ソフトウェア基礎論 31 － 1
（１９８９．９．２０）

ＴＲＳプログラムの自動合成

Synthesis of TRS Programs

富樫 敦 *　　千葉 和也 **　　野口 正一 *
Atsushi TOGASHI　　Kazuya CHIBA　　Shouichi NOGUCHI

＊　東北大学電気通信研究所　Research Institute of Electrical Communication, Tohoku University
＊＊　富士ゼロックス（株）　Fuji Xerox Limited

あらまし　入出力の例から，帰納的推論によって代数を実現する TRS プログラムを合成するアルゴリズムを示す．合成方法としては，基本的には Shapiro がモデル推論で用いたような漸増的な方法である[12]．アルゴリズムは，事実の枚挙（入出力例の無限列）を読み込んでいき，現在の推測を表わす TRS R_c を用いて，読み込んだ例の左辺を計算する．計算結果が正しくなければ，R_c から正しくないルールを取り去るか，または新しいルールを R_c に加えて推測を少しずつ修正していく．R_c が読み込んだ例に対して正しくなれば，また次の例を読み込み同様の処理を行なう．

合成アルゴリズムの正当性に関しては，簡約順序$\succ$のもとで停止し，代数を実現する TRS プログラムが存在するならば，アルゴリズムは，このような代数を実現する TRS プログラムに極限において収束するということを示す．

１．　はじめに

人工知能，あるいはソフトウェア工学（科学）の究極の目的の１つに，自動プログラミングが挙げられる．プログラムの仕様から，（半）自動的に意図するプログラムを合成することである．自動プログラミングに関しては，現在までに色々なアプローチが提案され，実験システムが試作されてきた[9]．

自動プログラミングの手法は，そのスタイルに応じて，仕様からプログラムをまったく新しく生成しようとする生成的アプローチと，既存のモジュールをいくつか組み合わせてプログラムを合成しようとするアプローチ（たとえば，部品合成による方法）に分けることができる．生成的アプローチも仕様が完全であるかどうかに応じて，完全仕様による方法と不完全仕様による方法に分割できる．前者は，完全に与えられた仕様から目的とするプログラムを合成する方法であり，演繹的推論による自動プログラミング[11]やプログラム変換による自動プログラミング[4]がこの範疇に入る．後者は具体例あるいは例題から一般的なプログラムを生成する方法であり，帰納的推論による方法[2, 12, 14]などが挙げられる．

本論文では，入出力の例から，帰納的推論によって代数を実現する TRS プログラムを合成するアルゴリズムを提案し，その正当性について述べる．主な結果として，簡約順序$\succ$のもとで停止し，代数を実現する TRS プログラムが存在するならば，アルゴリズムは，このような代数を実現する TRS プログラムに極限において収束するということを示す．

２．　項書き換えシステムと代数の実現

この章では，合成の対象となる項書き換えシステム(TRS)に関する定義を簡単に与え，代数を実現するシステムとして TRS を見ることができることについて述べる．

２．１　項書き換えシステム(TRS)

はじめに，項書き換えシステムを簡単に定義する．詳しい定義については，文献[6,7,8]を参照されたい．

関数記号の集合Ｆを仮定する．関数記号は f, g, h などの英小文字で表わされ，各関数記号は０以上の決まった数の引数を取る．０引数の関数記号は，定数である．変数を X, Y, Z などの英大文字で表わし，変数の集合をＶとする．項は，関数記号と変数から次のようにして帰納的に定義される．

(1)　変数 $X \in V$ は項である．

(2)　$f \in F$ を n 引数($n \geq 0$)の関数記号，$t_1, \cdots, t_n$ を項と

すると，$f(t_1,\cdots,t_n)$は項である．

項を表わすのに，s，t，u，v，l，r などの文字を用いる．FとVから作られる項全体の集合を T(F,V)で表わす．また，変数を含まない項を基礎項(ground term)と呼び，Fから作られる基礎項全体の集合を T(F)で表わす．

文脈(context)は，空の場所を表わす特別な定数記号□を1つだけ含む項であり，t[ ]と表わされる．文脈 t[ ]中の□を項 s で置き換えて得られる項を t[s]で表わす．s は t[s]の部分項である．変数を項で置き換える操作を代入(substitution)と呼び，$\theta = \{t_1/X_1,\cdots,t_n/X_n\}$ で表わす．項 t にθを施した結果 tθは，t 中の変数 X_i に t_i を同時に代入して得られる項である．

【定義2．1】　書き換え規則(あるいは，単にルール)は，2つの項 l，r を▷で結んで，l▷r の形で表わした項の順序対である．ただし，Var(l)⊃Var(r)であり，l は変数ではない．ここで，Var(t)は，項 t に出現する変数全体の集合を表わす．　□

一般に，任意の2つの項 l，r を▷で結ぶとルールができるが，この中で Var(l)⊃Var(r)でなかったり，l が変数であったりするルールは TRS のルールとして不適当である．以下，このようなルールを不適当なルールと呼ぶ．

【定義2．2】　項書き換えシステム (Term Rewriting System，以下 TRS と略記) は，書き換え規則の集合 R である．　□

以下，≡で2つの項が表記上同一であることを表わす．ルール l▷r が項 t≡t[s]に適用可能であるとは，ある代入θに対して s≡lθとなる場合である．s を t のリデックス(redex)と呼ぶ．このとき，l▷r が t に適用され，項 u≡t[rθ]を得る．$t\rightarrow_R u$ で，u が R 中のルールの1回の適用によって t から得られることを示す．$\rightarrow_R$ の反射・推移的閉包を $\overset{*}{\rightarrow}_R$ で表わす．$t\overset{*}{\rightarrow}_R u$ のとき，t は u に簡約されたと言う．項 t にリデックスがない場合，t を正規形(normal form)という．$t\overset{*}{\rightarrow}_R u$ かつ u が正規形ならば，u は t の正規形と呼ばれ，$t\downarrow_R$ とも書かれる．R によっては，t の正規形 $t\downarrow_R$ は一般に複数個存在する．以下，前後から R が明らかなときは R を省略する．

【例2．1】　例えば，足し算(plus)を用いた掛け算(times)のプログラムは，次の TRS で記述される．

times(0,Y)▷0
times(s(X),Y)▷plus(Y,times(X,Y))
plus(0,Y)▷Y
plus(s(X),Y)▷s(plus(X,Y))

