Metis-AS における代数的仕様の検証手続き

大須賀 昭彦† 坂井 公†† 本位田 真一†

本稿では，代数的仕様を自動検証する手続きの実現について述べる．この手続きは，仕様によって定まる始代数上の定理を項書換え技術によって証明するもので，理論的には等式論理の帰納的定理証明手続きを多ソート代数の場合へ拡張したものとなっている．ここでは，手続きの効率化手法に焦点をあてるとともに，手続きが無限実行される問題の救済についても考察する．

A Verification Procedure for Algebraic Specifications by *Metis-AS*

AKIHIKO OHSUGA,† KÔ SAKAI†† and SHIN'ICHI HON'IDEN†

We describe an implementation of a verification procedure for algebraic specifications. This procedure proves theorems in the initial algebra defined by given specifications. From the theoretical point of view, it can be seen as an adaptation of inductive theorem proving to many-sorted algebra. We focus on the techniques to improve the efficiency of the procedure and consider several remedies to avoid nontermination.

1. はじめに

代数的仕様記述法（algebraic specification）には，形式性，記述性，理解性，最小性といった，仕様記述言語に求められる重要な性質が備わっている．このため，代数的仕様をソフトウェアの開発に適用すると，開発者間の正確なコミュニケーションや，仕様の直接実行による動作の確認，数学的証明を用いた厳密な仕様検証，形式的な方法による実現への変換などの支援が受けられ，ソフトウェアの信頼性や生産性を高めることができる[12]．

これらの支援のなかでも，信頼性向上のための仕様検証の意義は特に大きいと考えられる．しかし，一般のユーザがこの恩恵を受けるためには，できる限り検証が自動化されていなければならない．代数的仕様における種々の検証の中では，仕様によって定まる始代数上の定理，すなわち帰納的定理を証明する技術が重要である．これは，始代数がわれわれの直観的な仕様のモデルに近いことによる．たとえば，自然数の加算の定義を代数的仕様で普通に与えた場合，交換律 $x+y=y+x$ や結合律 $(x+y)+z=x+(y+z)$ は通常の定理ではなく，帰納的定理である．しかし，この種の定理を効果的に自動証明する支援環境は今のところ少ない[3),6)]．

帰納的定理を証明するアプローチとしては，（1）構造帰納法や（2）被覆帰納法[18),19)]，（3）潜在帰納法[2),11)]を適用するものなどが考えられる．これらを自動化の観点から評価すると，（1）は自動化に不向きで，これを実装した BMTP においても証明に多くのヒューリスティクスが必要とされる[4)]．（2）では，被覆集合を求めた後の証明は機械的であるが，この集合の発見について自動化が難しい．（3）は，ほとんどの処理を項書換え技術によって機械的に行え，このなかでは最も自動化に向いている．この結果をふまえ，われわれは代数的仕様によるソフトウェア開発支援環境 *Metis-AS* に，潜在帰納法に基づく拡張手続きの実装を行った．この手続きは，理論的には等式論理の帰納的定理証明手続き[14)]を多ソート代数の場合へ拡張したものとなっている．本稿では，この手続きの実装と効率化について述べ，文献 14) では未解決であった，手続きが無限実行される問題の救済についても考察する．

まず，2章で用語や記法について述べ，手続きで用いる項書換え技術を3章で定義する．基本的な手続きを4章で導入した後，5章では無限実行の救済について，6章では手続きの効率化について考察し，最後に7章で仕様の検証例を見る．

2. 準　備

本章では用語と記法を導入する．ここで説明しない

† (株)東芝 研究開発センター システム・ソフトウェア生産技術研究所
Systems and Software Engineering Laboratory, Research and Development Center, Toshiba Corporation
†† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

事柄については，文献 8), 12), 17) などを参照されたい．

S をソートの有限集合とする．ソートとは，直観的には，ある定まった領域の名前である．各ソート σ には，関数記号の有限集合 F_σ と変数の可算無限集合 V_σ が対応している．関数記号にはランクと呼ばれるソートの列が対応し，F_σ の要素 f のランクが $\sigma_1, \cdots, \sigma_n, \sigma$ $(n \geq 0)$ であることを $f : \sigma_1, \cdots, \sigma_n \rightarrow \sigma$ と書く．これは，直観的には，f がソート $\sigma_1, \cdots, \sigma_n$ の表す領域内の要素を引数にとり，σ の表す領域内の要素を値として返す関数であることを表す．f が引数をとらない定数であるときは，とくに $f : \sigma$ と書く．また，変数 x が V_σ の要素であることを明示するときは，$x : \sigma$ と書く．このとき x は，直観的には σ の表す領域の不特定の要素を表している．$F = \cup_{\sigma \in S} F_\sigma$，$V = \cup_{\sigma \in S} V_\sigma$ とする．$T_\sigma(F, V)$ によって F と V から構成される σ の項の集合を表し，$T(F, V) = \cup_{\sigma \in S} T_\sigma(F, V)$ とする．同様に，$T_\sigma(F)$ によって F のみから構成される σ の項，つまり σ の基礎項の集合を表し，$T(F) = \cup_{\sigma \in S} T_\sigma(F)$ とする☆．

同一ソートの項 l，r を $=$ で結んだ式 $l = r$ を等式と呼び，E を等式の集合とする．等式論理の演繹[12),17)] によって E から $l = r$ が導けるとき，l と r は E 等価であるという．本稿では，文献 14) と同様に，各 σ について F_σ が演算子 (defined symbol) の集合 D_σ と構成子 (constructor) の集合 C_σ に分割されていることを仮定し $(F_\sigma = D_\sigma \cup C_\sigma, D_\sigma \cap C_\sigma = \phi)$，$D$，$C$ をこれまでと同様に定める．演算子を含まない項を構成子項，少なくとも 1 つの演算子を含む項を非構成子項と呼ぶ．等式が矛盾するとは，等式の両辺が異なる構成子項であることをいう．

