数理解析研究所講究録
第1843巻 2013年 119-130

対話型数式ユーザインタフェースMathTOUCHにおける数式表記表現の代数的ルールによる正準化の方法

武庫川女子大学　福井 哲夫*
TETSUO FUKUI
MUKOGAWA WOMEN'S UNIVERSITY

1　はじめに

教育の情報化が推進される中，2011年4月に文部科学省より発行された2020年に向けた「教育の情報化ビジョン」[9]の提言に伴い，双方向授業・協働学習・個別学習を支援するデジタル教科書の時代がこようとしている．特に，電子的個別学習の場面では，生徒・学生も数式を扱う必要性がますます高まりつつある[10],[11]．しかし，理数系教育において問題となるのが数式のデジタル入出力である．数式は文と異なり，2次元的な表記構造をもっており，従来のデジタル端末は数式の表記を十分考慮されておらず，数式の取り扱いを困難にしている．

そこで我々は，2011年8月に，数式のデジタル入力を平易にするための新しい数式入力方式を提案した[12]．その後，本方式の数式入力における効果[14]・効率[13]および満足度を高めるための追加・訂正機能の改善[15]について報告した．また，2012年8月には提案方式を実装したシステムMathTOUCHを開発して，簡単な数式ワークドリルへの応用事例を紹介し，その有効性を示した．

しかし，学習者が回答入力した数式の正誤判定を行うためには，数式内部表現の正準性（一意に定まること）を保証する必要がある．数学的意味入力しか行わないCASでは当たり前のこの仕組みが，初・中等教育の数学ソフトウェアと連携させる場合には大きな問題があることを指摘したい．本研究では，MathTOUCHで採用した代数的ルールによる正準化の方法について報告し，その解決策を提案したい．これにより，数学の数式を扱うデジタル学習への応用が可能となる．

第2章では，提案技術の概要について紹介し，第3章では，正準形の定義と正準化の方法について述べ，最後に，まとめを行う．

2　対話型数式ユーザインタフェースMathTOUCHの概要

ここでは，対話型数式ユーザインタフェースをMathTOUCHと呼び，その概要について述べる．

2.1　数式入力方式

まず初期入力のための数式文字列表記法を次のように定める．

*fukui@mukogawa-u.ac.jp

数式指示文字列表記法

所望する数式を，ユーザが読む順番にその数式要素に対応するキー文字（列）によって区切りなく線形に並べる．

例えば，変数 a や α などはいずれもキー"a"で表す．このように，数式要素に対するキー文字（列）はASCIIコードからなり，数式要素を連想しやすい頭文字やLaTeXなどに準じた文字列を採用してもよい．また，演算子の分数記号は"/"，積分記号は"int"などで表し，その前後には分数の分母・分子など，作用範囲の式を表す文字列が並ぶ．ユーザの負担を少なくするために，キーを並べる順番として，人がその数式を読む順番を採用した．特に他の表記法と大きく異なる点は，暗黙積やべき乗演算のように表記されない記号は，数式指示文字列に含めないところである．例えば，式 a^2 は"a2"と表記する．

上記のように，本システムは入力すべき数式文字列が従来方式に比べてとても単純になっている．

その代わり，所望する数式を構成している数式記号のスタイルや要素間の区切りや各演算子に対する作用範囲などが省略されており，入力された指示文字列形式の情報だけでは2次元的数式表記が一意的に定まらず，完全にフォーマットすることはできない．そこで，本システムはキー辞書データを用いて数式候補を提示し，仮名漢字変換のように，ユーザに不足情報を補うための簡単な指示（次の(指示1)〜(指示5)のいずれか）を要求する．

(指示1) 候補列の中から次候補を要求，

(指示2) 対象要素の現候補を採択，

(指示3) （候補が演算子の場合）作用範囲変更，

(指示4) 作用範囲が複数ある場合の変更対象切替，

(指示5) 候補を棄却し，文字列のキーを区切り直す．

本数式入力方式によって，ユーザによる入力・操作ミスがなければ，この5つのユーザ指示による手続きから所望する数式を確実に構築できる[14]．そのような数式の構築過程を図1に示す．ユーザは数式構成要素ごとに判断すればよいので，複雑な式であっても迷うことはない．すべての構成要素が確定すれば2次元的表記の構築が完了する[12]．

