論　文

高階の差分を用いた入出力例からの関数の合成

正　員　犬塚　信博[†]　　正　員　高橋　健一[†]　　正　員　石井　直宏[†]

Construction of Functions from Examples by Using High Order Differences

Nobuhiro INUZUKA [†], Kenichi TAKAHASHI [†] *and* Naohiro ISHII [†], *Members*

あらまし　有限個の入出力例から，そのような入力-出力の関係を示す関数を導く問題は，プログラム自動合成の基本的な問題である．対象とする関数は，LISPのS式上の関数である．この問題を解決する方法として，Summersが与えたアルゴリズムがある．それは，与えられた入出力例から，入力-出力間の変換関数を作り，それに対してアルゴリズミックなマッチング手続きを施すことで，例の間の繰返し的な関係を見出し，これを一般化することで関数をプログラムとして生成する．本研究では，高階の差分を導入することで，この繰返し関係を見出す手続きの拡張を行った．本研究の拡張により例から導き出される関数の範囲は，Summersの基本的なアルゴリズムにより生成される関数の範囲を包含するものである．そして，この拡張は，以下の点で自然な拡張であると考えられる．すなわち，この拡張はSummersの方法の反復的な適用に基づいている．また，彼の方法の二つの特徴を継承している．つまり，適用される関数の特徴に依存せず一般性があること，ほとんど探索を含まないことである．更に，拡張した方法により得られたプログラムが，例から見出された繰返し的な関係によって一般化した関数と等しいことについても証明を与えた．

1. まえがき

有限個の事実から一般的な法則を見出す推論の方式を帰納的推論と言い，近年，この方式についての研究が注目されるようになってきた[(1),(2)]．その理由として，帰納的推論はプログラムの自動合成の分野で重要な手法とみなされていることにもよる．プログラム合成の基礎は，目的とするプログラムの機能を示す入出力の組みと，そのプログラムを記述するためのプログラム言語とから，意図するプログラムを得ることである．従って，プログラム合成を帰納的推論の問題として取り上げることは，問題空間の明確化，実際上の応用範囲の広さから，有用であると考えられる[(3),(4)]．

本研究では，LISPのS式からS式への関数を，その有限個の入出力例からプログラムとして自動合成する問題を取り上げる．帰納的推論のプロトタイプとして，入出力例よりLISPのプログラムを導くという分野があり，そこでは多くの研究が行われてきている．Summersは，LISPプログラムをその例から合成する問題について，一般的な方法を提案した[(5)]．そこでは，与えられたいくつかの入出力例を，car，cdr，consを用いて，入力と出力の間の変換として記述し，それにアルゴリズミックなマッチングの手続きを施すことによって，入出力例間に繰返し的な構造を見出す．従って，この方法では複数個の例が必要である．複数個の例を用いることは，帰納的推論としての確からしさを高めることにも増して，例の間の構造的差異の発見を可能にする．そのため，関数の導出をより一般的なものとすることができる．

この方法は，多くの研究に引き継がれている[(1)]．彼の方法の特徴として，特殊なヒューリスティックを与えないという意味で一般性に優れている点がある．また，手続き中にはほとんど探索を含まず，効率が良いことも挙げられる．但し，ここで考慮に入れる関数のクラスは，S式を定義域および値域とする関数の中で，出力が，入力に含まれるアトムの性質に依存しないようなものに限定されている．

本研究では，優れた一般性と効率の良さといったSummersの方法の主眼から逸脱しないで推論の枠を広

† 名古屋工業大学工学部電気情報工学科，名古屋市
Faculty of Engineering, Nagoya Institute of Technology, Nagoya-shi, 466 Japan

げるという意味で，自然な拡張を試みる．具体的には，与えられた入出力の例から導出される入力-出力の変換式に関する，アルゴリズミックなマッチング手続きの反復的な使用に基礎を置くものである．この拡張により，適用範囲を広げることができる．文献(5)では，上で述べたもの以外に，変数付加(variable addition)の手法を取り入れることにより適用範囲を広げている．しかし，これは探索を含んでおり，効率の良さを損なうものである．本研究で提案する拡張は，この探索による効率の悪さを避けるための手法でもある．

更に本研究では，この拡張された方法の妥当性として，生成された関数が，与えられた例，およびそれに自然と思われる一般化をした結果と一致するという意味で，正当であることを証明した．ここで，目標関数が無限に存在する場合の，有限個の例からの一般化について，厳密な意味での妥当性を論ずることはできない．ここでは，一般化自身は正しいものと認めた上で，単に，生成されたプログラムが推論の結果を正しく表現しているという意味での正当性を保証するものである．

以下，**2.** では基本的な用語を導入し，Summers の方法を概説する．また，**3.** では本研究で提案する高階差分の手続きを述べ，**4.** では本研究での拡張した方法により生成される関数の正当性について証明する．

2. 入出力例からの関数の生成

2.1 対象と関数フラグメント

1. で述べたように，LISP での S 式を定義域，値域とする関数を扱う．但し，その出力は，入力された S 式に含まれるアトムそのものの性質には依存しないようなものに限定する．例えば，リスト (2 7 1) を入力とし，そこに含まれる要素を数として加算し，10 を得るような関数は，考慮に入れない．アトムの数としての性質を利用しているからである．例えば，ここで考える関数は，次の集合 EX_1 で示される入出力例をもつようなものである (ここで，「→」は入力と出力の区切りを示す)．

$EX_1=\{(\ \ )\rightarrow(\ \ )\ ;$
$(A)\rightarrow((A))\ ;$
$(A\ B)\rightarrow((A)\ (B))\ ;$
$(A\ B\ C)\rightarrow((A)\ (B)\ (C))\}$

この集合が意図する関数は，もちろん一意的ではないが，入力リストの各要素からそれのみを用いたリストを作り，それらを更にリストにしたものと考えられる．このような関数の出力は，入力 S 式中の各アトムの性質ではなく，リストに含まれる要素の個数など，S 式の構造のみに式が左右される．このような範囲の関数は，基本関数として car，cdr，cons，述語として atom，それにマッカーシの条件式と再帰を含む制御構造を用いることで実現できる (アトムの判別をする必要はないため，eq 等はいらない)．

