ICOT Technical Report: TR-554

TR-554

項書換えシステムと完備化手続き

大須賀昭彦(東芝)

April, 1990

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

項書換えシステムと完備化手続き

(株) 東芝 システム・ソフトウェア技術研究所
大須賀昭彦

e-mail: ohsuga@sscl.toshiba.co.jp

要旨

本稿では項書換えシステムについて解説する．項書換えシステムとは，書換え規則と呼ばれる式の集合によって計算を定義する数学的なモデルである．これを関数型言語など，等式に基づくプログラミング言語の計算モデルに採用すると，計算の意味や性質の解析，プログラムの変換・検証といった技術を数学的に捉え，厳密に議論することが可能となる．

1 はじめに

近年における計算機処理能力の向上と安価なハードウェアの出現は，自然な流れとしてソフトウェアに対する要求の多様化や高度な知識情報処理への期待をもたらした．しかしながら，FORTRAN,COBOL,C,PASCAL や PL/I などに代表されるいわゆる手続き型のプログラミング言語には，これらに応えるソフトウェアの開発に十分対処できない面がある．手続き型言語によるプログラムは，あらっぽく言えば，データの取出し／加工／格納指示とそれらの実行順序記述の集まりであるが，実行時において，実行の順序は格納されたデータ値に依存して決定され，実行順序の変化は格納するデータ値に影響を及ぼすというように両者が複雑に絡んで計算が進められる．それゆえプログラムの動きは静的に捉えにくく，大規模化の進むソフトウェアに対して全体的な正しさを保証することは，ほとんど不可能となっている．また，計算機を「いかに動かすか」という記述が中心の言語では，高度な処理の対象が持つ論理的性質を十分表現しきれないことも事実である．

こういった状況をふまえ，これまでと違う新しいパラダイムに基づくプログラミング言語が最近次々と提案されている．これらの言語に共通した特徴は，ベースとなる論理体系が明確に定まっていることである．このことによって，演繹的手法によるプログラムの検証が可能となり，また「何をするのか」という宣言的な記述によりプログラムが簡潔で読み易いものになるなど，手続き型言語の弱点を補うことができる．しかし，新しい言語がこれらの利点を生かして従来の言語に置き換わるためには，言語に適した計算機環境や開発支援環境の整備が必要である．なぜなら，これらの言語を従来の計算機上に実現しても，計算機の動

きを直接的に指示する手続き型言語に比べ実行効率が劣ることは避けがたく，また従来のデバッグ・テスト支援環境を転用しても，言語の特徴を十分に生かすことができないからである．言語がいくら検証可能であっても，具体的な検証支援系がなければ実用的に意味のないことが，その典型例である．

このような面から比較的見通しのよいアプローチの1つとして，ここでは項書換えシステムを考える．これは，(1) 項書換えシステムにおいては「参照の透明さ」(referential transparency) により計算の並列化が容易で，これに基づく並列計算機環境を構築すれば実行効率の向上が見込める．(2) 項書換えシステムは古くから理論・実用の両面にわたり研究の行なわれてきた数学的な技術で，その豊富な技術蓄積を開発支援環境の構築において活用できる．つまり，我々は項書換えシステムに基づくプログラミング言語に関して，既に数学を通してかなりの経験と知識を持っていることになるなどの理由からである．

まず，等式による仕様の記述と計算がいかにして結びつくかを具体例により説明し，第3節, 第4節で，計算における2つの重要な性質，停止性と合流性について触れ，第5節において，仕様の論理的な意味を変えずにこれらの性質を併わせ持つ理想的なシステムへ変換するKnuth–Bendix (クヌス・ベンディクス) 完備化手続きについて解説する．最後に第6節では，プログラム検証など，論理と計算の境界に位置するような問題に対しこの完備化手続きがいかに応用できるかを簡単に紹介する．

2 項書換えシステム

まず，簡単な項書換えシステムの例をみてみよう．リストの反転や2つのリストを結合するプログラムの仕様は，次のような等式の集合 E_{list} で記述できる．

$$
\begin{aligned}
E_{list}: \quad reverse([\,]) &= [\,] && \text{(e1)}\\
reverse([X|Y]) &= append(reverse(Y),[X]) && \text{(e2)}\\
append([\,],X) &= X && \text{(e3)}\\
append([X|Y],Z) &= [X|append(Y,Z)] && \text{(e4)}
\end{aligned}
$$

ここで，$reverse, append$ は関数記号，X, Y, Z は変数である．また，$[\,]$ は空リスト，$[X|Y]$ は $cons(X,Y)$ などと同様のリスト表現である．項書換えシステムでは，こういった関数や変数，リストなどの式をすべて項 (term) と呼ぶ．2つの項を '=' で結んだこれらの等式は，たとえば (1) は，$[\,]$ の反転は $[\,]$ に等しい，(2) は，$[X|Y]$ の反転は Y の反転に $[X]$ を結合したものに等しいといった関係を表わしている．

このとき次の項について考える．なお，$[a,b]$ はリスト $[a|[b|[\,]]]$ の略記である．

$$reverse([a,b])$$

この項は，E_{list} の等式を使って以下のように変形できる．

$$\begin{aligned}
&reverse([a,b]) \quad (\equiv \underline{reverse([a|[b]])}) && \text{(e2) を適用}\\
&= \overline{append(\underline{reverse([b])},[a])} && \text{(e2) を適用}\\
&= append(\overline{append(\underline{reverse([\,])},[b])},[a]) && \text{(e1) を適用}\\
&= append(\underline{append([\,],[b])},[a]) && \text{(e3) を適用}\\
&= \underline{append([b],[a])} && \text{(e4) を適用}\\
&= \overline{[b|\underline{append([\,],[a])}]} && \text{(e3) を適用}\\
&= [b|\overline{[a]}] \quad (\equiv [b,a])
\end{aligned}$$