5 組 (S, D, C, V, E) を代数的仕様と呼び，E を代数的仕様の公理系と呼ぶ．等式 $l = r$ が代数的仕様の帰納的定理であるとは，任意の基礎代入 θ について $l\theta$ と $r\theta$ が E 等価になることをいう．ここで基礎代入とは，項 t について $t\theta \in T(F)$ となる代入 θ をいう．

3. 項書換え技術

$T(F, V)$ 上には順序 $\succ$ が定められているとする．ここで $\succ$ は，$T(F)$ 上の全順序となる単純化順序[5)]で，しかも任意の基礎項 t，u について，t が非構成子項で u が構成子項ならば $t \succ u$ となるものとする．このような順序の構成方法はすでにいくつか知られているが，ここでは以降の例で用いる順序を導入する．

D 上の全順序 $>_D$ と C 上の全順序 $>_C$ が適当に与えられているとする．辞書式部分項順序[16)] (lexicographic subterm ordering) $\succ_{lex}$ は，以下のように再帰的に定義される☆．

(1) 変数 x は，いかなる項 u に対しても $x \succ_{lex} u$ であることはない．

(2) 項 $t \equiv f(t_1, \cdots, t_m)$ が項 u に対して $t \succ_{lex} u$ となるのは，(2-1)，(2-2) のいずれかの場合，かつそれらの場合に限られる．

(2-1) ある t_i $(1 \leq i \leq m)$ について $t_i \succ_{lex} u$ または $t_i \equiv u$ である．

(2-2) $u \equiv g(u_1, \cdots, u_n)$ で，各 u_j $(1 \leq j \leq n)$ について $t \succ_{lex} u_j$ であり，しかも以下のいずれかが成立する．

(2-2-1) $f \in D$ かつ $g \in C$ である．

(2-2-2) $f >_D g$ または $f >_C g$ である．

(2-2-3) $f = g$ で，ある i $(1 \leq i \leq m)$ があって $t_i \succ_{lex} u_i$，かつすべての k $(1 \leq k < i)$ について $t_k \equiv u_k$ である．

ここで $t \equiv u$ は，項 t と u の構文的な一致を表す．

同一ソートの項 l，r を $\Leftrightarrow$ で結んだ式 $l \Leftrightarrow r$ を書換え規則（または規則）と呼び，規則の集合を項書換え系 (term rewriting system) と呼ぶ．R を項書換え系，$l \Leftrightarrow r$ を R の規則，$t[s]$ を項とする☆☆．ここで，$l \Leftrightarrow r$ は $l \Leftrightarrow r$ または $r \Leftrightarrow l$ の略記である．代入 θ があって，$s \equiv l\theta$ かつ $l\theta \succ r\theta$ となるとき，$t[s]$ は $l \Leftrightarrow r$ によって $t[r\theta]$ へ簡約されるといい，このような $t[s]$ を R 可約であるという．$\succ$ が単純化順序であることより，簡約は停止的である．つまり，いかなる項からも無限の簡約列は生成されない．

等式 $g[s] \doteqdot d$ における s の出現位置を p とする．任意の基礎構成子代入 θ について $s\theta$ 全体が R 可約であるとき，p を $g[s] \doteqdot d$ の R 完全な出現と呼び，R 完全な出現をもつ等式を R 完全な等式と呼ぶ．ここで基礎構成子代入とは，変数に基礎構成子項のみを割り当てる代入をいう．このとき条件より，s は変数

☆ 簡単のため，本稿では異なるソートの間に関数記号や変数名の共有はないものと仮定する．これを許した場合には，記号とそのランクを組にするなどして，見かけ上同一の記号を区別すればよい．

☆ この定義は辞書式経路順序[5)] (lexicographic path ordering) を D，C について特殊化したものとなっているが，両者の間に本質的な違いはない．たとえば，辞書式経路順序において，C の要素が D の要素よりも必ず小さくなるように F 上の全順序を構成すれば，定義と同様の順序が得られる．

☆☆ 項 t が部分項 s をもつことを $t[s]$ で表し，この s を u へ置き換えた結果を $t[u]$ で表す．

でないとしてよい．R 完全性の判定は，基礎可約性（ground reducibility）を調べる方法[15]を用いて，有限の R について決定可能である．具体的には，一定の条件を満たす有限の基礎構成子代入 θ について，$s\theta$ の R 可約性を検査することになる．

$g[s]\Leftarrow d$，$l\Leftarrow r$ を2つの規則，s を変数でない部分項とし，$g[s]\Leftarrow d$ における s の出現位置を p とする．代入 θ があって，$s\theta\equiv l\theta$，かつ $g[s]\theta\not\leq d\theta$，$l\theta\not\leq r\theta$ となるとき，等式 $g[r]\theta=d\theta$ を，$g[s]\Leftarrow d$（の出現位置 p）に $l\Leftarrow r$ を重ねた要対と呼ぶ．

通常の項書換え系では規則の向きを $l\Rightarrow r$ のように固定し，これによる簡約が停止的であることを保証するために，各規則について $l\succ r$ となることを示す．しかし，$\succ$ は $T(F,V)$ 上の全順序にはなりえないので，場合によっては停止性を保証できないことがある．これが原因となって手続きが失敗するのを避けるために，われわれは文献2),14)などと同様に，規則を両方向に適用可能な形で定式化している．ただし，$l\Leftarrow r$ について $l\succ r$ が成立するときには，これを従来の $l\Rightarrow r$ なる規則と同様に扱う方が効率的である．順序 $\succ$ は代入に関して安定，つまり $l\succ r$ であれば任意の θ について $l\theta\succ r\theta$ が成立するので，このように扱っても問題は起こらない．以降の例においては，このような規則について断りなく $l\Rightarrow r$ なる記法を用いる．

