数理解析研究所講究録
第1785巻 2012年 32-44

線形文字列変換による対話型数式入力方式の効果

福井哲夫
TETSUO FUKUI
武庫川女子大学
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」[1]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来ようとしてる．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[2],[3]．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．

数式入力のための従来技術として1）LaTeXなどの線形文字列コンパイル方式[4],[5]，2）数式エディタ[6]などのGUIテンプレート方式，3）InftyEditor[7]などで使われている手書き入力方式が代表的であるが，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．

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[8]．その後，本方式の数式入力効率が従来方式の数式エディタに比べて約1.75倍良いことを報告した[9]．

本研究は，提案した数式入力方式のアルゴリズムの定式化とその効果（タスクが達成できるか）について議論することが目的である．

第2章では，本方式と処理の流れが似ている仮名漢字変換について紹介し，第3章では，本方式の数式入力の概要を紹介し，仮名漢字変換との違いを示す．第4章で，本方式のアルゴリズムを定式化し，その効果を議論する．最後に，本研究の将来展望をまとめる．

1.1 用語

本研究を説明するためにまず，いくつかの用語を定義しておく．

「線形文字列形式(linear string format)」という言葉は，例えばポーランド式プレフィックス形式（ポーランド記法）や逆ポーランド記法，TeXやLaTeXなど，線形表記法を用いた数式の線形テキストに基づく表現を指す．LaTeXによる線形文字列形式の式の例は，"¥frac{1}{a^2+1}"（aの二乗足す1分の1）である．

「2次元形式」という言葉は，数式が，非線形表記法を用いて表される形式を指す．例えば，分子，分離文字，分母を上下に，指数部を上付き添え字など2次元的に配置する．2次元形式の例は，$\frac{1}{a^2+1}$ である．

*fukui@mukogawa-u.ac.jp

この２次元形式で表された数式は「数式最小単位記号」自身であるか,「演算子」およびその演算子が作用している１個または複数のオペランドによって構成されている．オペランドはまた一つの数式である．

「数式最小単位記号」と言う言葉は，数，変数，文字などを指す．

「演算子」と言う言葉は，本論文では，２次元形式の構造をきめる関数，二項演算子，積分演算子，縮約演算子，括弧，冪乗，添え字などを指し，演算子記号および／またはオペランド表記との相対的配置および大きさによってその構造が決まる．

以下では，２次元形式で表された数式を構成している数式最小単位記号または演算子を数式要素と呼ぶ．

2　従来の技術（仮名漢字変換）

ここでは，数式入力のアルゴリズムと対比して議論するため，従来からある仮名漢字変換のアルゴリズムについて紹介する．

2.1　仮名漢字変換の歴史

仮名漢字変換のはじまりは，1967 年に九州大学の栗原俊彦教授らの論文からと言われている [10]．当時は，テレックスによる仮名伝聞を漢字かな混じり文へ機械的に自動変換しようとして始まった．1973 年の相沢らの論文では，すでに日本語入力を意識していた．その後，1978 年にはじめてのワープロ（東芝 JW-10）が発売された．初期の頃のアルゴリズムは単文節ごとのひらがな文を漢字に変換するものであった．近年になって，ひらがなで文をそのまま入力して変換できる連文節変換の方式がとられている．1990 年代頃から，自然言語処理の分野で統計・機械学習ベースのアルゴリズム [11] が開発され，現在の仮名漢字変換のエンジンに導入されるようになった [12]．

2.2　仮名漢字変換の方式 [12],[13],[14]

仮名漢字変換のユーザモデル操作は次のようになっている．

（１）所望する文のかな文字列を入力し，

（２）デコーダにより漢字かな混じり文へ変換された候補が提示され，

（３）必要に応じて変換結果の訂正・確定指示をおこなう．

このとき，仮名漢字変換器に必要な要素は，１）データ構造（辞書），２）デコーダ，３）学習アルゴリズム（学習器）である．辞書データには，かなキーに属する単語群の情報やそれぞれの単語の出現頻度に応じたスコアが記録されている．学習器は正文からそのスコアを算出する．また，単語どうしの接続の強さについてもスコアが付けられ，辞書データ構造に記録される仕組みになっている．

２）のデコーダによる仮名漢字変換の処理の流れは KKT-1)～KKT-6) のようになる．

[仮名漢字変換の処理]