基本関数の単なる組合せ (合成) からなる関数として，S 式から S 式への単純な変換を表す関数を定義する．これを関数フラグメント (function fragment) と呼び，便宜上，ラムダ記法を用いて表す．

[定義 2.1] n 引数の関数フラグメントの集合 FF_n は，次の①～④のみで導かれる．

① a がアトムならば $\lambda x_1\cdots x_n.\ a\in FF_n$
② $\lambda x_1\cdots x_n.\ x_i\in FF_n,\ 1\leqq i\leqq n$
② $f\in FF_n$ ならば
$\lambda x_1\cdots x_n.\ \mathrm{car}[f[x_1\ ;\cdots\ ;x_n]]\in FF_n,$
$\lambda x_1\cdots x_n.\ \mathrm{cdr}[f[x_1\ ;\cdots\ ;x_n]]\in FF_n$
④ $f_1, f_2\in FF_n$ ならば
$\lambda x_1\cdots x_n.\ \mathrm{cons}[f_1[x_1\ ;\cdots\ ;x_n]\ ;$
$f_2[x_1\ ;\cdots\ ;x_n]]\in FF_n$ □

また，述語のクラスとして，次の P を定義する．この述語は，生成される関数の定義において用いるものであり，それぞれの S 式に適した操作をするために，入力された S 式のタイプを区別するものである．

[定義 2.2] 述語の集合 P は，1 引数の関数フラグメント FF_1 から，次のように導かれる．

$f\in FF_1$ ならば $\lambda x.\ \mathrm{atom}[f[x]]\in P$ □

2.2 Summers のアルゴリズム

有限個の入出力例の集合から，それが示すような振舞いをする関数を導くためのアルゴリズムが，Summers によって提案されている(5)．ここで，k 個の入出力例の集合 $\{x_i\rightarrow y_i \mid i\leqq k\}$ が与えられているとすると，そのアルゴリズムは，次のような手順からなる．

① 各入出力例 $x_i\rightarrow y_i$ から，x_i を y_i に変換する関数フラグメント $f_i\in FF_1$ を求める．つまり，$y_i=f_i[x_i]$ の関係をもつ関数フラグメントである．

② 関数フラグメント間の規則性を，繰返し的関係として見出し，次のように記述する．これは，ある式と，その一つ前の添字をもつ式とのマッチングを行うことで実現できる．

$$\forall x, f_{i+1}[x]=a[f_i[b[x]]\ ;\ x],\quad i\leqq k-1 \qquad (1)$$

③ $\{x_1, x_2, \cdots, x_k\}$ から，次のような述語の集合 $\{p_1, p_2, \cdots, p_k\}$ を導く．

$p_i[x_j]$=true ($i=j$) or false ($i<j$)
(その他の i, j では任意)

④ 述語間の規則性を繰返し的関係として，次のように記述する．

$$\forall x, p_{i+1}[x]=p_i[b[x]], \quad i\leqq k-2 \qquad (2)$$

⑤ 関数フラグメントの規則性，述語の規則性を，与えた例についてだけでなく，一般的に成り立つものとして汎化する．つまり，式(1)，(2)の i に関する条件を外す．

⑥ 汎化された規則性から，関数を再帰的プログラムとして生成する．

このような手順で関数を生成するには，与える例にもいくつかの条件が必要である．その詳細については本論文では述べないが，少なくとも，S式の集合 S 上での次の擬順序 $\leqq$ において，小さい順に並んで与えられなければならない．すなわち，この擬順序は，a を任意のアトム，s，s_1，s_2，s_3，s_4 を任意のS式とすると，(ア) $a\leqq s$，(イ) $s_1\leqq s_2$，$s_3\leqq s_4$ ならば $(s_1.s_3)\leqq(s_2.s_4)$ である．

最後⑥の，規則性からのプログラムの生成は，繰返し関係で記述された関数と，ある再帰的な関数との等価性を示す次の定理を用いることで行われる．

[定理 2.3] (基本構成定理) 繰返し関係，

$$f_1, f_{i+1}=\lambda x.\ a[f_i[b[x]]; x]$$
$$p_1, p_{i+1}=\lambda x.\ p_i[b[x]] \quad i\geqq 1 \qquad (3)$$

として記述される関数は，次の再帰的な関数 F と等価である．ここで，t は真であることを示す定数．

$$F=\lambda x.\ [p_1[x]\rightarrow f_1[x];$$
$$t\rightarrow a[F[b[x]]; x]] \qquad \square$$

ここで，式(3)のように繰返し関係を言った場合，初めの式は与えられており，以下がそれに続く関係式で与えられることを示す．そして，それによって記述される関数と言った場合，次の関数 G を意味する．これは，次のような無限に続くマッカーシの条件式で与えられる関数 G' を使って定義される．

$$G=\lambda x.\ G'[x; x]$$
$$G'=\lambda xy.\ [p_1[y]\rightarrow f_1[x];$$
$$p_2[y]\rightarrow f_2[x];$$
$$\cdots$$
$$p_n[y]\rightarrow f_n[x];$$
$$\cdots \quad]$$

この定理は，Summersのアルゴリズムを支えるものである．証明は文献(5)を参照．

2.3 例　　題

例の集合 EX_1 を上の手続きに適用する．EX_1 の各例から，それぞれ次の関数フラグメントが得られる．

$$f_1=\lambda x.\ \text{nil}$$
$$f_2=\lambda x.\ \text{cons}[\text{cons}[\text{car}[x]; \text{nil}]; \text{nil}]$$
$$f_3=\lambda x.\ \text{cons}[\text{cons}[\text{car}[x]; \text{nil}];$$
$$\text{cons}[\text{cons}[\text{cadr}[x]; \text{nil}]; \text{nil}]]$$
$$f_4=\lambda x.\ \text{cons}[\text{cons}[\text{car}[x]; \text{nil}];$$
$$\text{cons}[\text{cons}[\text{cadr}[x]; \text{nil}];$$
$$\text{cons}[\text{cons}[\text{caddr}[x]; \text{nil}]; \text{nil}]]]$$

例えば，関数フラグメント f_3 が，$f_3[(A\ B)]=\text{cons}[\text{cons}[A; \text{nil}]; \text{cons}[\text{cons}[B; \text{nil}]; \text{nil}]]=((A)(B))$ であるように，各関数フラグメントは入出力間の関係を示している．これらの関数フラグメントの間には，次のような規則性を見ることができる．

$$f_{i+1}=\lambda x.\ \text{cons}[\text{cons}[\text{car}[x]; \text{nil}]; f_i[\text{cdr}[x]]]$$
$$(i\leqq 3) \qquad (4)$$

