北海道大学工学部研究報告
第166号（平成5年）

Bulletin of the Faculty of Engineering
Hokkaido University No. 166 (1993)

ATMSのデータ構造に基づいた複数の簡約順序を扱う完備化手続き

近藤　久　栗原　正仁　大内　東

（平成5年6月24日受理）

Completion with Multiple Reduction Orderings based on ATMS Data Structure

Hisashi KONDO, Masahito KURIHARA and Azuma OHUCHI

(Received June 24, 1993)

Abstract

The Knuth-Bendix completion procedure either succeeds, fails or diverges indefinitely. Success of the procedure heavily depends on the choice of a reduction ordering for orienting equations.

In this paper, we propose a completion procedure which are run with multiple reduction orderings. Actually, the procedure simulates the concurrent computation in which the ordinary completion procedures are run in parallel for each ordering. To gain efficiency as much as possible, we use the idea of ATMS(Assumption-based TMS). As a result, most of the results inferred in a concurrent process for one ordering can be reused in processes for other orderings.

1．まえがき

等式システムは代数的仕様記述，関数型プログラミング，定理自動証明などの理論的基礎として非常に重要である。等式システムに基づく計算モデルとして，等式を向きの付いた書換え規則とみなし，項を書換えることによって計算を行う項書換えシステム（*term rewriting system* : TRS）[1)~3)]の研究が活発に行われている。人工知能やソフトウェア工学においてはプログラムの合成[4)]，変換[5)]，検証[6)]の対象としてTRSを用いた研究が行われている。

TRSの重要な性質として停止性と合流性がある。停止性は無限に書換えが続かないこと，合流性は計算結果が一意であることに相当する。一般に停止性，合流性の検証は決定不能な問題であるが，いくつかの十分条件が知られている[3)]。停止性と合流性の両方をみたすTRSは完備であるという。

KnuthとBendixが提案した完備化手続きは，与えられた等式集合と等価で完備なTRSの生成を試みる(半)アルゴリズムであり，定理証明など種々の応用において非常に有用である。しかし，完備化の成功は与えられた簡約順序に大きく依存し，完備化手続きの問題点として少なくとも以下の3点が挙げられる。

情報工学科　システム工学講座

1．生成されるTRSの停止性を保証するために必要な簡約順序を利用者に要求する。

2．簡約順序が不適当な場合，手続きが無限に継続する場合がある。

3．簡約順序が適切な場合でも（完備なTRSが存在するにもかかわらず）失敗する場合がある。

第1の問題点は利用者に停止性に関する知識を要求する。また，多少その知識があったとしても，利用者が適切な簡約順序を与えることは難しい。第2の問題点に関しては，Hermannによる5つの発散解決法が提案されている[9]。第3の問題点に関しては，Bachmair等による基礎項上に限定した無失敗完備化（*unfailing completion*）[10]や順序付完備化（*ordered completion*）[3]と呼ばれる手続きが提案されている。但し，通常の完備化手続きが発散する場合，これらの手続きでも発散する。

手続きの発散を避ける明らかな方法は簡約順序の変更である。この方法はある書換え規則の向きを逆向きにすることによって発散の原因を取り除く。但し，候補となる簡約順序は複数存在するため適切な簡約順序を予め決定することは難しい。従って，適切な簡約順序の候補を複数個用意し，各簡約順序のもとで完備化を試みるしかない場合がある。その場合，手続きは発散する可能性があるため，複数の簡約順序を逐次的に試していく方法は不適切である。従って，手続きを各簡約順序毎に並行に動作させることが考えられるが，この方法では各プロセス間に重複した推論が生じ，効率が良くない。

本論文では，利用者の簡約順序の決定の負担を軽減し，不適当な簡約順序による発散を避けるため複数の簡約順序を扱う完備化手続きを提案する。本手法は効率化をはかるため人工知能の分野で提案されている問題解決のためのアーキテクチャであるATMS（*Assumption-based Truth Maintenance System*）のデータ構造[11]に基づいたノードと呼ばれるデータを用い，各簡約順序のもとで通常の完備化を並行に実行することをシミュレートしている。その際，ATMSの多重文脈を扱う性質を用い，推論結果を各簡約順序間で共有することによって重複した推論を避ける。

以下，2．でTRSと完備化手続きに関する予備知識をまとめ，完備化手続きの問題点2についてさらに詳しく述べる。3．ではその問題点を解決するための複数簡約順序完備化を提案し，実験結果を示し，本手法の効率を考察する。4．は結論である。