4. 基本検証手続き

本章では基本的な検証手続きを導入し，次章以降でこれを改良する．まず，検証で扱うことのできる代数的仕様のクラスを明らかにする．

定義1 代数的仕様 (S,D,C,V,E) が入力条件を満たすとは，任意の基礎項 $t\in T(F)$ について，t と E 等価な基礎構成子項 $u\in T(C)$ がただ一つ存在することをいう． ∎

この条件は，仕様に対応する始代数が基礎構成子項によって一意に表現されることを保証する自然なものである．この条件の判定方法については，6.1節で詳しく述べる．以下に基本検証手続きを示す．この手続きは，入力条件を満たす代数的仕様と証明すべき定理の集合を入力とし，証明結果とともに停止する．誤った定理に対しては必ず停止してその不成立を告げるが，定理が正しいときには，停止してその成立を告げる場合と，停止せずにいつまでも実行が続く場合がある．

手続き1（基本検証手続き）

第0段：（初期化） E の初期値を代数的仕様の公理系，G の初期値を証明すべき定理の集合とし，R，S の初期値を空とする．第1段へ進む．

第 i 段：（終了判定） G が空ならば終了．このとき定理は成立．

（等式の選択） E または G から等式を1つ選択し，これを $l=r$ とする．元の等式は削除する．ただし G から選択する等式は，現在の R によって R 完全なものに限る．

（公理の演繹） $l=r$ が E から選択されたものである場合，以下を行う．$l=r$ が自明（$l\equiv r$）ならば $i+1$ 段へ進む．そうでなければ $l\Leftrightarrow r$ を新しい規則として R へ加える．この規則と R の各規則を相互に重ねて得られる要対を E へ追加し，$i+1$ 段へ進む．

（定理の演繹） $l=r$ が G から選択されたものである場合，以下を行う．l，r を $R\cup S$ によって簡約した結果（の任意の1つ）を l'，r' とする．$l'=r'$ が矛盾するなら終了．このとき定理不成立である．$l'=r'$ が自明ならば $i+1$ 段へ進む．どちらでもなければ $l'\Leftrightarrow r'$ を新しい規則として S へ加える．この規則に R の各規則を重ねて得られる要対の中で自明でないものを G へ追加し，$i+1$ 段へ進む． ∎

等式の選択を行う際には，以下に述べる2つの条件を考慮しなければならない．1つは，E または G から公平に等式を選ぶことである．ここで公平とは，一度も選択されずに残されたままの等式が存在しないことをいう．たとえば，等式を構成する関数記号の数が少ないものを常に選ぶようにすると，公平な選択が実現される．もう1つは，G から選択する等式を，その時点での R によって R 完全なものに限ることである．最初に与えた仕様が入力条件を満たしていれば，証明を進めていく過程で G の等式はいつか必ず R 完全となる．これら2つの条件は，文献14)で議論されているように，手続きを完全なものにする上で重要である．

手続き1を用いて簡単な証明を行う．**図1**のように自然数の加算を定義したときに，$0+x=x$ が成立することを示す☆．nat の任意の基礎項は $s^n(0)\,(n\geq 0)$ なる形をした唯一の基礎構成子項と E 等価なので，この仕様は入力条件を満足している．関数記号間の順序を $s>c0$ とし，項の順序に $\succ_{lex}$ を用いると，以下の証明が得られる．

☆ 本稿では，等式に現れる変数はすべて全称束縛されていると考えるので，異なる等式の間に変数の共有はないと仮定する．共有がある場合には，暗黙のうちに新しい変数で置き換えられているものとする．

S = {nat}
D = {+:nat, nat→nat}
C = {0:nat, s:nat→nat}
V = {x:nat, y:nat, ⋯}
E = {$x+0=0$, $x+s(y)=s(x+y)$}

図 1 自然数の加算の代数的仕様例
Fig. 1 Algebraic specification example of natural number addition.

第 0 段：$E=\{x+0=0, x+s(y)=s(x+y)\}$，$G=\{0+x=x\}$ とし，R，S は空とする．

第 1 段：Eから $x+0=0$ を選び $x+0\Rightarrow 0$ をRへ加える．

第 2 段：Eから $x+s(y)=s(x+y)$ を選び $x+s(y)\Rightarrow s(x+y)$ を R へ加える．

第 3 段：Gから $0+x=x$ を選び $0+x\Rightarrow x$ をSへ加える．この規則にRの各規則を重ねた要対として，$0=0$ と $s(0+x)=s(x)$ が存在するが，前者は自明なので後者のみをGへ加える．ここまでで，Eは空，$R=\{x+0\Rightarrow 0, x+s(y)\Rightarrow s(x+y)\}$，$G=\{s(0+x)=s(x)\}$，$S=\{0+x\Rightarrow x\}$ となる．

第 4 段：Gから $s(0+x)=s(x)$ を選び $R\cup S$によって簡約すると，自明な等式 $s(x)=s(x)$ となる．

第 5 段：Gが空なので終了．この結果，定理は成立． ∎

関数記号数の少ないものから等式を選択した場合，初期状態において定理 $0+x=x$ の記号数が最小となるが，第 2 段が終わるまでこの等式はR完全にならない．つまり，第 2 段が完了して初めて $0+s^n(0)=s^n(0)$ $(n\geq 0)$ がR可約となる．したがって，これ以前にこの定理が選択されることはない．第 4 段における簡約に $0+x\Rightarrow x$，つまり定理を規則化したものが用いられている点に注意されたい．証明を完了させる上で，このような定理による簡約が重要である．

5. 無限実行への対処

手続き 1 は潜在帰納法と同様に，無限の要対を生成し続け停止しないことがある．この問題に対しては，メタ規則を導入する方法[13]や，項の構造を拡張する方法[1]などが提案されているが，いずれも実用的な技術にまでは至っていない．われわれは，実現性を考慮したいくつかの方法によってこの問題に対処している．