2.2 アルゴリズムの考え方と内部表現

ここで，システムが数式指示文字列形式からいかにして2次元形式を構築し得るかについて解説する．

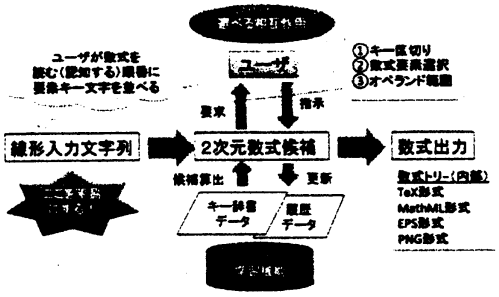


図 1: 数式構築過程

上述のように，2次元形式の数式は，数式最小単位記号自身でなければ，1個の演算子とその演算子が作用する1個または複数のオペランドによって構成される．すなわち，一般の2次元形式の数式は数式最小単位記号自身であるか，演算子を上位としてそのオペランドを下位にもつような階層構造をもち，オペランドもまた同様の階層構造をもった数式となっている．

このことから，階層構造をもつ場合の数式指示文字列形式は，その最上位にあたる1個の演算子を代表するキーとオペランドに相当する指示文字列の部分が演算子タイプに応じた順番で並んでいると解釈できる．すなわち，この解釈が数式の階層構造を把握し，2次元形式を構築することに他ならない．本論文では，数式指示文字列形式に対する上記の解釈を「構造解釈」と呼ぶ．このとき，オペランドに相当する指示文字列の部分はまた独立した数式指示文字列形式を成し，同様に構造解釈される．

例えば，式 $\frac{1}{a^2+1}$ を入力するための指示文字列は"1/a2+1"となる．このとき，分数演算子キー"/"に対して，2個の部分文字列"1"および"a2+1"がオペランドに対応していると解釈できる．ただし，数式指示文字列だけでは，キー"/"が最上位の演算子であるかどうかは判断できない．

したがって，一般に，本表記法で定めた数式指示文字列形式はユーザが所望する2次元形式に対して，数式を構成している数学記号のスタイルや要素間の区切りや各演算子に対する作用範囲区切りなどの情報が不足している．上記の不足情報は，自動的に決定することはできないため，本システムは，与えられた数式指示文字列に含まれる各キー文字（列）に対して，ユーザにその不足情報を要求し，その指示を受諾することによって確定する．

デコーダ（数式変換エンジン）による数式変換および不足情報1)〜3)のユーザ指示受諾処理の流れは**EXT-1)〜EXT-6)**のようになる．

[数式変換・構築の処理]

EXT-1) 入力（線形文字列）

EXT-2) キー文字列分解

EXT-3) キー文字列の辞書検索

EXT-4) 優先順位による数式要素候補の算出

EXT-5) 演算子とオペランドの構造解釈による数式木の構築

EXT-6) 訂正・確定

上述の例で解説すると，入力された数式指示文字列"1/a2+1"はキー文字列分解処理**EXT-2)**で，式(1)のように数式要素キー辞書と照らし合わせて部分文字列に分解される．

$$\{1, /, a, SP, 2, +, 1\} \tag{1}$$

このとき数式指示文字列形式の区切りポイントでは仮の暗黙積演算子 SP が挿入される．キー文字列の辞書検索処理**EXT-3)**で各部分文字列をキーとする数式要素の候補群を辞書から検索し取り出す．優先順位による数式要素候補の算出処理**EXT-4)**ではキーが演算子でなければ，仮名漢字変換と同様に最小単位記号辞書から各要素のスコアに基づいた優先順位によって取り出すことができるが，演算子の場合はデータ構造が複雑で，演算子記号群だけでなく演算子が作用するオペランドへの接続関係の情報も含み，ちょうど仮名漢字変換辞書の単語と接続関係の両方の情報を併せ持っている．数式木の構築処理**EXT-5)**では演算子とオペランドの構造解釈が行われ，数式木（例えば図2）を構築する．このように，構築結果がグラフの道ではなく複雑な木構造となるため，仮名漢字変換[7]とは全く異なり，最短経路探索問題で定式化することはできない．最後の処理**EXT-6)**が不足情報1)〜3)の必要な指示をユーザから受諾し訂正・確定処理を行う段階である．数式要素の表記記号がユーザの所望する記号でなかった場合，仮名漢字変換と同様の訂正・確定指示を行うが，所望する数式構造が異なる場合は，オペランド範囲の選択指示が必要となる．

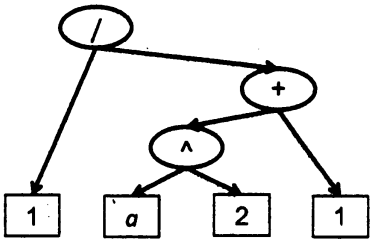


図 2: 数式木候補構築例

以上により，すべての構成要素キーが確定したとき数式の構築が完了する．

2.3 数式エディタ MathTOUCH（実装システム）

本数式デジタル化技術を応用して，Java アプレット版数式エディタ (MathTOUCH) を開発した．操作手順は，ユーザ経験を生かすため，仮名漢字変換とできるだけ類似の操作で行えるように配慮した．