KKT-1) 入力

KKT-2) キー文字列分解

KKT-3) キー文字列の辞書検索

KKT-4) グラフの構築

KKT-5) 最短経路探索

KKT-6) 訂正・確定操作

例えば「かがくとがくしゅう」がデコーダによって「科学と学習」へ変換される過程を解説する.(この例は,徳永拓之氏の文献[12]に大変分かり易く解説されていたので借用させていただく.)入力された文字列は処理KKT-2)で{か,かが,かがく,かがくと}のように先頭文字からn文字までの部分文字列に分解され,処理KKT-3)で各部分文字列をキーとする変換文字(単語)全てを辞書から検索し取り出す.例えば,"か"→{火,蚊},"かがく"→{科学}となる.処理KKT-4)で単語をノードとし,入力仮名の順に単語どうしをエッジで結ぶグラフを構築する.開始ノード(BOS)から最初の単語候補全てを結び,それらに続く単語を可能なつながり全てに対してエッジで接続する.そうして終了ノード(EOS)までのグラフを構築する.図1は構築されたグラフの一例である.このとき,辞書にはそれぞれの単語および単語間の接続関係に対してそれぞれ出現のしやすさおよび前後の接続のしやすさに応じたスコアが付けられており,グラフ内のノード(単語)のコストおよびエッジ(接続関係)のコストが計算できるようになっている.したがって,処理KKT-5)の漢字かな混じり文の候補を算出するアルゴリズムは,開始ノードから終了ノードまでの可能な経路の内,経路内のノードおよびエッジのスコアを合計したコストを距離と解釈した場合の最短経路探索問題に他ならない.処理KKT-6)の訂正・確定操作では,ユーザの所望する文でなかった場合,ユーザ指示により確定したノードを制限条件とした制限付き経路探索問題に切り替えて処理が進行し,変換が完了する.

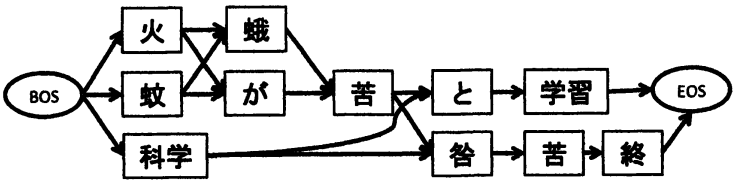


図1: 仮名漢字変換のグラフ構築 (文献[12]より)

3 線形文字列変換による対話型数式入力方式

ここでは,新数式入力方式の概要を述べ,数式変換アルゴリズムを解説すると共に,仮名漢字変換との違いを示し,最後に,本方式のユーザビリティ効果について議論する.

3.1 新数式入力方式の概要

本研究では,初期入力を平易にするため,線形文字列形式で表された次のような表記法を定める.

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.

本表記法で定めた数式要素に対応するキー文字(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TeX,LaTeXなどに準じた文字列を採用してもよい.

例えば,変数aやαなどはいずれもキー"a"で表す.また,演算子の場合は+記号や積分記号のように同じ記号であっても単項演算子や二項演算子など,構造の異なるものが存在する.

例えば，次の２次元形式の数式 (A)〜(D) 左辺の本表記法に従った線形文字列形式は，それぞれ右辺のように表される．

$$\begin{array}{lcll} \alpha = 2 & \leftrightarrow & \text{a=2} & \text{(A)} \\ 2a\beta & \leftrightarrow & \text{2ab} & \text{(B)} \\ \frac{1}{a^2+1} & \leftrightarrow & \text{1/a2+1} & \text{(C)} \\ \int_a^b c = 2 & \leftrightarrow & \text{intabc=2} & \text{(D)} \end{array}$$

特に，他の表記法と大きく異なる点は，暗黙積や冪乗演算子のように表記されない記号は，線形文字列に含めないところである．例えば，式 (C) に含まれる a^2 は，"a2" と表記する．この複数の数式要素キーが連続して並ぶキーとキーとの間を " 区切りポイント " と呼ぶことにする．

このように，本発明で使用する線形文字列形式表記法は，単純・簡潔になっている．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オペランドの範囲などが省略されており，入力された線形文字列形式の情報だけでは２次元形式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．そのため，本システムは線形文字列形式を解釈して構成要素を代表するキーの列に分解し，２次元形式の候補を算出するために，表１のようなキー辞書データを保持しており，各キーに関連付けられたさまざまな数式要素が登録された候補領域（対応する要素の集合）からデータの優先順位に基づいて候補を算出できるようになっている．そこで，本システムはそのキー辞書データを用いて，数式構成要素ごとに不足情報を補った候補を提示し，ユーザに候補を選択するための簡単な指示を要求する．そのような数式の構築過程を図２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次元形式の構築が完了する．

表 1: 「数式要素キー辞書データ」の例

数式要素キー	数式要素の候補領域	優先順位
"2"	2	1
"a"	a, **a**, a, α	4, 2, 1, 3
"b"	b, **b**, b, β	4, 2, 1, 3
"alpha"	α	1
"beta"	β	1
"="	$\square_1 = \square_2$, $\square_1 \doteq \square_2$, $\square_1 \equiv \square_2$, $\square_1 \approx \square_2$, $\square_1 \cong \square_2$	1, 2, 3, 4, 5
"+"	$\square_1 + \square_2$, $+\square_1$, $\square_1 \quad \square_2$, $\square_1{}^+$, $\square_1 \oplus \square_2$	1, 2, 3, 4, 5
"/"	$\square_1/\square_2$, $\frac{\square_1}{\square_2}$, $\square_1 \oslash \square_2$	1, 2, 3
省略キー (SP 演算子)	$\square_1\square_2$, $\square_1{}^{\square_2}$, $\square_{1\square_2}$, ${}^{\square_1}\square_2$, ${}_{\square_1}\square_2$	1, 2, 3, 4, 5
"i@" または "int"	$\int \square_1$, $\int_{\square_1} \square_2$, $\int_{\square_1}^{\square_2} \square_3$	1, 2, 3