5.1 包含処理の導入

無限実行を回避するための簡単な方法に包含処理がある．新たに生成される要対が既存の等式や規則に包含される場合に，その要対を捨てるというものである．この方法は単純だが，無限実行の回避に有効であり，また手続きの効率化にも役立つ．

定義 2 $l=r$ を等式，Eを等式の集合とする．$l=r$ は以下のいずれかのとき，Eに包含される．

(1) $l\equiv r$ である．
(2) E の等式 $g\triangleq d$ と代入 θ があって，$l\equiv g\theta$ かつ $r\equiv d\theta$ である．
(3) $l\equiv f(l_1,\cdots,l_n)$ かつ $r\equiv f(r_1,\cdots,r_n)$ で，各 i $(1\leq i\leq n)$ について $l_i=r_i$ が E に包含される． ∎

定義より，自明な等式は任意の集合に包含される．等式の集合による包含の定義は，規則の集合による包含の定義へ自然に拡張できる．EやRに包含される要対が論理的に冗長であることは明らかである．

5.2 再帰対による無限実行の検出

より効果的に無限実行を回避するために，再帰対を用いた無限実行の検出法を導入する．再帰対の考え方は，Knuth-Bendix 完備化手続きの無限実行を検出するための交差書換え系[9),10)] (crossed rewriting system) に基づいている．

定義 3 $g[s]\triangleleft d$ を規則，この規則における s の出現位置を p，$l_n\triangleleft r_n[t_n]$ $(n\geq 0)$ をある 1 つの規則について変数を置き換えたものの列とする．代入 θ_0 があって以下の条件が満たされるとき，等式 $g[r_0[t_0]]\theta_0=d\theta_0$ を $g[s]\triangleleft d$ (の出現位置 p) に $l_0\triangleleft r_0[t_0]$ を重ねた再帰対と呼ぶ．

(1) $s\theta_0\equiv l_0\theta_0$，かつ
(2) $g[s]\theta_0\not\equiv d\theta_0$，
(3) $l_0\theta_0\not\equiv r_0[t_0]\theta_0$，
(4) 各 i (≥ 0) について $t_i\theta_i\theta_{i+1}\equiv l_{i+1}\theta_{i+1}$ であるような代入 θ_{i+1} が存在する． ∎

定義より，再帰対は $g[s]\triangleleft d$ の出現位置 p に $l_0\triangleleft r_0[t_0]$ を重ねた要対である．再帰対が存在し，さらに各 i (≥ 0) について $l_i\theta_i\not\equiv r_i[t_i]\theta_i$，かつ $g[r_0\cdots r_i[t_i]\cdots]\theta_0\cdots\theta_i\not\equiv\theta_0\cdots\theta_i$ が成立すると，以下の形をした要対が無限に生成される．

$g[r_0[t_0]]\theta_0=d\theta_0$
$g[r_0[r_1[t_1]]]\theta_0\theta_1=d\theta_0\theta_1$
……
$g[r_0\cdots r_n[t_n]\cdots]\theta_0\cdots\theta_n=d\theta_0\cdots\theta_n$

これらを簡約または包含する等式や規則が出現しなければ，手続きは無限に実行される．定義 3 における条件(4)の判定には文献 10) の十分条件が適用できるので，手続きの実行中に再帰対の存在を調べることで，無限実行の可能性を検出できる．ただし，再帰対はすべての無限実行を検出するわけではない．たとえば，

相互再帰（mutual recursion）を用いて定義される公理が無限実行を引き起こす場合などには再帰対が存在しない．しかし，多くの無限実行において再帰対が存在することは，実験的に確認されている．

例1 図1の仕様において交換律 $a+b=b+a$ $(a, b \in V_{nat})$ が成立することを証明しようとすると，ある段において R の規則 $x+s(y)\Rightarrow s(x+y)$ (r1) と S の規則 $a+b\Leftrightarrow b+a$ (s1) が得られる．このとき，(s1) に (r1) を重ねた再帰対として $s(x+y)=s(y)+x$ が存在する．以降の証明を続けると，$s^n(x_n+y_n)=s^n(y_n)+x_n$ なる要対が無限に生成され，実際に手続きは停止しない． ∎

5.3 無限実行を回避する要対の選択

定理を規則化してそこから要対を得る際には，すべての要対を獲得しなくても，R 完全な出現を1つ選び，その位置にのみ R の規則を重ねるだけで十分である[2),7),14)]．この選択をうまく行うと，それまで無限実行されていた証明が止まる場合がある[7)]．しかし，R 完全な出現が複数存在するときに，どの出現を選べばよいかの指針は与えられていない．われわれは，再帰対の獲得を避けて無限実行を回避するために，以下の選択条件を設定した．複数の R 完全な出現が存在するときには，以下の条件を上から優先的に評価し，いずれか1つを満たす出現 p を選択する．

(S1) p における要対を $R \cup S$ で簡約した結果が $E \cup R \cup G \cup S$ に包含される．

(S2) p における再帰対が存在しない．

(S3) p における再帰対が $R \cup S$ 可約である．

(S4) p は最外部に現れる．

例2 図1の仕様において結合律 $(a+b)+c=a+(b+c)$ $(a, b, c \in V_{nat})$ が成立することを証明しようとすると，ある段において $E=\phi$，$R=\{x+0\Rightarrow x$ (r1), $x+s(y)\Rightarrow s(x+y)$ (r2)$\}$，$G=\{(a+b)+c=a+(b+c)$ (g1)$\}$，$S=\phi$ となる．G から (g1) を選び規則化すると，$(a+b)+c\Rightarrow a+(b+c)$ (s1) が得られる．この規則の左辺に R 完全な出現は2箇所あるが，どちらの出現にも (r2) を重ねた再帰対，$s(x_1+y_1)+c=x_1+(s(y_1)+c)$ (c1) と $s((a+b)+z_1)=a+(b+s(z_1))$ (c2) が存在する．$R \cup \{$(s1)$\}$ によって2つの再帰対を簡約すると，(c1) は変化しないが，(c2) は $s(a+(b+z_1))=s(a+(b+z_1))$ なる自明な式へ書き換えられる．そこで，選択基準 (S1) によって (c2) を得る出現位置を選択して証明を続けると，以降で手続きは停止する． ∎