2．TRSと完備化手続き

本章では，TRSと完備化手続きの基本的な概念と記法，及び完備化手続きの問題点について簡単に述べる。

2.1 TRS

演算子の集合をF，演算子を$f,g,h,\cdots$で，変数の集合をV，変数を$x,y,z,\cdots$で表す。引数を持たない演算子は定数記号である。F,Vから構成される項の集合を$T(F,V)$，Fのみから構成される項の集合を$T(F)$とする。

特別な定数記号□を1つ含む項を文脈といい，$c[]$で表す。$c[]$の□を項sで置き換えて得られる項を$c[s]$と表す。ここで，項sを$c[s]$の部分項という。特に$c\neq$□のとき項sは$c[s]$の真部分項である（≡は構文的に等しいことを表す）。

代入θはVから$T(F,V)$への写像である。θは$T(F,V)$から$T(F,V)$への写像に拡張される。すなわち，$t\equiv f(t_1,\ldots,t_n)$とすると，$\theta(t)\equiv f(\theta(t_1),\ldots,\theta(t))$である。以後，$\theta(t)$を$t\theta$と表す。

等式$s\leftrightarrow t$は$T(F,V)$中の項の非順序対である。Eを等式の集合とする。このとき$l\leftrightarrow r$がEの等

式であり，$s \equiv c[l\theta]$, $t \equiv c[r\theta]$となるような文脈$c[]$と代入θが存在するとき$s \leftrightarrow_E t$または$t \leftrightarrow_E s$と表す。

書換え規則（以後，ルールと記述）は$l \rightarrow r$の形をした$T(F, V)$中の項の順序対である。但し，lは変数ではなく，rに含まれる変数はlにも含まれる。

項書換えシステム（TRS）Rはルールの集合である。$l \rightarrow r$がRのルールであり，$s \equiv c[l\theta]$, $t \equiv c[r\theta]$となるような文脈$c[]$と代入θが存在するとき$s \rightarrow_R t$と表し，項sは項tに書換えられるという。$\rightarrow_R$の反射推移閉包を$\rightarrow_R^*$で表す。項tがそれ以上書換えられなければtは既約（*irreducible*）であるという。もし$s \rightarrow_R^* t$かつtが既約ならば，tはsの正規形（*normal form*）である。

$l \rightarrow r$と$s \rightarrow t$をRのルールとする。ある文脈$c[]$と非変数uに対して$s \equiv c[u]$と仮定する。$u\theta \equiv l\theta$となる代入θ（但し，θはuとlの最汎単一化子）が存在するとき項$s\theta \equiv c\theta[u\theta]$は，$t\theta$にも$c\theta[r\theta]$にも書換えられる。この2つの項から作られる等式$t\theta \leftrightarrow c\theta[r\theta]$を危険対（*critical pair*）という。R中のルール間から得られる危険対の集合を$cp(R)$と表す。

TRS：Rは，もし$t_0 \rightarrow_R t_1 \rightarrow_R \cdots$のような無限の書換え列が存在しなければ停止する（*terminating*）といわれる。任意の項s，tに対し，$s \leftarrow_R^* \circ \rightarrow_R^* t$ならば$s \rightarrow_R^* \circ \leftarrow_R^* t$（$\circ$は関係の合成）が成り立つとき$R$は合流性（*confluence*）をみたすという。また，$s \leftarrow_R \circ \rightarrow_R t$ならば$s \rightarrow_R^* \circ \leftarrow_R^* t$が成り立つとき$R$は弱合流性（*local confluence*）をみたすという。$s \leftarrow_R u \rightarrow_R t$の形をした2ステップの（$s \leftrightarrow_R^* t$であることの）証明をピーク（*peak*）という。

簡約順序（*reduction ordering*）$>$は置き換え，代入に関して閉じた，項の集合$T(F, V)$上の整礎な半順序（*well-founded partial ordering*）である。つまり，$s > t$ならば任意の文脈$c[]$に対して，$c[s] > c[t]$，任意の代入θに対して$s\theta > t\theta$である。$>$が整礎であるとは$T(F, V)$中の項の無限の下降列$t_1 > t_2 > \cdots$が存在しないことである。ある簡約順序$>$に対しTRS：Rのすべてのルール$l \rightarrow r$が$l > r$ならばRは停止する。

2.2 完備化手続き

最初の完備化手続き（以後，KBと略記）はKnuthとBendixによって与えられた[7]。HuetはKBが正しいことを証明した[13]。Bachmair等はKBを以下の抽象的な推論規則として定式化した[14]。