ここで，$\equiv$ は左右の項が構文的に同じものであることを，下線は等式を適用した場所を，上線は等式による変形の結果を表わしている．たとえば，最初の 2 行によって，$reverse([a,b])$ は $reverse([a|[b]])$ の略記であり，この項に等式 (e2) を適用した結果は，$append(reverse([b]),[a])$ になることがわかる（(e2) 左辺の変数 X に a を，Y に $[b]$ を割り当てて考えよ）．このようにして等式による項の変形をうまく繰り返すと，$reverse([a,b])$ に対し反転されたリスト $[b,a]$ を得ることができる．ここでは，こういった項変形を等号論理の推論といい，推論により結ばれる関係を等価 (equivalent) な関係という[†1]．見方を変えれば，等号論理の推論は，関数 $reverse$ の引数に $[a,b]$ を与えたものから解として $[b,a]$ を得る一種の計算である．しかも，この計算において等式は左から右への方向を持った規則として用いられている．そこで，等式を単なる関係としてではなく，初めから方向を持った**書換え規則** (rewrite rule) としてみると，その書換え規則の集合によってある種の計算を規定することができる．このような**書換え規則の集合を項書換えシステム** (term rewriting system; 以降 TRS と略す) という．

[†1] 一般的な定義によれば，等号論理の推論は以下の推論規則によって定義される．つまり，λ と ρ が等価であるためには，公理からこの推論規則によって $\lambda = \rho$ が導けることが必要十分である．任意の項 τ, υ, σ について

(1) $\tau = \tau$ （反射律）

(2) $\tau = \upsilon$ ならば $\upsilon = \tau$ （対称律）

(3) $\tau = \upsilon, \upsilon = \sigma$ ならば $\tau = \sigma$ （推移律）

(4) 任意の代入 Θ について，$\tau = \upsilon$ ならば $\Theta(\tau) = \Theta(\upsilon)$ （代入律）

(5) 任意の関数 f について，$\tau = \upsilon$ ならば $f(\ldots\tau\ldots) = f(\ldots\upsilon\ldots)$ （置換律）

ここで，$\Theta(\tau)$ は τ 中の変数に適当な値を割り当てたものを表わす．

書換え規則は，その方向性を明示するために2つの項を矢印'$\rightarrow$'で結んで表現する．すると，先の等式集合 E_{list} からは次のような TRS R_{list} が得られる．

$$
\begin{aligned}
R_{list}: \quad reverse([\,]) &\rightarrow [\,] && \text{(r1)}\\
reverse([X|Y]) &\rightarrow append(reverse(Y),[X]) && \text{(r2)}\\
append([\,],X) &\rightarrow X && \text{(r3)}\\
append([X|Y],Z) &\rightarrow [X|append(Y,Z)] && \text{(r4)}
\end{aligned}
$$

これらの書換え規則は「左辺の項を見つけたならば，右辺の項で置き換えよ」という操作を表わしており，通常，右辺は何らかの意味で左辺より単純なものになっている．また，X,Y,Z は変数なので任意の値を割り当てて考えてよい．こういった書換え規則を適用し，項をより簡単なものへ変形する計算のことを**リダクション** (reduction) といい，項 τ を υ にリダクションすることを $\tau \Rightarrow \upsilon$ で表わす．すると，R_{list} による $reverse([a,b])$ のリダクション列は，E_{list} による推論列の'$=$'を'$\Rightarrow$'に置き換えたものとなる．

$$
\begin{aligned}
&\underline{reverse([a,b])} && \text{(r2) を適用}\\
&\Rightarrow \overline{append(\underline{reverse([b])},[a])} && \text{(r2) を適用}\\
&\Rightarrow append(\overline{append(\underline{reverse([\,])},[b])},[a]) && \text{(r1) を適用}\\
&\qquad\vdots\\
&\Rightarrow [b,\overline{a}]
\end{aligned}
$$

$[b,a]$ のようにそれ以上リダクションできない項を**既約項** (irreducible term) という．既約項は，リダクションの結果，すなわち計算の解に相当する項である．また，リダクションの各ステップに興味がない場合，$reverse([a,b]) \stackrel{*}{\Rightarrow} [b,a]$ といった記述で0回以上の任意のリダクションを表わすことにする．

ここで，TRS の計算能力について簡単に触れておくと，計算理論上はすべての計算可能な関数が TRS で実現できることが知られている[†2]．これは，TRS がプログラム言語の計算モデルとして十分な表現能力を持つことを表わしている．

3 TRS の停止性

ここで，TRS によるリダクションの性質について考える．E_{list} の例で，もし $reverse([\,])$ の方が $[\,]$ よりも単純であると判断し，R_{list} において $[\,] \rightarrow reverse([\,])$ なる書換え規則を獲

[†2] TRS が部分再帰的関数 (partial recursive function) をシミュレートできることが示せる．

得したらどうなるであろうか. [] をこの規則でリダクションすると,

$$[] \Rightarrow reverse([]) \Rightarrow reverse(reverse([])) \Rightarrow \cdots$$

なる無限列となり, 明らかにリダクションは止まらない.

他のこれほど明らかでない例をみてみよう. コンビネータによる計算も等式によって表現可能である. たとえば, S, K コンビネータの仕様は次の E_{SK} で与えられる.

$$\begin{aligned} E_{SK}: \quad sXYZ &= XZ(YZ) \quad \text{(e1)} \\ kXY &= X \quad \text{(e2)} \end{aligned}$$

ここでは左結合を仮定して括弧を省略した. つまり, $sXYZ$ は $(((s\ X)\ Y)\ Z)$ の略記である. すると, (e1) が S コンビネータの, (e2) が K コンビネータの定義となっている. これを左から右へ向きづけると, 次の TRS R_{SK} を得る.

$$\begin{aligned} R_{SK}: \quad sXYZ &\to XZ(YZ) \quad \text{(r1)} \\ kXY &\to X \quad \text{(r2)} \end{aligned}$$

この R_{SK} によって項 $ssk(ssk)(ssk)$ をリダクションすると, 以下のようになる.

$$\begin{aligned} & \underline{ssk(ssk)}(ssk) && \text{(r1) を適用} \\ \Rightarrow\ & \overline{\underline{s(ssk)(k(ssk))}(ssk)} && \text{(r1) を適用} \\ \Rightarrow\ & \overline{ssk(ssk)(\underline{k(ssk)(ssk)})} && \text{(r2) を適用} \\ \Rightarrow\ & ssk(ssk)\overline{(ssk)} \end{aligned}$$

まだリダクションの途中であるが, 最後に得られた項は最初に与えた項そのものであり, これ以降リダクションを続けてもこの繰り返しとなる.