ただし，$\square_1, \square_2, \square_3$ はオペランドを表す．

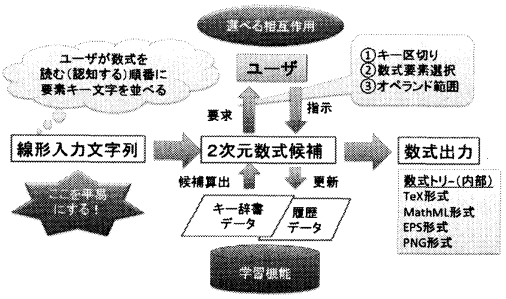


図 2: 新方式の数式構築過程

表 2: 代表的な演算子構造に応じた数式要素キーを並べる順番

構造の異なる演算子タイプ	演算子例	順番 1	2	3	4
暗黙積演算子	$\square_1\square_2$	$\square_1$	$\square_2$		
内挿二項演算子	$\square_1+\square_2$	$\square_1$	+	$\square_2$	
前置単項演算子	$\angle\square_1$	<	$\square_1$		
上置単項演算子	$\tilde{\square}_1$	~	$\square_1$		
下置単項演算子	$\underline{\square}_1$	_	$\square_1$		
後置単項演算子	$\square_1'$	$\square_1$	'		
括弧	$(\square_1)$	(	$\square_1$	)	
上付添字	$\square_1{}^{\square_2}$	$\square_1$	$\square_2$		
下付添字	$\square_1{}_{\square_2}$	$\square_1$	$\square_2$		
前置上付添字	${}^{\square_1}\square_2$	$\square_1$	$\square_2$		
前置下付添字	${}_{\square_1}\square_2$	$\square_1$	$\square_2$		
下付上付添字	$\square_1{}_{\square_2}^{\square_3}$	$\square_1$	$\square_2$	$\square_3$	
分数	$\frac{\square_1}{\square_2}$	$\square_1$	/	$\square_2$	
二乗根	$\sqrt{\square_1}$	root	$\square_1$		
n 乗根	$\sqrt[\square_1]{\square_2}$	$\square_1$	root	$\square_2$	
積分単項演算子	$\int\square_1$	int	$\square_1$		
積分二項演算子	$\int_{\square_1}\square_2$	int	$\square_1$	$\square_2$	
積分三項演算子	$\int_{\square_1}^{\square_2}\square_3$	int	$\square_1$	$\square_2$	$\square_3$
縮約三項演算子	$\sum_{\square_1}^{\square_2}\square_3$	sum	$\square_1$	$\square_2$	$\square_3$
内挿三項演算子	$\square_1\overset{\square_2}{\rightarrow}\square_3$	$\square_1$	->	$\square_2$	$\square_3$
前置上付演算子	$\sin^{\square_1}\square_2$	sin	$\square_1$	$\square_2$	
前置下付演算子	$\log_{\square_1}\square_2$	log	$\square_1$	$\square_2$	
lim 演算子	$\lim_{\square_1}\square_2$	lim	$\square_1$	$\square_2$	

3.2 数式変換アルゴリズムと仮名漢字変換との違い

ここで，システムが線形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．

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線形文字列形式は，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線形文字列の部分が表 2 の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，線形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線形文字列の部分はまた独立した線形文字列形式を成し，同様に構造解釈される．

例えば，式 (C) は分数演算子キー"/"に対して，2 個の線形文字列"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，線形文字列 (C) 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，本表記法で定めた線形文字列形式はユーザが所望する 2 次元形式に対して，次のような情報が不足している．

不足情報 1) 各構成要素キーの分離点はどこか？

不足情報 2) 各構成要素キーの所望する構成要素は候補領域のどの要素か？

不足情報 3) もし確定した要素が演算子の場合，そのオペランドは線形文字列のどの部分に相当するか？

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，与えられた線形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

例えば"1/a2+1"がデコーダによって $\frac{1}{a^2+1}$ へ変換される過程を解説する．入力された線形文字列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．

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

このとき線形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（例えば，表 1）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（例えば，表 2）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 3）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．詳細については次節で解説する．

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．

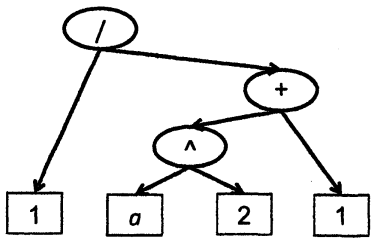
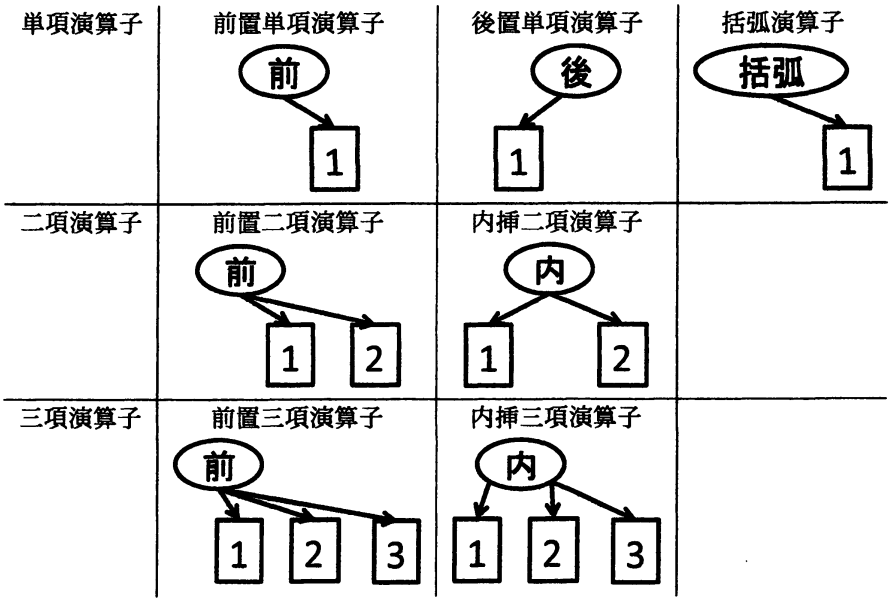


