(X,Z) \leftarrow p(X,Y), f(Y,Z)}{gf(a,c) \leftarrow p(a,b), f(b,c)}$$

$$\frac{gf(a',c') \leftarrow p(a',b'), f(b',c');\ p(a',b'), f(b',c')}{gf(a',c')}$$

により$gf(a', c')$が類推できる．

5．類推によるプログラムの再利用[6]

類推をプログラムの開発に応用することも可能である．一般にプログラム開発に際しては，既存のプログラムの再利用がその作業の大部分を占めるとさえいわれている．再利用には，開発しようとしているプログラムに似たプログラムが使われることが多い．以下に，そのような例を1つ挙げておこう．いま，リストの反転を行う論理プログラム，

$$S_1 = \{C_1, C_2\},$$

$$C_1 : \mathrm{rev}([\,], [\,]),$$

$$C_2 : \mathrm{rev}([X|X_s], W) \leftarrow \mathrm{rev}(X_s, Z_s),\ \mathrm{append}(Z_s, [X], W)$$

とリストの結合を行うプログラム append は，すでに開発済みであるとしよう．このとき，リストの要素がまたリストであるような，リストの反転をするプログラム fullrev を作成することが，現在の問題であるとする．fullrev は，たとえば，

$$C_3 : \mathrm{fullrev}([\,], [\,]).$$

$$C_4 : \mathrm{fullrev}([1,2], [2,1]).$$

$$C_5 : \mathrm{fullrev}([[1,2], 3, [4, [5,6]]], [[[6,5], 4], 3, [2,1]]).$$

に対して成功するようなプログラムである．簡単のためC_3は目的のプログラムの一部として最初から用いることにして，

$$S_2 = \{C_3\}$$

とする．このS_2に対して，C_4はもちろん成功しない．そこで，S_1のルールの変換によってC_4を導く．実際，rev を fullrev と同一視し，類比，

$$\varphi = \{<[\,], [\,]>, <1, 1>, <2, 2>\}$$

のもとで，

$$\frac{\mathrm{rev}([X|X_s], W) \leftarrow \mathrm{rev}(X_s, Z_s),\ \mathrm{append}(Z_s, [X], W)}{\mathrm{rev}([1,2],[2,1]) \leftarrow \mathrm{rev}([2],[2]),\ \mathrm{append}([2],[1],[2,1])}$$

fullrev ([1, 2], [2, 1])←fullrev ([2], [2]),
append ([2], [1], [2, 1])

およびルールの変換

rev ([2], [2])←rev ([ ], [ ]),
append ([ ], [2], [2])

fullrev ([2], [2])←fullrev ([ ], [ ]),
append ([ ], [2], [2])

によって，fullrev ([1, 2], [2, 1]) が類推される．そこで，ルールの変換に使われたS_1のルールをφのもとでS_2に取り込み，

$S_2 = \{C_3$: fullrev ([ ], [ ]),
C_6 : fullrev ([X|X_s], W)
←fullrev (X_s, Z_s),
append (Z_s, [X], W)$\}$

を得る．このS_2はC_5に対して失敗し，また，ルールの変換によって新たなルールを取り込むことにも失敗する．この時点で，帰納推論における条件アトムおよび新たなプログラム節の生成手続きを呼ぶ．実際，C_6の本体中に条件アトム atom (X) を付加して，

C_7 : fullrev ([X|X_s], W)←
atom (X), fullrev (X_s, Z_s),
append (Z_s, [X], W).

を作り，また

C_8 : fullrev ([X|X_s], W)←fullrev (X, Y),
fullrev (X_s, Z_s),
append (Z_s, [Y], W).

を新たに生成して目的のプログラム，

$S_2 = \{C_3, C_7, C_8\}$.

を得る．

このようにして，類推に帰納推論を併用した形で，既知のプログラム群から新しく必要なプログラムを開発する方式が考えられる．この問題は，現在研究に着手したばかりであるが，有望な応用問題であるように思われる．

以上，帰納推論と類推の応用という観点から，プログラムの知的な合成という問題について説明した．理論と現実とのギャップは，まだ極めて大きいが，現実を直視して理論化することにより，このギャップは徐々に埋められるであろう．なお，本稿では紙数の関係で十分な解説ができなかったところが多いので，興味ある読者は筆者らによる文献を参照されたい．

