ICOT Technical Memorandum: TM-0631

TM-0631

述語生成に関する一考察

石坂裕毅

November, 1988

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

述語生成に関する一考察

石坂裕毅

昭和63年11月

1 はじめに

例による論理プログラムの自動合成において，もっとも本質的でかつもっとも難しい問題は，合成の対象となる述語以外の補助的述語をいかにしてシステムが獲得するかという問題である．たとえば，リスト上の2項関係 $reverse(List1, List2)$ ("$List2$ は $List1$ を反転したリストである" という関係を表す) に対する以下のような標準的プログラムを考える．

$$reverse([X|Y], L) \leftarrow reverse(Y, Z), concat(X, Z, L).$$
$$reverse([], []).$$
$$concat(X, [Y|L1], [Y|L2]) \leftarrow concat(X, L1, L2).$$
$$concat(X, [], [X]).$$

この例では，$concat(\_,\_,\_)$ が補助的述語として使われている．もちろん，この他にも $appcnd(\_,\_,\_)$ を使ったものや，3引数の $reverse$ を使ったもの等いくつか考えられる．重要な点は，そのいずれもが $reverse(\_,\_)$ 以外の補助的述語を利用していることであり，$reverse(\_,\_)$ に関する正負の例だけから，$concat(\_,\_,\_)$ のような補助的述語を発見したり，そのような述語の定義を獲得することが一般には困難であるということである．

その困難さの原因の一つには，与えられる情報が $reverse(\_,\_)$ に関する正負の例だけであるという問題設定自体にもある．しかし，目標の述語 (概念) を効果的に定義するために，それまでに存在しなかったまったく新しい述語を生成するということは，人間が行なう発想や帰納や類推といった高次推論の本質的な部分に関わる極めて難しい問題であり，たとえ，それ以外の付加的情報が与えられたとしても，そう簡単に解決できる問題ではないだろう．

本稿では，この難問に対する最近のいくつかの試みを基に，この問題に対処する際に考慮されるべき問題点を明確にし，それに対する一つのアプローチを提案する．

2 問題設定

ここでは，本稿での考察の対象となる問題設定を，Shapiro のモデル推論問題 [6, 7] (論理プログラムを対象としたものに限定して考える) を基に明確にしておく．

以下では，すべての一階言語[1] L は関数記号の集合と変数記号の集合に関しては共通であるとする．L の述語記号の集合を $\mathrm{Pred}(L)$ で表す．任意の一階言語 L に対し，$\mathrm{Pred}(L) \subseteq \mathrm{Pred}(L')$ なる L' を L の**拡張言語**といい，$L \subseteq L'$ で表す．一階言語 L 上の Herbrand 基底 (Herbrand base)，すなわち，L 上のすべてのグランドアトムの集合を B_L で表し，B_L の任意の部分集合を L 上のモデルと呼ぶ．

L 上のモデル M に関するオラクルとは，B_L の任意の要素 α に対して，α が M の要素ならば "True" を返し，それ例外の場合には，"False" を返すような装置である．L 上のモデルに関するオラクルは B_L の要素に関してだけ答えられることに注意されたい (B_L の要素以外のアトムに関してはその真理値を

[1] 一般的には，整式 (well formed fomula) の集合を指す場合もあるが，ここでは単に述語記号の集合と関数記号の集合と変数記号の集合の3つ組のことを指す (定数記号は0引数関数記号とみなす)．

知らないものとする). L 上のモデル M に関する事実とは，B_L の要素 α と α を M のオラクルに与えたときの出力 V との対 $\langle \alpha, V \rangle$ である．$\langle \alpha, \mathrm{True} \rangle$ なる α を**正事実**と呼び，$\langle \alpha, \mathrm{False} \rangle$ なる α を**負事実**と呼ぶ．L 上のモデル M の**枚挙**とは，B_L の任意の要素 α に対し，ある $i \geq 1$ が存在して $F_i = \langle \alpha, V \rangle$ を満たすような M に関する事実の列 $F_1, F_2, \ldots$ である．L 上のモデル M に関するオラクルは，B_L の要素を枚挙する装置を組み込むことによって M の枚挙を与えることができる．