また，各例の入力を識別するための述語は，次のように求められる．

$$p_1=\lambda x.\ \text{atom}[x]$$
$$p_2=\lambda x.\ \text{atom}[\text{cdr}[x]]$$
$$p_3=\lambda x.\ \text{atom}[\text{cddr}[x]]$$
$$p_4=\lambda x.\ t$$

これらの間にも，次のような規則性がある．

$$p_{i+1}=\lambda x.\ p_i[\text{cdr}[x]] \quad (i\leqq 2) \qquad (5)$$

式(4)，(5)について，i に関する条件を外し，汎化する．

$$f_{i+1}=\lambda x.\ \text{cons}[\text{cons}[\text{car}[x]; \text{nil}]; f_i[\text{cdr}[x]]]$$
$$p_{i+1}=\lambda x.\ p_i[\text{cdr}[x]] \qquad (6)$$

この繰返し的な規則性に，定理 2.3 を適用することで次の関数 unpack を得る．

$$\text{unpack}=\lambda x.\ [\text{atom}[x]\rightarrow\text{nil};$$
$$t\rightarrow\text{cons}[\text{cons}[\text{car}[x]; \text{nil}];$$
$$\text{unpack}[\text{cdr}[x]]]]$$

この関数は，与えた例の振舞いを記述するだけでなく，その一般化にもなっている．

しかし，このようにして関数を一般化するには，与えられた例から導かれる関数フラグメントに，明白な規則性が必要である．つまり，関数フラグメントの列において，ある関数フラグメントが，一定の形式で，一つ前の関数フラグメントによって，記述されなければならない．述語も同様である．この規則性の発見に伴う条件は，かなり厳しいものであり，広い範囲において適用できるとは限らない．そのため文献(5)では，変数付加の手法に用いている．しかし，ここでは変数

付加を用いず，多少なりともこの方法の能力を高めることを考える．

3. 高階の差分を用いた関数の生成

3.1 規則性発見の拡張へ

2. で述べた方法では，関数フラグメントが各々，

$f_{i+1}[x]=a[f_i[b[x]]\,;\,x]$

といった一定の形式を用いて，関数フラグメントの列中，一つ若い添字をもつもので表されなければならなかった．すなわち，関数フラグメント f_{i+1} が，一つ前の関数フラグメント f_i，2引数関数 a，および，1引数関数 b を用いて表現される必要があった．ところが，実際には，a は一定にならず，規則性をもちながらも各々異なったものになることが多い（同様に，b についても添字 i に関して一定にならない場合を考えることができる．しかし，その可能性は低く，そのためここでは a の変化のみを考えることにする）．そこで，a を添字に依存する形で表してみる．

$f_2=\lambda x.\,a_1[f_1[b[x]]\,;\,x]$

$f_3=\lambda x.\,a_2[f_2[b[x]]\,;\,x]$

$\cdots$

$f_{i+1}=\lambda x.\,a_i[f_i[b[x]]\,;\,x]$

$\cdots$

そして，この a_i についての規則性を，f_i の規則性を求めるときに用いたものと同様の方法により，発見することを考える．つまり，Summers の方法の反復的な適用を考える．

3.2 差分と繰返し関係

2. で述べたような，ある関数フラグメントを別の関数フラグメントで表すという操作を，「差分」という．

[定義 3.1] 二つの1変数関数フラグメント $f_1, f_2\in FF_1$ に対し，任意の入力 x で，

$f_2[x]=a[f_1[b[x]]\,;\,x]$

となる $a\in FF_2$，$b\in FF_1$ を求めることを，「差分する」という．また，a を「f_2 の f_1 に対する差分関数フラグメント」，b を「引渡し関数」という． □

[定義 3.2] 関数フラグメントの k 組 $(f_1,\cdots,f_k)$ に対し，f_i' が f_{i+1} の f_i に対する差分関数で，そのときの引渡し関数がすべて等しいとき，$(f_1',\cdots,f_{k-1}')$ を，「$(f_1,\cdots,f_k)$ の差分」という． □

Summers のアルゴリズムは，与えられた例の集合から作られた関数フラグメントの組の差分が，すべて等しい差分関数フラグメントになることで，規則性を見つけるものであった．そうした差分関数フラグメントを見つけることにより，関数フラグメントの組を繰返し的関係として記述できる．ここでは，差分の概念を次のように拡張する．このことにより，一定にならなかった関数フラグメントの中に規則性を見出す手段を与える．

[定義 3.3]（定義 3.1 の拡張） 二つの n 変数関数フラグメント $f_1, f_2\in FF_n$ に対し，任意の入力 $x_1,\cdots,x_n$ で，

$f_2[x_1\,;\cdots;\,x_n]$

$=a[f_1[b_1[x_1]\,;\cdots;\,b_n[x_n]]\,;\,x_1\,;\cdots;\,x_n]$

となる $a\in FF_{n+1}$，および，$b_1,\cdots,b_n\in FF_1$ を求めることを，「差分する」という．また，a を「f_2 の f_1 に対する差分関数フラグメント」，$b_1,\cdots,b_n$ を「引渡し関数」という． □

定義 3.2 で定義した「差分」は，定義 3.3 での拡張された定義でも，そのまま言うことにする．

[定義 3.4] 関数フラグメントの k 組 $(f_1,\cdots,f_k)$ に対し，その差分 $(f_1',\cdots,f_{k-1}')$ を，「1階差分」という．また，1階差分中の関数フラグメント f_i' を，「1階差分関数フラグメント」という．

m 階差分 $(f_1^{(m)},\cdots,f_{k-m}^{(m)})$ の差分，$(f_1^{(m+1)},\cdots,f_{k-m-1}^{(m+1)})$ を「$m+1$ 階差分」という． □