図 3: 数式木候補構築例

表 3: 7つの基本演算子構造



3.3 演算子とオペランドの訂正・確定方法

(1) 7つの基本演算子構造

数式変換を複雑にしている要因は，演算子が演算子記号で表されるばかりでなく，複数のオペランドへの接続関係構造を持つことにある．本研究では，演算子が最大3つのオペランドに作用し，全ての演算子を表3に示すような7つの構造に分類して処理を行う実施例で議論する．

本実施例が提案した数式入力方式の基本原理を限定するものではないが，この7つの構造によってLaTeXで表現できる数式をほとんど網羅することができる．例えば，本線形文字列形式におけるプラス演算子「+」は内挿二項演算子の場合（例：$x+y$），前置単項演算子の場合（例：$+x$），後置単項演算子の場合（例：x^{+}）などが考えられる．ユーザはいずれの場合であっても単に「+」と表現すればよいが，システム側はプラス演算子の構造解釈がいずれかに一意的に定まらなければ，構造の訂正・確定を要求するように設計しなければならない．

後の議論のために，表3のような演算子とオペランドの階層関係をあらわす数式木構造を数式処理システムの内部表現で多く使用されているような次のリスト表現で表すことにする．式(2)は表3における3つの単項演算子，式(3)は2つの二項演算子，式(4)は2つの三項演算子を表す．

$$\{\text{前単},\{\square_1\}\},\ \{\{\square_1\},\text{後}\},\ \{\text{左括弧},\{\square_1\},\text{右括弧}\} \tag{2}$$

$$\{\text{前二},\{\square_1\},\{\square_2\}\},\ \{\{\square_1\},\text{内二},\{\square_2\}\} \tag{3}$$

$$\{\text{前三},\{\square_1\},\{\square_2\},\{\square_3\}\},\ \{\{\square_1\},\text{内三},\{\square_2\},\{\square_3\}\} \tag{4}$$

ここで，$\{\square_1\}$,$\{\square_2\}$,$\{\square_3\}$は第1，第2，第3オペランドの数式木リストを表しており，「前単」,「内二」,「左括弧」などはそれぞれ，ある前置単項演算子記号，内挿二項演算子記号，左括弧記号などを表している．ただし，リスト$\{\square\}$の要素が1つだけの場合は{}を省く．例えば図3の数式木構造は，このリスト表現によって式(5)で表される．

$$\{1,/,\{\{a,\hat{},2\},+,1\}\} \tag{5}$$

(2) キー文字列分解

キー文字列分解処理 **EXT-2)** の結果において，もし不足情報 1) のように，分解された一つのキー文字列が所望するキーと異なっている場合はユーザによる再分解指示を受け付ける．例えば，入力文字列"int"は $\{\int\}$ ではなく $\{i,n,t\}$ としたい場合があるかもしれない．

次に，キー文字列の辞書検索処理 **EXT-3)** において n 個に分解されたキー列は，キー辞書データと照らし合わせて，キーのそれぞれに対応する数式要素候補ベクトルを取り出した列

$$\{\mathbf{e}_1, \mathbf{e}_2, \quad , \mathbf{e}_n\} \tag{6}$$

を生成する．このベクトル列を数式要素候補列と呼ぶ．ここで，ベクトル $\mathbf{e}_i$ $(i = 1, \quad , n)$ は表 1 のようなキー辞書データの対応するキーに属する候補領域を表しており，各成分は数式要素の候補，すなわち数式最小単位記号であるか演算子を表す．例えば，キー"a"や"+"や SP の候補ベクトルは次のように便宜的に表すことにする．

$$\mathbf{e}_{"a"} = \begin{pmatrix} a \\ \mathbf{a} \\ \alpha \\ \mathrm{a} \end{pmatrix}, \quad \mathbf{e}_{"+"} = \begin{pmatrix} \square_1 + \square_2 \\ +\square_1 \\ \square_1 \quad \square_2 \\ {\square_1}^{+} \\ \square_1 \oplus \square_2 \end{pmatrix} \quad \mathbf{e}_{SP} = \begin{pmatrix} \square_1\square_2 \\ {\square_1}^{\square_2} \\ \square_{1\square_2} \\ {}_{\square_1}\square_2 \\ {}_{\square_1}\square_2 \end{pmatrix} \tag{7}$$

