正規表現を用いた数式検索

静岡大学　博士論文

正規表現を用いた数式検索

2015 年 12 月

大学院　自然科学系教育部

情報科学専攻

渡部　孝幸

要旨

数式は，数理的な概念やモデルを記述するために使われる．数学はもちろん，物理学や経済学など，数理的な手法を用いるさまざまな分野で用いられる重要な形式である．web ページや電子書籍中に数式を記述したり，数式の変形や数式のグラフの描画を自動的に行ったりなど，計算機上での数式の利用も進んでいる．その結果，数式を検索するための技術の必要性が高まっている．

数式の検索に関する研究は複数行われている．それらの既存研究の多くは，問い合わせとして入力した数式と検索対象となる数式との関連性の強さをもとに検索を行う．このような手法を用いると，検索結果を順序付けて提示できる．この特徴は，検索エンジンを実現する上で特に有用である．一方，問い合わせの数式の位置を特定することができない．このため，「数式中の，問い合わせが記述されている箇所を探す」といった処理を実現できない．

そこで本研究では，パターンマッチング（文字列探索）に基づく数式の検索手法を提案する．また，パターンマッチングの，「問い合わせ（パターン）の存在する位置を特定することができる」という特徴を利用し，検索した数式のハイライト表示や，数式に対する置換処理といった処理も実装する．自然言語に対するハイライト表示は web ブラウザに搭載されている機能であり，数式に対するハイライト表示を行うことができれば，数式を含む文章を読む際に高い利便性が得られる．例えば，さまざまな関数について解説している web ページにおいて，正弦関数を分母に持つ関数について調べたい場合，その関数をハイライト表示できれば，高い利便性が得られるであろう．置換処理は，テキストエディタや文書編集ソフトウェアに搭載されている機能であり，数式の置換処理は，数式を含む文書を編集している際に効果を発揮する．例えば，2 人の著者が共同で文書を執筆しており，微分の表記を 1 人は$\frac{\mathrm{d}}{\mathrm{d}x}f(x)$，もう 1 人は$f'(x)$としていた場合を考える．ここで，$\frac{\mathrm{d}}{\mathrm{d}x}f(x)$という表記を$f'(x)$に置換できれば，表記を統一する作業の負担が軽減される．

本手法では，検索対象の数式の見た目が同一であれば同一の検索結果が得られるようマッチングを行う．一方，本研究で対象とする数式のデータ形式である MathML では，同一の数式を表示するためのデータが一意に定まらない．例えば，ab^2という数式を表すデータを作成する際，bの右上に上付き文字として 2 を配置することも，ab というまとまりの右上に 2 を配置することもできる．そこで，検索処理を行う前に MathML データに対して正規化処理を行うことにより，このようなデータのゆらぎを解消し，数式の表示が同一ならば検索結果も同一となるようなマッチングを実現した．

パターンマッチングの利便性を高めるために，正規表現を用いてパターンを記述する機能も実装した．正規表現における，ワイルドカード，記号の繰り返し，論理和や後方参照などの機能により，パターンの記述能力を飛躍的に高め，例えば「同一の分母を持つ分数同士の加算」といった数式をパターンマッチングで検索できるようにしている．ここで，数式の

データは木構造として表現されるが，正規表現には木構造を扱う能力がない．そこで本研究では，パターンを再帰的に処理できるよう拡張が施された正規表現エンジン Onigmo を利用することにより，正規表現で木構造を扱うことを可能とし，数式に対する正規表現を用いた検索を実現した．