ここで，自然数は 0 と後者関数 s によって，0，s(0)，s(s(0))，… と表現される．例えば，2×1を表わす項 times(s(s(0)),s(0))は，次々に簡約されて正規形 s(s(0))まで書き換えられる．　□

本論文では，関数記号には，構成子(constructor)と定義関数記号(defined function symbol)の2種類があると仮定する．構成子は，データ構造を作るために用いられる．構成子だけから作られた項は，通常正規形になっている．構成子および定義関数全体の集合をそれぞれCとDで表わす．したがって，F＝C∪D．構成子の例をいくつか示す．

s(_)，0，cons(_,_) ([_|_]とも書く)，nil ([]とも書く)，true，false

【定義2．3】　構成子だけから作られる変数を含まない項をデータ項(data-term)といい，データ項全体の集合を T(C)で表わす．　□

例えば，0，s0，ss0，…（ここで，s0 は s(0)の略記である）がデータ項の例である．

定義2．1より，書き換え規則は

$f(l_1,\cdots,l_n)$▷r

の形で現わされる．簡約の定義より，この規則は f で始まる項を書き換えることから，f に関する規則を定めている．データ項はデータの値を項として表現したものであるから，データ項はもうこれ以上書き換えられないとする．従って，本論文では書き換え規則に関して次の条件を仮定する．

【仮定】　規則の左辺は必ず定義関数記号で始まり，その引数には定義関数記号が現われない．

以上の仮定から，データ項は必ず正規形である．

項書き換えシステムについては，次の合流性と停止性が重要となる．

【定義2．4】　R を TRS，T を項の集合とする．

(1)　R が T について合流するとは，任意の項 t∈T について，$t\overset{*}{\rightarrow}t_1$，$t\overset{*}{\rightarrow}t_2$ ならば，ある項 s が存在して，$t_1\overset{*}{\rightarrow}s$，$t_2\overset{*}{\rightarrow}s$ となること．

(2)　R が T について停止するとは，任意の項 t∈T について，t から始まる長さ無限の簡約系列 $t\rightarrow t_1\rightarrow\cdots\rightarrow t_i\rightarrow\cdots$ が存在しないこと．

(3)　R が T について合流しかつ停止するとき，R は T について完備であるという．　□

定義2．4で，特に，T が項全体から成る集合のとき，T について合流する，停止する，完備な TRS は，それぞれ単に合流する，停止する，完備であると呼ばれる．

2．2　代数を実現する TRS

TRS によって，代数を実現することができる．一般に，

代数は台と呼ばれるデータの集合Aと，台上での（全域的な）演算の集合Σの2項組＜A,Σ＞で定義され，台だけで参照される場合が多い．代数＜A,Σ＞を項を用いて具体的に表現するために，次に述べる代数の表現を考える．まずはじめに，台A中の要素をいくつかの構成子を用いて表現し，AとT(C)の間に1対1の写像を定める．次に，各演算子σ∈Σを定義関数記号を用いて表現する．

【定義2．5】　代数＜A,Σ＞の表現は，4項組Γ＝＜C,D,γ,δ＞である．ここで，

(1)　Cは，構成子の集合；

(2)　Dは，定義関数記号の集合；

(3)　γは，1対1の写像γ：T(C)→A；

(4)　δは，1対1の写像δ：D→Σ；ただし，対応する関数記号と演算子の引数の数は同じ．　□

【例2．2】　例えば，自然数上で足し算とかけ算が定義された代数＜N，{＋,×}＞の表現は，構成子の集合{0,s}，定義関数の集合{plus, times}と次の2つの写像からなる．

γ ：$0\mapsto 0_N$, $s0\mapsto 1_N$, $ss0\mapsto 2_N$, …

δ ：plus(_,_)↦＋, times(_,_)↦×

添字Nは代数の台の要素を示すために付けた．　□

代数Aとその表現γが定まると，T(F)からT(C)への写像を定義することができる．この写像を定義するために少し準備をする．

【記法2．1】　次に定める項の集合をT_oとする．

T_o＝ {d(c1,…,cn) | d∈D, ci∈T(C), n≧0}

【定義2．6】　代数Aの表現Γによって定まる関数は，T_oを定義域T(C)を値域とする関数Val[Γ]：T_o→T(C)であり，Val[Γ](d(c1,…,cn))＝cとなるのは，γ(ci)＝ai (1≦i≦n)，δ(d)＝σ，σ(a1,…,an)＝a，γ^{-1}(a)＝cとなる場合であり，またその場合だけである．ここで，γ^{-1}はγの逆関数である．　□

【例2．3】　例えば，Valを例2．2で与えた代数表現によって定まる関数とすると，γ(ss0)＝2_N，γ(s0)＝1_N，δ(plus)＝＋，$1_N+2_N=3_N$，γ(sss0)＝3_Nであるから，Val(plus(ss0,s0))＝sss0となる．

3章で合成の例をあげるときは，例2．2の代数表現によるValを断わりなく用いることにする．　□

例からも分かるように，Valを用いると，定義関数記号はデータ項の組を引数としてデータ項を返す関数とみなせる．この意味で，定義関数記号を単に関数と呼ぶことがある．代数とその表現Γが与えられると関数Val[Γ]:T_o→T(C)が定義されることをみた．逆に，関数記号の集合F＝CUDを仮定した場合，関数Val:T_o→T(C)が与えられると，Valから T(C)を台Dを演算子の集合とする代数が定まり，Val＝Val[Γ]を満たす表現Γを逆に定義することができる．このことは，代数と関数Valを同一視できることを意味する．そこで，以下本論文では，Valを代数の代わりとして用いる．その際，代数の表現Γは既知として，Val[Γ]を単にValと記すことにする．