Delete: $(E \cup \{s \rightarrow s\} \ ; \ R) \vdash (E \ ; \ R)$

Compose: $(E \ ; \ R \cup \{s \rightarrow t\}) \vdash (E \ ; \ R \cup \{s \rightarrow u\})$ if $t \rightarrow_R u$

Simplify: $(E \cup \{s \leftrightarrow t\} \ ; \ R) \vdash (E \cup \{s \leftrightarrow u\} \ ; \ R)$ if $t \rightarrow_R u$

Orient: $(E \cup \{s \leftrightarrow t\} \ ; \ R) \vdash (E \ ; \ R \cup \{s \rightarrow t\})$ if $s > t$

Collapse: $(E \ ; \ R \cup \{s \rightarrow t\}) \vdash (E \cup \{u \leftrightarrow t\} \ ; \ R)$ if $s \rightarrow_R u$ by $l \rightarrow r \in R$ with $s \rhd l$

Deduce: $(E \ ; \ R) \vdash (E \cup \{s \leftrightarrow t\} \ ; \ R)$ if $s \leftarrow_R u \rightarrow_R t$

$>$は簡約順序，Eは任意の等式集合，Rはすべてのルール$l \rightarrow r$に対し$l > r$が成り立つ任意のTRS，$\rhd$は包含順序（*encompassment ordering*）である。$s \rhd t$であるとはsの部分項がtの代入例であり，逆が成り立たないとき，かつそのときに限る。

Deleteは自明な等式$s \leftrightarrow s$をEから削除する。**Compose**はルール$s \rightarrow t$の右辺tを書換える。**Simplify**は等式$s \leftrightarrow t$の左辺，右辺どちらかを書換える。**Orient**は等式$s \leftrightarrow t$を簡約順序に従ってルール$s \rightarrow t$あるいは$t \rightarrow s$に変換する。**Collapse**はルール$s \rightarrow t$の左辺sを書換え，等式$u \leftrightarrow t$とする。但し，sに適用しようとするルール$l \rightarrow r$は左辺lが$s \rhd l$のものに限る。**Deduce**は新たな等式$s \leftrightarrow t$を

Eに加える。但し，$s \leftarrow_R u \rightarrow_R t$。

2.3 KBの問題点

KBの問題点は1．で述べたが，本節では問題点2について例を用いて述べる。

KBを用いるためには，推論規則**Orient**で用いる簡約順序$>$を必要とする。この簡約順序に依存して，完備化が発散する場合がある。例えば，以下の2本の等式と簡約順序として先行順序$>^1=\{(+,f)\}$とした辞書式経路順$>^1_{lpo}$で完備化を行う。

$$(x+y)+z \leftrightarrow x+(y+z) \quad (1)$$
$$f(x)+f(y) \leftrightarrow f(x+y) \quad (2)$$

等式(1)は**Orient**を適用することによってルール$r_1:(x+y)+z \rightarrow x+(y+z)$となり，等式(2)はルール$r_2:f(x)+f(y) \rightarrow f(x+y)$となる。$r_1$，$r_2$から**Deduce**を用いて新たな等式を求め，**Orient**を適用すると，$r_3:f(x+y)+z \rightarrow f(x)+(f(y)+z)$が得られる。さらに，$r_2$，$r_3$から同様の推論規則の適用によって新たなルール$f^2(x+y)+z \rightarrow f^2(x)+(f^2(y)+z)$が得られる。この**Orient**，**Deduce**の推論規則の繰り返しの適用は無限に続き，ルール$f^n(x+y)+z \rightarrow f^n(x)+(f^n(y)+z)$の無限の族を生成する。

先行順序を$>^2=\{(f,+)\}$とし，辞書式経路順序のクラスを変更することによって等式(2)は逆に向き付けることができ，次の2本のルールで完備化は成功する。

$$(x+y)+z \rightarrow x+(y+z) \quad (1')$$
$$f(x+y) \rightarrow f(x)+f(y) \quad (2')$$

この例からわかるように，与える簡約順序によって完備化の発散を避けることができる。この例では簡約順序を辞書式経路順序として，完備化に成功したが，簡約順序のクラス変更をしなければならない場合もある[9]。

既存のシステム（REVEやRRL等）では，簡約順序を適当な経路順序に定め，利用者と対話しながら先行順序を順次拡張しながら完備化を行う。利用者が誤りに気がついたときは，割り込みによって適当な選択点にバックトラックして間違った先行順序の取り消しや新たな選択を行う。しかし，利用者は停止性に関する知識が必要であり，バックトラック（深さ優先探索）を利用しているため，利用者が適切な割り込みをしなければ解があっても得られない場合がある。