◇ 参 考 文 献 ◇

(1) Angluin, D. : Inductive inference of formal languages from positive data, Inform. and Contr., Vol. 45, No. 2 , pp. 117-135 (1980).
(2) Angluin, D. and Smith, C. H. : Inductive Inference ; Theory and methods, Computing Surveys, Vol. 15, pp. 237-269 (1983) (大谷木重夫訳：帰納的推論－理論と方法－, bit 別冊, acm computing surveys '83, コンピュータ・サイエンス, pp. 107-135, 共立出版 (1985)).
(3) 有川節夫：帰納推論と類推－理論と応用－, 古川ほか編「知識の学習メカニズム」, pp. 23-51 共立出版 (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」, pp. 147-197, オーム社 (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」, pp. 199-219, オーム社 (1987).
(6) 有川節夫, 原口　誠：類推の理論, 大須賀, 佐伯編「知識の獲得と学習」, pp. 221-251, オーム社 (1987).
(7) 有川節夫：述語論理と推論, 電子通信学会誌, Vol. 69, No. 11, pp. 1113-1119 (1986).
(8) Bauer, M. A. : Programming by examples, Artif. Intell., Vol. 12, No. 1 , pp. 1-21 (1979).
(9) Biermann, A. W. : Regular LISP programs and their automatic synthesis from examples, Duke Univ., CS-1976-12 (1976).
(10) Carbonell, J. G. : A Computational model of analogical problem solving, Proc. of IJCAI-81, pp. 147-152 (1981).
(11) Gold, E. M. : Limiting recursion, J. Symbolic Logic, Vol. 30, No. 1 , pp. 28-48 (1965).
(12) Gold, E. M. : Language identification in the limit, Inform. and Contr., Vol. 10, No. 4 , pp. 447-474(1967).
(13) Haraguchi, M. : Towords a mathematical theory of analogy, Bull. Inform. Cybernetics, Vol. 21, No. 3, 4, pp. 29-56 (1985).
(14) Haraguchi, M. : Analogical reasoning using transformation of rules, Bull. Inform. Cybernetics, Vol.22, No. 1 ～ 2 , pp. 1-8 (1986).
(15) 原口　誠, 有川節夫：類推の定式化とその実現, 人工知能学会誌, Vol. 1, No. 1, pp. 132-139 (1986).
(16) Haraguchi, M. and Arikawa, S. : A Foundation of Reasoning by Analogy : Analogical union of logic programs, Proc. of the Logic Programming Conf. '86, pp. 103-110 (1986).
(17) 原口　誠：類推の機械化について, 古川ほか編「知識の学習メカニズム」, pp. 125-154 (1986).
(18) Hardy, S. : Synthesis of LISP functions from examples, Proc. of IJCAI-4, pp. 240-245 (1975).
(19) 石坂裕毅：汎化を用いたモデル推論, 九州大学総合理工学修士論文 (1986).
(20) Kling, R. E. : A paradigm for reasoning by analogy, Artif. Intell., Vol. 2, No. 2 , pp. 147-178 (1971).
(21) Nix, R. P. : Editing by Example, RR-280, Dept. Comp. Sci., Yale Univ. (1983).
(22) Shapiro, E. Y. : Inductive Inference of Theories from Facts, RR-192, Dept. Comp. Sci. Yale Univ. (1981) (有川節夫訳：知識の帰納的推論, 共立出版 (1986)).
(23) Shapiro, E. Y. : Algorithmic Program Debugging, The MIT Press (1983).
(24) Shinohara, T. : Polynomial time inference of pattern

languages and its application, Proc. 7th IBM Symp. Math. Found. of Comp. Sci., pp. 191-209 (1982).
(25) Shinohara, T. : Polynomial time inference of extended regular pattern languages, Proc. RIMS Symp. Software Sci. & Eng. (1982), LNCS-147, pp. 115-127, Springer-Verlag (1983).
(26) Shinohara, T. and Arikawa, S. : Learning Data Entry System ; An application of inductive inference of pattern languages, RR-102, Research Institute of Fundamental Information Science, Kyushu Univ. (1983).
(27) 篠原 武：言語の帰納推論，古川ほか編「知識の学習メカニズム」, pp. 53-70, 共立出版 (1986).
(28) Summers, P. D. : Program construction from example, IBM Res. PC-5637 (1975).
(29) Summers, P. D. : A methodology for LISP program construction from examples, JACM, Vol. 24, No. 1, pp. 161-175 (1977).
(30) Tangwangsong, S. and Fu, K. S. : An application of learning to robot planning, J. Comp. Inform. Sci., Vol. 8, No. 3, pp. 303-333 (1979).
(31) Winston, P. H. : Learning and reasoning by analogy, CACM, Vol. 23 (1980).
(32) Winston, P. H. : Learning new principles from precedents and exercises, Artif. Intell., Vol. 19, No. 3, pp. 321-350 (1982).

著 者 紹 介

有川 節夫（正会員）

昭和 39 年 3 月九州大学理学部数学科卒業．現在，九州大学理学部基礎情報学研究施設教授，大学院総合理工学研究科情報システム学専攻教授を兼務．理学博士．現在の主要研究テーマは，計算理論，情報検索論，知識情報処理，特に各種推論機構．第 11 回丹羽賞学術賞受賞．日本数学会，情報処理学会，ソフトウェア科学会，LA 各会員．