代数を実現するTRSを合成する場合，Valを用いることによって，議論をいちいち代数までさかのぼって行う必要がなくなり，項の間の関係だけに着目すればよい．

【定義2．7】　Val^+ は T(F)から T(C)への関数であり，Valを用いて次のように帰納的に定義される．

$Val^+(f(t_1,\cdots,t_n))$

＝$f(Val^+(t_1),\cdots,Val^+(t_n))$　　if　f∈C

＝$Val(f(Val^+(t_1),\cdots,Val^+(t_n)))$　if　f∈D　□

定義から明らかに，データ項s_1，s_2について，$Val^+(s_1)=Val^+(s_2)$であることと$s_1\equiv s_2$であることは同値．

【命題2．1】　文脈t[ ]と項u，vについて，$Val^+(u)=Val^+(v)$ならば$Val^+(t[u])=Val^+(t[v])$が成り立つ．

（証明）　Val^+の定義から明らか．　□

Val^+により，Valに関するルールの真偽を次のように定義する．

【定義2．8】　ルール l▷r が Valに対して真であるとは，すべての基礎代入θに対して，$Val^+(l\theta)=Val^+(r\theta)$が成り立つこと．　□

従って，ルール l▷r が Valに関して偽であるならば，ある基礎代入θについて，$Val^+(l\theta)\neq Val^+(r\theta)$となる．すべてのルールがValについて真であるTRSは，Valに関して真であるという．命題2．1により，真なルールによる項の書き換えは，Val^+の値を保存する．即ち次の命題が成り立つ．

【命題2．2】　Rを Valに関して真な TRS，t，sを基礎項とするとき，$t\xrightarrow{*}s$ ならば $Val^+(t)=Val^+(s)$．　□

【命題2．3】　Rを TRSとし，$t\xrightarrow{*}s$とする．$Val^+(t)\neq Val^+(s)$ならば，tをsまで簡約するときに使われたルールの集合 R'⊂R の中に偽なルールが必ず存在する．

（証明）　R'が Valに関して真であるならば，命題2．2より $Val^+(t)=Val^+(s)$．これは仮定に反する．従って，R'の中に偽なルールが必ず存在する．　□

Valを実現する TRSは，Valに関して真であり，かつ項 t∈T_oの正規形がただ1つだけ存在して Val(t)と一致するような TRSである．正確には，Valを実現する TRSは次のように定義される．

【定義２．９】　TRS R が Val の実現であるとは，

(1)　R は Val に関して真であり；

(2)　任意の $t \in T_\Omega$ とデータ項 s について，$Val(t)=s$ ならば $t \xrightarrow{+}_R s$.

【命題２．４】　R を Val を実現する TRS，t を任意の基礎項，s，s_1，s_2 を任意のデータ項とすると，R について，次の性質が成り立つ．

(1)　$Val^+(t)=s$ ならば $t \xrightarrow{+}_R s$.

(2)　$t \xrightarrow{+}_R s_1$，$t \xrightarrow{+}_R s_2$ ならば $s_1 \equiv s_2$．従って(1)より，$Val^+(t)=s$ であることと $t \xrightarrow{+}_R s$ であることは同値．

（証明）　(1)については，実現の定義から明らか．(2)については，R は Val に関して真であるから，基礎項 t，データ項 s について，$t \xrightarrow{+}_R s$ であるならば $Val^+(t)=Val^+(s)=s$・従って，(2)が成り立つ．　□

【命題２．５】　R を Val を実現する TRS とすると，R について，次の性質が成り立つ．

(1)　R は基礎項全体の集合 T(F)について合流する．

(2)　基礎項 t の正規形はただ１つだけ存在し，$Val^+(t)$と一致する．従って，t が R のもとで $Val^+(t)$ と異なる正規形を持つことはない．

（証明）　(1)について示す．(2)については，$Val^+(t)$が t の正規形であることと(2)から明らか．基礎項 t, t_1, t_2 について $t \xrightarrow{+} t_1$，$t \xrightarrow{+} t_2$ とすると，R は Val に関して真であるから，$Val^+(t)=Val^+(t_1)=Val^+(t_1)$．従って，$t_1 \xrightarrow{+} Val^+(t)$，$t_2 \xrightarrow{+} Val^+(t)$．　□

３．代数を実現するＴＲＳプログラムの合成アルゴリズム

この章では，代数を実現する TRS の合成アルゴリズムについて述べる．このアルゴリズムは，TRS のルール全体の枚挙を考え，すでに読み込んでいる事実に対して正しいルールを拾い上げ，間違っているルールを捨てるという方法である．基本的には，Shapiro がモデル推論で用いたような漸増的な方法[12,13]である．

３．１　枚挙と神託

本アルゴリズムでは，ルール全体の集合 R_h を仮定する．R_h の要素は線形順序で並んでいるとする．アルゴリズムによって推測された現在の TRS を R_c とする．アルゴリズムは，事実を読み込んでいき，R_c を用いて読み込んだ事実の左辺を簡約する．簡約した結果が正しくなければ，３．３節，３．４節で述べる２つの場合のいずれかが起こり，それぞれ R_c 中の偽なルールが捨てられるか，R_h から新しいルールが加えられる．R_c が読み込んだ事実に対して正しくなれば R_c を出力して，また次の事実を読み込んでいく．

合成アルゴリズムにユーザーが与える事実は，Val の枚挙である．

【定義３．１】　$t \in T_\Omega$ について，$Val(t)=s$ が成り立つとき，$t=s$ は Val についての事実であるという．また，Val についての事実の無限列 $l_1=r_1$，$l_2=r_2$，… で，T_Ω 中の任意の項 l が，事実 $l=r$ の左辺としてどこかに必ず現われる場合，この無限列を Val の枚挙という．　□

合成アルゴリズムでは，以下で定義する神託も仮定するが，理論的には神託を仮定することと枚挙を仮定することは本質的には同じである．

【定義３．２】　T_Ω 中の項 t を受け取って，答として Val(t)を返す仕掛けを Val についての神託という．　□

３．２　停止性の保証

最終的に合成される TRS は，停止性を満足しなければならない．また，合成の途中で得られた TRS についても，停止性はつねに保証されていないといけない．なぜなら，停止性を満足しない TRS に対しては，合成アルゴリズム自身が止まらなくなってしまうことがあるからである．そこで，項の集合上に適当な順序を定めることによって停止性を保証する．

【定義３．３】　$\succ$を項の集合上の半順序とする．

(1)　$\succ$が整礎順序(well-founded ordering)であるとは，

$t_0 \succ t_1 \succ t_2 \succ \cdots$

と無限に減少していく項の無限列が存在しないこと．

(2)　$\succ$が安定(stable)であるとは，任意の項 u，v，文脈 t[ ]，代入 θ に対して，$u \succ v$ ならば $t[u] \succ t[v]$，$u\theta \succ v\theta$ となること．　□

