記　号　処　理 78－4
（１９９５．３．２４）

論理型プログラムにおける同値関係の分析

小谷善行

東京農工大学工学部電子情報工学科情報工学講座

論理的なプログラム表現において、記述の間に同値関係を定義することによりプログラム代数が構成できることを示す。われわれは、述語の引数がなく、述語をメタ述語という演算子で結合することにより表されるひとつの論理型プログラム表現 LONG を提案した。これを用いてプログラム代数を構成する。

同値関係として、強・弱二つの同値性を定めた。前者は、非決定的に得られる計算結果の順序を含めた式の等価性を意味し、後者は順序を含めない等価性を意味する。まず、メタ述語で結び付けられた表現の間での等価性を定義から証明することにより、公理的規則群を導き出し、メタ述語の代数的性質を明らかにした。次に、これら規則を式にほどこして式を変形する手順で二つの式の等価性を証明することにより、プログラム変換を厳密に行えることを実例で示した。実例としては探索アルゴリズムを用い、非決定的記述と手続き的記述が強い等価性を持つこと等を証明した。本表現の計算モデルとしての有用性を明らかにするとともに論理的、非決定的プログラム言語における代数的手法の一方向を与えた。

Analysis of Equivalence Relations between Logical Programs

Yoshiyuki KOTANI

Department of Computer Science, Tokyo University of Agriculture and Technology
Koganei, Tokyo, Japan
kotani@cc.tuat.ac.jp

It is shown that program algebra can be constructed in logical program description, by defining specific equivalence relations between expressions of the description. We have proposed a logical program description. Two equivalence relations are introduced, strong and weak. The former means the equivalence of expressions in the computation which includes the temporal order to execute non-determinism. The latter allows the disorder.

First, a set of axiomatic rules is derived by proving the equivalence between simple expressions generated by meta-predicates. Program transformation is shown to be constructed by applying the rules to an example, which is a set of search algorithms written in LONG. It is proved that the non-deterministic recursive algorithm of depth-first search is strongly equivalent to the procedural one of it.

This description is useful as a computational model. Also the result gives a perspective of new algebraic methods of logical or non-determinstic programming languages.

1．はじめに

プログラム代数とは、Ｐｒｏｌｏｇのように、述語論理に近い表現でプログラム代数を実現する方向も考えられる。しかし、Ｐｒｏｌｏｇには引数があり、しかも引数には複合項という構造を許している。これが複雑なプログラム変換のネックになっている（もちろん folding/unfolding などの方向も一つの解決方法である）。

非決定性をもたない関数プログラミング言語では、ＦＰが一定の成功をおさめた[3,4]。その手法が成立し、λ計算を用いるのと異なる成果が得られた理由はやはり引数を表に出さない表現を用いたことであった。

われわれは当初、ユーザインターフェースの立場から、自然言語に近い表現として、述語の引数を隠ぺいし、ユーザ親和性をもたらす表現 LONG を設計した。[1] この表現は同時に計算モデルとしての体系性を持つ。また、結果としてプログラム代数が構築できることが予想された。[1]

ここではこの LONG を用いてプログラム代数が構築されることを示す。そのための準備としてまず LONG の仕組みや意味論を厳密に定義することから始める（第２章）。さらに、プログラムを構成する道具であるメタ述語などの数学的定義を与える（第３章）。第４章ではわれわれの提案する２種類の同値関係を示す。この同値関係を用いて、第５章では、これらの同値関係に関する有用な規則（等式）をまとめる。第６章では二つのプログラムの間を、規則を用いて変形することにより、同値性を証明する。ここでは探索に関するプログラム表現を例にしている。最後にこれらプログラム代数の意義をまとめる。

2．論理型プログラム表現 LONG 及び非決定性の記述方式

プログラム表現 LONG は引数を明示的に示さない一つの論理的プログラム記述である。われわれは利用者親和性の高い Prolog プログラム表現から出発し、２，３，自然言語においては引数がほとんど明示されないことを考慮しこの表現を設計した。

一例として、「祖母とは父や母の母のことである」という知識を LONG で表現してみる。これは

祖母とは父や母の母。

となる。個々の人間に関する父や母のデータがあれば、このプログラムで人の祖母を求めることができる。このプログラムに対応する Prolog プログラム

```
祖母（X,Sobo）:-
    （父（X,Y）;母（X,Y））,母（Y,Sobo）.
```