複数の簡約順序を扱う方法としては，各簡約順序毎にKBを並行に動作させることが考えられるが，各プロセス間に重複した推論が生じる。例えば，上記の例の場合，等式(1)はどちらの簡約順序においても同じ方向に向き付けられる。

3．複数簡約順序完備化

本章では，2.3で述べたKBの問題点を避けるため，ATMSのデータ構造に基づき複数の簡約順序を扱う完備化手続きについて述べる。以下，この手続きをMKBと記す。

3.1 複数簡約順序完備化

任意の簡約順序の集合を$O=\{>_1,\ldots,>_n\}$，そのインデックスの集合を$I=\{1,\ldots,n\}$とする。MKBは並行プロセス$\{P_1,\ldots,P_n\}$をシミュレートする(以後，プロセスP_iを単にプロセスiと呼ぶ)。ここでプロセスiは簡約順序$>_i$のもとでのKBを実行している。

KBはEとRに対する推論規則であった。それに対し，MKBはノードと呼ばれるデータ構造の集

合Nに対する推論規則として形式化される。

[定義 3.1(ノード)]ノードは4つ組$\langle s:t, L_1, L_2, L_3\rangle$である。ここで，$s:t$は項$s:t$の順序対，$L_1$, L_2, L_3はラベルである。ラベルはインデックス集合Iの部分集合である。

ノードの各ラベルは，プロセス$i\in L_1$においては$s\rightarrow t\in R$，プロセス$i\in L_2$においては$t\rightarrow s\in R$，プロセス$i\in L_3$においては$s\leftrightarrow t\in E$であることを意味する。以後，ノード$\langle s:t, L_1, L_2, L_3\rangle$と$\langle t:s, L_2, L_1, L_3\rangle$を同一視する。

MKBの推論規則を以下に示す。Nはノードの集合である。各ノード$\langle s:t, L_1, L_2, L_3\rangle\in N$に対し，$i\in L_1$ならば$s>_i t$，$i\in L_2$ならば$t>_i s$が成立する。演算子\は集合の差を表す。

Delete-1: $N\cup\{\langle s:s,...,...,...\rangle\}\vdash N$

Delete-2: $N\cup\{\langle s:t,\{\},\{\},\{\}\rangle\}\vdash N$

Merge: $N\cup\left\{\begin{matrix}\langle s:t, L_1, L_2, L_3\rangle\\ \langle s:t, L'_1, L'_2, L'_3\rangle\end{matrix}\right\}\vdash N\cup\{\langle s:t, L_1\cup L'_1, L_2\cup L'_2, L_3\cup L'_3\rangle\}$

Rewrite-0: $N\cup\{\langle s:t, L_1\cup\{i\},...,L_3\cup\{i\}\rangle\}\vdash N\cup\{\langle s:t, L_1\cup\{i\},...,L_3\rangle\}$

Rewrite-1: $N\cup\{\langle s:t, L_1, L_2, L_3\rangle\}\vdash N\cup\left\{\begin{matrix}\langle s:t, L_1\backslash L, L_2, L_3\backslash L\rangle\\ \langle s:u, L\cap L_1,\{\}, L\cap L_3\rangle\end{matrix}\right\}$

if$\langle l:r, L,...,...\rangle\in N, t\rightarrow_{\{l\rightarrow r\}}u$ and either $t\not\rhd l$ or $L\cap L_2=\{\}$

Rewrite-2: $N\cup\{\langle s:t, L_1, L_2, L_3\rangle\}\vdash N\cup\left\{\begin{matrix}\langle s:t, L_1\backslash L, L_2\backslash L, L_3\backslash L\rangle\\ \langle s:u, L\cap L_1,\{\}, L\cap(L_2\cup L_3)\rangle\end{matrix}\right\}$

if$\langle l:r, L,...,...\rangle\in N, t\rightarrow_{\{l\rightarrow r\}}u, t\rhd l, L\cap L_2\neq\{\}$

Orient: $N\cup\{\langle s:t, L_1, L_2, L_3\rangle\}\vdash N\cup\{\langle s:t, L_1\cup\{i\}, L_2, L_3\backslash\{i\}\rangle\}$

if $s>_i t, i\in L_3$

Deduce: $N\vdash N\cup\{\langle s:t,\{\},\{\}, L\cap L'\rangle\}$

if$\langle l:r, L,...,...\rangle,\langle l':r', L',...,...\rangle\in N, s\leftarrow_{\{l\rightarrow r\}}u\rightarrow_{\{l'\rightarrow r'\}}t$