TRS R において, 任意の項について上のような無限のリダクション列が存在しなければ, R は停止性 (termination) を持つという. ここまでは, 等式を直観的に複雑な項から単純な項へ向きづけてきた. しかし, そうして得られた TRS が停止性を持つか否かを判定するには, もう少し客観的な基準が必要である. 残念なことに, この判定は決定不能, つまり任意の TRS が停止性を持つかどうかを一般的に調べる手立てはないことが理論的に知られている. しかし, 「具体的に与えられた等式をどちら向きの書換え規則にすれば, TRS の停止性が保証されるのか」という実際の問題に対し, 一部でも解くことのできるメカニズムがあれば実用上十分役に立つ. これに関する研究は進んでいるので, 主なものを紹介する.

3.1 意味順序

ある方法で項に対して自然数を対応づけたとしよう．その自然数を項の重みと呼ぶと，Rによる任意のリダクション$\tau \Rightarrow \upsilon$について常に$\upsilon$の重みが$\tau$の重みより小さくなるようなら，$R$が停止性を持つのは明らかである．そこで，等式が与えられたときに何らかの方法で両辺の重みを比較し，重みが小さくなる方向に向きをつけることを考える．これによって得られた書換え規則は，任意のリダクションについて項の重みを減少させるだろうか．まず，次のような方法を考える．

【例】(項の長さによる重みづけ)

$$E_{nat}: \quad f(f(X)) = g(X) \quad (e1)$$
$$g(g(X)) = f(X) \quad (e2)$$

からTRSを得るために，各項τに次の自然数$[\![\tau]\!]_{nat}$を対応させる．

$$[\![\tau]\!]_{nat} = (\tau\text{に出現する関数記号の数} + \text{定数の数} + \text{変数の数})$$

たとえば，(e1)に関して両辺の値を調べると，

$$[\![f(f(X))]\!]_{nat} = (2+0+1) = 3$$
$$[\![g(X)]\!]_{nat} = (1+0+1) = 2$$

となり，[左辺]$_{nat}$ > [右辺]$_{nat}$により，

$$f(f(X)) \to g(X)$$

を得る．(e2)も同様に向きづけると，TRS R_{nat}が得られる．

$$R_{nat}: \quad f(f(X)) \to g(X) \quad (r1)$$
$$g(g(X)) \to f(X) \quad (r2)$$

ここで，R_{nat}によるリダクションが項の重みを小さくすることを確認しよう．以下に，項$f(f(g(f(a))))$のリダクション列と$[\![\ ]\!]_{nat}$によるその重みを示す．

リダクションされる項	重み	
$\underline{f(f(g(f(a))))}$	5	(r1)を適用
$\Rightarrow \overline{\underline{g(g(f(a)))}}$	4	(r2)を適用
$\Rightarrow \overline{\underline{f(f(a))}}$	3	(r1)を適用
$\Rightarrow \overline{g(a)}$	2	

この方法は直観的にわかりやすく，停止性保証のしくみを大まかに理解する上でよい助けとなる．しかし，このように単純な方法では，書換え規則の停止性を保証できないことがすぐにわかる．たとえば，書換え規則 $f(f(f(X))) \to g(X,X)$ は，この方法によれば任意のリダクションについて項の重みを小さくするはずである．しかし，$f(f(f(\text{“大きな項”})))$ なる項をこの規則でリダクションすると，$g(\text{“大きな項”},\text{“大きな項”})$ を獲得し全体として項の重みを増してしまう．この例から，書換え規則中に現れる変数の扱いに注意が必要なことがわかる．そこで，変数にどのような値が代入されても重みが安定するように，各項に多項式を対応させて考える．

【例】(多項式による重みづけ)

$$E_{pol}: \quad p(p(X,Y),Z) \;=\; p(X,p(Y,Z)) \quad \text{(e1)}$$

が与えられたとき，各項 τ に多項式 $[\![\tau]\!]_{pol}$ を対応させる．

$$\begin{aligned} [\![p(X,Y)]\!]_{pol} &= [\![X]\!]_{pol} * [\![Y]\!]_{pol} + [\![X]\!]_{pol} \\ [\![\text{変数}]\!]_{pol} &= \text{変数} \end{aligned}$$

すると，(e1) 両辺の $[\![\ ]\!]_{pol}$ による解釈は

$$\begin{aligned} &[\![p(p(X,Y),Z)]\!]_{pol} \\ &\quad = [\![p(X,Y)]\!]_{pol} * [\![Z]\!]_{pol} + [\![p(X,Y)]\!]_{pol} \\ &\quad = ([\![X]\!]_{pol} * [\![Y]\!]_{pol} + [\![X]\!]_{pol})Z + ([\![X]\!]_{pol} * [\![Y]\!]_{pol} + [\![X]\!]_{pol}) \\ &\quad = (XY+X)Z + (XY+X) \\ &\quad = XYZ + XZ + XY + X \\ &[\![p(X,p(Y,Z))]\!]_{pol} \\ &\quad = [\![X]\!]_{pol} * [\![p(Y,Z)]\!]_{pol} + [\![X]\!]_{pol} \\ &\quad = X([\![Y]\!]_{pol} * [\![Z]\!]_{pol} + [\![Y]\!]_{pol}) + X \\ &\quad = X(YZ+Y) + X \\ &\quad = XYZ + XY + X \end{aligned}$$

となる．$X,Y,Z \geq 2$ で両辺の解釈を比較すると，

$$XYZ + XZ + XY + X > XYZ + XY + X$$

つまり,$XZ > 0$

の成立は明らかで，これにより TRS R_{pol} を得る．

$$R_{pol}: \quad p(p(X,Y),Z) \quad \rightarrow \quad p(X,p(Y,Z)) \quad (\mathrm{r1})$$

この方法は多項式によって重みづけを行なうところから**多項式解釈** (polynomial interpretation)[8]と呼ばれ，この方法で向きづけされた書換え規則はその停止性が保証される．このように，書換え規則の重みづけのためにある領域[†3]とその上の順序を設定し，項とその領域を対応させて停止性を調べる方法を一般に**意味順序** (semantic ordering) による方法という．

3.2 構文順序

我々は，意味順序による方法という，書換え規則の停止性を紙の上で客観的に示す手段があることを知った．しかし，実際の場面にこれを適用すると，人間が各項と領域の対応を定め，その領域上の不等式を解く作業には，かなりの労力を必要とすることがわかる．そのうえ，不等式がうまく解けない場合には，領域や対応の見直し・修正といった後戻りまで発生する．では，停止性保証をある程度自動化して，この手間を軽減することはできないだろうか．残念ながら，意味順序による方法は人間の直観や経験にたよって領域や対応を定めることが多く，また各領域毎にその上の不等式を解くメカニズムが必要であるなどの理由で自動化が困難である．そこで，**構文順序** (syntactical ordering) による方法を考える．これは，関数記号上に半順序を導入し，項の構造と関数記号の大きさから2項の相対的な重さを比較するもので，**単純化順序** (simplification ordering) の方法がその代表である．