実行画面を図 3 に示す．変換・確定指示の対象である数式要素（候補記号）が黄色で強調表示されている．図 4 は指示対象が演算子の場合で，作用範囲（この例では分数演算子に対する分子，分母）の式がグレーで網掛けされ，←と→キーで範囲調整が可能である．

2.3.1 基本機能

扱える数式の範囲は，LaTeX で構築できる数式のほぼ 9 割をカバーしており，大学の教養数学程度の数式は入力可能である．

しかも，確定した数式の挿入ポイントで式を削除したり，追加・訂正が可能となっている [15]．

また，構築した数式は現時点で次の（形式 1）～（形式 4）の出力が可能である．

（形式 1） LaTeX 形式
（形式 2） MathML 形式
（形式 3） PNG 形式
（形式 4） EPS 形式

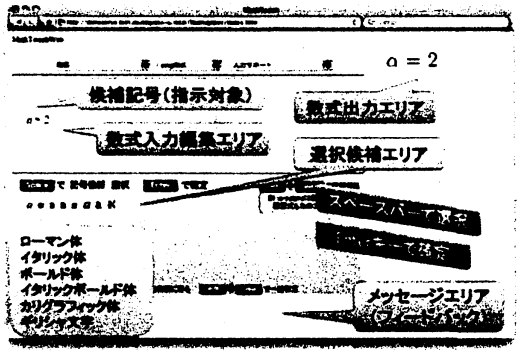


図 3: MathTOUCH 実行画面（記号選択）

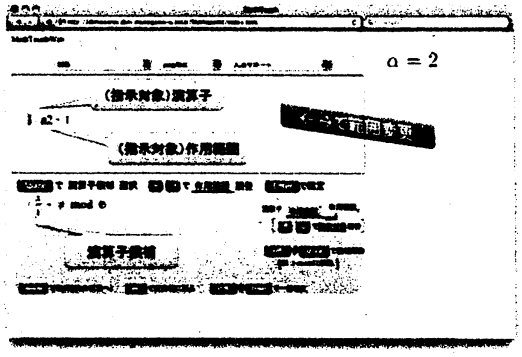


図 4: MathTOUCH 実行画面（演算子選択）

ウィンドウ（図 3）の選択メニューで希望の出力形式を指定すると，システムが数式オブジェクトを生成し，クリップボードに出力されるので，必要に応じて他のアプリケーションの挿入ポイントで貼り付け（ペースト）できるようになっている．

2.3.2 数式の行列配置

今回開発した数式エディタの機能の一つが数式の行列配置機能である．これにより連立方程式の表記やベクトル・行列の成分表記が可能となる．また，さらに，行列配置した数式に罫線枠を付けることも可能となっている．その出力例を表 1 に示す．詳細な入力手順は MathTOUCH ユーザーズマニュアル [16] を参照していただきたい．

表 1: 行列配置の数式例

$$a=1,\quad b=2,\qquad \begin{cases} y=3x-2 \\ y=x+1 \end{cases} \qquad A=\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

$$c=3,\quad d=4$$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}=\begin{pmatrix} ax+by \\ cx+dy \end{pmatrix}$$

$f(x)$	x	x^2
$f'(x)$	1	$2x$
$\int f(x)dx$	$\frac{x^2}{2}$	$\frac{x^3}{3}$

2.3.3 数式入力の従来技術との効率比較

数式入力のための従来技術として次が代表的である．

従来技術 1） 組版指示文字列コンパイル方式 （例：LaTeX[3],MathML[5] など），

従来技術 2） GUI テンプレート方式 （例：数式エディタ [6] など），

従来技術 3） 手書き入力方式 （例：Windows7 の数式入力パネルなど [8]）

1) は組版指示文字列を入力し，整形プログラムによって数式を構築するタイプで，文法が複雑である．2) は数式構造を表すテンプレートを GUI によって選択しながら構築するもので，長い式の編集では煩わしい場合がある．3) の場合，入力は最も自然であるが，誤認識や入力ミスによる編集操作は文書編集に比べ煩わしく感じる．また，入力効率は紙に直接記入する場合と変わらない．