L 上の論理プログラム (具体的には純 Prolog プログラム) を単に**プログラム**と呼び，プログラム P の最小 Herbrand モデルを $M(P)$ で表す．述語名，述語記号，述語 を以下のように区別して用いることにする (関数名，関数記号，関数についても同様)．

述語名:	単なる文字列	e.g.	$reverse, concat$, etc.
述語記号:	述語名＋引数の情報	e.g.	$reverse(\_,\_), concat(\_,\_)$, etc.
述語:	述語記号＋その述語	e.g.	$reverse(\_,\_) \Longleftrightarrow \{reverse([],[]),$
	記号に関するモデル		$reverse([a],[a]), reverse([a,b],[b,a]) \ldots\}$

L' を一階言語 L の任意の拡張言語とする．L' 上の任意のプログラム P に対し，$M(P)$ の要素を B_L の要素に制限したものを $M(P)_L$ で表す．

以上の定義を基に Shapiro のモデル推論問題は以下のように表現できる．

> 有限個の述語記号と有限個の関数記号をもつ一階言語 L，および L 上のモデル M に関するオラクルが与えられたとき，$M(P) = M$ なる L 上のプログラム P を見つける．

この問題を解くモデル推論アルゴリズムの枠組を図1 に示す．

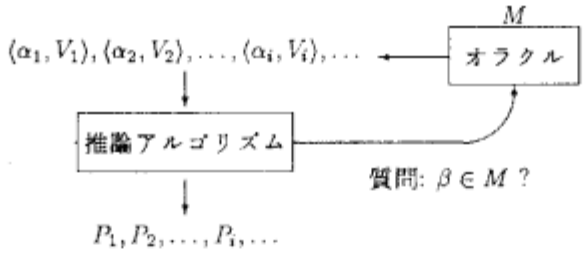

図 1: モデル推論アルゴリズムの枠組

アルゴリズムは，オラクルから与えられるモデル M の枚挙に対し，各時点 i において，i 番目の事実 $\langle \alpha_i, V_i \rangle$ を読み込む．その後 B_L のある要素 β の M における真理値に関する有限回の質問を行った後，その時点での仮説 P_i を出力する．一般にモデルの枚挙は無限の列であるから，この過程は無限に続くことになる．このとき，出力の無限列 $P_1, P_2, \ldots$ が，あるプログラム P に収束して[2]，$M(P) = M$ を満たすとき，アルゴリズムは M を**極限において同定する**という．

本稿では，Shapiro のモデル推論問題を次のように一般化したものについて考える．

> 有限個の述語記号と有限個の関数記号をもつ一階言語 L，および L 上のモデル M に関するオラクルが与えられたとき，$M(P)_L = M$ なる L の拡張言語 L' 上のプログラム P を見つける．

たとえば，L として唯一つの述語記号 $reverse(\_,\_)$ と 2 引数関数記号 $.(\_,\_)$[3] および [] と有限個の 0 引数関数記号をもつような一階言語を考える．M として $\{reverse(t_1,t_2) \in B_L | t_2$ は t_1 を反転したリストである.$\}$ を考える．このとき，M に関するオラクルを用いて，すなわち，$reverse(\_,\_)$ に関する真