と比較すると理解されるように、 LONG では引数がない。ここで Prolog の引数である、X, Y, Soboに当たるものは明示されない。

LONG の基本的単位は述語である。述語をメタ述語と呼ばれる演算子で記述して、式を構成する。表現形式の核の部分は次のＢＮＦで表される。

```
<式>：：＝
    <述語>｜
    <リスト>｜
    <式><２項メタ述語><式>｜
    <前置単項メタ述語><式>｜
    <式><後置単項メタ述語>
```

LONG の実行は二つのポート（入力ポート、出力ポート）によって理解される。２項メタ述語は、二つの式の結合のされ方を指定するものである。個々のメタ述語の結合は次章に示される。上の例では「や」や「の」が２項メタ述語である（「とは」も２項メタ述語であるが、述語を定義するという副作用をもつ特殊なメタ述語である）。

個々の式は入力を受け取り、非決定的にデータを出力する。その出力を受ける側が失敗すると次の出力をそこに与える。すなわち、非決定性は深さ優先（depth-first)方式で実行される。

3．LONG のメタ述語及び組み込み述語の数学的定義

まず、準備となる概念及びその表記を述べる。

「オブジェクト」

計算の単位となるものをオブジェクトと呼ぶ。オブジェクトをx，y，zなどの記号で示す。

オブジェクトはＬＩＳＰのデータと同様、アトム（文字列定数）及びリストからなる。

リストはオブジェクトの並びを［　］でくくったものである。

「列」

列とは有限個のオブジェクトの並びである。列をα，β，γなどの記号で表す。

長さ0の列をλで示す。オブジェクトは長さ1の列とみなされる。列αの長さ（列中のオブジェクトの数）を$|\alpha|$で示す。また無限列や例外をまとめて表現する記号として⊥を使う。

「連接」

列には連接（・）が定義される。すなわち、二つの列α、βを順につないだものを$\alpha \cdot \beta$で表す。また　$\alpha 1, \cdots, \alpha n$をつないだものを

$$\mathop{\cdot}_{i=1}^{n} \alpha i$$

のように示す。

また列はつねにオブジェクト列として記述できる：

$$\alpha = a1, a2, a3, \cdots, an = \mathop{\cdot}_{i=1}^{n} ai$$

LONG は非決定的に定まるプログラムの出力を、深さ優先に生成する。すなわち一つの出力を得るごとにバックトラックさせることにより、出力列が得られる。列は具体的対象としてはこの出力列を表現する。

ここで、無限列は単一のもの（⊥）としてみなす。実際に生じる現象は単一ではない。

「式」

述語をメタ述語で結び付けた表現を式と呼ぶ。これが実行の単位である。式はp，q，rなどの記号で表す。

「計算」

プログラムの計算は、オブジェクトをプログラム（式）に与えることにより行われ、結果として出力を得る。オブジェクトaをプログラムpに与え、それをバックトラックさせて得られる出力列をa：pで表す。

さらに、列を式に与えた結果を、オブジェクトを与えた結果を連接したものとして定める：

$$\alpha : p = \mathop{\cdot}_{i=1}^{n} (ai : p)$$

ただし$\alpha = a1, \cdots, an$

メタ述語の定義

LONG の主要な組み込みメタ述語に、「の」、「や」、「で」、「と」、「各」、「全」、「総」がある。また主要な述語として「メンバー」、「アペンド」がある。これらを定義する（これ以外にもメタ述語、組み込み述語は存在するが、以下の議論に無関係なので省略する）。

以下にオブジェクトxをメタ述語表現に与えて得られる結果として、メタ述語等を定義する。列に対する結果は上記から明らかである。

定義3．1（メタ述語）　任意の式（プログラム）p，qと組み合わせて以下の等式が成り立つように各メタ述語を定める。

x：（pのq）＝（x：p）：q

x：（pやq）＝（x：p）・（x：q）

x：（pでq）＝（x：p）∩（x：q）

ただし

$$a1, a2, \cdots, am \cap b1, b2, \cdots, bn = \underset{i=1}{\overset{n}{-}} \underset{j=1}{\overset{m}{-}} \delta(ai, bj)$$

また

$a = b$ならば$\delta(a, b) = a$,
$a \neq b$ならば$\delta(a, b) = \lambda$

$$x : (p \text{と} q) = (x : p) \times (x : q)$$

ただし

$$a1, a2, \cdots am \times b1, b2, \cdots, bn = \underset{i=1}{\overset{n}{-}} \underset{j=1}{\overset{m}{-}} [ai, bj]$$

$$x : \text{各} p = \underset{i_1=1}{\overset{n_1}{-}} \cdots \underset{i_k=1}{\overset{n_k}{-}} [y1i_1, \cdots, yki_k]$$