安定な整礎順序を簡約順序と呼ぶ．明らかに，すべてのルール $l \triangleright r \in R$ について，$l \succ r$ が成り立つ簡約順序$\succ$が見つかれば，R は停止性を満たす．そこで，TRS の停止性を保証するために，簡約順序$\succ$を合成アルゴリズムにパラメータとして与えることにする．自動的に順序が判定できる簡約順序としては，再帰的経路順序(recursive path ordering)や，辞書式経路順序(lexico graphic path ordering)[5]などが知られている．これらの順序は，関数記号の順序を与えるだけで項の順序が決定される．

合成アルゴリズムでは，与えられた簡約順序$\succ$に従ってルールを調べ，R_c には $l \succ r$ を満たすルールしか加えないようにして，R_c の停止性を保証している．

【定義３．４】　TRS R が簡約順序$\succ$のもとで停止する(terminate under $\succ$)とは，すべてのルール $l \triangleright r \in R$ につ

いて $l \succ r$ が成り立つこと．　□

3．3　正しくない答を出す場合

例えば R_c が，次のプログラムであったとする．

plus(0,Y)▷s(Y)

plus(s(X),Y)▷s(plus(X,Y))

このプログラムを用いて，事実 plus(s0,0)＝s0 の左辺を計算すると，

plus(s0,0)→s(plus(0,0))→s(s0)

となり正規形は s0 とはならない．この原因は，計算に用いたプログラム中に偽のルールがあるからである．正しいプログラムを合成するためには，誤りのあるプログラムからその原因を見つけなければいけない．以下では，その方法について述べる．

事実 t＝s の左辺 t を TRS R_c を用いてデータ項（従って，正規形）まで簡約できたとする．

$(t \equiv) t_0 \rightarrow t_1 \rightarrow t_2 \rightarrow \cdots \rightarrow t_n (\equiv t\downarrow)$

$t_n \equiv s$ であれば問題はないが，$t_n \neq s$ であれば R_c から偽なルールを見つけないといけない．命題2．3より，R_c には必ず偽なルールが存在する．どのルールも，右辺に出現する変数はすべて左辺に出現するので，基礎項を書き換えて得られた項は必ず基礎項である．従って，各 t_i は基礎項である．そこで，各 t_i について，最も内側に位置する T_σ の項の値を神託によって求めることによって，$Val^+(t_i)$ の値を求めることができる．

$Val^+(t_i) \neq Val^+(t_{i+1})$

となる i，$0 \leq i < n$，が見つかれば，t_i から t_{i+1} を得るときに用いた規則が偽である．上の例では s0，s0，ss0 となり，2番目の書き換えで使われたルール plus(0,Y)▷s(Y) が偽であることがわかる．アルゴリズム3．4で述べるように，偽なルールは R_c から取り除かれる．

以上の方法により，誤った答を出した場合は，神託によってその原因となる偽のルールを見つけだすことができる．次に，そのアルゴリズムを示す．

【アルゴリズム3．1】　偽なルールの削除

入力：　Val についての神託

停止性が保証された TRS R_c

簡約列 $(t \equiv) t_0 \rightarrow t_1 \rightarrow t_2 \rightarrow \cdots \rightarrow t_n (\equiv t\downarrow)$

ここで，$t \in T_\sigma$，t_n はデータ項であり，$t_n \neq Val(t)$．また，$t_i \rightarrow t_{i+1}$ の簡約には，ルール $l_i \triangleright r_i$ が用いられたとする．

出力：　R_c 中の偽なルール

アルゴリズム：

```
begin
 i ← 0
 j ← n
M ← Val(t)
N ← t_n
repeat
    k ← ⌊(i+j)/2⌋
    Val+(t_k)の値を神託によって求める．
    if  M≠Val+(t_k)
        then  j ← k
              N ← Val+(t_k)
        else  i ← k
              M ← Val+(t_k)
until  i + 1 = j
ルール l_i▷r_i を出力
end
```

補題3．1が，アルゴリズム3．1の正当性である．

【補題3．1】　アルゴリズム3．1の仮定のもとで，アルゴリズム3．1は，必ず停止し，R_c のある偽なルールを出力する．

（証明）　省略．　□

3．4　答が出ない場合

例えば，R_c による項の書き換えが plus(plus(5,3),2) のようなデータ項でない形で止まるときがある．この原因は R_c のルールが不足しているからである．この場合は，R_h の先頭からその左辺が，内側の項 plus(5,3)にマッチするルールを捜してきて，Rc に加えればよい．例えば，plus(X,3)▷X がそうである．このルールは偽であるが，偽なルールはアルゴリズム3．1でいつか取り除かれるので，この段階ではルールの真偽は考えなくてもよい．また，R_c の停止性を保証するため，新しいルールを加えるときに順序のチェックを行なう．

次に，簡約結果がデータ項でない場合，R_h から新しいルールを捜してくるアルゴリズムを示す．R_h の各ルールには最初にマークを付けないでおき，使用されたときにマークを付けていく．こうすることによって，1回使われたルールを再度使わないようにする．

【アルゴリズム3．2】

パラメータ：　ルール全体の線形順序集合 R_h

停止性を保証する簡約順序 $\succ$

入力：　項 t（ただし，t はデータ項でない基礎項である）

出力： R_h 中のまだマークされていないルール $l \triangleright r$ で，$l \succ r$ を満たすもの

アルゴリズム：

t の最も内側の部分項で，T_Ω に含まれる部分項を1つ見つけて s とする．

```
i ← 1
repeat
  if  l_i ▷ r_i にマークが付いていなくて，かつ l_i は
      s とマッチする．（ある θ があって，l_i θ ≡ s ）
    then  l_i ▷ r_i  にマークを付ける
          if  l_i ≻ r_i  then  l_i ▷ r_i を出力し end へ．
  i ← i + 1
forever
end
```

アルゴリズム 3．2 の正当性は後で示される．

3．5　精密化演算子によるルールの枚挙

合成アルゴリズムで用いるルール全体の集合を構成するために，次の精密化演算子を考える．

【定義 3．5】　精密化演算子(refimement opetator) ρ_1 は，ルール δ からルールの集合 $\rho_1(\delta)$ への関数であり，$\rho_1(\delta)$ は次のルールからなる．