単純化順序による方法は，機械的な手順で2項の比較ができることから実装も容易で，したがって複雑な項にも十分対応し，また前提となる関数記号上の半順序は項の構造から逆に推論できるなど，自動化という観点から最も有望な方法である．そのため，停止性保証の方法として今のところ最も広く用いられ，試作・実験も多く，また研究も活発に行なわれている．単純化順序の推論のしくみは多少複雑になるので，詳細は文献 [3] にゆずる．具体的な単純化順序としては，**再帰経路順序** (recursive path ordering)，**再帰分割順序** (recursive decomposition ordering)，**辞書式部分項順序** (lexicographic subterm ordering) などが知られている．ただし，停止性保証の能力については，どの方法が優位とは一概に言えないので，単純化順序でだめなときは多項式解釈を持ち出すというように，状況に応じてこれらの方法を使い分けているのが実情である．

[†3]必ずしも自然数でなくてよいが，整礎 (well founded) でなければならない．

4 TRS の合流性

次のような仕様 E_{group} を考える．これらは群の公理，つまり左単位元の存在，左逆元の存在，結合律を表わしている．

$$
\begin{aligned}
E_{group}: \qquad 0 + X &= X && \text{(e1)} \\
-X + X &= 0 && \text{(e2)} \\
(X + Y) + Z &= X + (Y + Z) && \text{(e3)}
\end{aligned}
$$

停止性を保証しながら等式の書換え規則化を行なうと，停止性を持つ R_{group} が得られる．

$$
\begin{aligned}
R_{group}: \qquad 0 + X &\to X && \text{(r1)} \\
-X + X &\to 0 && \text{(r2)} \\
(X + Y) + Z &\to X + (Y + Z) && \text{(r3)}
\end{aligned}
$$

これにより，

$$(-0 + 0) + (-0) \qquad \text{(s1)}$$

なる項をリダクションすると，次のリダクション列を得る．

$$
\begin{aligned}
&\underline{(-0 + 0) + (-0)} && \text{(r3) を適用} \\
&\Rightarrow \overline{-0 + \underline{(0 + (-0))}} && \text{(r1) を適用} \\
&\Rightarrow -0 + \overline{(-0)}
\end{aligned}
$$

項の中で書換え規則が適用できる場所（下線で示した部分）をリデックス (redex) と呼ぶと，(s1) には他のリデックスが存在する．別のリデックスを選択してリダクションを行なうと，今度は次の列を得る．

$$
\begin{aligned}
&\underline{(-0 + 0)} + (-0) && \text{(r2) を適用} \\
&\Rightarrow \overline{0} + \underline{(-0)} && \text{(r1) を適用} \\
&\Rightarrow \overline{-0}
\end{aligned}
$$

TRS では書換え規則によってのみ計算が進められるので，$-0 + (-0)$ も -0 も R_{group} における既約項，つまり共に (s1) のリダクション結果である．このように，項のリダクション結果は一つに定まるとは限らない．

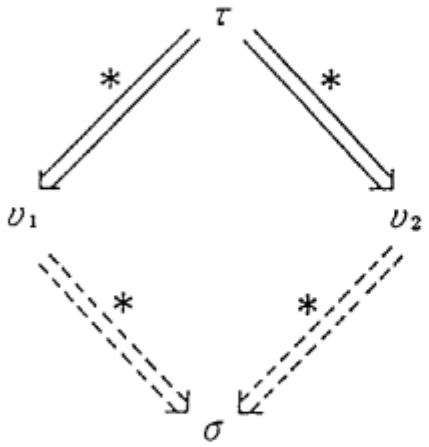

図 1　合流性

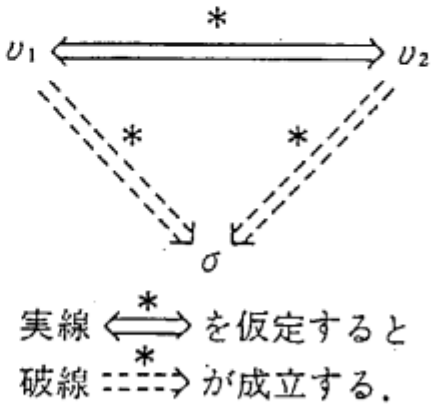

図 2　Church-Rosser 性

TRS R において任意の項からのすべてのリダクション列を考えたとき，上のようなリダクション列の分岐が発生しても，うまくその先を続けていくことによってそれらの列が互いに同じ項に到達するならば，R は合流性 (confluence) を持つという．これは，形式的には図 1のように表わされる．また，合流性は図 2の **Church–Rosser** 性とも等しい概念である．ここで，$v_1 \stackrel{*}{\Leftrightarrow} v_2$ は v_1 と v_2 が等号論理の上で等価であることを表わしている．合流性を持たない TRS では，すべての解を得るために Prolog におけるバックトラック (backtracking) のような後戻りのしくみが必要となるので，計算の効率が低下する恐れがある．これは，計算モデルとしてあまり歓迎すべきものではない．逆に，TRS が合流性を持てばどのように計算を進めても結果の一意性が保証されるので，計算順序を気にすることなく効率よく計算を進めることができる．もちろん，バックトラックのような処理は不要となる．

4.1　完備な TRS

停止性は任意の計算が必ず止まり結果が得られることに対応し，合流性は計算結果が（求まれば）一つに定まることに対応するのは既に述べた通りである．この 2 つの性質を同時に持つ TRS を完備 (complete) な TRS というと，当然のことながら，完備な TRS においては任意の項に対するリダクション結果が常に唯一求まる．そこで，項 τ から求まる唯一の既

約項を τ の**正規項** (normal term) といい，$\tau\downarrow$ で表わすことにする．すると，完備な TRS では次の重要な性質が成り立つ．

$$\tau \stackrel{*}{\leftrightarrow} \upsilon \text{ のとき，かつこのときに限り } \tau\downarrow \equiv \upsilon\downarrow \text{ である}$$