またこのとき，候補ベクトルの成分の順番が優先順位を表しており，候補算出処理 **EXT-4)** の段階で第 1 成分を候補として算出する．

(3) 数式要素の訂正・確定

上述のように，演算子とオペランドの構造解釈による数式木の構築処理 **EXT-5)** によって算出された数式候補が，所望の数式でない場合は訂正・確定処理 **EXT-6)** によって正しい式に修正する．

ユーザに訂正・確定させるとは，式 (7) のような候補ベクトルの成分の一つを選択させることに他ならない．例えば，プラス演算子「+」の $a+b$ を $a \quad b$ に変換する場合がそれに当たる．この処理を次のように表す．

$$\underline{e_i} = \mathrm{Select}(\mathbf{e}_i) \tag{8}$$

ここで，選択を受諾する処理を Select とし，選択・確定された要素を $\underline{e_i}$ とした．

さて，ユーザに所望する要素を選択させる実施例としては，優先順位に基づいた候補ベクトルの成分順に要素候補を，例えば，仮名漢字変換のように，キーボードに割り当てられた指示キー（例えば，スペースキー）を押すことによって 1 個ずつ切り替えて提示し，所望する要素が表示されたときに採択させる方法と候補ベクトルのすべての要素を同時に提示し，ポインティングデバイスなどを用いて選択させる方法のいずれかまたはその両方によって実施することが考えられる．

(4) 演算子構造の変換

上述の，プラス演算子「+」の例でも説明したように，数式変換では演算子構造を別の構造へ訂正しなければならない場合が多々ある．演算子構造の変換公式を数式要素列で表すと次のようになる．

内挿二項演算子の場合：

$$\{\{\square_1\}, \text{内二}, \{\square_2\}\} \quad \leftrightarrow \quad \{\{\square_1\}, SP, \{\text{前単}, \{\square_2\}\}\} \tag{9}$$

$$\{\{\square_1\}, \text{内二}, \{\square_2\}\} \quad \leftrightarrow \quad \{\{\{\square_1\}, \text{後}\}, SP, \square_2\} \tag{10}$$

式 (9) の右辺は内挿演算子が前置演算子に変わり $\{\square_2\}$ のみに作用するようになり，$\{\square_1\}$ とは SP 演算子が区切りとして挿入される．同様に，式 (10) は後置演算子に変換される場合である．ここで，SP 演算子は式 (7) のように演算子記号を持たない内挿二項演算子であることに注意せよ．

括弧演算子を左右片括弧へ分離する場合：

$$\{\text{左括弧}, \{\square_1, \square_2\}, \text{右括弧}\} \leftrightarrow \{\{\text{左片括弧}, \{\square_1\}\}, \square_2, \text{右片括弧}\} \tag{11}$$

式(11)の左辺にある括弧演算子のオペランド$\{\square_1, \square_2\}$が右辺では$\square_1$と$\square_2$の間で分離されている．分離ポイントについては次項で説明する．

前置演算子の場合：

$$\{\text{前単}, \{\square_1, SP, \square_2\}\} \leftrightarrow \{\text{前二}, \{\square_1\}, \{\square_2\}\} \tag{12}$$

$$\{\text{前単}, \{\square_1, SP, \square_2, SP, \square_3\}\} \leftrightarrow \{\text{前三}, \{\square_1\}, \{\square_2\}, \{\square_3\}\} \tag{13}$$

$$\{\text{前二}, \{\square_1\}, \{\square_2, SP, \square_3\}\} \leftrightarrow \{\text{前三}, \{\square_1\}, \{\square_2\}, \{\square_3\}\} \tag{14}$$

オペランドどうしの区切りは，必ず区切り演算子SPで起こり，SP演算子は削除される．逆に2つのオペランドが1つに結合される場合はSP演算子が挿入される．ただし，内挿二項演算子の場合も変換公式(9)あるいは(10)のように前置あるいは後置演算子に変換され得る場合は，SP演算子を含むことになるので，オペランドの区切り候補となり得ることに注意する必要がある．

ところで，内挿三項演算子についても上述のように別構造の演算子に変更可能な設計にすることもできるが，本論文では簡単のため，内挿三項演算子は他の構造へは変換しない設計として議論する．このような設計はキー辞書の設計次第によって可能である．

(5) 演算子右オペランドの終端

訂正・確定処理 **EXT-6)** において，不足情報3)のオペランドに相当する線形文字列の部分（オペランド範囲）を確定する方法は，次に示すオペランド範囲の終点候補をユーザに選択させることによって実施する．ある演算子の右に位置するオペランド（例えば，前置演算子の第1，第2，第3オペランドや内挿二項演算子の第2オペランド）のオペランド範囲の開始点となるキーは，与えられた線形文字列の内，必ず(a)先端キー，(b)演算子直後のキー，(c)直前オペランドの終点直後のキーのいずれかとして自動的に決まる．一方，オペランド範囲の終点となり得るキーは，一意的には決まらず，開始点以降の(d)区切りポイントSP直前のキー，(e)演算子直前のキー，(f)終端キーのいずれかである．システムは，確定した演算子タイプから表2のようなオペランドの個数と演算子およびオペランドが並ぶ順番の情報を用いて，与えられた線形文字列形式を構造解釈し，終点候補となるキー(d)〜(f)を機械的に算出できる．例えば，線形文字列(C)の演算子キー"/"が分数の場合に，分母にあたるオペランド範囲の終点は式(1)から"a"，"2"，"b"の3通りあり，それぞれ(d)，(e)，(f)の場合に該当する．