(1)　δ の左辺に現われる変数 Z について，δ 中の Z の全ての出現を $f(X_1,\cdots,X_n)$ で置き換えることによって得られるルール．ここで，f は $n \geq 0$ 引数の構成子であり，$X_1,\cdots,X_n$ は δ に現われない互いに異なる新しい変数である．

(2)　δ の右辺だけに現われる変数 Z について，右辺中の Z の全ての出現を $f(X_1,\cdots,X_n)$ で置き換えることによって得られるルール．ここで，f は $n \geq 0$ 引数の関数（構成子，あるいは定義関数）であり，$X_1,\cdots,X_n$ は δ に現われない互いに異なる新しい変数である．

(3)　δ の2つの異なる変数 Z と W について，δ 中の Z の全ての出現を W で置き換えることによって得られるルール．　□

δ が変数を含まなければ，またそのときに限り $\rho_1(\delta) = \{\}$ である．$\rho_1(\delta)$ の要素を δ の ρ_1 による精密化という．定義からわかるように，δ が不適当でないルールであっても δ の ρ_1 による精密化の中には不適当なルールが出てくる．

【例 3．1】　ルール $times(X,Y) \triangleright Z$ の ρ_1 による精密化は，$times(0,Y) \triangleright Z$，$times(X,Y) \triangleright Y$，$times(sX,Y) \triangleright Z$ などである．精密化演算子の動作を精密化グラフと呼ばれるグラフで表わすことができる．ρ_1 によって導かれる精密化グラフの一部を下に示す．

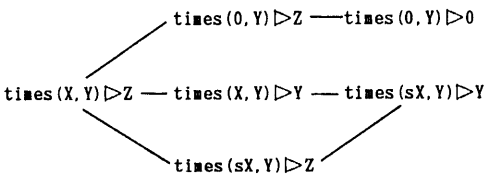

図 3．1　精密化グラフ　□

精密化演算子 ρ_1 が漏れのない枚挙をするということを示すのが，次の命題である．

【命題 3．1】　ρ_1 はルール全体の集合に対して次の意味で完全である．すなわち，【仮定】を満足する不適当でない任意のルール $f(l_1,\cdots,l_n) \triangleright r$ は，基本となるルール $f(X_1,\cdots,X_n) \triangleright Y$ から，ρ_1 による精密化を有限回繰り返して作ることができる．

（証明）　任意のルール δ について，$\rho_1(\delta)$ が有限であることと，ループする精密化がないこと（精密化グラフが非循環であること）は明らかである．$f(l_1,\cdots,l_n) \triangleright r$ 中の変数の各出現をすべて異なる新しい変数で置き換えたルールを $f(l_1',\cdots,l_n') \triangleright r'$ とする．$f(l_1,\cdots,l_n) \triangleright r$ は，$f(l_1',\cdots,l_n') \triangleright r'$ から定義 3．5 の規則(3)を何回か適用することによって得られる．また，$f(l_1',\cdots,l_n') \triangleright r'$ が $f(X_1,\cdots,X_n) \triangleright Y$ から定義 3．5 の規則(1)，(2)を何回か適用して作られることは，項 $f(l_1',\cdots,l_n')$，r' の構造に関する帰納法で示すことができる．　□

次に，ルールの枚挙が簡単な順番で並び要求に応じて作られるように，精密化演算子 ρ_1 を用いて R_h を作る手続きを示す．

【アルゴリズム 3．3】　精密化演算子 ρ_1 による R_h の生成手続きを以下に示す．

(0)　ルールの線形順序集合を表わす R_0，R_h を用意し，$R_0 = R_h = \{\}$ と置く．

(1)　R_0 に $d(X_1,\cdots,X_n) \triangleright Y (n \geq 0)$ の形をしたルールを，各 $d \in D$ について1つづつ順番に入れる．ここで，n は d の引数の数を表わす．

(2)　R_0 からまだマーク付けされていず，最も前に位置するルール δ を選び，δ にマークを付ける．

(3)　δ の ρ_1 による精密化のすべてを R_0 の後ろに付け加える．ただし，その精密化がすでに R_0 に存在しているならば付け加えない．変数名の付け替えによって同じになるルールは，同じものとみなす．

(4)　δ が変数条件を満たすならば，δ を R_h の後ろに付け

加える．変数条件を満足しなければ，R_h には付け加えない．

(5)　(2)から(4)までを繰り返す．　□

3．6　合成アルゴリズム

以上で述べた三つの手続きと精密化演算子を用いて，目的の合成アルゴリズムを述べる．アルゴリズムについて述べる前に，例を用いてアルゴリズムの動作を説明する．

現時点までに合成された停止性を満たす TRS を

R_c：　(1) plus(X, ssY) ▷ plus(sX, plus(s0, Y))

(2) plus(ssX, Y) ▷ ss(plus(X, Y))

(3) plus(X, 0) ▷ X

(4) plus(sX, s0) ▷ ssX

(5) plus(sX, X) ▷ sX

とする．R_c は今まで読み込んだ事実について正しく振る舞うとする．そこで，アルゴリズムは次の事実 plus(s0, ss0)) ＝ssss0 を読み込み，この事実に対しても R_c が正しく振る舞うかどうかをチェックする．初めに，事実の左辺 t＝plus(s0, ss0))を根とする　R_c による簡約木を求める．R_c は停止性を満たすので，簡約木は必ず有限である．実際求めると次の簡約木を得る．