5.4 再帰対を簡約する補題の追加

再帰対の獲得を避けることができず，しかもこれが簡約されないものである場合（選択条件 (S4) の場合），無限実行の可能性が高くなる．このようなときには，再帰対を簡約するための定理を後から G に追加する方法が有効である．適切な定理を追加し，この定理によって再帰対が包含されるものへと簡約されれば，その再帰対が原因となる無限実行は回避される．このような役目を果たす定理をここでは補題と呼ぶ．適切な補題の自動獲得は難しいので，*Metis-AS* では，無限に生成されるであろう要対の一般形を提示し，ユーザの判断を待つ．ユーザはこれに対し，補題を手で与えるか，この再帰対を凍結状態にして他の処理を続けるか，項の深さに制限を設けるか（後述），処理をそのまま継続するかを選択する．要対の一般形 $g[r_0\cdots r_n[t_n]\cdots]\theta_0\cdots\theta_n=d\theta_0\cdots\theta_n$ は再帰対の形から推測でき，有効な補題は，左辺 $g[r[\cdots]]\theta_0\cdots\theta_n$ を $g[\cdots]\theta_0\cdots\theta_{n-1}$ へ簡約する等式か，または右辺 $d\theta_0\cdots\theta_n$ を $d\theta_0\cdots\theta_{n-1}$ へ簡約する等式であることが多い．

例3 例1において補題 $s(x)+y=s(x+y)$ を入力したとする．これを規則 $s(x)+y\Rightarrow s(x+y)$ に変換した後，この規則を加えた $R \cup S$ で再帰対 $s(x+y)=s(y)+x$ を簡約すると，右辺が簡約されて，$s(x+y)=s(y+x)$ となる．この等式は S の規則 $a+b\Leftrightarrow b+a$ に包含されるので，以降で手続きは停止する． ∎

5.5 その他の方法

再帰対の凍結とは，再帰対を要対生成の対象からはずすことをいう．このようにして処理を続けると，後から獲得される規則によって再帰対が簡約されて取り除かれることがある．また，再帰対を凍結した状態で手続きが矛盾を発見せずに停止した場合，他の方法でこの再帰対の成立を証明するか，または証明すべき対象をこの再帰対に絞って再び手続きを適用するなどして，元の定理の正しさを保証できる．

また，手続きを停止させる最後の手段に，項の深さに制限を設ける方法がある．構成子項の深さに最大値を設け，新たに求まる要対がこの制限を越えているときには，それを捨ててしまうというものである．当然のことながら証明手続きの完全性は失われるが，これにより手続きは確実に停止することになる．この方法では，深さの最大値を徐々に大きくして，結果の信頼性を段階的に上げることができる．また，どの種の項が制限に触れて捨てられたかを記録しておくことで，検証結果の妥当性を判断するための材料が提供される．

6. 手続きの改良

本章では，検証手続きの効率化や，検証を支援する

ための機能について説明する．

6.1 入力条件を満たさない仕様への対処

実際の証明においては，与えられた仕様が入力条件を満たさない場合も考慮しなければならない．これは，次のように検査できる．まず，基礎項 t と E 等価な基礎構成子項 u の存在は，任意の演算子 $f\in D$ と任意の基礎構成子項 t_i について $f(t_1,\cdots,t_n)$ が R 可約であるならば保証される．この判定は文献15)の方法によって有限のRに対して決定可能であるため，証明の任意の時点でこれを調べることができる．Eが空となってもR可約とならない項が存在すれば，入力条件は満たされない．これは，Eが空でGにR完全でない等式が存在するときも同様である．一方，Eが空でなく，R可約でない項が存在している時点でGが空となった場合には，仕様が入力条件を満たしているという前提の下で定理の成立を帰結できる．このとき，ユーザの判断によって手続きを継続し，入力条件の成立を確認することもできる．Gが空となっても手続きを続けた場合，以降の実行は公理の無向完備化[2]に相当する．基礎項 t とE等価な基礎構成子項uが唯一でないときには，証明を進めていく過程でいつか必ずEに矛盾が生じるので明らかとなる．

6.2 冗長な定理の除去による効率化

あるソート σ について $T_\sigma(C)$ が2点以上を含まないとき，σ の任意の2項を結んだ等式は矛盾することがないので，すべて正しい定理となる．この条件は構文的に調べられるので，このような等式が証明すべき定理の集合に含まれているときには，それを取り除いて無駄な演繹を避けることができる．ただし，この場合には仕様に誤りがある可能性も高いので，*Metis-AS* ではこのようなソートの存在について警告が発せられる．この他にも，定理の中に公理系Eに包含される等式が存在する場合，それを取り除けることは明らかである．手続きを開始する時点でこういった処理を行うことで，仕様の誤りの発見を早めたり，証明を効率化することができる．

6.3 等式の分配による手続きの効率化

証明手続きでは，E や G の等式 $l=r$ について，$l\equiv c(l_1,\cdots,l_n)$，$r\equiv c(r_1,\cdots,r_n)$ で，しかも $c\in C$ であるならば，これを取り除いて，代りに $l_1=r_1,\cdots,l_n=r_n$ を追加しても手続きの完全性に影響がない[11]．この操作を等式の分配と呼ぶ．等式の分配を行うことで，無駄な要対の獲得を減らすことができる．

また，等式の分配が常に行われていれば，矛盾の検出を簡単化することができる．項 t の頂点に現れる関数記号を t の主記号と呼ぶと，等式 $l=r$ は以下のときに限り矛盾する．