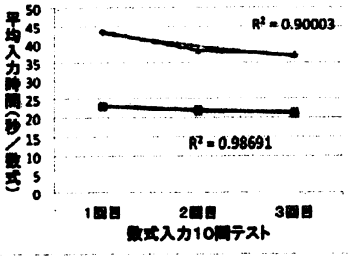
いずれの方式も理数系を専門としない一般ユーザにとっては依然使い易いとは言い難い．この理由は，システムが完全に数式構築できる情報をユーザの操作開始時に要求する仕組みにある．

本システムの優位性を示すために，数式入力のタスクテストによって実際に比較実験を行った [13]．

被験者（武庫川女子大学・情報メディア学科 3 年生）10 人に同一タスク問題を，本方式実装システム（MathTOUCH）と Microsoft（登録商標） Word for Mac2011 ver.14.1.1 に付属する数式エディタ [6] とで実施してもらい，数式入力時間（タスク達成時間）を測定して比較を行った．タスク問題は次のような分数係数の 2 次多項式である．

$$-\frac{5}{7}x^2-\frac{5}{8}x+3,\quad -7\alpha^2+\frac{9}{7}\alpha-\frac{5}{4},\quad -\frac{9}{7}\beta^2-\frac{7}{4}\beta-4$$

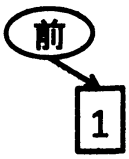
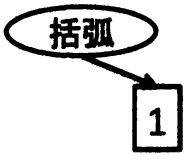
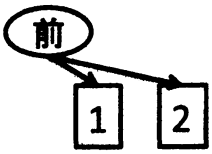
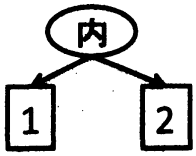
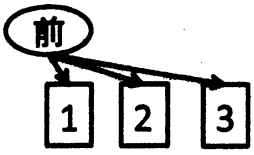
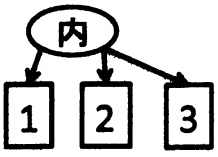
10 問テストを 3 回実施した後の結果を図 5 に示す．縦軸が一つの数式を入力するのに要する平均入力時間（秒）であり，上側の曲線が数式エディタの結果で，下側の曲線が本システムの結果を表す．t 検定の結果，有意差 ($t(9)=4.83, p<.01$) があり，このタスクでは，図 5 のように，本システムの方が約 1.75 倍速いことが判った．



比較項目	タスク達成時間（秒／数式）（3回目の平均入力時間）
数式エディタ	34.4 ± 6.2
本システム	19.6 ± 2.4

図 5: タスク達成時間の比較（3回の習熟曲線）

表 2: 7つの基本演算子構造

単項演算子	前置単項演算子	後置単項演算子	括弧演算子
			
二項演算子	前置二項演算子	内挿二項演算子	
			
三項演算子	前置三項演算子	内挿三項演算子	
			

3 MathTOUCHにおける内部表現の問題点と解決方法

ここでは，本題であるMathTOUCHにおける数式の内部表現の問題点について紹介し，その解決策について述べる．

3.1 数式木表現の問題点

上記のように，本アルゴリズムにおける数式の内部表現は数式木構造をもつ．MathTOUCHで扱う数式は最小単位要素自身でなければ表2に示した7つの基本演算子構造で成り立っている [14]．しかし，この数式木表現には以下で示すような問題点がある．

3.1.1 代数的ルールの問題

例えば $3x^2$ という数式は表記は変わらないが図 6-a,b に示すような 2 通りの数式木で表すことができる．ここで，$\otimes$ は暗黙積を，^ 記号はべき乗演算子を明示的に表した．木構造が示す数学上の意味は，図 6-a では $3 \otimes x^2$ であり，図 6-b では $(3x)^2$ に当たる．しかし，木構造は内部表現なのでユーザにはその違いが見えないが，数学的な意味が大きく異なってしまう．筆記や活字の世界では，我々はこれを避けるために代数的ルールを採用しており，$3x^2$ と書けば図 6-a の意味を表す．

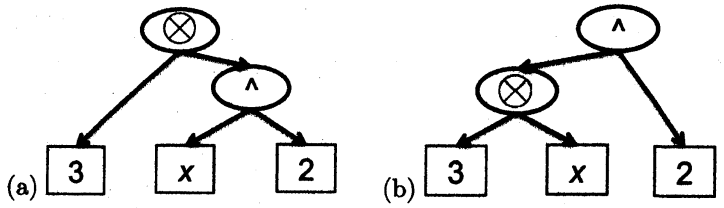


図 6: **数式木表現の問題点 1**

もう一つの例として，$y = ax^2 + b$ を考える．これも木構造として多数の表現が可能であるが，代数的ルールにおいて正しい表現は図 7-a である．しかし，図 7-b のように式の意味が $(y = ax^2) + b$ となっていても表記上は図 7-a と変わらないことに注意して欲しい．