ただしここで $x = [x1, \cdots, xk]$ であり、

$$xj : p = yji_1, \cdots, yji_{nj}$$

$$x : \text{総} p = \underset{i_1=1}{\overset{n1}{-}} \cdots \underset{i_{k-1}=1}{\overset{n_{k-1}}{-}} y i_1, \cdots, i_{k-1}$$

ただしここで $x = [x1, \cdots, xk]$ であり、

$y1 = x1$

$y i_1, \cdots, i_m, i_{m+1} =$
列$[y i1, \cdots, im, xm+1]$: pのi_{m+1}番目の
オブジェクト　　$(m = 1, 2, \cdots)$

$x : \text{全} p = [a1, a2, \cdots, ak]$
ただし$x : p = a1, a2, \cdots, ak$

定義3．2（組み込み述語）　メンバーとアペンドを以下のように定める。

$x : \text{メンバー} = x1, x2, \cdots, xk$
ただし$x = [x1, x2, \cdots, xk]$

$x : \text{アペンド} = [x1, \cdots, xm, y1, \cdots, yn]$
ただし$x =$
$[[x1, \cdots, xm], [y1, \cdots, yn]]$

4．二つの同値関係

以上の表記を用いて二つの LONG の式の間に成り立つ同値関係を定義する。

第一のものは式の非決定的動作を含めて完全に一致することを表すものである。これを「強い等価性」と呼ぶ。

定義4．1（強い等価性）

二つの式p，qの間の関係≡を以下のように定義する。

全てのオブジェクトxに対して

$x : p = x : q$

ならばまたそのときに限り、

$p \equiv q$

定義2．（弱い等価性）

二つの式p，qの間の関係～を以下のように定義する。

「全てのオブジェクトxに対して、
列x：pの要素は列x：qに含まれ、
また列x：qの要素は列x：pに含まれる」

ならばまたそのときに限り、

$p \sim q$

これら関係が同値関係となるのは、定義から容易に示される。すなわちすべてのp、q、rに対して、

p≡p
p≡qならばq≡p
p≡qかつq≡rならばp≡r
p～p
p～qならばq～p
p～qかつq～rならばp～r

これらの関係は以下の処理では煩雑であるので断らず用いる。

ここで概念「決定性」を定めておく。

「決定性」

すべてのオブジェクトxについてx：pが空列かオブジェクトならば、式pを「決定的（deterministic）」であるという。

5．規則集合

まず、同値関係の間の関係を示す。

p≡q　ならば　p～q
(1.1)

p及びqが決定的であり、かつp～qならば、
p≡q
(1.2)

式に対する代入の可能性を与える規則を示す。

f(x)を、xを含む任意の式とすると、
p≡q　ならば　f（p）≡f（q）
(2.1)

f(x)を、xが含まれる場合、「全g(x)」という形を含まないとすると、
p～q　ならば　f（p）～f（q）
(2.2)

このうち(2.1)は、すべてのメタ述語について、

p≡q　ならば
pのr≡qのr，rのp≡rのq，・・・
pやr≡qやr，rやp≡rやq，・・・
各p≡各q，全p≡全q・・・

といった関係を各々示すことで証明される。(2.2)についても同様にできるが、例外として「全」を含む上記の「～」の式は成り立たないのでそのようになる。すなわち、あるxについて
x：p＝a，b　かつ　x：q＝b，a
ならp～qとなりうるが、
x：全p＝［a，b］≠［b，a］＝x：全q
であるので、全p～全qとはならない。

次に結合則を示す。「と」はリストが生成されるので結合則は成り立たない。「の」、「や」、「で」は下のように成り立つ。

（pのq）のr≡pの（qのr）　(3.1)
（pやq）やr≡pや（qやr）　(3.2)
（pでq）でr≡pで（qでr）　(3.3)

交換則としては次の二つが成り立つ。

pやq～qやp　(4.1)
pでq～qでp　(4.2)

次に分配則を示す。「の」、「や」、「で」、「と」の組み合わせの可能性は２４通りあるが、が、そのうち次の１１通りが成り立つ。またさらにそのうち3通りに強い等価性が成り立つ。

（pやq）のr　≡　（pのr）や（qのr）(5.1)
pの（qやr）　～　（pのq）や（pのr）(5.2)