つまり，2 つの項が等号論理の上で等価であることは，それらの正規項が一致することと同値である．この性質は，もし TRS が完備ならば，元の等号論理における論理的性質をリダクションという計算によって判定できるということを意味している．すなわち，等式を仕様とするとき，それから得られる完備な TRS はこの意味で正しいプログラムとみなすことができる．では，このような TRS を求める方法はあるのだろうか．それが，これから説明する完備化手続きである．この手続きは，等式集合をある程度自動的に完備な TRS に変換するもので，提案者の名をとって **Knuth–Bendix 完備化手続き** (Knuth–Bendix completion procedure; 以降 KB と略す) と呼ばれている．この手続きの概要を理解するためには，KB で重要な役割を果たす危険対とそれを用いた合流性の判定方法について知っておく必要がある．

4.2 危険対

項の中に現れる変数に具体的な項を割り当てる操作を**代入** (substituion) といい，項 τ, υ がある代入によって構文的に等しい項となるとき，τ と υ は**単一化可能** (unifiable) であるという．

先の R_{group} における (s1) のリデックスは $(-0+0)+(-0)$ と $(-0+0)$ の 2 つであったが，これらを見比べると，(s1) の同一部分 $(-0+0)$ が両者に含まれていることがわかる．実は，このように 1 つの項の重複した部分に異なる書換え規則の適用されることが，こういった分岐を引き起こす原因なのである．この分岐の可能性については，以下のように書換え規則の左辺の単一化可能性をみて一般的に調べることができる．まず，(r2), (r3) の変数名が衝突しないように名前のつけ換えを行なう．

$$\underline{-A+A} \;\rightarrow\; 0 \qquad \text{(r2')}$$

$$(\underline{C+D})+E \;\rightarrow\; C+(D+E) \qquad \text{(r3')}$$

すると，(r2') の左辺全体と (r3') の左辺の一部が単一化可能となる（下線部分）．そこで，(r3') の左辺にその代入を施し，

$$(-A+A)+E \qquad \text{(s0)}$$

なる項 (s0) を得ると，これが重複したリデックスを持つ項の 1 つになる．たしかに，(s1) をみると (s0) の具体例になっていることがわかる ((S0) の各変数 A, E に $0, -0$ を代入して考えよ)．この (s0) を (r2'), (r3') で別々にリダクションすると，(s0) の定め方より両書換え規則共適用でき，次の 2 項を得る．

$$\underline{(-A+A)+E} \;\Rightarrow\; \overline{-A+(A+E)} \quad \text{(r3') を適用}$$

$$\overline{(\underline{-A+A})+E} \;\Rightarrow\; \overline{0}+E \quad \text{(r2') を適用}$$

先の例におけるリダクション列の分岐が，この 2 項の具体例から始まっていることもすぐに確認できる．こうして求まる項の対，

$$< -A+(A+E), 0+E > \qquad \text{(c1)}$$

を危険対 (critical pair) または要対といい，これらをさらにリダクションしても互いに同じ項に到達しないとき，その危険対は発散するという．(c1) は，
$< -A+(A+E), E >$ にリダクションされるが 2 項は一致しないので，発散する危険対である．

4.3 危険対を用いた TRS の合流性判定

停止性を持つ TRS R が合流性を持つための必要十分条件は，R が発散する危険対を持たないことである[11]．R の書換え規則は有限で，それらの間の危険対も有限個であるから，すべての危険対は R の書換え規則の左辺を重ねることによって求まる．それらが発散するか否かは，実際にリダクションを行なえば判定できるので，これにより停止性を持つ R の合流性は決定可能となる．

逆に，発散する危険対をなくすことを考えてみよう．危険対 $< \zeta_1, \zeta_2 >$ について考えると，等号論理の上で ζ_1 と ζ_2 は等価である．たとえば先の例では，
E_{group} の推論によって $-A+(A+E)=E$ の成立は簡単に証明できる．そこで，この $-A+(A+E)=E$ を新たな等式として E_{group} に追加して E'_{group} を構成しても，両者の論理的な意味は変わらないことになる．これから再び R'_{group} を得ると，

$$\begin{aligned}
R'_{group}: \qquad 0+X &\rightarrow X && \text{(r1)}\\
-X+X &\rightarrow 0 && \text{(r2)}\\
(X+Y)+Z &\rightarrow X+(Y+Z) && \text{(r3)}\\
-A+(A+E) &\rightarrow E && \text{(r4)}
\end{aligned}$$

となり，少なくとも1つの発散する危険対(c1)はなくすことができる．ただし，この場合には新しい書換え規則(r4)と(r1)~(r3)の間に，また新たな危険対の発生する可能性がある．したがって，すべての発散する危険対をなくすためには，こういった危険対が発生しなくなるまでこの手続きを繰り返す必要がある．これが，次に述べるKBの基本的なアイデアである．

5 Knuth-Bendixの完備化手続き

既に3節では，等式を停止性を持つ方向に向きづける方法を検討した．4節では，停止性を持つTRS Rが合流性を持つための必要十分条件を知り，発散する危険対を等式として獲得する方法を学んだ．この2つの操作，等式の向きづけと発散する危険対の等式化を繰返し行なっていけば，Rの停止性を保証したまま任意の発散する危険対を合流させることができる．新たな書換え規則を含めて考えても発散する危険対がなくなっていれば，Rは停止性に加えて合流性を持つことになる．このようにしてRを繰返し変形し，完備なTRSへ近づけていくのがKBの基本的な考え方である[11]．

歴史的には，KBは普遍代数(universal algebra)の語の問題(word problem)を解くアルゴリズムとして，60年代後半に提案されたものである．その後しばらくは情報科学とは遠いところにあったが，80年代に入りTRSを通じて関数型言語や代数的仕様などの応用の場を与えられると，急速に情報科学と接近し，特に最近ではその拡張，改良などの新しい提案が続出するといった状況である．

図3にKBのアルゴリズム概要を示す．ここで，Eは等式の集合，Rは書換え規則の集合であり，$\succ$は停止性保証のための順序である．このアルゴリズムは新たに獲得された等式の両辺が$\succ$で比較できないとき，つまりRの停止性が保証できなくなったときは失敗に終わる．また，完備なRが有限集合として存在しない場合は新たな等式を獲得し続け停止しないので，正確には半アルゴリズムである．初期状態では常に等式のみが与えられ書換え規則は空であるものと仮定すると，KBは図4のように等式による仕様を入力し，完備なTRS(プログラム)を出力する変換システムとみなせる．