終点候補（(d)〜(f)）は線形に並んでいるため，これをユーザに選択させることは容易であろう．例えば，次のような公式(15)を使えばオペランドの終端を変更，すなわち，オペランド範囲を変更することができる．

右オペランド範囲変更公式：

$$\{\{\quad, \text{演}, \{\square_1\}\}, \text{演}_2, \square_2, \quad\} \leftrightarrow \{\{\quad, \text{演}, \{\square_1, \text{演}_2, \square_2\}\}, \quad\} \tag{15}$$

ここで，対象の演算子記号を「演」，そのオペランドを$\{\square_1\}$とし，続く演算子記号を「演_2」で表した．「演_2」は上述のオペランド終端(d)または(e)を決める演算子である．(15)式：右矢印→の変換によってオペランド範囲が$\{\square_1, \text{演}_2, \square_2\}$のように拡大しており，左矢印←の変換によって縮小する．したがって，ユーザは(15)式の拡大または縮小の2つの指示（例えば，キーボードの右または左矢印キーを押下）を繰り返すだけで所望するオペランド範囲へ修正することができる．

また，演算子が複数オペランドに作用する場合は，まず，第1オペランド範囲が確定すれば第2オペランドの先頭要素が決まり，第2オペランドの終端は公式(15)の方式で決めることができる．第3オペランドも同様である．この実施方法例としては，オペランド範囲変更対象を第1〜3のいずれか1つにし，対象を

切替ながら範囲変更手続きを行えばよい．変更対象の切替は，前後のオペランドへ移す２つの指示ができれば可能であろう．

3.4 本方式の効果（タスク達成）

以上から数式変換の訂正・確定処理 **EXT-6)** は，以下の「文字列順確定アルゴリズム」によって実施できる．ここでは，簡単のために入力・操作ミスが無いものとする．以下では，アンダーラインの数式要素は確定済みを表し，式 (6) の数式要素候補列をこのアルゴリズムで処理した結果を $\mathbf{Proc1}(\{\mathbf{e}_1, \quad, \mathbf{e}_n\})$ で表す．

[文字列順確定アルゴリズム]

Procedure name: Proc1

Input: 式 (6) の数式要素候補列 $\{\mathbf{e}_1, \mathbf{e}_2, \quad, \mathbf{e}_n\}$

Output: 確定された数式木 $\mathcal{O}$

— Start of Proc1 —

Step1)(i=1) 第 l 要素 $\mathbf{e}_1$ は 1）最小単位記号，2）前置単項演算子，3）前置二項演算子，4）前置三項演算子，5）括弧演算子のいずれかである．$\mathbf{e}_1$ の確定は 3.3 節の式 (8) の候補選択の手続きによって行う．$\underline{e_1}$ が演算子の場合，オペランド範囲の変更指示は公式 (15) で述べた手続きによって確定できる．構造が異なる変換や括弧の分離のある場合も公式 (11)〜(14) によって確定できる．したがって，確定した演算子 $\underline{e_1}$ とオペランド範囲をリスト表現で表し，残りの要素は次の case:1)〜5) のように処理する．

case:1）最小単位記号 $\mathcal{A} = \{\underline{e_1}, \mathbf{e}_2, \quad, \mathbf{e}_n\}, i = 2$, Step2) へ

case:2）前置単項演算子

公式 (15) によってオペランド範囲が $\{\mathbf{e}_2, \quad, \mathbf{e}_j\}$ $(j > 1)$ に決まったとして，

$\mathcal{A} = \{\{\underline{e_1}, \mathbf{Proc1}(\{\mathbf{e}_2, \quad, \mathbf{e}_j\})\}, \mathbf{e}_{j+1}, \quad, \mathbf{e}_n\}, i = j + 1$, Step2) へ

case:3）前置二項演算子

公式 (15) によってオペランド範囲が $\{\mathbf{e}_2, \quad, \mathbf{e}_j\}, \{\mathbf{e}_{j+1}, \quad, \mathbf{e}_k\}$ $(k > j > 1)$ に決まったとして，

$\mathcal{A} = \{\{\underline{e_1}, \mathbf{Proc1}(\{\mathbf{e}_2, \quad, \mathbf{e}_j\}), \mathbf{Proc1}(\{\mathbf{e}_{j+1}, \quad, \mathbf{e}_k\})\}, \mathbf{e}_{k+1}, \quad, \mathbf{e}_n\},$

$i = k + 1$, Step2) へ

case:4）前置三項演算子

公式 (15) によってオペランド範囲が $\{\mathbf{e}_2, \quad, \mathbf{e}_j\}, \{\mathbf{e}_{j+1}, \quad, \mathbf{e}_k\}, \{\mathbf{e}_{k+1}, \quad, \mathbf{e}_l\}$ $(l > k > j > 1)$ に決まったとして，