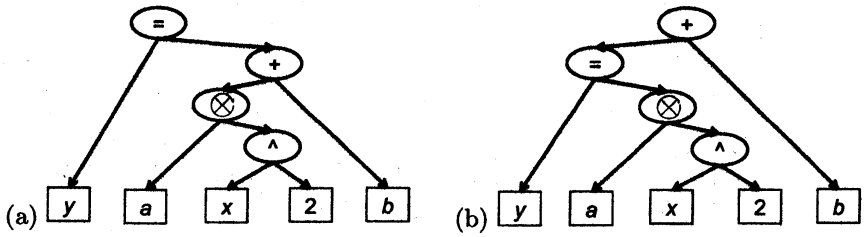


図 7: **数式木表現の問題点 2**

3.1.2 表現の一意性の問題

もう一つの問題は，数学的に正しく，代数的意味も表記も変わらないが，異なる表現が存在する場合があることである．

例えば，$a+b+c$ を考えよう．これは図 8-a,b どちらで表しても正しい．しかし，同じ表記で複数の表現を許すと数式辞書の履歴記録時にそれぞれの表現へスコアが分散し，機械学習システム部分の処理として不都合である．また，将来デジタル学習などの数式入力を本提案方式で行わせる場合，表現の一意性が保証されていないと，回答入力された式の正誤判定を行う際にも不都合が起こる．

3.2 解決策（代数的ルールによる正準化）

3.1 節の問題点を解決するヒントは 1980 年代の数式処理の研究にあった．特に，佐々木氏，元吉氏，渡辺氏により共同執筆された文献 [4] が大いに参考になる．その中で，数式処理における正準表現とは次のように定義される．

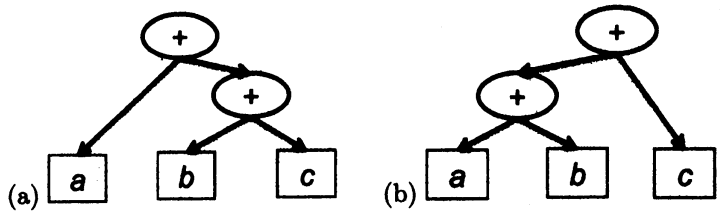


図 8: **数式木表現の問題点３**

数式処理における正準表現

数学的に等価な表式はすべて計算機内で一意的に表現されるような表現形式

例えば，多項式の正準表現として採用されてきたのが，変数の順序付けによる主変数，第２変数，…ごとの階層に分けて展開する表現で再帰的表現と呼ばれる．他にも正準表現としてさまざまな表現方法が存在するが，いずれの表現法においても必ず順序が定義されていることが重要である．このおかげで一意性が保証されるからである．

一方，本研究のように数式入力を主題にした場合は，表現の一意性として必要なのは数学的意味ではなく表記上の問題として考えなければならない．したがって，本論文における正準表現とは次のように定義する．

表記における正準表現

表記が同じ数式はすべて計算機内で一意的に表現されるような表現形式

正準化方法の考え方は，やはり数式処理の方法に習って実現した．数式処理における問題は，与えられた数式を数学的に等価なより簡単と思われる数式にいかにして書き換えるかにあった．この簡単化の理論は，「項書き換えの理論」として一般化されており [2]，本研究テーマである表記のための数式木表現の正準化においても当てはめることができる．

項書き換えの理論における項とは，多項式の項ではなく，一般に定数，変数，および関数形式から成る表式である [4].

＜項＞::=＜定数＞｜＜変数＞｜＜関係形式＞

ここで＜定数＞とは，数値であるか，あるいは自分自身としかマッチしない記号であり，＜変数＞とは任意の定数，変数，あるいは関数形式とマッチする記号である．また，＜関数形式＞とは，op を関数記号とするとき $op(arg_1,\cdots,arg_n)$（ただし，$arg_1,\cdots,arg_n$ は項）なる形式をしている．文献 [2],[4] でも述べられているように，項は非常に広い概念であり，例えばプログラムコードのような多くの実態が項と見なせる．本研究では MathTOUCH の数式内部表現である表記のための数式木を項と見なす．