（1） l, r の主記号が異なる構成子である．

（2） l の主記号が構成子で，r が変数である．

(2′) 上記(2)の対称である．

（3） l, r が異なる変数である．

6.4 証明済規則の導入による能力の向上

正しいことがすでに確認されている定理を規則化し，証明すべき定理を簡約する際にこの規則も併せて適用すると，証明の効率を落とさずに，証明能力を大幅に向上させることができる[2]．これは，人間が証明を行う際に，必要な補題を前もって証明しておき，目的とする定理の証明にその補題を利用することに対応する．手続きにおいては，こういった規則を S の初期値に含ませるだけでよい．

6.5 誤りを含む部分仕様の提示

証明において矛盾が検出されたときに，元の仕様の誤り箇所をさがす作業は人間にとって容易なものでない．手続きで証明過程の履歴（要対の生成過程，定理に対する代入の履歴など）を保存しておくと，矛盾が発見されたときに，矛盾の原因を遡って調べることができる．これにより，元の仕様における誤った公理の候補や，矛盾が発見された時点での定理に対する代入の値などをユーザに示すことができ，仕様の誤りをさがす手がかりが得られる．

6.6 改良された検証手続き

これまでに述べた改良技術を導入した定理証明手続きを以下に示す．ここでは，手続きの効率化のために，公理から演繹される等式についても簡約を行っている．また，すべての定理を各段において簡約することで，矛盾の発見を早めている．

手続き2（*Metis-AS* における検証手続き）

第0段：（**初期化**）証明すべき定理の中に冗長なものがあれば取り除く．証明済み定理の集合を規則化して S の初期値とする．他は手続き1に同じ．

第 *i* 段：（**終了判定**）E，Gがともに空ならば終了．このとき定理は成立する．Gのみが空ならばユーザの判断により終了．終了した場合，仕様が入力条件を満足しているという前提の下で定理は成立．

（**等式の選択**）手続き1に同じ．ただし，G に R 完全な等式が存在せず，かつEが空ならば終了．このとき仕様は入力条件を満たさない．

（**公理の演繹**）$l=r$ が E から選択されたものである場合，以下を行う．l, r を R によって簡約した結果を l', r' とする．$l'=r'$ が矛盾するなら

終了．このとき仕様は入力条件を満足しない．$l'=r'$ が $E \cup R$ に包含されるならば $i+1$ 段へ進む．$l'=r'$ が分配可能ならば分配結果を E に追加した後，$i+1$ 段へ進む．さもなくば $l' \Leftrightarrow r'$ を新しい規則として R へ加える．この規則と R の各規則を相互に重ねて得られる要対を E へ追加する．

(定理の演繹) $l=r$ が G から選択されたものである場合，以下を行う．$l \Leftrightarrow r$ を新しい規則として S へ加える．この規則の R 完全な出現を選択条件に基づいて1つ選び，その出現位置に R の各規則を重ねて得られる要対を G へ追加する．このとき再帰対が存在していれば，ユーザの判断によって補題の獲得などを行う．

(定理の簡約) G のすべての等式を，$R \cup S$ によって簡約したものと置き換える．このとき，分配可能な等式は分配し，$E \cup R \cup G \cup S$ に包含される等式は取り除く．この結果，G に矛盾する等式が現れれば終了．このとき定理は不成立．さもなくば $i+1$ 段へ進む． ∎

7. 検 証 例

手続き2は代数的仕様によるソフトウェア開発支援環境 *Metis-AS* に実装されている．この証明手続きを用いて実際に仕様の検証を行う．

7.1 自然数の加算の検証

Metis-AS のユーザ・インタフェースを見るために，これまでの例を検証した様子を**図2**に示す．ここでは，左単位元の存在，結合律，交換律の3つの性質を同時に証明している．左単位元の存在は問題なく証明され，結合律の証明については要対の選択条件が機能して無限実行が回避される．交換律の証明については，再帰対の獲得が避けられないため，以降の処理についてユーザへの問合せが行われる．ここで適切な補題を入力すると，証明は完了する．この結果，定理の成立を表す "PROVED" の文字と，導出された等式や規則の数，処理時間とその内訳などの情報が表示される．手続きの開始から完了までに要した時間は約0.4秒である．

7.2 エレベータボタン操作の仕様検証

図3はエレベータ内部におけるボタン操作の一部を *Metis-AS* の言語によって記述したものである．ここで，記述の4～6行が D の定義に，7～11行が C に，12～14行が V に，15～22行が E の定義に対応している．ただし，V には記述で用いる変数名のみが宣言されている．「ボタン押下げ」(宣言は5行，公理は16～19行) は，階番号を停止階情報へ登録する関数で，利用者が停止階ボタンを押す操作を表す．「動作指示参照」(宣言は6行，公理は20～22行) は，停止階情報に階番号が登録されているかを調べる関数で，システムが各階への指示(停止または通過)を参照する操作を表す．ここでは，これ以外の操作 (外部からの呼び出しなど) は考えないことにする．この仕様における主要な性質として，以下のものが挙げられる．