[2] ある時点 n 以降のすべての出力 P_i $(n \leq i)$ に対して，$P_i = P$ となること．

[3] 通常の DEC10 Prolog の記法に従い $.(X,Y)$ を $[X|Y]$ で表す．

理知の情報だけから前節に挙げたようなプログラムを見つけるのである[4]．本稿では，このような問題を拡張モデル推論問題 と呼ぶ．

3 拡張モデル推論における問題点

拡張モデル推論問題の定義から明らかなように，それを解くアルゴリズムにとってもっとも厄介な問題は，$\mathrm{Pred}(L') - \mathrm{Pred}(L)$ 中の述語記号としてどのようなものを生成するのか，さらにそれらを P 中でどのように定義するのか，すなわち，述語としてどのようなものを生成するのかということである．本節では，この問題に対する最近の試み [1, 4] について概観し，この問題に対処する際のポイントについて指摘する．

Muggleton と Buntine は， Inverting Resolution と呼ばれる Resolution のちょうど逆の操作を行う手法を提案し，それに基づいたシステム CIGOL を試作している． Inverting Resolution は TRUNCATION, ABSORPTION, INTRA-CONSTRUCTION と呼ばれる 3 つのオペレータによって実行される． TRUNCATION オペレータは， Plotkin の最小汎化 (least generalization) を利用して正事実の集合から単一節 (unit clause) を生成する． ABSORPTION オペレータは，既知の述語記号だけから非単一節を生成する． INTRA-CONSTRUCTION オペレータによって新たな述語記号の生成が行なわれる． CIGOL では，オラクルとして前節で定義したものよりはるかに強力なものが仮定される．たとえば，一般の節

$$reverse([X|Y], L) \leftarrow reverse(Y, Z), concat(X, Z, L).$$

が真であるか否かを判定できなければならない．さらに， CIGOL は述語記号の生成は行なうが，述語の生成までは行なわない．すなわち，生成した述語記号に対する意味 (解釈) の付与はユーザ (オラクル) に任される．

ここで，彼等が論文 [3] の中で与えている例をもとに，述語記号生成段階で生じる問題を指摘する．図2は $arch(Arch)$ ($Arch$ は 3 つ組 $(Column, beat, Column)$ であり，$Column$ は $block$ または $brick$ からなるリストである.) の CIGOL による学習過程の 1 場面である．最初の 3 つの正事実は予めファイル "arch" の中に

```
| ?- cigol.
|- [-arch].
|- show_clauses.
    arch(([],beam,[])).
    arch(([block],beam,[block])).
    arch(([brick],beam,[brick])).
|- arch(([block,brick],beam,[block,brick])).
 TRUNCATION (-71)
   Is arch((A,beam,A)) always true? n.
 TRUNCATION (-65)
   Is arch(([A|B],beam,[A|B])) always true? n.
 TRUNCATION (-53)
   Is arch(([block|A],beam,[block|A])) always true? n.
 INTRA-CONSTRUCTION (-32)
    arch((A,beam,A)):-p110(A).
    p110([]).
    p110([block]).
    p110([block,brick]).
    p110([brick]).
  What shall I call p110? column.
 ABSORPTION (2)
    New clauses:[(column([block|A]):-column(A))]
    cover new facts: [column([block,block]),column([block,block,brick]),...]
  Are new clauses always true? y.
```

図 2: Arch の学習過程 (Muggleton and Buntine [3])

[4] この場合，$\mathrm{Pred}(L') = \mathrm{Pred}(L) \cup \{concat(\_,\_)\}$

登録しておいたものをロードしたものである．次に新たな正事実 $arch(([block, block], beam, [block, brick]))$ が与えられている．まず，これら4つの正事実に対して TRUNCATION オペレータが適用されている．3つの単一節の候補が生成されているが，いずれも単一節としては不適切である．次に INTRA-CONSTRUCTION オペレータが適用され新たな述語記号 p110(_) が生成されている．CIGOL はこの新しい述語記号の名前をユーザに指定してもらい，引続きその述語に対する定義の作成を行なう．