与えられた例の集合から作られた関数フラグメントに対し，1階差分，2階差分，3階差分，…と取り，ある所で一定になれば規則性を見出すことができる．

例えば，次の例を取り上げる．

$EX_2=$

$\{(\ )\quad\rightarrow(\ )\,;$

$(A)\quad\rightarrow(A)\,;$

$(A\ B)\quad\rightarrow(B\ B\ A)\,;$

$(A\ B\ C)\quad\rightarrow(C\ C\ C\ B\ B\ A)\,;$

$(A\ B\ C\ D)\quad\rightarrow(D\ D\ D\ D\ C\ C\ C\ B\ B\ A)\,;$

$(A\ B\ C\ D\ E)\rightarrow(E\ E\ E\ E\ E\ D\ D\ D\ D\ C\ C\ C\ B\ B\ A)\}$

ここから構成される関数フラグメントは，それぞれ次のようになる．

$f_1=\lambda x_1.\,\mathrm{nil}$

$f_2=\lambda x_1.\,\mathrm{cons}[\mathrm{car}[x_1]\,;\,\mathrm{nil}]$

$f_3=\lambda x_1.\,\mathrm{cons}[\mathrm{cadr}[x_1]\,;\,\mathrm{cons}[\mathrm{cadr}[x_1]\,;\,\mathrm{cons}[\mathrm{car}[x_1]\,;\,\mathrm{nil}]]]$

$f_4=\lambda x_1.\,\mathrm{cons}[\mathrm{caddr}[x_1]\,;\,\mathrm{cons}[\mathrm{caddr}[x_1]\,;\,\mathrm{cons}[\mathrm{caddr}[x_1]\,;\,\mathrm{cons}[\mathrm{cadr}[x_1]\,;\,\mathrm{cons}[\mathrm{cadr}[x_1]\,;\,\mathrm{cons}[\mathrm{car}[x_1]\,;\,\mathrm{nil}]]]]]]$

$f_5=\lambda x_1.\,\mathrm{cons}[\mathrm{cadddr}[x_1];\mathrm{cons}[\mathrm{cadddr}[x_1];$
$\mathrm{cons}[\mathrm{cadddr}[x_1];\mathrm{cons}[\mathrm{cadddr}[x_1];$
$\mathrm{cons}[\mathrm{caddr}[x_1];\mathrm{cons}[\mathrm{caddr}[x_1];$
$\mathrm{cons}[\mathrm{caddr}[x_1];\mathrm{cons}[\mathrm{cadr}[x_1];$
$\mathrm{cons}[\mathrm{cadr}[x_1];\mathrm{cons}[\mathrm{car}[x_1];\mathrm{nil}]]]]]]]]]]$

$f_6=\lambda x_1.\,\mathrm{cons}[\mathrm{caddddr}[x_1];\mathrm{cons}[\mathrm{caddddr}[x_1];$
$\mathrm{cons}[\mathrm{caddddr}[x_1];\mathrm{cons}[\mathrm{caddddr}[x_1];$
$\mathrm{cons}[\mathrm{caddddr}[x_1];\mathrm{cons}[\mathrm{cadddr}[x_1];$
$\mathrm{cons}[\mathrm{cadddr}[x_1];\mathrm{cons}[\mathrm{cadddr}[x_1];$
$\mathrm{cons}[\mathrm{cadddr}[x_1];\mathrm{cons}[\mathrm{caddr}[x_1];$
$\mathrm{cons}[\mathrm{caddr}[x_1];\mathrm{cons}[\mathrm{caddr}[x_1];$
$\mathrm{cons}[\mathrm{cadr}[x_1];\mathrm{cons}[\mathrm{cadr}[x_1];$
$\mathrm{cons}[\mathrm{car}[x_1];\mathrm{nil}]]]]]]]]]]]]]]$

従って，$(f_1, f_2, f_3, f_4, f_5, f_6)$ の 1 階差分 $(f_1^{(1)}, f_2^{(1)}, f_3^{(1)}, f_4^{(1)}, f_5^{(1)})$ は，次のようになる．

$f_1^{(1)}=\lambda x_1x_2.\,\mathrm{cons}[\mathrm{car}[x_2];x_1]$

$f_2^{(1)}=\lambda x_1x_2.\,\mathrm{cons}[\mathrm{cadr}[x_2];\mathrm{cons}[\mathrm{cadr}[x_2];x_1]]$

$f_3^{(1)}=\lambda x_1x_2.\,\mathrm{cons}[\mathrm{caddr}[x_2];\mathrm{cons}[\mathrm{caddr}[x_2];$
$\mathrm{cons}[\mathrm{caddr}[x_2];x_1]]]$

$f_4^{(1)}=\lambda x_1x_2.\,\mathrm{cons}[\mathrm{cadddr}[x_2];\mathrm{cons}[\mathrm{cadddr}[x_2];$
$\mathrm{cons}[\mathrm{cadddr}[x_2];\mathrm{cons}[\mathrm{cadddr}[x_2];x_1]]]]$

$f_5^{(1)}=\lambda x_1x_2.\,\mathrm{cons}[\mathrm{caddddr}[x_2];\mathrm{cons}[\mathrm{caddddr}[x_2];$
$\mathrm{cons}[\mathrm{caddddr}[x_2];\mathrm{cons}[\mathrm{caddddr}[x_2];$
$\mathrm{cons}[\mathrm{caddddr}[x_2];x_1]]]]]$

このときの引渡し関数は $\lambda x.\,x$ である．

もう一度差分することで 2 階差分 $(f_1^{(2)}, f_2^{(2)}, f_3^{(2)}, f_4^{(2)})$ を得る．

$f_1^{(2)}=\lambda x_1x_2x_3.\,\mathrm{cons}[\mathrm{cadr}[x_3];x_1]$

$f_2^{(2)}=\lambda x_1x_2x_3.\,\mathrm{cons}[\mathrm{caddr}[x_3];x_1]$

$f_3^{(2)}=\lambda x_1x_2x_3.\,\mathrm{cons}[\mathrm{cadddr}[x_3];x_1]$

$f_4^{(2)}=\lambda x_1x_2x_3.\,\mathrm{cons}[\mathrm{caddddr}[x_3];x_1]$

このときの引渡し関数は，それぞれ $\lambda x.\,x$，$\lambda x.\,\mathrm{cdr}[x]$ である．

更にもう一度差分することで 3 階差分 $(f_1^{(3)}, f_2^{(3)}, f_3^{(3)})$ を得る．

$f_1^{(3)}=\lambda x_1x_2x_3x_4.\,x_4$

$f_2^{(3)}=\lambda x_1x_2x_3x_4.\,x_4$

$f_3^{(3)}=\lambda x_1x_2x_3x_4.\,x_4$

このときの引渡し関数は，それぞれ，$\lambda x.\,x$，任意，$\lambda x.\,\mathrm{cdr}[x]$ である．

このとき 3 階差分に含まれる関数フラグメントがすべて一致しており，これで規則性が見出された．規則性は次のような繰返し的関係である．

すなわち，関数フラグメント f_i は $f_i^{(1)}$ を用いることで繰返し的に記述され，また，$f_i^{(1)}$ は $f_i^{(2)}$ を用いることで繰返し的に記述される．そして各 f_i は，そこに現れる定まっていない関数フラグメントに，順次繰返し関係を適用してゆくことで得られる．