(1) 初期状態では，どの階に対する停止指示もない．

```
[METIS-AS]-> prove
[ inductive theorem proving for "Nat" ]
(retrieving...  5 data, 17 msec)
Prove 0+A==A ? (*y/n) y
Prove A+B==B+A ? (*y/n) y
Prove (A+B)+C==A+B+C ? (*y/n) y
Rul r1: A+0 ==> A (e1)
Rul r2: A+s(B) ==> s(A+B) (e2)
Smp r3: 0+A ==> A (e3)
Smp r4: (A+B)+C ==> A+B+C (e4)
Smp r5: A+B <=> B+A (e5)
##  s(A+B)==s(B)+A is a recursive pair which generates
##  s(//...s(A+B)...\\)==s(/...s(B)...\)+/A\
[1] introduce new lemma
[2] set maximum depth
[3] freeze this RP
[4] give up
[*] go on
Which? 1
Lemma> s(x)+y==s(x+y).
Smp r6: s(A)+B ==> s(A+B) (e6)

####### PROVED #######
E(quations)  : 2 generated, 0 remain.
R(ules)      : 2 generated.
T(heorems)   : 5 generated, 0 remain.
S(implifiers): 4 generated.
CP(s)        : 0 found, 0 asserted.
Narrow(s)    : 8 found, 1 asserted.
Reduction    : 7 steps.
Runtime      : 393 msec
( 37% for selection,  0% for superposition, 19% for narrowing,
   6% for ordering,  13% for E-reduction,  10% for T-reduction,
  15% for others )
Save this result ? (*y/n) y
(saving...  16 data, 83 msec)
```

図2 自然数の加算の検証例
Fig. 2 Verification example of the natural number addition specification.

(2) 任意の状態において，ボタンが押されない階に対する指示は変わらない．

(3) 任意の状態において，ボタンが押された階に対する指示は停止となっている．

これらの性質は，定理として仕様の 23〜27 行に記述されている．たとえば 27 行の記述は，「任意の状態 s で，f 階のボタンを押した結果から f 階への指示を参照すると，指示は停止になっている」と読むことができ，性質の(3)に対応していることがわかる．これらの性質の証明を試みると，手続きは約 3 秒ですべての定理の成立を告げる．

次に，誤りを含んだ仕様の検証を見るために，図 3 の 22 行を以下の記述と入れ換える．

==if f1=f2 then **通過** else 動作指示参照 (f1, s)；

この結果，動作指示参照の定義に誤りが生じる．この仕様を用いて同じ定理を証明した様子を**図 4** に示す．証明の結果，定理の不成立を表す "DISPROVED" の文字と，矛盾の種類，不成立の定理，矛盾が導かれた際の代入の値，誤っている公理の候補などが表示される．実際に，誤りを埋め込んだ公理が候補に含まれていることが，図から確認できる．ユーザは，この情報を手がかりに仕様の誤りをさがして修正することになる．たとえば，表示された代入を定理に施し，これを公理によって簡約すると，矛盾が導かれる過程を目で追うことができる．この様子を**図 5** に示す．

```
1. spec エレベータ内ボタン操作 has
2.     use Bool;
3.     sorts 階番号, 停止階情報, 動作指示;
4.     functions
5.         ボタン押下げ : 階番号, 停止階情報 ⟶ 停止階情報;
6.         動作指示参照 : 階番号, 停止階情報 ⟶ 動作指示;
7.     constructors
8.         InitStop : ⟶ 停止階情報;
9.         AddStop : 階番号, 停止階情報 ⟶ 停止階情報;
10.        F1, ... ,F3 : ⟶ 階番号;
11.        停止, 通過 : ⟶ 動作指示;
12.    forall
13.        f, ... ,f3 : 階番号;
14.        s : 停止階情報;
15.    axioms
16.        ボタン押下げ (f,InitStop)==AddStop(f,InitStop);
17.        ボタン押下げ (f1,AddStop(f2,s))
18.        ==if f1=f2 then AddStop(f1,s)
19.                   else AddStop(f2, ボタン押下げ (f1,s));
20.        動作指示参照 (f,InitStop)== 通過;
21.        動作指示参照 (f1,AddStop(f2,s))
22.        ==if f1=f2 then 停止 else 動作指示参照 (f1,s);
23.    theorems
24.        動作指示参照 (f,InitStop)== 通過;
25.        動作指示参照 (f1, ボタン押下げ (f2,s))== 動作指示参照 (f1,s)
26.        when not f1=f2;
27.        動作指示参照 (f, ボタン押下げ (f,s))== 停止;
28. end.
```

図 3 エレベータ操作の代数的仕様例
Fig. 3 Algebraic specification example of elevator operation.

8. おわりに

本稿では，代数的仕様を自動検証する手続きの実現について述べた．この手続きは反駁完全性をもち，無限実行の可能性がある場合を除いて完全に自動実行でき，しかも証明が効率よく行える．無限実行についても人間の助けを借りる対処法はいくつか実装されているが，これを自動化することが今後の重要課題である．また，代数的仕様を記述するための言語は，条件付公理の導入や順序ソートへの拡張，オブジェクト指向技術の導入など，記述力の面から次々と拡張されている状況にあり，これらの拡張に，いかに検証を適合させていくかは長期的な課題である．

謝辞 本研究は，第 5 世代コンピュータプロジェクト (FGCS) の一環として行われたものである．研究の機会をいただいた，ICOT 渕一博所長 (現在，東京大学教授)，長谷川隆三部長 (現在，ICOT 次長)，(株)東芝 研究開発センター システム・ソフトウェア生産技術研究所 西島誠一所長 (現在，研究開発センター次長)，大筆豊 研究主幹，三亀和雄 部長に感謝致します．また，本論文に関する誤りの指摘や有益な助言をいただいた査読者の方々に深く感謝致します．

```
######## DISPROVED ########
By 停止 = 通過
######## CHECK THE FOLLOWING ########
Theorem(s):
  動作指示参照 (f, ボタン押下げ (f,s)) == 停止 with {InitStop/s}
Axiom(s):
  ボタン押下げ (A,InitStop) == AddStop(A,InitStop)
  動作指示参照 (A,InitStop) == 通過
  動作指示参照 (A,AddStop(B,C))
                == if(A=B, 通過, 動作指示参照 (A,C))
```

図 4 誤りの検出
Fig. 4 Detection of inconsistency.

```
[METIS-AS]-> execute
(retrieving...  26 data, 83 msec)
Term> 動作指示参照 (f, ボタン押下げ (f,InitStop))== 停止.
==> 動作指示参照 (f,AddStop(f,InitStop))== 停止
==> if(f=f, 通過, 動作指示参照 (f,InitStop))== 停止
==> if(true, 通過, 動作指示参照 (f,InitStop))== 停止
==> 通過 == 停止
[ 3 msec ]
```

図 5 仕様実行による誤りの追認
Fig. 5 Confirmation of incorrectness by specification execution.