さらに詳しく知りたい読者のために，実行の効率まで考慮した詳細アルゴリズムを図5に示す．このKBは，等式集合Eを引数として入力し，完備なTRS Rを返す関数である．この関数が正常にRを返したときは，Rは完備であることに加え既約でもある．つまり，Rの各書換え規則は自分を除いた他の規則によってリダクションされることがない．すると当然，冗長な書換え規則を含むこともない．このようなRは，変数名の違いを無視すれば等式選択の順序によらず一意に定まる．

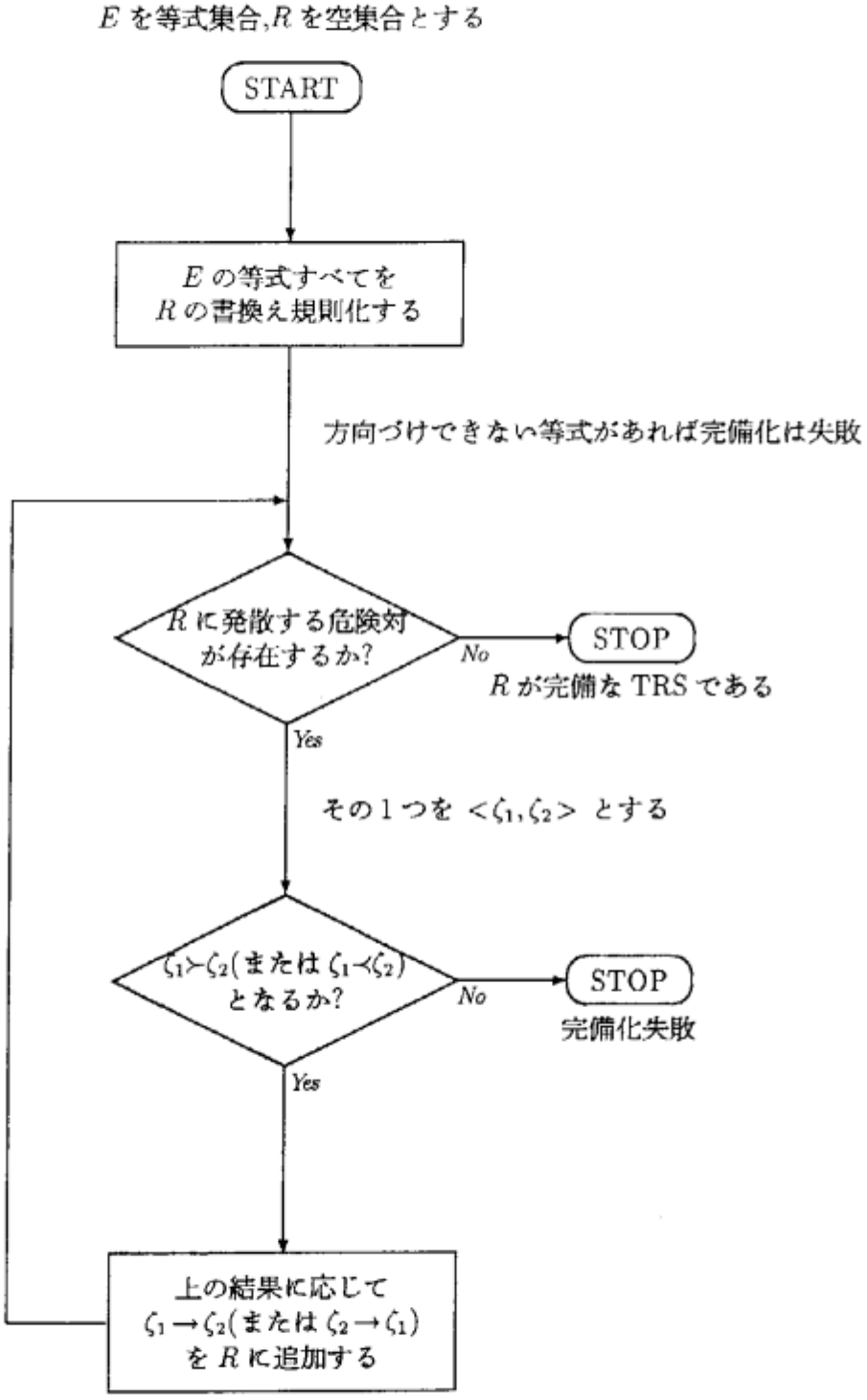

図 3: Knuth–Bendix の完備化手続き (概要)

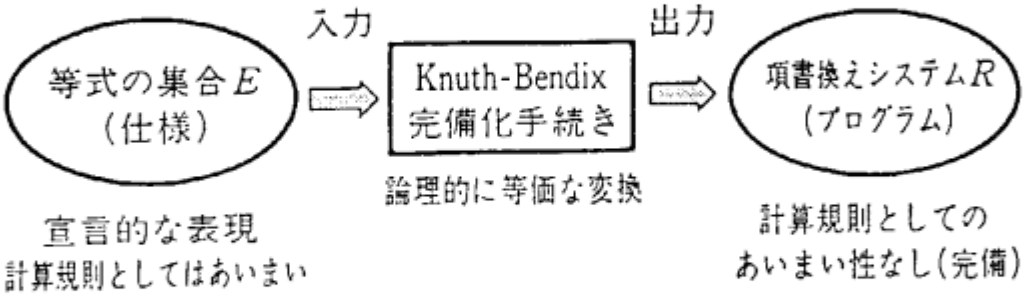

図 4: KB によるプログラム変換

【例】次の等式を図 3の KB で完備化する.

$$E_{easy}: \quad f(g(X)) = a \quad (e1)$$
$$g(b) = b \quad (e2)$$

2 つの等式は向きづけ可能で，次を得る.

$$f(g(X)) \to a \quad (r1)$$
$$g(b) \to b \quad (r2)$$

この 2 規則は $f(g(b))$ なる項で重なり，$< a, f(b) >$ なる危険対を持つ．この対は発散するので，向きづけをして書換え規則化する.

$$f(b) \to a \quad (r3)$$

(r1)～(r3) の間には発散する危険対が存在しないので,

$$R_{easy}: \quad f(g(X)) \to a \quad (r1)$$
$$g(b) \to b \quad (r2)$$
$$f(b) \to a \quad (r3)$$

が完備な TRS となる.