$f_1=\lambda x_1.\,\mathrm{nil},$

$f_{i+1}=\lambda x_1.\,f_i^{(1)}[f_i[x_1];x_1]$

但し，

$f_1^{(1)}=\lambda x_1x_2.\,\mathrm{cons}[\mathrm{car}[x_2];x_1],$

$f_{i+1}^{(1)}=\lambda x_1x_2.\,f_i^{(2)}[f_i^{(1)}[x_1;\mathrm{cdr}[x_2]];x_1;x_2]$

$f_1^{(2)}=\lambda x_1x_2x_3.\,\mathrm{cons}[\mathrm{cadr}[x_3];x_1],$

$f_{i+1}^{(2)}=\lambda x_1x_2x_3.\,f_i^{(2)}[x_1;x_2;\mathrm{cdr}[x_3]]$ (7)

述語についても同様である．例の集合から生成される述語は次のとおりである．

$p_1=\lambda y.\,\mathrm{atom}[y]$

$p_2=\lambda y.\,\mathrm{atom}[\mathrm{cdr}[y]]$

$p_3=\lambda y.\,\mathrm{atom}[\mathrm{cddr}[y]]$

$p_4=\lambda y.\,\mathrm{atom}[\mathrm{cdddr}[y]]$

$p_5=\lambda y.\,\mathrm{atom}[\mathrm{cddddr}[y]]$

これは，1 階差分によって次のような規則性にまとめられる．

$p_1=\lambda y.\,\mathrm{atom}[y],$

$p_{i+1}=\lambda y.\,p_i[\mathrm{cdr}[y]]$ (8)

3.3 関数の生成

3.2 で述べたように，差分によって求められた規則性から，関数が求められる．これは，定理 2.3 を拡張した次の定理に従う．この定理は，**4.** で証明される．

[定理 3.5] 次の繰返し関係を考える．

$f_1, f_{i+1}=\lambda x.\,f_i^{(1)}[f_i[b_1[x]];x]$

$p_1, p_{i+1}=\lambda x.\,p_i[bp[x]]\quad i\geq 1$

但し，$f^{(j)}\,(j=1,\cdots,n-2)$ は，

$f_1^{(j)}, f_{i+1}^{(j)}=\lambda x_1\cdots x_{j+1}.\,f_i^{(j+1)}[f_i^{(j)}[b_i^{(j)}[x_1];$
$\cdots;b_{j+1}^{(j)}[x_{j+1}]];x_1;\cdots;x_{j+1}]$

また，$f^{(n-1)}$ は，

$f_1^{(n-1)}, f_{i+1}^{(n-1)}$
$=\lambda x_1\cdots x_n.\,a[f_i^{(n-1)}[b_1^{(n-1)}[x_1];$
$\cdots;b_n^{(n-1)}[x_n]];x_1;\cdots;x_n]$

で与えられる．このように記述される関数は，次の関数 G と等価である．

$G=\lambda x.\,F[x;x]$

但し，F は，

$F=\lambda x_1y.$
$[p_1[y]\rightarrow f_1[x_1];$
$t\rightarrow F^{(1)}[F[b_1[x_1];bp[y]];x_1;y]]$

また，$F^{(j)}\,(j=1,\cdots,n-2)$ は，

$F^{(j)}=\lambda x_1\cdots x_{j+1}y.$
$[p_{j+1}[y]\rightarrow f_1{}^{(j)}[x_1;\cdots;x_{j+1}];$
$t\rightarrow F^{(j+1)}[F^{(j)}[b_1{}^{(j)}[x_1];\cdots;$
$b_{j+1}{}^{(j)}[x_{j+1}];bp[y]]x_1;$
$\cdots;x_{j+1};y]]$

そして，$F^{(n-1)}$ は，

$F^{(n-1)}=\lambda x_1\cdots x_ny.$
$[p_n[y]\rightarrow f_1{}^{(n-1)}[x_1;\cdots;x_n];$
$t\rightarrow a[F^{(n-1)}[b_1{}^{(n-1)}[x_1];\cdots;$
$b_n{}^{(n-1)}[x_n];bp[y]];x_1;$
$\cdots;x_n]]$ □

である．

これを **3.2** の例題で得た規則性に適用することで，次のような関数 G が得られる．

$G=\lambda x.F[x;x]$

但し，

$F=\lambda x_1y.$
$[\mathrm{atom}[y]\rightarrow\mathrm{nil};$
$t\rightarrow F^{(1)}[F[x_1;y];x_1;y]]$
$F^{(1)}=\lambda x_1x_2y.$
$[\mathrm{atom}[\mathrm{cdr}[y]]\rightarrow\mathrm{cons}[\mathrm{car}[x_2];x_1];$
$t\rightarrow F^{(2)}[F^{(1)}[x_1;\mathrm{cdr}[x_2];y];x_1;x_2;y]]$
$F^{(2)}=\lambda x_1x_2x_3y.$
$[\mathrm{atom}[\mathrm{cddr}[y]]\rightarrow\mathrm{cons}[\mathrm{cadr}[x_3];x_1];$
$t\rightarrow F^{(2)}[x_1;x_2;\mathrm{cdr}[x_3];y]]$

である．

この手法は，関数生成の一つの方法であり，Summers の基本的な手法を包含するものである．また，関数フラグメント，および各階の差分関数フラグメントが，常に，隣接するフラグメントの部分式とマッチング可能であるような場合，彼の変数付加の方法と同等か，それ以上の振舞いをする．例えば，リストの反転をする関数は，文献(5)で変数付加によって求められている．本研究で提案した方法では，これは 2 階の差分によって求めることが可能である．しかし，Summers の変数付加の方法に取って代わるという意味ではない．これを他の手法と組み合わせて用いることで，より効果があると考える．