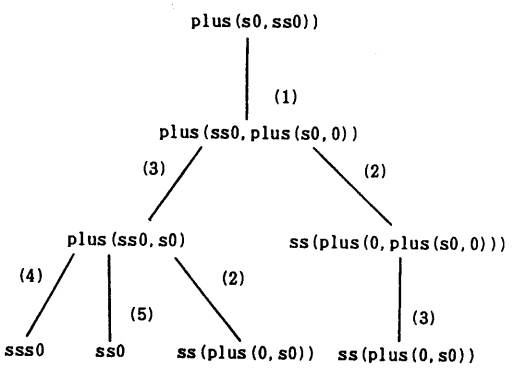

図 3．2　簡約木

木の先端つまり葉は，根の正規形になっている．合成アルゴリズムでは，それぞれの葉について，葉がデータ項でなければアルゴリズム 3．2 を適用して新しいルールを R_c に追加する．今の場合，項 ss(plus(0, s0))がその例である．また，データ項であるが Val(t)でない場合はアルゴリズム 3．1 を適用して，根からその葉までを簡約するのに使われたルールから偽なルールを探し出して R_c から削除する．今の場合，項 ss0 がその例で，その項を得るまでに用いられたルール(1)，(3)，(5)の中に偽のルールがあり，それを削除する．もしすべての葉が Val(t)と一致すれば，次の事実を読み込み，同様な処理を行なう．

次に代数 Val を実現する TRS の合成アルゴリズムを述べる．アルゴリズムにパラメータとして与えられる R_h は，アルゴリズム 3．3 によって精密化演算子 ρ_1 から作られるルール全体の線形順序集合である．

【アルゴリズム 3．4】　合成アルゴリズム

パラメータ：　ルール全体の線形順序集合 R_h

停止性を保証する簡約順序≻

入力：　対象となる Val の枚挙 $t_1 = s_1$，$t_2 = s_2$，…

対象となる Val についての神託

出力：　推測の TRS プログラム R_1，R_2，… の列

(R_k は k 個までの事実に対して正しく振る舞う)

アルゴリズム：

```
k ← 0
Rc ← {}  (現在までの推測を表わす TRS を {} にする)
repeat
  k ← k + 1
  次の事実 tk = sk を読む
  repeat
    現在までに読み込んだ事実 t=s を 1 つ選び，Rc による t の簡約木 G(t)を構成する．
    repeat
      G(t)の 1 つの葉 (t の正規形) t↓ を選ぶ．
      if  t↓がデータ項でない
        then  アルゴリズム 3．2 により，l≻r となる新しいルール l▷r を Rh から見つけて Rc に付け加える．
        else if  t↓ ≠ s
               then  アルゴリズム 3．1 により，t を t↓まで簡約するのに用いたルールの中から偽のルールを見つけだし，Rc から取り除く．
               else  Rc はこの葉については正しい．次の葉を調べる．
    until  t の全ての正規形 t↓について t↓ ≡ s．
  until  現在までに読み込んだ全ての事実 t=s の左辺 t を根とする簡約木 G(t)の全ての葉 t↓について t↓ ≡ s となる．
  Rc を Rk として出力する
forever
```

合成アルゴリズムは永久に続く手続きであるから，その

正当性は極限においての収束という概念を用いて示される．

【定義３．６】　Val の枚挙が与えられたとき，枚挙上の事実を順番に読み込みプログラムの推測を出すようなアルゴリズムが，ある時点以降プログラム R とは異なる推測を二度と出力しなければ，そのアルゴリズムは，Val の枚挙上で R に極限において収束するという．　□

アルゴリズム３．４の正当性は，次の定理で保証される．

【定理３．１】　アルゴリズム３．４の仮定のもとで，簡約順序≻のもとで停止し Val を実現する TRS が存在するならば，アルゴリズム３．４は，Val を実現する TRS に極限において収束する．　□

（証明）　R_h の n 番目までのルールの集合を $R_h(n)$で示す．仮定より，簡約順序≻のもとで停止し，Val を実現する TRS が（一般に複数）存在する．そこで，n_o を $R_h(n)$が簡約順序≻のもとで停止し，Val を実現する TRS のルールを全て含む最小の n とする．$R_h(n_o)$に含まれ，簡約順序≻のもとで停止し，Val を実現する TRS を R_oとする．

（一時中断）

定理の証明を一時中断し，定理を示すためにいくつかの命題と補題を準備する．

そこで以下では，合成の対象となる Val，ルール全体の線形順序集合 R_h（精密化演算子 ρ_1 により，アルゴリズム３．３から作られる），簡約順序≻が与えられているとする．また，簡約順序≻のもとで停止し Val を実現する TRS が存在するとし，上で述べた n_o と R_o を仮定する．アルゴリズムから分かるように，アルゴリズム動作中，現時点までの TRS の推測を表わす R_c は色々変化する．そこで，アルゴリズム動作中の時点を τ，τ_0，τ_1 などで表わし，動作中の時点 τ における R_c を特に表現したいときには，$R_c(\tau)$で表わす．

【命題３．２】　アルゴリズム３．４動作中のどの時点 τ においても，推測 $R_c(\tau)$は停止性を満たす．

（証明）　初めは $R_c=\{\}$ であり，アルゴリズム動作中 R_c に付け加えられるルール $l\triangleright r$ はアルゴリズム３．２から得られるルールだけである．よって，$l\succ r$ を満たす．半順序≻は簡約順序であるから，どの時点 τ においても，推測 $R_c(\tau)$は停止性を満たす．　□

上の命題から，アルゴリズムが出力する TRS R_k は，すべて停止性を満たす．この命題より，基礎項の簡約木について次の系が成り立つ．

【系３．１】　アルゴリズム３．４動作中どの時点 τ においても，$R_c(\tau)$による基礎項 t の簡約木 G(t)は必ず有限である．

（証明）　τ をアルゴリズム動作中の任意の時点とし，$R_c(\tau)$による基礎項 t の簡約木を G(t)とする．命題３．１より $R_c(\tau)$は停止性を満たすので，$R_c(\tau)$による長さ無限の簡約列は存在しない．また，$R_c(\tau)$は有限集合であり，どの項に対しても $R_c(\tau)$によるリデックスの数は有限．従って，Konig の補題[10]より G(t)は必ず有限である．　□

【補題３．２】　アルゴリズム３．４動作中のある時点 τ_0 で，$R_c(\tau_0)$が Val を実現する TRS R を含んでいれば，τ_0 以降のどの時点 τ においても，R_c に新しいルールが追加されることはない．