以下，各推論規則について述べる。

- **Delete-1**は**Delete**に対応し，すべての適当なプロセスにおいて自明な等式を削除する。
- **Delete-2**はすべてのプロセスがルール$s\rightarrow t$あるいは$t\rightarrow s$あるいは等式$s\leftrightarrow t$のいずれも持たないことを表すノードを削除する。
- **Merge**は第1スロットが等価なノードを1つにまとめノード数を減少させる。この推論規則はATMSに基づき同じデータを1つのノードで表すために用いられる。
- **Rewrite-0**はルール$s\rightarrow t(t\rightarrow s)$がプロセス$i$における$R$に存在し，同時に$E$中に等式$s\leftrightarrow t$が存在するならば，$E$からこの等式を削除する。等式$s\leftrightarrow t$はすでに簡約順序$>_i$のもとで向き付けられているため必要がない。
- **Rewrite-1**は適当なルールと等式が存在するすべてのプロセスにおいて**Simplify**と**Compose**を同時に行う。次の2つのノードを考える。但し，ルール$l\rightarrow r$によりtはuに書換え可能であるとする。また$t\not\rhd l$または$L\cap L_2=\{\}$であるとする。

$\langle l:r, L,...,...\rangle$ (3)

$\langle s:t, L_1, L_2, L_3\rangle$ (4)

$L \cap L_3$に属するプロセス中にはルール$l \to r$と等式$s \leftrightarrow t$が存在するので**Simplify**を適用できる。その結果，等式$s \leftrightarrow t$が削除され，等式$s \leftrightarrow u$が生成される。これは，ノード(4)を$\langle s:t, L_1, L_2, L_3 \backslash L \rangle$に変更し，ノード$\langle s:u, \{\}, \{\}, L \cap L_3 \rangle$を生成することに対応する。

同様に，$L \cap L_1$に属するプロセス中にはルール$l \to r$と$s \to t$が存在するので**Compose**を適用できる。その結果，ルール$s \to t$が削除され，ルール$s \to u$が生成される。これはノード(4)を$\langle s:t, L_1 \backslash L, L_2, L_3 \rangle$に変更し，ノード$\langle s:u, L \cap L_3, \{\}, \{\} \rangle$を生成することに対応する。

従って，**Simplify**と**Compose**を同時に行うことは，ノード(4)を$\langle s:t, L_1 \backslash L, L_2, L_3 \backslash L \rangle$に変更し，ノード$\langle s:u, L \cap L_1, \{\}, L \cap L_3 \rangle$を生成することに対応する。これが**Rewrite-1**である。

- **Rewrite-2**は**Simplify**，**Compose**に加え，**Collapse**を行う。上記と同じ2つのノードを用いて説明する。但し，$t \rhd l$とする。$L \cap L_2$に属するプロセス中にはルール$l \to r$と$t \to s$が存在するので**Collapse**を適用できる。その結果，ルール$t \to s$が削除され，等式$u \leftrightarrow s$が生成される。これはノード(4)を$\langle s:t, L_1, L_2 \backslash L, L_3 \rangle$に変更し，ノード$\langle s:u, \{\}, \{\}, L \cap L_2 \rangle$を生成することに対応する。従って，**Simplify**，**Compose**，**Collapse**を同時に適用することは推論規則**Rewrite-2**で表現できる。
- **Orient**はあるプロセスiに与えられた簡約順序$>_i$のもとで等式$s \leftrightarrow t$が$s >_i t$ならばルール$s \to t$に向き付ける。
- **Deduce**は適切なプロセスにおいて存在する2つのノードから危険対を求め，新たなノードを生成する。

3.2 基本アルゴリズム

NEをノードの集合，NRを相互の推論が済んでいるノードの集合，Iを簡約順序のインデックスの集合とする。複数簡約順序完備化アルゴリズムを図1に示す(手続きの記述は基本的にModula-2の構文による)。

関数手続き$mkb(NE, NR, I)$は複数簡約順序のもとで完備化を行い，完備化に成功した時点のノードの集合NRを返す。初期値はNEが第1ラベルと第2ラベルが空で第3ラベルに全簡約順序のインデックスを持つノードの集合，NRが空集合，Iが簡約順序のインデックスの集合である。アルゴリズムの基本的な流れを明確に記述するため，便宜上，**Delete-1**，**Delete-2**，**Rewrite-0**，**Merge**の操作を無視して記述していない。これらは適当な時点で適宜実行される。