（pやq）でr　≡　（pでr）や（qでr）(5.3)
pで（qやr）　～　（pでq）や（pでr）(5.4)

（pやq）とr　≡　（pとr）や（qとr）(5.5)
pと（qやr）　～　（pとq）や（pとr）(5.6)

（pでq）やr ～ （pやr）で（qやr）(5.7)
pや（qでr） ～ （pやq）で（pやr）(5.8)

（pでq）とr ～ （pとr）で（qとr）(5.9)
pと（qでr）～ （pとq）で（pとr）(5.10)

pの（qとr）～（pのq）と（pのr） (5.11)

このほかに

p～pやp (6.1)
p～pでp (6.2)

がある。

単項メタ述語及び組み込み述語に関しては次のようなさまざまな規則が抽出される。

まず「メンバー」と「アペンド」に関係する規則を示す。

メンバー≡最初や（残のメンバー） (7.1)

これはよく知られたメンバーの再帰的定義になっている。ここで最初と残はLISPのcar, cdrに相当するリスト演算子である。

次に「各」などに関係するものを順に示す。

総アペンド≡全（メンバーのメンバー）)

各pのメンバー ～ メンバーのp (8.1)
全pのメンバー ≡ p (8.2)
各自身 ≡ 自身 if リスト (8.3)
全メンバー ≡ 自身 if リスト (8.4)

全各pの総アペンド ～ 全（メンバーのp）
(8.5)

一例として（8．5）を証明する。このように他の規則から証明できるものと、証明するにはメタ述語の定義にまでさかのぼる必要があるものがある。

証明
全（メンバーのp）
～全（各pのメンバー）
～全各pの（各全メンバーの総アペンド）
≡全各pの（各自身の総アペンド）
≡全各pの（自身の総アペンド）
≡全各pの総アペンド ■

各（pのq） ～ 各pの各q (8.6)
各（pやq） ～ （各pと各q）のアペンド
(8.7)
pでq ≡ pとqの同 (8.8)
各（pでq） ～ 各pと各qの分配の各同(8.9)

各（pとq） ≡ 各pと各qの分配
～全（メンバーの（pとq））
～全（（メンバーのp）
と（メンバーのq））
(8.10)

pとqのアペンド ～
全（（pのメンバー）や（qのメンバー））
(8.11)
全（pのq） ≡ 全pの（各全qの総アペンド）
(8.12)
全（pやq） ≡ （全pと全q）のアペンド
(8.13)

総p ≡ 残の「 [] 」であるなら自身ほかは
[最初と（残の最初）のp | 残の残] の総p
(8.14)

次のようにメタ述語の演算に単位元を考えることができる。ここで「と」には単位元はない。「空」とは常に失敗する述語である。「自身」とは入力と同じものを一度だけ出力に出す述語である。「何」はあらゆるオブジェクトと等しいと判定される変数である（現在の LONG の仕様にはない）。

pの自身 ≡ 自身のp ≡ p
pや空 ≡ 空やp ≡ p
pで何 ≡ 何でp ≡ p

6．等価なプログラム表現の生成

Prolog と同様 LONG は非決定性を縦型に実行する。この LONG の特性をそのまま利用して表現すると、縦型探索の計算は次のように定義される。このプログラムに対して入力として始状態を与えると、可能な状態を順にたどりつつ出力する。なお、この出力について各々ゴールか否かを判定すれば縦型探索が実行されるが、その部分は本質的な意味がないので以下の議論では省略する。

解とは自身や（次の解）。

一方、縦型探索は、手続き的アルゴリズムとしては、一次元リストをスタックとして用いることで表現される。この方法のもとで LONG で表現したものを以下に示す。

解２とは［自身］の解二。
解二とは
　先頭や（先頭の全次と残のアペンドの解二）。

LONG は手続き的表現を記述することを目的として設計されていない。そこで、ここではスタックのデータを式の入力から出力へ引き渡し、決定的アルゴリズムによる末尾再帰（tail recursion）を用いて表現している。すなわち、スタックに未処理状態列を保存してあるとすると、たどる解は、まずスタックの先頭データであり、そのあと先頭の全次状態を残りのスタックの先頭に付けたものに対して再び解を求める。このプログラムを非決定的に効率よく実行するように処理系を設計することは可能であると考えられる。