（証明）　アルゴリズム動作中のある時点 τ_0 で，$R_c(\tau_0)$が Val を実現する TRS R を含んでいたとする．R は定義から真なルールの集合であり，命題３．１より真なルールが R_c から取り除かれることはないから，τ_0 以後のどの時点 τ においても R に属するルールが R_c から取り除かれることはない．そこで，τ_0 以降のある時点 τ_2 で R_c に新しいルールが追加されたとする．R_c に新しいルールが追加される前の時点を τ_1 とする．アルゴリズム３．４より，これは既に読み込んだ事実 t=s の左辺 t の $R_c(\tau_1)$ による正規形の中で，まだデータ項になっていない正規形 $t\downarrow$があることを意味する．$R\subset R_c(\tau_1)$ であり，R は Val を実現する TRS であるから，$t\downarrow$をデータ項 $Val^+(t\downarrow)$まで書き換える $R_c(\tau_1)$ による簡約列が存在する．このことは，$t\downarrow$が $R_c(\tau_1)$ による正規形であることに反する．従って，$R\subset R_c$ まで達したら，R_c に新しいルールが追加されることはない．　□

【補題３．３】　定理３．１の仮定のもとで，アルゴリズム３．４動作中，アルゴリズム３．２が起動されたとき，アルゴリズム３．２は，必ず停止してあるルール $l\triangleright r$を見つけだす．

（証明）　アルゴリズム３．２が起動され，項 t の部分項で T_o に含まれる項 s にマッチするルールを捜しているとする．R_o は Val の実現であるから，s にマッチするルールが必ず $R_o\subset R_h$ にある．このようなルールのうちいちばん最初に現われるルールを $l\triangleright r$ とする．R_o のルールは取り除かれないので，$l\triangleright r$ にマークが付けられていたとするとそれは R_c にあることになり，アルゴリズム３．２が起動されたという仮定に反する．よって，$l\triangleright r$ にはマークが付いていない．アルゴリズム３．２が起動されると，$l\triangleright r$ は s にマッチし，$l\succ r$ を満たすから，アルゴリズムはルール $l\triangleright r$ を出力して終了する．　□

【補題３．４】　定理３．１の仮定のもとで，アルゴリズム３．４を Val の枚挙に適用すると，枚挙上の任意の事実

はいつかは読み込まれる．

（証明）　補題を示すためには，$k \geqq 1$ 番目の事実 $t_k = s_k$ を読み込んだら，アルゴリズムは k 個の事実による推測 R_k を必ず出力することを示せば十分．

そこで，アルゴリズムは $k \geqq 1$ 番目の事実 $t_k = s_k$ を読み込んだとする．今まで読み込んだ事実の1つを $t = s$ と置く．系3．1より t の簡約木 G(t)は有限であるから，G(t)の構成手続きは必ず終了する．G(t)の葉を調べる過程で，アルゴリズム3．1とアルゴリズム3．2が呼ばれることがある．これらのあるゴリズムは，それぞれ補題3．1と補題3．3により必ず終了し，アルゴリズム3．1が起動されたときは，R_c から偽なルールが取り除かれ，アルゴリズム3．2が起動されると，R_c に新しいルールが付け加わる．仮定から $R_o \subset R_h(n_o)$ であり，1回取り除いたルールは復活しないから，補題3．1と3．2より，アルゴリズム3．4の動作中，R_c は高々 $2n_o$ 回しか変化しない．従って，この2つのあるゴリズムが起動されるのは，高々 $2n_o$ 回であり，いつかは R_c が変化しなくなる．G(t)の葉は有限個であり，今まで読み込んだ事実は k 個であるから，すべての事実を調べる手続きも終了する．このことは，現在までに読み込んだ全ての事実 $t = s$ の左辺 t を根とする簡約木 G(t)の全ての葉 $t\downarrow$ について $t\downarrow \equiv s$ となることを意味する．よって，一番外側の until の条件が満たされて，アルゴリズムは k 個の事実による推測 R_k を必ず出力する．　□

補題3．4より，全ての事実はいつかは読み込まれることが分かった．アルゴリズムより明らかに，k 個の事実から推測された R_k について，次の命題が成り立つ．

【命題3．3】　アルゴリズム3．4により，k 個の事実から推測される TRS R_k に関して，今まで読み込まれた任意の事実 $t = s$ に対して，$t \xrightarrow{=} s$ が成り立ち，さらに，$t \xrightarrow{=} t\downarrow$ ならば $t\downarrow \equiv s$ が成り立つ．　□

【補題3．5】　任意の基礎項 t について，ある $k \geqq 0$ があって，アルゴリズム3．4によって k 個の事実から推測される R_k について，$t \xrightarrow{=} Val^+(t)$ が成り立つ．

（証明）　t に含まれる定義関数記号の数に関する帰納法で示す．

t が定義関数記号を含まないデータ項であるならば，0個の事実から推測される R_0 について，明らかに $t \xrightarrow{=} Val^+(t)$ $(\equiv t)$ が成り立つ．

t が n+1 個の定義関数記号を含む基礎項とする．t の一番内側に位置し T_o に属する部分項 u を1つ勝手に選ぶ．事実 $u = s$ が与えられた枚挙で m 番目に現われるとする．命題3．3から，m 個の事実から推測される R_m について，$u \xrightarrow{=} s$ $(\equiv Val^+(u))$ が成り立つ．t の部分項 u を s で置き換えた項 t[s] は，k 個の定義関数記号を含む基礎項である．従って，帰納法の仮定より，ある $k' \geqq 0$ があって，k' 個の事実から推測される $R_{k'}$ について，$t[s] \xrightarrow{=} Val^+(t[s])$ が成り立つ．m と k' のうち大きい方を $k = max\{m, k'\}$ とすると，k 個の事実から推測される R_k について，$(t \equiv)\ t[u] \xrightarrow{=} t[s] \xrightarrow{=} Val^+(t[s])\ (\equiv Val^+(t))$．　□

【補題3．6】　アルゴリズム3．4動作中の任意の時点 τ で，$R_c(\tau)$ が Val に関して真でなければ，τ 以後のある時点でアルゴリズム3．1が必ず呼び出され，R_c から偽なルールが取り除かれる．