さて，述語生成における問題として次の2つが考えられる．

1. どの時点で新たな述語を生成するのか？

2. どのような意味をもつ述語を生成するのか？

Muggleton 等は2の問題をオラクルの能力によって回避しているが，これが最も本質的な部分である．2と関連する問題として何引数の述語記号を生成するのかということも問題になる．1の問題は推論アルゴリズムの収束性に関係する．たとえば，図2では，$arch(\_)$ を定義するのに新たな述語記号 $p110(\_)$ を導入しているが，次のようなプログラムを考えると述語記号は $arch(\_)$ だけで十分である．

$$arch(([block|X], beam, [block|X])) \leftarrow arch((X, beam, X)).$$
$$arch(([brick|X], beam, [brick|X])) \leftarrow arch((X, beam, X)).$$
$$arch(([], beam, [])).$$

推論アルゴリズムの収束性を考慮するならば，できるだけ既知の述語記号あるいは述語のみでプログラムを記述してしまうような手法が望ましい．Banerji も彼の論文 [1] の中で述べているように，新たな述語を生成する枠組では，アルゴリズムの収束性の保証を与えることが難しくなる．

Banerji は，[1] の中で新述語の生成を行なう手続き DREAM の1つの利用法を示している．そこでは，新述語の生成は仮説の簡略化のために用いられる．たとえば，仮説の中に以下のような2つの節が存在していると仮定する．

$$p \leftarrow A, D. \tag{1}$$
$$p \leftarrow B, D. \tag{2}$$

ただし，D は2つの節の本体に共通に出現しているアトムの集合であり，A，B は各々の本体から D を除いた残りのアトムの集合である．このとき，この2つの節を仮説から取り去り，代わりに次の3つの節を導入する．

$$p \leftarrow new(t_1, \ldots, t_n), D.$$
$$new(X_1, \ldots, X_n) \leftarrow A'.$$
$$new(X_1, \ldots, X_n) \leftarrow B'.$$

ただし，$t_1, \ldots, t_n$ は A，B 中に出現しているすべての項であり，$new(\_, \ldots, \_)$ は新たな述語記号であり，A', B' は A，B における項 t_i の出現を変数 X_i で置き換えたものである．

明かに，この形での述語生成においては，先に挙げたような問題は起こらない．すなわち，述語生成のタイミングは，プログラム中に (1), (2) を満たす節が存在するときであり，その述語記号の引数の個数も述語としての意味も A，B 中の既知の述語によって決定される．しかし，これでは，述語生成というよりもむしろ単なる述語変換であり，我々の動機にとって満足できるものではない．この外にも，Banerji 自身がこの方法におけるいくつかの問題点を指摘している．

4 一つのアプローチ

推論対象のモデルが，構文的に十分制限されたプログラムのクラスで特徴付けられるような場合には，前節で挙げたような問題に対しても容易に対処することができよう．ここでは，各問題点に対処するための構文上の制限について考察してみる．

4.1 生成時点について

まず，新たな述語が生成されなければならない状況について整理しておく．当然のことながら，新たな述語が生成されるからには，既知の事実と現在の仮説 P の間に矛盾が生じているわけである．矛盾の仕方は大きく分けて2通りの場合，すなわち，仮説が強すぎる場合[5]と，仮説が弱すぎる場合[6]がある．仮説が強すぎる場合には，モデル M [7] において偽であるような P 中の節 C が単に取り除かれるだけであるから，新たな述語の導入は必要ない．仮説が弱すぎる場合には，P 中のどの節によっても M において被覆されない[8] M の要素 α が存在する [7]．この場合，M において α を被覆するような新たな節 C を P に付加することによって仮説の修正を行なう．新たな述語記号が生成されるのは，既知の述語だけから生成可能な節の中に C として適切なものが存在しない場合である．

したがって，候補節の探索におけるある時点で，“すべての可能な節は調べ尽くした”ということをシステムが判定できなければならない．すなわち，任意の正事実 α に対して，α を被覆する可能性のある節は有限個である必要がある．このための構文上の制限としては，いろいろなものが考えられるが，少なくとも，本体中に出現するアトムの個数や，それらの引数に現れる項の深さに関しては，予め上限が与えられなければならないだろう．また，推論アルゴリズムの効率を考えるならば，その上限は極めて小さなものに制限されるべきであろう．