このアルゴリズムの実行中NRは常に以下の条件をみたしている。

- すべてのノード$n \in NR$は**Orient**済み。
- すべての異なる2つのノード$m, n \in NR$は互いに**Rewrite-(1,2)**済み。
- すべての2つのノード$m, n \in NR$間で**Deduce**済み。

アルゴリズム中で用いている述語*success-p*と関数手続き*choose*，*rewrite*，*orient*，*cp*について述べる。

述語*success-p*(NE, NR, I)は完備化成功かどうかをチェックするために用いる。*success-p*はI中のあるインデックスiがNE中のすべてのノードのどのラベルにも存在せず，かつNR中のすべてのノードの第3ラベルに存在しないならば真を，さもなくば偽を返す。*success-p*が真を返すならば，簡約順序$>_i$のもとで完備化成功を表す。

関数手続き*choose*(*nodes*)はノード集合*nodes*から1つのノードを非決定的に選択する。この選択は公平(*fair*)であるものとする。つまり，どのノードも有限ステップの間に必ず選択されるということである。

```
1:  procedure mkb(NE, NR, I)
2:  begin
3:    while not success-p(NE, NR, I) do
4:      if NE = {} then return(fail)
5:      else
6:        n := choose(NE);
7:        NE := NE\{n} ∪ rewrite({n}, NR);
8:        (* n のラベルが副作用的に変更される *)
9:        if n ≠ ⟨..., { }, { }, { }⟩ then
10:         n := orient(n);
11:         (* n のラベルが副作用的に変更される *)
12:         if n ≠ ⟨..., { }, { }, ...⟩ then
13:           NE := NE ∪ rewrite(NR, {n});
14:           (*NR中のノードのラベルが
15:             副作用的に変更される *)
16:           NE := NE ∪ cp(n, NR)
17:         end;
18:         NR := NR ∪ {n}
19:       end
20:     end
21:   end
22:   return(NR)
23: end mkb
```

図1 複数簡約順序完備化アルゴリズム

関数手続き*rewrite*(*nodes* 1, *nodes* 2)は**Rewrite-1**, **Rewrite-2**を繰り返し適用することによってノード集合*nodes* 1中のノードをノード集合*nodes* 2により書換える。そして，書換えによって新たに生成されたノードの集合を返す。このとき，一般に*nodes* 1中のノードのラベルが副作用的に変更される(*nodes* 1中のノードは3.1の各推論規則のノード⟨*s*; *t*,...,...,...⟩に対応させる)。

関数手続き*orient*(*node*)は**Orient**を繰り返し適用し，ノード*node*の第3ラベル中の簡約順序によって等式を向き付け，副作用的にノードのラベルを更新する。

関数手続き*cp*(*node*, *nodes*)は**Deduce**の適用によってノード*node*とノード集合*nodes*中のノード間から得られるすべての危険対を求め，新たなノードを生成し，得られたノードの集合を返す(*node*と*node*間の危険対も含む)。

3.3 実験結果

Sun SPARC Station IPX(32Mbyte, 28.5MIPS) 上にLucid Common Lispを用いて，通常の完備化手続き(*kb*)と複数簡約順序完備化手続き(*mkb*)を実現し，表1に示す実験結果を得た。表中の○は成功を，×は失敗を表す。以下，実験結果について述べる。

実験1は以下の結合則(5)と自然数の加法(6)の2つの等式の完備化である。

表1

実験	簡約順序	インプリメント	
		kb	*mkb*
実験1	$\succ_{lpo}^{1}$	Divergent	Divergent
	$\succ_{lpo}^{2}$	0.03sec ○	0.04sec ○
	$\succ_{lpo}^{i=1,2}$		0.09sec ○
実験2	$\succ_{lpo}^{1}$	0.05sec ×	0.08sec ×
	$\succ_{lpo}^{2}$	0.05sec ×	0.08sec ×
	$\succ_{lpo}^{3}$	Divergent	Divergent
	$\succ_{lpo}^{4}$	Divergent	Divergent
	$\succ_{lpo}^{5}$	1.97sec ○	2.91sec ○
	$\succ_{lpo}^{6}$	1.98sec ○	2.83sec ○
	$\succ_{lpo}^{i=1,\ldots,6}$		9.39sec ○

$$(x+y)+z \leftrightarrow x+(y+z) \tag{5}$$
$$x+s(y) \leftrightarrow s(x+y) \tag{6}$$