さて，MathTOUCH における正準形を規定するために順序を定義しよう．MathTOUCH における正準表現は必ず代数的ルールに則って正しい表現でなければならないため，正準形を規定する順序として表３のような代数的ルールに則った順序になるように定義する．表３は BNF（バッカスの正規形式）を用いた再帰的定義となっており，左辺が MathTOUCH 内部で扱う 数式 あるいは 数式の構成要素 を定義している．したがって，数式は必ず最上位の，ある＜複合式＞であると見なすことができる．ここで，$\otimes$ は暗黙積，$\oplus$ は+または-の加算型二項演算子，$\pm$ は $\{+,-,\pm\}$ のいずれかの負号型単項演算子を代表している．例えば，＜項＞とは xy^2z などであり，暗黙積 $\otimes$ の右に表記される式は必ず＜因子＞であると規定している．また，＜右因子＞とは，項の右に表記される因子として許されるが，左に表記されることは禁止された因子のことで，例えば sin,lim,log,$\sum$,$\int$ 等である．なぜなら，$\int f(x)dx$ の右に因子 y が表記されると，積分変数と区

別できないため代数的ルールとして避ける必要があるためである．同様の理由で，負号の付いた $-x$ のような場合も＜項＞の右に表記されると引き算と区別できず，不都合なため＜左因子＞として定義されており，処理を区別して行う．

表 3: 代数的ルールのための順序 (BNF による)

＜数＞	::=	＜数値＞ | ＜数値＞. ＜数値＞;
＜非数値記号＞	::=	＜記号＞ | ＜テキスト＞ | ＜履歴式＞ | ＜ $\hat{記号}$ ＞ | ＜ $\dot{記号}$ ＞ | ＜ $\ddot{記号}$ ＞ | ＜ $\overline{記号}$ ＞;
＜因子＞	::=	＜非数値記号＞ | (＜因子＞＜後置演算子＞) | (＜括弧＞＜複合式＞) | $\sqrt{＜多項式＞}$ | ＜因子＞$^{＜多項式＞}$ | ＜因子＞$_{＜多項式＞}$ | $\frac{＜多項式＞}{＜多項式＞}$;
＜項＞	::=	＜数＞ | ＜因子＞ | ＜項＞ ⊗ ＜因子＞;
＜右因子＞	::=	(＜前置演算子＞＜数式列＞);
＜右項＞	::=	＜右因子＞ | ＜項＞ ⊗ ＜右因子＞ | ＜右項＞ ⊗ ＜右因子＞;
＜左因子＞	::=	±＜因子＞ | ±＜左因子＞;
＜左項＞	::=	＜左因子＞ | ＜左項＞ ⊗ ＜因子＞;
＜左右項＞	::=	±＜右因子＞ | ＜左項＞ ⊗ ＜右因子＞ | ＜左右項＞ ⊗ ＜右因子＞;
＜多項式＞	::=	＜項＞ | ＜右項＞ | ＜左項＞ | ＜左右項＞ | ＜多項式＞ ⊕ ＜項＞ | ＜多項式＞ ⊕ ＜右項＞;
＜比較式＞	::=	＜多項式＞ | (＜比較式＞＜比較演算子＞＜多項式＞);
＜関係式＞	::=	＜比較式＞ | (＜関係式＞＜関係演算子＞＜比較式＞);
＜複合式＞	::=	＜関係式＞ | ＜行列＞ | (＜複合式＞ , ＜関係式＞) | (＜複合式＞＜空白＞＜関係式＞);

3.3 代数的ルールによる正準化アルゴリズム

前節で定義した順序に基づいて，正準形でない数式木を表記を変えずに正準形へ変換する項書き換えアルゴリズムを解説する．

さて，次のような関係式で表された項書き換え規則 R を考える．

$$R : 項 \rightarrow 項$$

与えられた数式木 t を構成している各項を一定の順番に走査して，t 自身あるいは t の部分が R の左辺とマッチすればそれを右辺に置き換える．この操作を簡約と呼ぶ．

したがって，本研究では右辺の項が表 3 で定義した正準形となるように項書き換え規則を定めている．そして，MathTOUCH では簡約処理を行うプログラムを reform と呼ぶ．正準形でない数式 ex を項として reform(ex) は定められた項書き換え規則を適用し，書き換えられた正準形を返す．

具体的な項書き換え規則の 3 例を図 9〜11 に示す． 規則 1 は，べき乗の仮数部が＜因子＞であるべきところ＜項＞となっている場合で，＜項＞の右端は必ず＜因子＞なので，その＜因子＞のべき乗に書き換えることによって正準化を行うものである．

規則 2 は，暗黙積演算子の右側が＜因子＞であるべきところ＜項＞となっている場合を表している．この書き換えでは再び＜項＞ ⊗ ＜項＞の形が生成されるが，reform を再帰的に呼び出すことによって正準化を実現している．このとき，再帰的に reform に入力する式は元の式より因子の数が減少していることに注意して欲しい．これにより reform 手続きの停止性 [4] が保証される．