4.2 生成される述語の意味

上で述べたように，新たな述語は，ある既知の述語 p に関する正事実 α に対し，α を被覆するような節の本体中の1つのアトムとして生成される．新たな述語の意味は，それが導入される節に現れる他の述語の意味に依存する．このことを逆に利用することによって，その述語の意味を決定するような方法が考えられる．たとえば，以下のような節によって導入された述語 $new(\_,\_,\_)$ の場合，

$$reverse([X|Y],L) \leftarrow reverse(Y,Z), new(X,Z,L).$$

$reverse(\_,\_,\_)$ に関するモデル $M = \{reverse(t_1,t_2) \in B_L \mid t_2$ は t_1 を反転したリストである $\}$ によって，$new(\_,\_,\_)$ のモデルを以下のように定義できる．

$$\{new(t_1,t_2,t_3) \mid reverse([t_1|t_4],t_3) \in M \text{ かつ } reverse(t_4,t_2) \in M\}.$$

すなわち，$new(\_,\_,\_)$ のすべての引数に，節中の他の場所に現れている変数を割り当てることによって，他の述語の意味と関連付けられた意味を割り当てるのである．

もちろん，これを行なうためには，以下のことを決定する必要がある．

1. 頭部および本体中の他のアトム (の形)．
2. new の引数の個数．
3. それぞれの引数に出現する項 (の形)．

一般的なプログラムを対象とする限り，これらを決定することは困難であるように思われる．しかし，逆に考えると，これらの問題が決定できるようなプログラムによって特徴付けられるようなモデルのクラスに対する拡張モデル推論問題は効率的に解けることが期待される．次節では，そのようなプログラムのクラスの例をいくつか紹介する．

---

[5] ある負事実 α に対し，$P \vdash \alpha$ である場合．

[6] ある正事実 α に対し，$P \not\vdash \alpha$ である場合．

[7] 既に新たな述語が導入されている場合には，その述語に関するモデルも含めたものを考える．

[8] アトム α がモデル M において節 $A \leftarrow B_1,\ldots,B_n$ によって被覆されるとは，ある代入 θ が存在して $A\theta = \alpha, B_i\theta \in M (1 \le i \le n)$ を満たすことをいう．

5 構文的制限の例

5.1 DRLP

DRLP (Deterministic Regular Logic Program) は，決定性有限オートマトン (deterministic finite state automaton) の論理プログラムによる自然な表現である [2]．したがって，DRLP で特徴付けられるモデルのクラスは正則言語のクラスと一致する．DRLP は次の 2 種類の節だけから構成され，さらに，各 i と各 a に対して，$q_i([a|X])$ を頭部とする節は高々 1 つしか存在しない．

$$q_i([a|X]) \leftarrow q_j(X).$$
$$q_i([]).$$

このように DRLP は構文的に極端に制限されているので，前節で挙げたようなことはまったく問題にならない．$q_i([a|w])$ を被覆するために導入された $q_j(\_)$ の意味は以下のように定義できる．

$$\{q_j(w) \mid q_i([a|w]) \in M\}$$

5.2 LMLP

LMLP (Linear Monadic Logic Program) は，決定性木オートマトン (deterministic tree automaton) と等価な論理プログラムである．LMLP は次の 2 種類の節から構成される．

$$q_i(f(X_1, \ldots, X_n)) \leftarrow q_{j_1}(X_1), \ldots, q_{j_n}(X_n).$$
$$q_i(a).$$

すなわち，DRLP の 1 つの自然な一般化になっている．Sakakibara [5] は，任意の LMLP P に対し，P 中の 1 つの述語 $q_0(\_)$ によって特徴つけられるモデルに関する拡張モデル推論問題が効率的に解けることを示している．