4. 構成定理の証明

2.2 で述べた定理 2.3 は，文献(5)にその証明が与えられている．そこでは，アルゴリズムによって定義される再帰的な関数を，次のような汎関数，

$\tau=\lambda\Phi.\lambda x.[p_1[x]\rightarrow f_1[x];$
$t\rightarrow a[\Phi[b[x]];x]]$

の不動点として表す．そして，これが不動点をもつこと，また，その最小不動点が繰返し関係として与えられる関数と等価であることを証明する．

この章では，定理 2.3 の拡張として本研究で考案した定理 3.5 を証明する．ここでの証明も，不動点意味論(6)に基づいて行う．そこで，定理 3.5 における再帰的な関数を，汎関数による記述に直し，その汎関数の不動点として論ずる．この定理 3.5 を汎関数を使った記述に直すと，次のようになる．

[定理 3.5] (汎関数による表現) 汎関数 $\tau_1,\cdots,\tau_{n-1}(\tau_j:D_{j+2}\rightarrow D_{j+2})$ を次のように定義する．

$\tau_j=\lambda\Phi.\lambda x_1\cdots x_{j+1}y.$
$[p_{j+1}[y]\rightarrow f_1{}^{(j)}[x_1;\cdots;x_{j+1}];$
$t\rightarrow F^{(j+1)}[\Phi[b_1{}^{(j)}[x_1];\cdots;$
$b_{j+1}{}^{(j)}[x_{j+1}];bp[y]];x_1;\cdots;$
$x_{j+1};y]]$
$(j=0,\cdots,n-2)$
$\tau_{n-1}=\lambda\Phi.\lambda x_1\cdots x_ny.$
$[p_n[y]\rightarrow f_1{}^{(n-1)}[x_1;\cdots;x];$
$t\rightarrow a[\Phi[b_1{}^{(n-1)}[x_1];\cdots;b_n{}^{(n-1)}[x_n];$
$bp[y]];x_1;\cdots;x_n]]$

上で $f_i{}^{(0)}$ は f_i を意味するものとする．このとき，$\tau_1,\cdots,\tau_{n-1}$ は不動点をもつ．但し，そのときの最小不動点をそれぞれ $F^{(1)},\cdots,F^{(n-1)}$ とする．このとき，τ_0 も不動点をもち，その最小不動点を F とする．すると，次のように定義される関数 G は，

$G=\lambda x.F[x;x]$

次の繰返し関係によって定義される関数 F と等価である．

$f_1,f_{i+1}=\lambda x.f_i{}^{(1)}[f_i[b_1[x]];x]$
$p_1,p_{i+1}=\lambda x.p_i[\mathrm{bp}[x]]$

但し，$f^{(j)}\ (j=1,\cdots,n-2)$ は，

$f_1,f_{i+1}{}^{(j)}=\lambda x_1\cdots x_{j+1}.$
$f_i{}^{(j+1)}[f_i{}^{(j)}[b_1{}^{(j)}[x_1];\cdots;$
$b_{j+1}{}^{(j)}[x_{j+1}]];x_1;\cdots;x_{j+1}],$

また，$f^{(n-1)}$ は，

$f_1,f_{i+1}{}^{(n-1)}=\lambda x_1\cdots x_n.$
$a[f_i{}^{(n-1)}[b_1{}^{(n-1)}[x_1];\cdots;$
$b_n{}^{(n-1)}[x_n]];x_1;\cdots;x_n]$

で与えられる． □

ここで，S 式の集合を S，未定義要素を ω で表すと，

$D_j=((S\cup\{\omega\})\times\overset{j}{\cdots}\times(S\cup\{\omega\})\rightarrow(S\cup\{\omega\}))$

である．但し，× は直積を表す．

3. での定理 3.5 にある再帰的関数 $F, F^{(1)}, \cdots$ は，上の定理の汎関数 $\tau_0, \tau_1, \cdots$ の最小不動点である．

D_j の元は，$(S\times\overset{j}{\cdots}\times S\to S)$ の元として表される関数の近似であると見ることができる．そこで，近似の度合による順序を導入したい．

[定義 4.1] $F, G\in Dn$ に対し，$F\leqq_F G \Longleftrightarrow$ 任意の $x_1, \cdots, x_n\in S$ に対して，$F[x_1; \cdots; x_n]\neq\omega$ ならば $F[x_1; \cdots; x_n]=G[x_1; \cdots; x_n]$ □

各 D_j に対して，別々の関係 $\leqq_F$ を定義する必要があるが，あいまいではないので同一の記号を用いる．

すると次の諸性質を示すことができる．

[性質 4.2] 関係 $\leqq_F$ は D_j において半順序をなす． □

[性質 4.3] D_j 上の $\leqq_F$ に関する任意の鎖 (chain)，つまり，$F_0, F_1, \cdots (F_l\in D_j,\ F_l\leqq_F F_{l+1})$ は，一意な極限 F (すなわち，$\{F_0, F_1, \cdots\}$ の一意な最小上界) をもつ．これを $\lim_{l\to\infty} F_l$ と書く． □

鎖 $F_0, F_1, \cdots$ の一意な極限は，次のように定まる F である．

ある $x_1, \cdots, x_n\in S$ に対し，

$$F[x_1; \cdots; x_n] = \begin{cases} a & \text{ある } l \text{ で } F_l[x_1; \cdots; x_n]=a\,(\neq\omega) \text{ となるとき．} \\ \omega & \text{その他のとき．} \end{cases}$$

[性質 4.4] D_j は，$\leqq_F$ に関して $\Omega y=\lambda x_1\cdots x_j.\ \omega$ なる最小元をもつ． □

汎関数 $\tau_j : D_{j+2}\to D_{j+2}$ についても次の性質がある．

[性質 4.5] 汎関数 $\tau_j : D_{j+2}\to D_{j+2}$ は連続である．すなわち，次の二つの条件を満たす．

① τ_j は単調である．つまり，任意の $F, F'\in D_{j+2}$ に対し，$F\leqq_F F'$ ならば $\tau_j[F]\leqq_F\tau_j[F']$.