演算子は$+, s$の2つであり，先行順序を$\succ^1=\{(+,s)\}$, $\succ^2=\{(s,+)\}$，とし，それぞれに対応する辞書式経路順序$\succ_{lpo}^{1}$, $\succ_{lpo}^{2}$を簡約順序として用いた。*kb*, *mkb*に1つずつ簡約順序を与えたとき$\succ_{lpo}^{2}$では共に非常に短い実行時間で以下の完備なTRSを生成して成功した。

$$(x+y)+z \rightarrow x+(y+z) \tag{5'}$$
$$s(x+y) \rightarrow x+s(y) \tag{6'}$$

$\succ_{lpo}^{1}$では2.3と同じ理由により共に発散した。*mkb*に全簡約順序$\succ_{lpo}^{i=1,2}$を与えた場合，$\succ_{lpo}^{2}$で完備化が成功した。

実験2は以下の群論の3つの等式の完備化である。

$$e*x \leftrightarrow x \tag{7}$$
$$x^{-}*x \leftrightarrow e \tag{8}$$
$$(x*y)*z \leftrightarrow x*(y*z) \tag{9}$$

演算子は$e, *, ^{-}$の3つであり，先行順序を$\succ^1=\{(e,^{-}),(^{-},*),(e,*)\}$, $\succ^2=\{(e,*),(*,^{-}),(e,^{-})\}$, $\succ^3=\{(*,^{-}),(^{-},e),(*,e)\}$, $\succ^4=\{(*,e),(e,^{-}),(*,^{-})\}$, $\succ^5=\{(^{-},*),(*,e),(^{-},e)\}$, $\succ^6=\{(^{-},e),(e,*),(^{-},*)\}$，とし，それぞれに対応する辞書式経路順序$\succ_{lpo}^{i}(i=1,\ldots,6)$を簡約順序として用いた。*kb*, *mkb*に1つずつ簡約順序を与えたとき$\succ_{lpo}^{5}$, $\succ_{lpo}^{6}$のもとで共に以下の完備なTRSを生成し成功した。

$$e*x \rightarrow x \tag{7'}$$
$$x^{-}*x \rightarrow e \tag{8'}$$
$$(x+y)+z \rightarrow x+(y+z) \tag{9'}$$
$$e^{-} \rightarrow e \tag{10}$$
$$x^{-}*(x*y) \rightarrow y \tag{11}$$
$$x*e \rightarrow x \tag{12}$$
$$x*x^{-} \rightarrow e \tag{13}$$
$$x^{--} \rightarrow x \tag{14}$$
$$(x*y)^{-} \rightarrow y^{-}*x^{-} \tag{15}$$
$$x*(x^{-}*y) \rightarrow y \tag{16}$$

$\succ_{lpo}^{1}$, $\succ_{lpo}^{2}$のもとでは失敗し，$\succ_{lpo}^{3}$, $\succ_{lpo}^{4}$のもとでは共に発散した。*mkb*に全簡約順序$\succ_{lpo}^{i=1,\ldots,6}$

を与えた場合，$>^5_{lpo}$($>^6_{lpo}$)のもとで完備化に成功した。実行時間は*kb*で$>^5_{lpo}$のもとで1.97秒に対し，全簡約順序を与えた*mkb*で9.39秒である。

*kb*と*mkb*の効率の比較をどのような基準で行えば良いかは難しい。*mkb*が6通りの簡約順序を扱っていることから，単純に*kb*の実行時間を6倍したとすると，$1.97\times 6=11.82>9.39$で*mkb*の方が効率がよい。しかし，実際には$>^1_{lpo}$，$>^2_{lpo}$は0.05秒しかプロセッサを使用していないので，$1.97\times 4+0.05\times 2=7.98<9.39$という式を考えれば*kb*の方が良い。また，*kb*は$>^1_{lpo},\ldots,>^6_{lpo}$の順で逐次的に実行されるとすれば*kb*の発散のため，$\infty>9.39$で*mkb*の方が良い。