LMLP の場合も DRLP の場合と同様に，前節で挙げた 3 つの問題は，LMLP の構文上の制限によって決定されている．一方，生成される述語の意味に関しては，本体中のアトムが変数を共有しないので，本体中の他の述語の意味とは独立に，DRLP と同様な方法で，その述語が導入された節の頭部の述語の意味だけから決定することができるように思われる．ところが，LMLP の場合には DRLP との場合と異なり，頭部の構造による節の一意性が仮定されない．したがって，次のような 2 つの節がプログラム中に存在する場合もある．

$$q_1(x \vee y) \leftarrow q_1(x), q_2(y).$$
$$q_1(x \vee y) \leftarrow q_1(x), q_3(y).$$

このような場合には，DRLP の場合のような意味の決め方では，$q_2(\_)$ と $q_3(\_)$ の意味の区別ができない．すなわち，OR 並列が本質的に必要となるようなプログラムのクラスに対しては，前節で述べたような単純な意味の与え方ではうまくいかないかもしれない．

5.3 SDG

LMLP は，本体中のアトムが変数を共有しないが，頭部の構造による節の一意性がないようなプログラムのクラスの例である．それとは逆に，本体中のアトムが変数を共有するが，節の一意性は仮定されるような例として，SDG (Simple Deterministic Grammer) に対応するプログラムのクラスがある．SDG とは以下の条件を満たす 2 標準形[9]文脈自由文法 $G = \langle N, \Sigma, P, S \rangle$ である．

$$A \to a\alpha \text{ と } A \to a\beta \text{ が共に } G \text{ の生成規則であるならば，} \alpha = \beta. \tag{3}$$

[9] 生成規則の右辺に現れる非終端記号が高々 2 個の Greibach 標準形

SDG によって生成される言語を SDL (Simple Deteministic Language) と呼ぶ.

SDG に対応する確定節文法 (Definite Clause Grammer) に基づく論理プログラム (SDDCG と呼ぶ) は，以下の 3 つの節から構成される.

$$A([a|X],Y) \leftarrow B(X,Z), C(Z,Y). \tag{4}$$

$$A([a|X],Y) \leftarrow B(X,Y). \tag{5}$$

$$A([a|X],X). \tag{6}$$

しかも，条件 (3) により，各非終端記号に対応する述語 $A(\_,\_)$ と各終端記号 a に対し，$A([a|X],Y)$ を頭部とする節はプログラム中に高々 1 個しか現れない．したがって，5.2節で述べたような問題は起きない．しかし，SDDCG の場合は，本体中のアトムが変数を共有するために新たな問題が生じる．たとえば，(4) のタイプの節によって，述語 $B(\_,\_)$ と $C(\_,\_)$ が同時に生成された場合，それぞれの意味は，

$$M'(B(\_,\_)) = \{B(t_1,t_2) \mid A([a|t_1],t_3) \in M,\ C(t_3,t_2) \in M'\}$$
$$M'(C(\_,\_)) = \{C(t_1,t_2) \mid A([a|t_3],t_2) \subset M,\ B(t_3,t_1) \in M'\}$$

のように自己再帰的に定義されることになり，帰納的集合にならない [10].

このように，本体中のアトムが変数を共有するような節が必要なプログラムに関しては，1 つの節で 2 つ以上の述語が生成される場合のそれぞれの述語の意味の決定が問題となる．一般的な論理プログラムを対象とした場合，これは克服し難い問題であるが，幸いなことに，SDG に関しては，(3) の拡張として次の性質が成り立つ.

$$A \Rightarrow_G^* w\alpha \text{ かつ } A \Rightarrow_G^* w\beta\ (w \in \Sigma^+, \alpha,\beta \in N^*) \text{ ならば，} \alpha = \beta. \tag{7}$$

したがって，(4) のタイプの節に対しては，$B(\_,\_)$ に関するある正事実，たとえば $B([w|X],X)$[11] が発見できれば，$C(\_,\_)$ の意味を次のように定義できる.