$\mathcal{A} = \{\{\underline{e_1}, \mathbf{Proc1}(\{\mathbf{e}_2, \quad, \mathbf{e}_j\}), \mathbf{Proc1}(\{\mathbf{e}_{j+1}, \quad, \mathbf{e}_k\}), \mathbf{Proc1}(\{\mathbf{e}_{k+1}, \quad, \mathbf{e}_l\})\},$

$\mathbf{e}_{l+1}, \quad, \mathbf{e}_n\}, i = l + 1$, Step2) へ

case:5）括弧演算子

公式 (11) のようにオペランド範囲 $\{\mathbf{e}_2, \quad, \mathbf{e}_{j-1}\}$ $(j > 1)$ が分離指示されなければ，

$\mathcal{A} = \{\{\underline{e_1}, \mathbf{Proc1}(\{\mathbf{e}_2, \quad, \mathbf{e}_{j-1}\}), \underline{e_j}\}, \mathbf{e}_{j+1}, \quad, \mathbf{e}_n\},$

$i = j + 1$, Step2) へ

ここで，$\underline{e_1}$,$\underline{e_j}$ がそれぞれ左括弧，右括弧を表す．(注) 分離された場合は case:2).

Step2) case: 2)〜5) の演算子 $\underline{e_1}$ が作用するオペランドの再帰呼び出し **Proc1** が確定したと仮定（帰納的仮定）する．このとき，$\{\underline{e_1}, \quad, \underline{e_{i-1}}\}$ は確定済みである．
もし，$i > n$ ならば $\mathcal{O} = \mathcal{A}$ を出力して終了．そうでなければ Step3) へ

Step3) 公式 (15) の議論より，第 i 要素 $\mathbf{e}_i$ $(i > 1)$ は 6）内挿二項演算子，7）後置単項演算子，8）内挿三項演算子のいずれかである．さらに，6）は 6.1) 内挿二項演算子に確定される場合と公式 (9),(10) によって 6.2) 前置演算子または 6.3) 後置演算子に構造変換される場合がある．

case:6.1）内挿二項演算子
内挿二項演算子 $\underline{e_i}$ の第 1 オペランド $\{\underline{e_1}, \quad, \underline{e_{i-1}}\}$ は既に確定しているから，第 2 オペランドの範囲を確定すればよい．公式 (15) によってオペランド範囲が $\{\mathbf{e}_{i+1}, \quad, \mathbf{e}_j\}$ $(j > i)$ に決まったとして，
$\mathcal{A} = \{\{\{\underline{e_1}, \quad, \underline{e_{i-1}}\}, \underline{e_i}, \mathbf{Proc1}(\{\mathbf{e}_{i+1}, \quad, \mathbf{e}_j\})\}, \mathbf{e}_{j+1}, \quad, \mathbf{e}_n\}$,
$i = j + 1$, Step2) へ

case:6.2）内挿→前置演算子
この場合，もともと内挿演算子 $\mathbf{e}_i$ が前置演算子に変わり，その前に SP 演算子が第 i 要素として挿入されたため，$\underline{e_i}$ とそのオペランド $\{\mathbf{e}_{i+1}, \quad, \mathbf{e}_j\}$ $(j > i)$ が確定した後，$\mathbf{Proc1}(\{\mathbf{e}_{i+1}, \quad, \mathbf{e}_j\})$ が確定できると仮定（帰納的仮定）して，SP 演算子も確定できる．
$\mathcal{A} = \{\{\{\underline{e_1}, \quad, \underline{e_{i-1}}\}, \underline{SP}, \{\underline{e_i}, \mathbf{Proc1}(\{\mathbf{e}_{i+1}, \quad, \mathbf{e}_j\})\}\}, \mathbf{e}_{j+1}, \quad, \mathbf{e}_n\}$,
$i = j + 1$, Step2) へ

case:6.3）内挿→後置演算子
この場合，もともと内挿演算子 $\mathbf{e}_i$ が後置演算子に変わり，その後に SP 演算子が第 i+1 要素として挿入される．後置演算子 $\underline{e_i}$ は $\{\underline{e_1}, \quad, \underline{e_{i-1}}\}$ を第一オペランドとして確定できる．
$\mathcal{A} = \{\{\{\underline{e_1}, \quad, \underline{e_{i-1}}\}, \underline{e_i}\}, SP, \mathbf{e}_{i+1}, \quad, \mathbf{e}_n\}$, $i = i + 1$, Step2) へ

case:7）後置演算子
後置演算子 $\underline{e_i}$ は $\{\underline{e_1}, \quad, \underline{e_{i-1}}\}$ を第一オペランドとして確定できる．
$\mathcal{A} = \{\{\{\underline{e_1}, \quad, \underline{e_{i-1}}\}, \underline{e_i}\}, \mathbf{e}_{i+1}, \quad, \mathbf{e}_n\}$,
$i = i + 1$, Step2) へ