【例】先の E_{list} に同 KB を適用すると，R_{list} と同じものが得られる．これにより，R_{list} には発散する危険対がなく，既に完備であったことがわかる.

```
function KB(E : 等式の集合) returns 書換え規則の集合
  var R, R_new : 書換え規則の集合;
    R := φ  /* R の初期化 */
    while E ≠ φ
        E から等式 λ=ρ を選択する
        if λ↓ ≡ ρ↓ then
            /* 自明な等式を E から削除する */
            E := E − {λ=ρ}
        else
            /* 等式を書換え規則化する */
            case λ↓ ≻ ρ↓ then R_new := {λ↓→ρ↓}
            case λ↓ ≺ ρ↓ then R_new := {ρ↓→λ↓}
            otherwise 「失敗」を表示して停止する
            /* リダクション可能な書換え規則を R から取り除く */
            while R の中に R_new でリダクション可能な規則 γ→δ が存在
                R := R − {γ→δ}
                E := E ∪ {γ=δ}
            end
            /* 新しい書換え規則を R へ，危険対を E へ追加する */
            R := R ∪ R_new
            E := E − {λ=ρ} ∪ {ζ1=ζ2 | <ζ1,ζ2> は R_new と R の各規則間の危険対}
        end
    end
    return R /* 完備な R のリターン */
end
```

図 5: Knuth–Bendix の完備化手続き (詳細)

【例】 E_{SK} に KB を適用すると，等式 $sXYZ = XZ(YZ)$ に向きづけができず，手続きそのものが失敗に終わる．

【例】 E_{group} に図 5 の KB を適用すると，次のような完備な TRS R''_{group} が得られる．

$$
\begin{array}{lrcll}
R''_{group}: & 0+X & \to & X & \text{(r1)} \\
 & (-X)+X & \to & 0 & \text{(r2)} \\
 & (X+Y)+Z & \to & X+(Y+Z) & \text{(r3)} \\
 & (-X)+(X+Y) & \to & Y & \text{(r4)} \\
 & X+0 & \to & X & \text{(r5)} \\
 & -(-X) & \to & X & \text{(r6)} \\
 & X+(-X) & \to & 0 & \text{(r7)} \\
 & -0 & \to & 0 & \text{(r8)} \\
 & X+((-X)+Y) & \to & Y & \text{(r9)} \\
 & -(X+Y) & \to & (-Y)+(-X) & \text{(r10)}
\end{array}
$$

この例では，わずか 3 個の等式から 10 個の書換え規則が得られた後完備化が完了し，これらの規則の中には最初に与えなかった右単位元 (r5) や右逆元 (r7) に関する規則も見事に含まれている．この完備な TRS によるリダクションを用いれば，群に関する任意の等式の成立を判定することができる．たとえば，

$$-(-X+(-Y)) = X+(-X+Y)+X$$

の成立は，以下のリダクションによって確認される．

$$-(-X+(-Y)) \Rightarrow Y+X \Leftarrow X+(-X+Y)+X$$

6 KB の応用

前節でみたように，KB は等式で表現された論理的関係を意味を変えずに完備で既約な計算規則へ変換する手続きであるが，これを論理と計算の両者にまたがる問題の解決に役立てようという試みも多い．その中から 2 つのものを簡単に紹介する．

6.1 帰納的な定理証明への応用

E_{list} において，次の性質は成り立つであろうか．

$$\forall X.\ reverse(reverse(X)) \ = \ X \quad (t1)$$

多くの人がこれは成立すると考えるのではないだろうか．しかし，残念ながら答は否である．つまり，等号論理の上でこの等式は成立しない．その証拠に，第5節で求めた完備な TRS R_{list} によって両辺の正規項を比較すると，既に両辺共に正規項になっており一致はしていない．しかし，我々の直観によると，この等式が正しく思えるのは何故だろうか．

実際，任意のリスト $[\alpha_1, \ldots, \alpha_n]$ を考えて計算を行なうと，

$$reverse(reverse([\alpha_1, \ldots, \alpha_n])) = reverse([\alpha_n, \ldots, \alpha_1]) = [\alpha_1, \ldots, \alpha_n]$$

となり (t1) は成立する．これは，我々がある制限されたモデルを想定してこの等式を眺めていることにほかならない．このモデルとして比較的ふさわしいと思われるのは，変数を持たない項だけからなる世界である．つまり，(t1) は等号論理上の定理ではないが，変数を持たない項だけに限った世界では定理となる[†4]．このような定理は，一般に帰納法によって証明されるので**帰納的定理** (inductive theorem) と呼ばれる．たとえば，(t1) はリストの構造に関する帰納法により，

(1) $reverse(reverse([])) = []$

(2) $\forall X, Y.\ reverse(reverse(Y)) = Y$ ならば $reverse(reverse([X|Y])) = [X|Y]$

を示すことで証明される．このような帰納的定理を帰納法を使わずに KB の拡張によって証明する方法が知られている．**潜在帰納法** (inductionless induction) または**無矛盾性による証明** (proof by consistency)[7, 12, 2] と呼ばれるこの方法は，代数的仕様の検証や TRS 型プログラムの検証においてかなり有望であると考えられる．

6.2 等号論理の定理証明への応用

E_{SK} の完備化例でみたように，KB は等式に向きづけができない場合，TRS の停止性保証ができなくなり手続きそのものが失敗する．しかし，一般のプログラムには実際に停止しないものも少なくないので，停止性は多少厳しすぎる条件となっている．そこで，停止性の保証がなくても KB を継続させる方法が考えられた．向きづけ不可能な等式 $\lambda = \rho$ を 2 つの

[†4] ここでは *reverse* の引数にはリストしかこないことを暗黙のうちに仮定している．厳密な議論では，これを明示するために引数の型づけが必要となる．

書換え規則 $\lambda \to \rho$ 及び $\rho \to \lambda$ の役目を果たす**向無し書換え規則** (orientation free rewrite rule) $\lambda \leftrightarrow \rho$ に変換するもので，リダクションや危険対，停止性保証のための順序などをうまく拡張すればKBでこれを扱うことができる．このように拡張されたKBは，従来のように失敗に終わることがないため**失敗しないKB** (Unfailing Knuth-Bendix; 以降UKBと略す) [1]，または向無し書換え規則の導入から**無向完備化** [12]などと呼ばれている．