$$M'(C(\_,\_)) = \{C(t_1,t_2) \mid A([a \cdot w|t_1],t_2) \in M\}$$

$B(\_,\_)$ の正事実は，$A(\_,\_)$ に関するある正事実 $A([a \cdot w'|X],X)$ における w' の 1 つの接頭語 w に対し，$B([w|X],X)$ として得ることができる．ただし，w の候補としては $|w'|$ 個のものが存在するから，$C(\_,\_)$ および $B(\_,\_)$ の意味を一意に決定できず，そのバリエーションによる効率の低下が予想される.

Yokomori [8] は，2 節で定義したオラクルよりかなり強力なオラクルを仮定することによって，SDL の学習が多項式時間で行なえることを示している．本稿での設定による SDL に対する拡張モデル推論問題については，別の機会により詳細な議論を行いたい.

5.4 単純な再帰的プログラム

これまでに紹介したプログラムのクラスは，オートマトンや形式文法の直接的な表現になっているようなものである．したがって，プログラム節の構造やそれを構成する述語の引数の個数などに関する一定のパターンが存在している．一般の論理プログラムにおいては，それほど強力なパターンが存在するとは言い難い．しかし，実際の人間が行なうプログラミングも，かなり一定のパターンに基づいている場合が多い．ここでは，Shapiro [6] が与えた項自由変換 (term-free transformation) と呼ばれる節からなるプログラムの具体例をもとに，そのようなパターンの 1 例を考えてみる.

乗算のプログラム.

$$times(0,X,0).$$
$$times(s(X),Y,Z) \leftarrow times(X,Y,W), plus(Y,W,Z).$$
$$plus(0,X,X).$$
$$plus(s(X),Y,s(Z)) \leftarrow plus(X,Y,Z).$$

[10] すなわち，$B(t_1,t_2), C(t_1,t_2)$ に関する真偽の判定ができない

[11] $[a_1a_2\cdots a_n|X]$ で $[a_1,a_2,\ldots,a_n|X]$ を表す.

部分集合関係.

$$subset([], X).$$
$$subset([A|X], Y) \leftarrow subset(X, Y), member(A, Y).$$
$$member(X, [X|L]).$$
$$member(X, [Y|L]) \leftarrow member(X, L).$$

挿入法による整列化プログラム.

$$sort([], []).$$
$$sort([A|X], Y) \leftarrow sort(X, Z), insert(A, Z, Y).$$
$$insert(A, [], [A]).$$
$$insert(A, [B|X], [A, B|X]) \leftarrow A < B.$$
$$insert(A, [B|X], [B|Y]) \leftarrow B \leq A, insert(A, X, Y).$$
$$X < X.$$
$$X \leq s(Y) \leftarrow X \leq Y.$$

以上の他に，冒頭で挙げたリストの反転プログラム等もその例である．いずれも単純な自己再帰型のプログラムである．これらのプログラムには以下のような特徴が見られる．

1. 本体中の補助述語が高々1個である．
2. 本体中のアトムの引数には変数しか現れない．
3. 本体中に自由変数[12]がない．
4. それぞれの頭部が共通のインスタンスをもたない．

これらの特徴は，先に指摘したいくつかの問題点に対して都合のよい性質である．1によって，5.3節で指摘したような，1つの節で複数の新述語が生成される場合を考慮しなくて済む．4によって，5.2節で述べたような問題がなくなる．2および3によって，4.2節で挙げた2,3に対する制限を与えることができる．さらに，新述語の引数の個数を限定する上で重要であると思われる特徴として，次の性質が成り立っている．