② D_j 上の任意の鎖 $F_0, F_1, \cdots$ に対して，$\tau_j[\lim_{l\to\infty} F_l]=\lim_{l\to\infty}\tau_j[F_l]$ が成り立つ． □

また，次のような定理が成り立つ．

[定理 4.6] (不動点定理)[6],[7] D が次の①～③を満たすとき，連続関数 $\theta : D\to D$ は，$x=\lim_{l\to\infty}\theta^l[\omega]$ で与えられる最小不動点をもつ．

① D はある関係 $\leqq$ において半順序集合をなす．

② D は $\leqq$ に関して最小元 ω をもつ．

③ D の鎖はすべて D に一意な極限をもつ． □

こうした性質 (性質 4.2～5)，および，不動点定理により，次のことが結論される．

[補題 4.7] $\tau_j\ (j=0, \cdots, n-1)$ は不動点をもち，その最小不動点は，$\lim_{l\to\infty}\tau_j^l[\Omega_{j+2}]$ である． □

このように最小不動点として与えられる関数が，どのようなものであるか，次の補題で示す．

[補題 4.8] $\tau_j\ (j=0, \cdots, n-1)$ の最小不動点は，繰返し関係

$$\begin{aligned} &p_{j+1}, p_{i+1}=\lambda x.\ p_i[\mathrm{bp}[x]] \\ &f_1^{(j)}, f_{i+1}^{(j)}=\lambda x_1\cdots x_i. \\ &\quad H[f_i^{(j)}[b_1^{(j)}[x_1]; \cdots; b_{j+1}^{(j)}[x_{j+1}]]; \\ &\qquad x_1; \cdots; x_{j+1}], \end{aligned} \tag{9}$$

によって決定される次の関数 $F^{(j)}$ に等しい．

$$\begin{aligned} &F^{(j)}=\lambda x_1\cdots x_{j+1}y. \\ &\quad [p_{j+1}[y]\to f_1^{(j)}[x_1; \cdots; x_{j+1}]; \\ &\quad p_{j+2}[y]\to f_2^{(j)}[x_1; \cdots; x_{j+1}]; \\ &\quad \cdots \\ &\quad p_{j+m}[y]\to f_m^{(j)}[x_1; \cdots; x_{j+1}]; \\ &\quad \cdots \quad] \end{aligned}$$

但し，

$0\leqq j\leqq n-2$ のとき，$H=(\tau_{j+1}$ の最小不動点)

$j=n-1$ のとき，$H=a$ である． □

(証明) この補題は，j が 0 から $n-1$ の各場合について述べているが，ある j での τ_j の最小不動点は，そこでの H，つまり，τ_{j+1} の最小不動点が定まらない限り求められない．しかし，$i=n-1$ のときの H は a であるので，τ_{n-1} は求められ，後ろから順次定まる．従って，τ_{j+1} の最小不動点が定まっているとして τ_j の最小不動点を考える．

まず，途中まで式(9)の繰返し関係に従う関数 $F_m^{(j)}$ を考える．

$$\begin{aligned} &F_m^{(j)}=\lambda x_1\cdots x_{j+1}y. \\ &\quad [p_{j+1}[y]\to f_1^{(j)}[x_1; \cdots; x_{j+1}]; \\ &\quad p_{j+2}[y]\to f_2^{(j)}[x_1; \cdots; x_{j+1}]; \\ &\quad \cdots \\ &\quad p_{j+m}[y]\to f_m^{(j)}[x_1; \cdots; x_{j+1}]; \\ &\quad t\to\omega] \end{aligned}$$

$F_m^{(j)}$ は，$\{y \mid p_{j+1}[y]\vee p_{j+2}[y]\vee\cdots\vee p_{j+m}[y]\}$ においてのみ，(未定義要素 ω ではない) 値をとる関数である．これを $F^{(j)}$ の第 m 近似という．

すると，この $F_m^{(j)}$ における m を無限大にしたものは，式(9)の繰返し関係によって与えられる関数の言換えになっている．つまり，$F^{(j)}=\lim_{m\to\infty} F_m^{(j)}$ である．これにより，$\lim_{m\to\infty}\tau_j^m[\Omega_{j+2}]=\lim_{m\to\infty} F_m^{(j)}$ を示すことで，τ_j の最小不動点が，繰返し関係で定義される関数 $F^{(j)}$ と等価なものになることを言うことができる．そのために，

$\tau_j^m[\Omega_{j+2}]=F_m^{(j)}$ を mに関する数学的帰納法を用いて示す．

まず，$m=1$ のときは次のように成り立つ．

$$\begin{aligned}&\tau_j^1[\Omega_{j+2}][x_1;\cdots;x_{j+1};y]\\&=[p_{j+1}[y]\to f_1^{(j)}[x_1;\cdots;x_{j+1}];\\&\quad t\to H[\Omega_{j+2}[b_1^{(j)}[x_1];\cdots;\\&\qquad b_{j+1}^{(j)}[x_{j+1}];bp[y]];x_1;\cdots;\\&\qquad x_{j+1};y]]\\&=[p_{j+1}[y]\to f_1^{(j)}[x_1;\cdots;x_{j+1}];\\&\quad t\to\omega]\\&=F_1^{(j)}[x_1;\cdots;x_{j+1};y]\end{aligned}$$