（証明）　アルゴリズム動作中のある時点 τ_0 で，$R_c(\tau_0)$ が Val に関して真ではなく，τ_0 以後アルゴリズム3．1は決して呼び出されないとする．従って，τ_0 以後 R_c から偽なルールが取り除かれることはない．$l \rhd r \in R_c(\tau_0)$ を偽なルールとすると，ある基礎代入 θ について，$Val(l\theta) = Val^+(l\theta) \neq Val^+(r\theta)$．$r\theta$ は基礎項であるから，補題よりある $k \geqq 0$ があって，k 個の事実から推測される R_k について，$r\theta \xrightarrow{=} Val^+(r\theta)$ が成り立つ．事実 $l\theta = Val^+(l\theta)$ が枚挙上で m 番目に現われるとし，$k' = max\{k, m\}$ と置く．τ_0 以後アルゴリズム3．1は呼び出されないから，k' 個の事実から推測される TRS $R_{k'}$ について，$l \rhd r \in R_{k'}$．また，$R_k \subset R_{k'}$ であるから，既に読み込まれた事実 $l\theta = Val^+(l\theta)$ について，$R_{k'}$ のもとで $l\theta \rightarrow r\theta \xrightarrow{=} Val^+(r\theta) \neq Val^+(l\theta)$．これは命題3．3に反する．　□

【系3．2】　アルゴリズム3．4動作中の任意の時点 τ で，$R_c(\tau)$ 中の偽なルール $l \rhd r$ はいつかは R_c から取り除かれる．

（証明）　アルゴリズム動作中の任意の時点を τ とし，$R_c(\tau)$ は偽なルール $l \rhd r$ を含んでいるとする．$R_o \subset R_h(n_o)$ であり，補題3．2から R_c が R_o を含むと R_c には新しいルールが追加されることはないので，τ 以後 Rc から偽なルールが削除されるのは高々 n_o 回である．また，補題3．6より Rc が偽であるときは必ずアルゴリズム3．2が適用されて偽なルールが削除されるから，いつかは偽なルール $l \rhd r$ が削除される．　□

【補題3．7】　アルゴリズムの動作が進むと，必ず $R_o \subset R_c$ の状態まで達する．

（証明）　$R_o \subset R_c$ の状態まで達せず，ある時点 τ_0 以降これ以上新しいルールが追加されないと仮定する．系3．2より，R_c が真でなければ偽なルールはいつか削除されるので，τ_0 以降のある時点点 τ_1 で $R_c(\tau_1)$ は真なルールだけからなる．$R_c(\tau_1)$ はまだ Val の実現にはなっていない

から，ある事実の t=s の左辺 t を簡約したときの正規形（R_c は真なルールの集合であるから，s と異なるデータ項になることはない）には定義関数記号が現われる．よって，新しいルールが R_h から追加されることになる．これは，τ_0 以降新しいルールが追加されないという仮定に矛盾する．　□

以上定理３．１を証明する準備が全て終わったので，これらの補題を用いて定理を証明する．

定理３．１の証明の続き：

n_o を $R_h(n)$が簡約順序≻のもとで停止し，Val を実現する TRS のルールを全て含む最小の n とし，$R_h(n_o)$に含まれ，簡約順序≻のもとで停止し，Val を実現する TRS を R_o とした．補題３．７と３．２より，アルゴリズムの動作が進行していくと，ある時点 τ_0 で $R_o \subset R_c(\tau_0)$を満たし，τ_0 以降 R_c に新しいルールが付け加えられることはない．また，系３．２より R_c が偽なルールを含めばいつかは削除されるから，ある時点 τ_1 で $R_c(\tau_1)=R_o$ となりこれ以後 R_c は変化しない．従って，τ_1 以降アルゴリズムは R_o を出し続けるようになる．すなわち，アルゴリズム３．４は，Val を実現する TRS に極限において収束する．　□

４．　おわりに

本論文では，入出力の例から，帰納的推論によって代数を実現する TRS プログラムを合成するアルゴリズムを与え，その正当性を示した．簡約順序≻のもとで停止し，代数を実現する TRS プログラムが存在するならば，このアルゴリズムは，このような代数を実現する TRS プログラムに極限において合成することができる．

例題からプログラムを合成する場合，可能性があるプログラムの候補を全て列挙し，例題に合う候補をプログラムとして出力するのでは余りにも原始的である．時間が掛かりすぎて，現実的規模のプログラムを合成するには至らない．明らかにそぐわない候補をばっさり刈り取る方法論や組織的に候補を絞り込む発見的なメカニズムが必要となる．プログラムの構造に応じたテンプレートを用意したり，例題を分析し構成的にプログラムの骨格を形成していく方法が不可欠となる．今後は，合成の効率化について考察していきたい．

参考文献

[1] 有川節夫，石坂裕毅「帰納的推論による自動プログラミング」，情報処理，Vol. 28, No. 10, pp 1312-1319, 1987.

[2] Biermann, A.W. "The inference of regular Lisp programs from examples", IEEE Transactions on Systems, Man and Cybernetics SMC-8, pp 585-600, 1978.

[3] 千葉，富樫，野口

[4] Darlington, J., "Program Transformation", Functional Programming and its Application, Cambridge Univ. Press, pp. 193-215, 1982.

[5] Dershowitz, N., "Termination of Rewriting", J. Symbolic Computation 3, pp 69-116, 1987.

[6] 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.

[7] Huet, G. "Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems", J. ACM, Vol.27, No.4, pp 797-821, 1980.

[8] Huet, G. and Oppen, D.C. "Equations and Rewrite rules: a Survey", Formal Languages: Perspectives and Open Problems, Ed. Book R, Academic Press, pp 349-393, 1980.

[9] 大特集：自動プログラミング，情報処理，Vol.28, No. 10, 1987.

[10] Knuth, D.E., "The Art of Computer Programming", Addison-Wesley, Reading , Masachusetts, 1968.

[11] Manna, Z., Waldinger, R., A Deductive Approach to Program Synthesis, TOPLAS, Vol. 2, No. 1, pp. 90-121, 1980.

[12] Shapiro, E.Y. "Inductive Inference of Theories From Facts", Technical Report 192, Yale Univ., 1981. (有川節夫訳「知識の帰納的推論」，共立出版，1986)

[13] Shapiro, E.Y. "Algorithmic Program Debugging", Ph. D. Thesis, The MIT Press, 1982.

[14] Summers, P.D. "A methodology for LISP program construction from examples", Journal of the ACM 24, pp 161-175, 1977.

[15] Togashi, A. and Noguchi, S. "A Program Transformation from equational Programs into Logic Programs", J. Logic Programming, 4, pp 85-103, 1987.