5. 再帰的に処理される項に対応する変数が補助述語中に現れない．

たとえば，$times(s(X), Y, Z)$ の X が $plus(Y, W, Z)$ には現れていない．

これらの構文的特徴は，前節までに指摘した問題点を回避するのに十分有効なものであると考えられる．この他にも，ShapiroのMIS [7] で採られたように，各述語の引数を入力引数と出力引数に分割することも重要である．入力引数と出力引数に関する情報を用いることによって，頭部のアトムの形(それぞれの引数に出現する項の形)や，本体中における変数の共有の仕方などをある程度限定することができる．そのような情報は，オラクルによって与えられる述語に関するものが分かれば，その他のシステムによって生成される述語に関しても自動的に決定できる．たとえば，$reverse(\_,\_)$ の第1引数が入力で第2引数が出力であるとする．節

$$reverse([X|Y], L) \leftarrow reverse(Y, Z), new(X, Z, L).$$

によって生成された述語 $new(\_,\_,\_)$ の場合，頭部の入力である X と本体中の $reverse(\_,\_)$ の出力である Z が出現している部分を入力引数とし，頭部の出力である L が出現している部分を出力引数とすればよい．述語を手続きと見なし，その入出力に関する情報を与えられるようにオラクルの能力を拡張することは，それほど無理な拡張ではないだろう．

[12] ここでいう自由変数とは，一般の1階述語論理における自由変数ではなく，本体中のある1ヶ所だけに出現しているような変数のことである

6 おわりに

以上，拡張モデル推論問題を解く際の述語生成に関する問題点と，それに対する最も単純なアプローチの1つについて考察を行なった．5節で紹介したような，構文的に極端に制限されたプログラムによって表現できるモデルのクラスは，極めて限られたものである．しかし，一方では，ある領域に固有の表現方法が，本稿で指摘したような問題点をクリアできるようなものであるならば，その領域における帰納的知識獲得が効果的に行なえる期待がもてる．また，5.4節で述べたように，実際，我々人間が行なうプログラミングにおいても，いくつかのパターンが存在していると考えられる．そのようなパターンの中から，5.4節で与えたような極く単純なものだけ抽出し，組み合わせることによって，より複雑なプログラムの合成も可能となるであろう．

述語生成に対する別の興味深いアプローチとして，既存のプログラムとの類似性を利用するような手法が考えられる．その場合には，単なる事実だけではなく既存のプログラムという情報が与えられることになる．人間が行なうプログラミングを考えるならば，このアプローチは極めて自然であり，効果が期待できるものである．

述語生成の問題は，人間が行なう知的情報処理の本質的な部分に関わる難問であり，今後の研究が期待される重要なテーマであると考える．

謝辞

日頃，御指導，御討論下さる古川康一ICOT次長ならびに長谷川隆三ICOT第1研究室長に感謝致します．また，セミナー等で議論して頂いたICOT研究員諸氏に感謝致します．

参考文献

[1] R. B. Banerji. Learning theories in a subset of a polyadic logic. In *Proc. Computational Lerning Theory '88*, pages 281–295, 1988.

[2] H. Ishizaka. Inductive inference of regular languages based on model inference. In *Proc. Logic Programming Conference '87*, LNCS 315, pages 178–184. Springer, 1988.

[3] S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. To appear in Proc. Machine Learning '88, 1988.

[4] S. Muggleton and W. Buntine. Towards constructive induction in first-order predicate calculus. TIRM 88-031, The Turing Institute, 1988.

[5] Y. Sakakibara. Inductive inference of logic programs bsed on algebraic semantics. Research Report 79, IIAS-SIS, FUJITSU LIMITED, 1987.

[6] E. Y. Shapiro. Inductive inference of theories from facts. Technical report, Yale University Computer Science Deprt., 1981.

[7] E. Y. Shapiro. *Algorithmic program debugging*. PhD thesis, Yale University Computer Science Dept., 1982. Published by MIT Press, 1983.

[8] T. Yokomori. Learning simple languages in polynomial time. In *Proc. of SIG-FAI*, pages 21–30. Japanese Society for Artificial Intelligence, June 1988.