$m=k$ のとき，$\tau_j^k[\Omega_{j+2}]=F_k^{(j)}$ と仮定する（帰納法の仮定）．すると，$m=k+1$ のとき，

$$\begin{aligned}&\tau_j^{k+1}[\Omega_{j+2}][x_1;\cdots;x_{j+1};y]\\&=\tau_j[\tau_j^k[\Omega_{j+2}]][x_1;\cdots;x_{j+1};y]\\&=\tau_j[p_{j+1}[y]\to f_1^{(j)}[x_1;\cdots;x_{j+1}]\\&\qquad\cdots\\&\qquad p_{j+k}[y]\to f_k^{(j)}[x_1;\cdots;x_{j+1}]\\&\qquad t\to\omega]\\&=[p_{j+1}[y]\to f_1^{(j)}[x_1;\cdots;x_{j+1}]\\&\quad t\to\\&\quad H[p_{j+1}[bp[y]]\\&\qquad\to f_1^{(j)}[b_1^{(j)}[x_1];\cdots;b_{j+1}^{(j)}[x_{j+1}]];\\&\qquad\qquad\cdots\\&\qquad p_{j+k}[bp[y]]\to f_k^{(j)}[b_1^{(j)}[x_1];\cdots;\\&\qquad\quad b_{j+1}^{(j)}[x_{j+1}]];\\&\qquad t\to\omega];x_1;\cdots;x_{j+1};y]\\&=[p_{j+1}[y]\to f_1^{(j)}[x_1;\cdots;x_{j+1}]\\&\quad p_{j+1}[bp[y]]\\&\qquad\to H[f_1^{(j)}[b_1^{(j)}[x_1];\cdots;\\&\qquad\quad b_{j+1}^{(j)}[x_{j+1}]];x_1;\cdots;x_{j+1};y]\\&\qquad\cdots\\&\quad p_{j+k}[bp[y]]\\&\qquad\to H[f_1^{(j)}[b_1^{(j)}[x_1];\cdots;\\&\qquad\quad b_{j+1}^{(j)}[x_{j+1}]];x_1;\cdots;x_{j+1};y]\\&\quad t\to\omega]\\&=[p_{j+1}[y]\to f_1^{(j)}[x_1;\cdots;x_{j+1}]\\&\quad p_{j+2}[y]\to f_2^{(j)}[x_1;\cdots;x_{j+1}]\\&\qquad\cdots\\&\quad p_{j+k+1}[y]\to f_{k+1}^{(j)}[x_1;\cdots;x_{j+1}]\\&\quad t\to\omega]\\&=F_{k+1}^{(j)}[x_1;\cdots;x_{j+1};y]\end{aligned}$$

となり，$k+1$ でも成り立ち，常に $\tau_j^m[\Omega_{j+2}]=Fm^{(j)}$ が言える．従って，m を極限に動かせば，$\lim_{m\to\infty}\tau_j^m[\Omega_{j+2}]=\lim_{m\to\infty}F_m^{(j)}$ が成り立つ．これにより，$\lim_{m\to\infty}F_m^{(j)}=F^{(j)}$ が τ_j の最小不動点であることがわかる． □

（定理 3.5 の証明） 補題 4.7，4.8 により，τ_0 は不動点をもち，その最小不動点 F は，繰返し関係

$$\begin{aligned}&f_1,f_{i+1}=\lambda x.\,f_i^{(1)}[f_i[b_1[x]];x]\\&p_1,p_{i+1}=\lambda x.\,p_i[bp[x]]\end{aligned}\qquad(10)$$

により，

$$\begin{aligned}F=\lambda x_1y.\,&[p_1[y]\to f_1[x_1];\\&\quad p_2[y]\to f_2[x_1];\\&\qquad\cdots\\&\quad p_m[y]\to f_m[x_1];\\&\qquad\cdots]\end{aligned}$$

と書かれる関数 F に等しい．故に，$G=\lambda x.F[x;x]$ とすれば，明らかに，これは式(10)で定義される関数である． □

5. むすび

本論文では，Summers のアルゴリズムにおける例間の規則性を発見するためのマッチングの手続きを，反復的に適用できる高階の差分手続きに拡張する方法について述べた．また，本方法は直線的に最終的な結果まで進むことができる．この特徴は，Summers のアルゴリズムが本来もっていた優れた特徴である．この高階の差分を用いることで，探索を必要とする変数付加の手法のいくつかを避けることができる．同じく Summers のアルゴリズムの特徴として，アドホックな方法を用いないという意味で一般性に優れている点についても，本方法は継承している．

しかし，与えた入出力例から関数フラグメントへの変換，また，二つの関数フラグメントから，差分関数フラグメントを求める過程には，多少の選択の余地があり，現在はその中の一つに固定してある．従って，これらの選択の違いが，結果に影響するかどうかの考察は，今後に残された課題である．また，このような拡張により，どの程度能力が上がったかの評価も今後に残されている．

このアルゴリズムは，LISP の S 式上の関数に対するものであるが，これがこの領域でうまくいっているのは，次の二つの理由によると考えられる．一つは，S 式を操作する有限で完備なオペレータ car，cdr，cons，が存在するため，入力-出力の変換式（関数フラグメント）をこれらを使って得ることができることである．も

う一つは，入力であるS式が束を構成しており，順序づけることができるということである．このような性質を満たせば，他の領域にも応用が可能であると考えられる．

文　献

(1) D. Angluin and C. H. Smith: "Inductive inference: theory and methods", ACM Comput. Surv., **15**, 3, pp. 237-269 (Sept. 1983).

(2) 仁木和久，石崎　俊："概念の帰納的学習"，人工知能学会誌，**3**, 6, pp. 695-703 (昭63-11).

(3) R. S. Michalski, J. G. Carbonell, T. M. Mitchell: "Machine Learning An Artificial Intelligence Approach vol. 2", Morgan Kuafman (1986).

(4) E. Y. Shapiro: "Algorithmic Program Debugging", MIT Press (1982).

(5) P. D. Summers: "A methodology for LISP program construction from examples", J. ACM, **24**, 1, pp. 161-165 (Jan. 1977).

(6) R. Bird: "PROGRAMS AND MACHINES An Introduction to the Theory of Computation", John Wiley & Sons Ltd. (1976).

(7) S. C. Kleene: "Introduction to Methematics", Van Nostrand, Princeton, N. J. (1952).

(昭和63年11月14日受付，平成元年4月13日再受付)

犬塚　信博

昭62名工大・工・情報卒．平成元年同大大学院博士前期課程電気情報工学専攻了．現在，同大学院博士後期課程電気情報工学専攻在籍．人工知能，特に帰納的推論に関する研究に従事．

高橋　健一

昭52名工大・工・情報卒．昭54同大大学院修士課程了．同年同大・工・助手．平成元年同大電気情報工学科助教授，現在に至る．この間，画像情報処理および画像通信の研究に従事．工博．IEEE，情報処理学会各会員．

石井　直宏

昭38東北大・工・電気卒．昭43同大大学院博士課程電気および通信工学専攻了．同年同大・医・助手．昭50名工大・工・助教授を経て，現在，同大・電気情報工学科教授．この間，しきい値論理，医用情報処理，および非線形処理の研究に従事．工博．