case:8）内挿三項演算子
内挿三項演算子 $\underline{e_i}$ の第 1 オペランド $\{\underline{e_1}, \quad, \underline{e_{i-1}}\}$ は既に確定しているから，第 2，第 3 オペランドの範囲を確定すればよい．公式 (15) によってオペランド範囲が $\{\mathbf{e}_{i+1}, \quad, \mathbf{e}_j\}, \{\mathbf{e}_{j+1}, \quad, \mathbf{e}_k\}$ $(k > j > 1)$ に決まったとして，
$\mathcal{A} = \{\{\{\underline{e_1}, \quad, \underline{e_{i-1}}\}, \underline{e_i}, \mathbf{Proc1}(\{\mathbf{e}_{i+1}, \quad, \mathbf{e}_j\}), \mathbf{Proc1}(\{\mathbf{e}_{j+1}, \quad, \mathbf{e}_k\})\}$,
$\mathbf{e}_{k+1}, \quad, \mathbf{e}_n\}$, $i = k + 1$, Step2) へ

— End of Proc1 —

さて最後に，入力された数式候補ベクトル列において，手続き $\mathbf{Proc1}(\{\mathbf{e}_1, \mathbf{e}_2, \quad, \mathbf{e}_n\})$ が完了できることを示しておく．

(I) $n = 1$ の場合は，$\mathcal{A} = \{\mathbf{e}_1\}$ が数式最小単位記号のみの場合であるから，手続き **Proc1** は 3.3 節で述べた式 (8) の手続きによって明らかに確定できる．

(II) 一般に $n > 1$ の場合は，手続き **Proc1** の Step1) によって，第 1 要素 $\mathbf{e}_1$ は，演算子であっても確定できることが示され，Step2) および Step3) によって処理すべき要素は n　1 個以下であり，処理を繰り返すたびに，処理すべき要素の個数は確実に減少している．したがって，手続き **Proc1** の中で再帰的に呼び出される手続き **Proc1** の入力数式候補ベクトル列の要素数も n　1 個以下であり，n が有限であることと，帰納法により処理は完結する．／／

以上により，本数式入力方式のユーザビリティ効果が示された．すなわち，本方式により数式入力のタスクが確実に達成できる．

4　まとめ

数式のデジタル入力を容易にするユーザインターフェースを開発するため 2011 年 8 月に提案した新しい数式入力方式 [8] は，数式の線形文字列形式で入力された文字列を数式辞書データを使って変換し，算出された数式候補をユーザとの対話形式で訂正・確定操作することによって数式を構築できるものである．

本提案方式の独創性は，数式に対する入力文字列をユーザの読む順番に対応させた点にある．したがって，従来技術に比べ，（1）数式構築のための入力文字列が平易，（2）候補を選択すればよいので，数式構築操作が単純，（3）数式辞書の機械学習機能により入力効率が向上するといった優位性がある．

ユーザビリティの評価は効果・効率・満足度の高さで測られる [15]．本研究では，そのユーザビリティを評価するため，本方式を実装した数式入力システムを試作し，本方式の効率についてその実装システムによる評価テストを実施した結果，分数係数の 2 次多項式の入力に関しては従来方式の数式エディタに比べて約 1.75 倍効率が良いことを 2011 年 9 月の研究 [9] において報告した．

今回の研究では，提案した数式入力方式におけるその効果について検証するため，線形文字列変換による対話型数式入力方式による手続きをアルゴリズム表現によって定式化した．その結果、次のことを示した．

1. 本方式は，仮名漢字変換と類似の操作性をもつが，数式のデータ構造は全く異なるため，仮名漢字変換のアルゴリズムは適用できない．

2. 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，次の 4 つのユーザ指示（1）数式要素選択，（2）オペランド範囲変更，（3）変更対象オペランドの切替，（4）キー再分解による手続きから確実に所望する数式を構築できる．

したがって，本方式のユーザビリティ効果が示された．

最後に，満足度の高い評価を得るためには，方式が優れているだけではだめで，ユーザの入力・操作ミスにも対処できなければならない．そのためには，ミスしても修正できるような数式編集機能を設計する必要がある．これについては，現在取組中で，今後の課題である．

参　考　文　献

[1] 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.

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

[3] 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, 数理解析研究所講究録 *1735*「数式処理と教育」,2011,73–84.

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

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

[6] Microsoft co.: Microsoft(R) 数式エディターユーザヘルプ, http://office.microsoft.com/ja-jp/word/.

[7] 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.

[8] 福井哲夫:数式のインテリジェントな線形入力方式, 数理解析研究所講究録「数式ソフトウェアと教育」,2012, 掲載予定.

[9] 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, 数理処理, 日本数式処理学会,2012, 掲載予定.

[10] 森健一, 八木橋利昭:日本語ワープロの誕生, 丸善,1989.

[11] 奥村学, 高村大也:言語処理のための機械学習入門, コロナ社,2010.

[12] 徳永拓之:作って学ぶ日本語入力,Web+DB press,Vol.64, 2011, 94-121.

[13] 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.

[14] 吉村賢治:Information Science & Engineering-T4 自然言語処理の基礎, サイエンス社,2000.

[15] ヤコブ・ニールセン:ユーザビリティエンジニアリング原論, 東京電機大学出版局,1999.