定理
　解≡解２

証明
　まず方程式

解二≡
　先頭や（先頭の全次と残のアペンドの解二）　①

を解くと

解二≡メンバーの解　②

となることを示す。これを、②を仮定すると①が成り立つことで示す。

①の右辺
≡先頭や
　（先頭の全次と残のアペンドのメンバーの解）
≡先頭や（先頭の全次と残の
　全（メンバーのメンバー）のメンバーの解）
≡先頭や
　（先頭の全次と残のメンバーのメンバーの解）
≡先頭や（先頭の全次や残のメンバーの解）
≡先頭や（（先頭の全次のメンバー）や
　（残のメンバー）の解）
≡先頭や（（先頭の次）や（残のメンバー）の解）
≡先頭や（先頭の次の解）や（残のメンバーの解）
≡先頭の（自身や（次の解））や
　（残のメンバーの解）
≡先頭の解や（残のメンバーの解）
≡先頭や（残のメンバー）の解
≡メンバーの解
≡解二

次に②を用いると、

解２
≡［自身］の解二
≡［自身］のメンバーの解
≡自身の解
≡解

証明終わり。

次に別の話題として、横型探索（広がり優先探索、breadth-first search）を考える。このアルゴリズムは、１次元リストとしてスタックの代わりに、待ち行列（つまりfirst-in first-outバッファ）を用いる。これはリストからのデータの取

り出し端と追加端が異なるものとして次のように表現できる。

解3≡［自身］の解三
解三≡
先頭や（残と（先頭の全次）のアペンドの解三）

さらに、探索の深さごとに生成される状態のリストを作成することによる横型探索アルゴリズムもまた考えられ、次のように表現できる。

解とは［自身］の解二。
解二とはメンバーや（各全次の総アペンドの解二）。
（横型：深さごとのリストによる）

これら二つの横型探索アルゴリズムと、上述の縦型探索アルゴリズムのあいだには、弱い等価性～が成り立つことが予想される。

7．議論及びまとめ

われわれが提起した論理型プログラム LONG において、その計算に関する同値関係を定義することにより、代数的構造があることを示した。これは非決定性プログラム言語の問題として新しい考えであるといえる。

同値関係としては二つの等価性を与えた。強い等価性（≡）は、非決定性を縦型に実行するときの順序を保存するものである。弱い等価性（～）はそれを保存しないものである。

実例として、再帰的縦型探索を表現した形式と、と、非再帰的手続き的縦型探索を表現した形式との間には強い等価性（≡）が成り立つことが代数的に証明された。

参考文献

1）小谷善行、日本語による論理プログラム表現、情報処理学会論文誌、Vol. 33, No. 11, pp. 1296-1305, 1993.

2）小谷善行，変数記述を減らしたPrologの仕様，情報処理学会第２９回全国大会講演論文集，pp. 519-520, 1984.

3） 小谷善行，ＬＯＧＯの「ヒューマン・フレンドリ」性とその日本語 Prolog への応用，pp. 207-212, ヒューマンフレンドリーなシステムシンポジウム報告集，情報処理学会，1986.

4) Backus, J., Can programming be liberated from the von Neuman style? — A functional style and algebra of programs, CACM, Vol. 21, No. 8, pp. 613-641, 1978.

5) Backus, J.:The algebra of functional programs: functional level reasoning, linear equations, and extended definitions, pp. 1-43, Lecture Notes in Computer Science, No. 107, Springer Verlag, 1981.

6）小谷善行，引数のない論理プログラム表現，情報処理学会第５３回記号処理研究会研究報告，pp. 1-8，1989.

7）新田健二，矢野稔裕，瀧口伸雄，小谷善行，西村恕彦，自然言語指向の論理型言語処理系の実現方式，情報処理学会第４４回全国大会講演論文集，vol. 5, pp. 17-18, 1992.

8）戸村哲，ＴＤ Prolog ：項記述可能な Prolog 処理系，Proc. of the Logic Programming Conference'85, pp. 237-245, 1985.

9)戸村哲，項記述単一化に基づく Prolog ，コンピュータソフトウェア Vol. 8, No. 6, pp. 53-65, 1991.

10) Tanaka, Y., Vocabulary-Based Logic Programming, Proc. of Info Japan'90, 1990.

11) Pereira, F. C. N. and D. H. D. Warren, Definite Clause Grammars for Language Analysis--A Survey of the Formalism and a Comparison with Augmented Transition Networks, Artificial Intelligence, vol. 13, pp. 231-278, 1980.

12) Dahl, V. and P. Saint-Dizier, Natural Language Understanding and Logic Programming, Elsevier, 1985.