しかし，UKBによって手続きが失敗に終わるケースを回避しても，手続きが停止するか否かについてはやはり何の保証もない．実際の例では，実行が無限に続く場合の方がむしろ多くなるかもしれない．これに対し，UKBを完備なTRSを得るためでなく，等号論理の定理を証明する目的で利用する試みがある．S戦略 (S-strategy)[5]は，証明すべき等式（目標）を否定した結果「矛盾」が導かれるなら目標は定理であるとする証明手続きで，目標の否定と公理から矛盾を導くためにUKBを利用する．これは，等号論理の証明手続きとして完全，つまり目標が正しければ必ずその証明を見つけ出して停止するものとなっている．その意味においては，この手続きの方が元のUKBと比べ，より実用的といえるかもしれない．

【例】(S戦略による証明)

E_{SK} において，$(I\ x) = x$ なる I コンビネータが存在することを証明する．I コンビネータの存在は，以下の論理式で表現される．

$$\exists I.\forall X.(I\ X) = X \quad \text{(目標)}$$

これを否定すると，$\forall I.\exists X.(I\ X) \neq X$．さらにスコーレム化なる操作を行なうと，

$$(I\ \ \$(I)) \neq \$(I) \quad \text{(t1)}$$

なる式を得る．ここで，$\$$ はスコーレム関数と呼ばれる新たな関数記号である．これを E_{SK} に追加すると E'_{SK} として，

$$\begin{aligned} E'_{SK}: \quad sXYZ &= XZ(YZ) \quad \text{(e1)} \\ kXY &= X \quad \text{(e2)} \\ I\$(I) &\neq \$(I) \quad \text{(t1)} \end{aligned}$$

が得られる．これに対しUKBを適用すると，ある段階でR'_{SK}は次のようになる．

$$
\begin{aligned}
R'_{SK}: \quad sXYZ &\leftrightarrow XZ(YZ) && \text{(r1)}\\
kXY &\rightarrow X && \text{(r2)}\\
skXY &\rightarrow Y && \text{(r3)}\\
I\$(I) &\neq \$(I) && \text{(t1)}\\
\$(skX) &\neq \$(skX) && \text{(t2)}
\end{aligned}
$$

ここで，(r3)は(r1)と(r2)の間の（拡張された）危険対，(t2)は(t1)と(r3)から導かれる新たな式である．これをみると，先の完備化において向きづけのできなかったSコンビネータの定義(e1)は，向無し書換え規則(r1)として獲得されていることがわかる．この時点でのE'_{SK}は空でなく，したがってR'_{SK}は完備でない．しかし，R'_{SK}における(t2)は，構文的に等しい項が等式の否定で結ばれており，明らかに矛盾している．この結果，最初に与えた目標が定理であることが判明し，証明手続きとしてはこれ以上実行を続ける必要がなくなる．このとき，目標の変数Iに代入された値skXに着目すると，これはIコンビネータの定義となっている．実際，$(skX)Y$なる項をR'_{SK}でリダクションすると，これが間違いなくIコンビネータであることが確認される．

$$
\begin{aligned}
&\underline{skXY} \quad \text{(r3) を適用}\\
&\Rightarrow \overline{Y}
\end{aligned}
$$

7 おわりに

本稿では，TRSの重要な性質や完備化手続きについて解説し，その応用についても簡単に触れた．完備化手続きの応用に関する研究は実に数多く，ここで取り上げたもの以外に一階述語論理の定理証明システムへの応用，関数型プログラムの合成・等価変換や検証への応用，論理型プログラムの単一化の拡張や等式制約の導入への応用などたくさんのものがある．TRSとは少し離れたところで，一階述語論理における導出原理 (resolution principle) や，数式処理におけるGröbner基底を求めるBuchbergerアルゴリズムも一種の完備化である[2]．その他には，TRSに基づく関数型言語として，Miranda, KRC, HOPE, Tなどが知られており[9]，完備化手続きを中心にプログラムの合成・変換や定理証明，検証などを行なう実験システムには，REVE, RRL, FORMEL, TERSA, 筆者らが試作中の *Metis*

などがある．本稿における TRS の完備化や定理の証明例は，すべてこの *Metis* 上で作成，検証したものである．

参考文献

[1] Bachmair, L., Dershowitz, N., and Plaisted, D. A.: Completion without failure, in *Proc. Colloquium on Resolution of Equations in Algebraic Structures* (1987).

[2] Buchberger, B.: History and Basic Features of the Critical-Pair/Completion Procedure, *J. Symbolic Computation*, Vol. 3, No. 1&2 (1987), pp. 3–38.

[3] Dershowitz, N.: Termination, in *Rewriting Techniques and Applications (RTA)*, LNCS 202, Springer-Verlag (1985), pp. 180–224.

[4] 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 (1983), pp. 133–146.

[5] Hsiang, J. and Rusinowitch, M.: On Word Problems in Equational Theories, in *14th International Colloquium Automata, Languages and Programming (ICALP)*, LNCS 267, Springer Verlag (1987), pp. 54–71.

[6] Huet, G.: A complete proof of correctness of the Knuth-Bendix completion algorithm. *J. Comput. Syst. Sci.*, Vol. 23, No. 1 (1981), pp. 11–21.

[7] Huet, G. and Hullot, J-M.: Proofs by induction in equational theories with constructors, *J. Comput. Syst. Sci.*, Vol. 25, No. 2, (1982), pp. 239–266.

[8] Huet, G. and Oppen, D. C.: Equations and Rewrite Rules: A Survey, *Formal Languages: Perspective and Open Problems* (R. Book eds.), Academic Press (1980), pp. 349–405.

[9] 井田哲雄, 田中二郎: 関数型言語の計算モデル, コンピュータソフトウェア, Vol. 3, No. 2 (1986), pp. 2–18.

[10] Klop, J.W.: Term Rewriting Systems: A Tutorial, *Bull. of the European Association for Theor. Comput. Sci. (EATCS)*, No. 32 (1987), pp. 143–182.

[11] Knuth, D. E. and Bendix, P. B.: Simple word problems in universal algebras, *Computational problems in abstract algebra* (J. Leech eds.), Pergamon Press, Oxford (1970), pp. 263–297; also in *Automation of Reasoning 2* (Siekmann and Wrightson eds.), Springer-Verlag (1983), pp. 342–376.

[12] 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 (1987), pp. 2–22.
in *Proc.1st IEEE Symp. Logic in Computer Science* (1986).