3.4 考　察

本節では，*mkb*の効率をさらに向上させ得る手法について考察する。

●経路順序の単調性の利用

経路順序の性質として単調性がある。つまり，もし$>^1\subseteq>^2$ならば$>^1_*\subseteq>^2_*$である。ここで，$>^i$は先行順序，$>^i_*$は対応する経路順序である。さらに，2つの項s,tと演算子の集合が与えられたとき，$s>_* t$となるすべての先行順序を求めることが可能である。この性質を用いて，先行順序に属する先行順序対の極小集合によって対応する経路順序を表すとする。例えば，辞書式経路順序により等式$f(h(x,y))\leftrightarrow g(h(y,x))$を向き付けたとする。演算子は$f$，$g$，$h$の3つであり，先行順序を全順序とすると先行順序は6通りある。つまり，6通りの辞書式経路順序がある。ここで，単調性を利用しノードを表すと$\langle f(h(x,y)):g(h(y,x)),\{\{(f,g),(f,h)\}\},\{\{(g,f),(g,h)\}\},\{\{(h,f),(h,g)\}\}\rangle$となり，第1ラベル中の集合$\{(f,g),(f,h)\}$は単調性の性質から，この集合を包含するすべての先行順序，つまり，$f>g>h$と$f>h>g$の2つの先行順序からつくられる2つの辞書式経路順序を代表することができる。第2，第3ラベルも同様である。このようにラベルを表すことによって関数手続き*orient*の効率の改善が期待できる。

●完備化成功判定の効率化

関数手続き*success-p*の効率を改善するために図1のアルゴリズムのループ間に*NE*と*NR*の各々に存在するノードのラベル中のインデックスの生起数を保持しておき，漸次増減させる。あるインデックスiのカウント値が0になったとき簡約順序$>^i$で完備化成功とする。

●ヒューリスティクスの利用

上記のカウント値を利用し，図1のステップ6：の*choose*によるノードの選択においてカウント値の小さい（完備化成功に有望な）インデックスを持つノードを選択する。この選択の仕方によって生成される危険対を少なくすることができる。

4. むすび

本論文では，利用者の簡約順序の決定の負担を軽減し，不適当な簡約順序による手続きの発散を避けるため，ATMSのアイデアに基づく複数の簡約順序を扱う完備化手続きを提案した。

もちろん，停止性，合流性の検証は決定不能な問題であるから，本手法にも限界があり，不適当な簡約順序のみが与えられたときには発散してしまう。但し，利用者との対話とバックトラック法を用いた従来のシステムと比較し，本手法は利用者の簡約順序の決定の負担を軽減することができる。

今後の課題として，

(1) 3.4で述べた効率の改善を行う。

(2) より幅広い完備化を行うために，AC(*Associative-Commutative*) 完備化[3),8)] を導入する。

ことが挙げられる。本手法はこれらの改善，拡張によりさらに有効なものになると期待される。

参考文献

1) 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，**24**，2，pp. 133－146(1983)．

2) 井田哲雄：“計算モデルの基礎理論”，pp. 223－296，岩波書店(1991)．

3) Dershowitz N. and Jouannaud J.P. : “Rewrite Systems”, in Handbook of Theoretical Computer Science, Vol.B, ed. J. van Leeuwen, Elsevier Science Publishers B. V., pp. 243－320(1990).

4) 富樫 敦，千葉和也，野口正一：“代数を実現する項書換えシステムの帰納的推論”，人工知能学会誌，**6**，4，pp. 520－531(1991)．

5) Toyama Y. : “How to Prove Equivalence of Term Rewriting Systems without Induction”, Theo. Comp. Sci., **90**, pp. 369－390(1991).

6) 近藤 久，栗原正仁，大内 東：“Reason Maintenance Systemによる項書き換えシステム停止性検証の効率化”，情処学論，**32**，9，pp. 1046－1056(1991)．

7) Knuth D. E. and Bendix P. B. : Simple Word Problems in Universal Algebras”, in Computational problems in abstract algebra, ed. J. Leech, Pergamon Press, pp. 263－297(1970).

8) 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，**4**，1，pp. 2－22(1987)．

9) Hermann M. : “Vademecum of Divergent Term Rewriting Systems“, Technical Report 88－R022, Centre de Recherche en Informatique de Nancy, 1988.

10) Bachmair L., Dershowitz N. and Plaisted D. A. : “Completion Without Failure”, in Resolution of equations in algebraic structures, Vol. II, eds. H. Aït-Kaci and M. Nivar, Academic Press, pp.1－30(1989).

11) de Kleer J. : “An Assumption based TMS”, Artif. Intell., **28**, pp. 163－196(1986).

12) Dershowitz N. “Termination of Rewriting”, J. Symbolic Computation, **3**, pp. 69－116(1987).

13) Huet G. : “A Completion Proof of Correctness of the Knuth-Bendix Completion Algorithm”, J. Comput. Syst. Sci., **23**, 1, pp. 11－21(1981).

14) Bachmair L., Dershowitz N. and Hsiang J. : “Orderings for Equational Proofs”, Proc. Sympo. on Logic in Computer Science, Cambridge, MA, pp. 346－357(1986).