参考文献

1) Avenhaus, J.: Proving Equational and Inductive Theorems by Completion and Embedding Techniques, *LNCS 488*, pp. 361-373, Springer-Verlag (1991).
2) Bachmair, L.: *Canonical Equational Proofs*, p. 135, Birkhäuser (1991).
3) Bidoit, M. et al. (eds.): Algebraic System Specification and Development, *LNCS 501*, p. 98, Springer-Verlag (1991).
4) Boyer, R. S. and Moore, J. S.: *A Computational Logic*, p. 397, Academic Press (1979).
5) Dershowitz, N.: Orderings for Term-rewriting Systems, *Theor. Comput. Sci.*, Vol. 17, No. 3, pp. 279-301 (1982).
6) Ehrig, H.: Overview of Algebraic Specification Languages, Environments and Tools, and Algebraic Specifications of Software Systems, *Bull. EATCS*, Vol. 39 & 40 (1989).
7) Fribourg, L.: A Strong Restriction of the Inductive Completion Procedure, *J. Symbolic Computation*, Vol. 8, No. 3 & 4, pp. 253-276 (1989).
8) 二木厚吉, 外山芳人: 項書き換え型計算モデルとその応用, 情報処理, Vol. 24, No. 2, pp. 133-146 (1983).
9) Hermann, M. and Privara, I.: On Nontermination of Knuth-Bendix Algorithm, *LNCS 226*, pp. 146-156, Springer-Verlag (1986).
10) Hermann, M.: Crossed Term Rewriting Systems, private communication (1988).
11) Huet, G. and Hullot, J.-M.: Proofs by Induction in Equational Theories with Constructors, *J. Comput. Syst. Sci.*, Vol. 25, No. 2, pp. 239-266 (1982).
12) 稲垣康善, 坂部俊樹: 抽象データタイプの代数的仕様記述法の基礎(1)～(4), 情報処理, Vol. 25, No. 1, 5, 7, 9, pp. 47-53, 491-501, 708-716, 971-986 (1984).
13) Kirchner, H.: Schematization of Infinite Sets of Rewrite Rules Generated by Divergent Complete Processes, *Theor. Comput. Sci.*, Vol. 67, No. 2 & 3, pp. 303-332 (1989).
14) 大須賀昭彦, 坂井 公, 本位田真一: 等式論理の帰納的定理を証明する手続き, 信学論, Vol. J 76-D-I, No. 3, pp. 130-138 (1993).
15) Plaisted, D. A.: Semantic Confluence Tests and Completion Methods, *Inf. Control*, Vol. 65, pp. 182-215 (1985).
16) Sakai, K.: An Ordering Method for Term Rewriting Systems, Tech. Report TR-062, ICOT (1984).
17) 坂井 公: Knuth-Bendix の完備化手続きとその応用, コンピュータソフトウェア, Vol. 4, No. 1, pp. 2-22 (1987).
18) 酒井正彦, 坂部俊樹, 稲垣康善: 代数的仕様の検証のための被覆集合帰納法, 信学論, Vol. J 75-D-I, No. 3, pp. 170-179 (1992).
19) Zhang, H., Kapur, D. and Krishnamoorthy, M. S.: A Mechanizable Induction Principle for Equational Specification, *LNCS 310*, pp. 162-181, Springer-Verlag (1988).

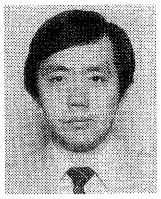
(平成5年1月18日受付)
(平成5年9月 8 日採録)

大須賀昭彦 (正会員)

1958 年生. 1981 年上智大学理工学部数学科卒業. 同年(株)東芝入社. 1985～1989 年(財)新世代コンピュータ技術開発機構に出向. 現在(株)東芝研究開発センターシステム・ソフトウェア生産技術研究所開発主務. 主として形式的仕様記述法の研究に従事. 項書換え技術や定理の自動証明技術に興味を持つ. 1986 年情報処理学会論文賞受賞. 電子情報通信学会, 日本ソフトウェア科学会, EATCS 各会員.

坂井 公 (正会員)

1953 年生. 1976 年東京工業大学理学部情報科学科卒業. 1978 年同大大学院修士課程修了. 同年日本電気(株)入社. 1982 年 10 月 ICOT 出向. 1992 年筑波大学電子情報工学系講師. 理学博士. 理論計算機科学, 特に自動定理証明, 項書換えシステム, 構成的数学, 型理論, 制約処理など, 数学基礎論の計算機応用の研究に従事. 日本ソフトウェア科学会, EATCS 各会員.



本位田真一 (正会員)

1953 年生. 1976 年早稲田大学理工学部電気工学科卒業. 1978 年同大学院理工学研究科電気工学専攻修士課程修了. 工学博士. 同年(株)東芝入社. 現在, 同社研究開発センターシステム・ソフトウェア生産技術研究所主任研究員. 1989 年より早稲田大学非常勤講師を兼任. 1991 年東京工業大学大学院非常勤講師. 主として, ソフトウェア工学, 人工知能の研究に従事. ソフトウェア工学, 人工知能の研究に従事. ソフトウェアの基礎理論に興味を持つ. 1986 年情報処理学会論文賞受賞. 著訳書「ソフトウェア開発のためのプロトタイピング・ツール」(共著),「KE 養成講座(2)エキスパートシステム基礎技術」(共著),「オブジェクト指向システム分析」(共訳),「オブジェクト指向システム開発」(共著) など. 日本ソフトウェア科学会, 人工知能学会, IEEE, AAAI 各会員.