規則 3 は，加算型演算子 $\oplus := +, -, \pm, \mp$ に対する書き換え規則で，規則 2 の考え方と同じである．

最後に reform のアルゴリズムを以下に示す．表 3 で定めた順序が代数的ルールにに基づいている理由は，演算子タイプの順位が次の不等式になるように設計されているからである．

一般的二項演算子 < 暗黙積⊗ < 加算型演算子⊕ < 比較演算子 < 関係演算子 < 複合型演算子

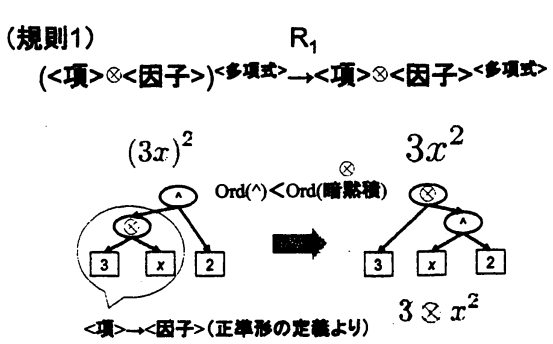


図 9: 代数的ルールによる書き換え規則 1

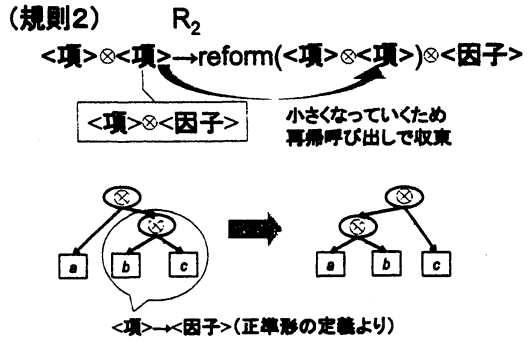


図 10: 代数的ルールによる書き換え規則 2

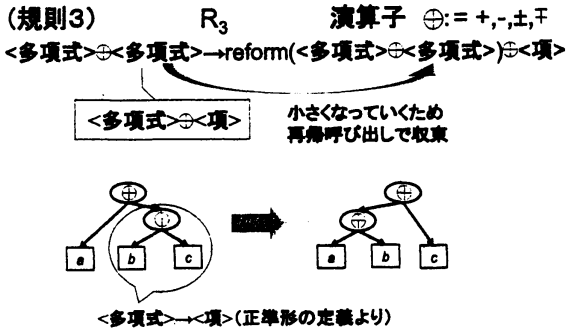


図 11: 代数的ルールによる書き換え規則 3

このように実際の正準化書き換え規則は演算子 op の種類ごとに処理が設計されており膨大な数におよぶため，演算子タイプの識別番号 N :=TypeNumberOf(op) を使用して全体の項書き換え規則を $\{R_N\}$ で表す．

[代数的ルールによる正準化アルゴリズム]

Procedure name: reform

Input: MathTOUCH を使用しユーザから入力された数式木 $ex := op(arg_1, \cdots arg_n)$, $n \leq 3$

Output: ex と表記が同じ正準形数式木 $\underline{ex}$, ただし，正準化できない場合は ERROR を返す

— Start of reform —

Step1) If Ord(ex)> 0 return ex;

Step2) for(i=1;i<=n;i++) if(($\underline{arg_i}$:=reform(arg_i))==ERROR) return ERROR;

Step3) If(Ord($op(\underline{arg_1}, \cdots \underline{arg_n})$)>0) return $op(\underline{arg_1}, \cdots \underline{arg_n})$;

Step4) N:=TypeNumberOf(op);
If(Ord($R_N(op(\underline{arg_1}, \cdots \underline{arg_n}))$)>0) return $R_N(op(\underline{arg_1}, \cdots \underline{arg_n}))$;

Step5) return ERROR;

— End of reform —

ここで，入力時の数式木 $ex := op(arg_1, \cdots arg_n)$, $n \leq 3$ とは，もし ex が構造をもたなければ正準形であることは自明であるから，演算子構造をもつ場合だけを考えており，op は表2にある演算子のいずれかを表し，arg_n が作用範囲の式を表している．また，$\mathtt{Ord}(ex)$ は，式 ex が表3で定義された代数的正準形ならばその順位を返し，そうでなければゼロを返す関数であり，式の正準形判定に使われる．**Step2)** では，まず op に対する全ての作用範囲の式 arg_i を reform によって正準化する．**Step4)** では，N を op の演算子タイプ識別番号とし，対応する項書き換え規則 R_N によって $op(\underline{arg_1}, \cdots \underline{arg_n})$ を変換し，正しく正準化されたならばその結果を返す．最後に，この項書き換え規則 R_N は，基本的に $\mathtt{Ord}(arg_i) \leq \mathtt{Ord}(op)$ （ただし，$i = 1, 2, 3$）ならば順位入替となるように作られている．この理由は，規則2の例でも述べたように，reform が再帰的に reform を呼び出しており，規則 R_N が必ず主演算子 op の順位を下げる処理となるように設計することによって，項書き換えの停止性を保証するためである．

4 まとめ

以上，暗黙積と加減算を中心とする代数的ルールに基づく順序づけによる正準形の定義と表記を変えない項書き換え規則の再帰的定義によって数式木（内部表現）の正準化に成功した．

正準化処理は MathTOUCH において候補算出やヒューマンエラー入力に対して代数的ルールに従った結果へと自動補正でき，とても重要である．すなわち，代数的ルールに基づく正準化にはつぎの4つの意義がある，

代数的ルールに基づく正準化の意義

I. 変換開始の数式候補算出時 代数的ルールに則った候補を算出できるので，候補選択の効率が向上する，

II. 数式候補の確定時 ヒューマンエラーによる非正準確定を補正できるので，入力の簡略化となる，

III. システムの機械学習時 更新される数式辞書のデータが正準化されるので，辞書検索時の一意性を保証できる，

IV. （応用） 確定式の同一性判定を可能にし，数学的意味をシステムが把握する上で役立つ，

しかし，本研究では数式木を「項」と見なした項書き換え理論における停止性を保証するように設計してはいるが，数学的な証明を与えるに至っていない．さらに，Knuth-Bendix が示したような完全性 (completeness)[1] の証明を示すことは今後の課題とする．

また，今後この正準化の成功により入力された数式の正誤判定や数学的意味解釈を応用し数式処理システムとの連携や数式を扱うデジタル学習システムへ発展していきたい．

参 考 文 献

[1] Knuth D.E., Bendix P.B.:Simple Word Problems in Universal Algebra, Proc. OXFORD 67,1967,263-298.

[2] 二木厚吉，外山芳人:項書き換え型計算モデルとその応用（解説），情報処理, **24**,1983,133-146.

[3] Donald Ervin Knuth: The TeXbook ,Addison-Wesley, 1984.

[4] 佐々木建昭, 元吉文男, 渡辺隼郎:数式処理システム -ソフトウェア講座 36-, 昭晃堂,1986.

[5] http://www.w3.org/TR/2001/REC-MathML2-20010221/

[6] Microsoft(R) co.:マイクロソフト数式エディター 3.0, http://office.microsoft.com/.

[7] 坂倉省吾:JIS X 4064 仮名漢字変換システムの基本機能, 財団法人日本規格協会,2002.

[8] M.Fujimoto, T.Kanahori and M.Suzuki: Infty Editor – A Mathematics Typesetting Tool with a Handwriting Interface and a Graphical Front-End to OpenXM Servers, Computer Algebra - Algorithms, Implementations and Applications, RIMS Kokyuroku Vol.1335 , 2003, 217–226.

[9] 文部科学省: 教育の情報化ビジョン,
http://www.mext.go.jp/b_menu/houdou/23/04/1305484.htm , 2011.

[10] 中村泰之:STACK と Moodle による数学 e ラーニング, 京都大学数理解析研究所講究録 *1735*「数式処理と教育」,2011,9–15.

[11] 白井詩沙香, 福井哲夫:数式処理を用いた教育を想定したタイピング能力の調査, 京都大学数理解析研究所講究録 *1735*「数式処理と教育」,2011,73–84.

[12] 福井哲夫:数式のインテリジェントな線形入力方式, 京都大学数理解析研究所講究録「数学ソフトウェアと教育」,**1780**,2012,160-171.

[13] 福井哲夫:数式のインテリジェントな線形入力方式と評価, 数式処理,vol.18 No.2, 日本数式処理学会,2012,47-50.

[14] 福井哲夫:線形文字列変換による対話型数式入力方式の効果, 京都大学数理解析研究所講究録「数式処理」,**1785**,2012,32-44.

[15] 福井哲夫:線形文字列変換による機械学習型数式入力インタフェースと編集機能の設計, 情報処理学会第 74 回全国大会論文集,2F-1,2012,4.1-4.2.

[16] 福井哲夫:MathTOUCH ユーザーズマニュアル, http://math.mukogawa-u.ac.jp, 2012.