正規表現を用いたパターンマッチングでは数式の意味を考慮した問い合わせを処理することができない．この問題を解決するための礎として，数式中の記号の意味推定を行う枠組みも提案および実装した．意味推定の例として，$|A|$という数式を考える．パターンマッチングにおいては，この数式は，‘|’, ‘A’, ‘|’という 3 つの記号が並んだものである．このため，「絶対値を検索する」,「行列式を検索する」,「集合の濃度（要素数）を検索する」といった問い合わせには対応できない．ここで，記号の意味を推定し，‘|’が「絶対値」「行列式」「濃度」のいずれの意味を持つかを判別できれば．意味的な問い合わせにも対応できるようになる．推定をする際の手がかりとして，「意味を推定する記号と同一の数式中に存在する他の記号」および「記号の順序」に着目した．例えば前者に関して，$|A \cap B|$という数式において，∩は集合に対する演算を表すため，‘|’という記号が「集合の濃度」を意味すると考えられるであろう．このような推定の手がかりを数値化した上で，機械学習を用いることにより，意味の推定を行った．

Mathematical expressions are commonly described in scientific documents. Electronic documents have made it simple to retrieve their contents. However, retrieving mathematical expressions correctly is still challenging because the characters in a mathematical expression are two-dimensionally located with its structure. In this study, a search method for mathematical expressions is proposed based on a pattern-matching algorithm. This method allows users to identify mathematical expressions in a large document. The author first normalizes the mathematical expression data in MathML Presentation Markup in order to eliminate data variety which can cause inaccuracies during the pattern-matching process. The proposed method also enables users to use regular expressions, such as wildcards, the Boolean “or,” and the iteration of character(s) to produce patterns more flexible. In addition, in this study, functions to highlight and replace the matched mathematical expressions are implemented based on the proposed method.

Mathematical expressions contain ambiguity in nature. The ambiguity often creates problems in their retrieval. For example, the mathematical expression $|A|$ can be interpreted as the absolute value of A, the cardinality of A or the determinant of A because of multiple meanings of the symbol “|.” As a result, when a user who searches for the absolute value of A inputs $|A|$ as the query, the results for all three possible interpretations will be generated. Here, to overcome this problem, a framework is proposed for labeling semantic information to each symbol through supervised learning.

目次

図目次

表目次

1. 序論

本論文は，数式の検索手法を提案するものである．本章では，数式検索に関する基礎的な考察を行い，提案する手法について概説する．まず，数式というものが持つ，表示および意味という 2 つの側面について説明する．次に，計算機上での数式利用，および数式のデータ形式について述べる．その後，数式の検索の動向といくつかの先行研究を紹介し，最後に，提案する手法の概要を示す．

1.1. 数式の表示と意味

数式は，数理的な概念やモデルを記述するために使われる．数学はもちろん，物理学や経済学など，数理的な手法を用いるさまざまな分野で用いられる重要な形式である．

数式は，厳密さを特徴とする学問である数学に端を発するため，数式の記法もまた厳格な規則に基づくものであると誤解されることが多い．しかし，数式の記法は注意深く設計されたものではなく，場当たり的な記法が慣用化される中で形成されてきたものである．実際，数学の先端分野では，現在も新たな記法が生み出され続けている[1]．そのため，数式の記法は非常に多様であり[2]，年代，地域，分野などによって，同一の内容が様々な数式で表現される．例えば，n 個のものから k 個を選ぶ組み合わせは，$\binom{n}{k}$や ${}_nC_k$と表現する地域もあれば，${}^nC_k$や$C_{n,k}$といった表記を用いる地域もある．分野による違いの例としては，虚数単位は数学ではiで表されるが，物理学（電磁気学）ではjで表される．

数式にはまた, 解釈が一意に定まらない場合もある. 例えば, $\binom{a}{b}\cdot\binom{c}{d}=s$という数式は, 2 つのベクトルの内積を求める式であると解釈できる．一方，$\binom{a}{b}$は a 個のうち b 個を選び出す組み合わせの数を表す場合もあるため，$\binom{a}{b}\cdot\binom{c}{d}=s$を 2 つの組み合わせの数の積であると解釈することもできる．このいずれの解釈が正しいかは，文脈に依存して決まる．

これらの 2 つの問題, つまり「同一の内容を様々な数式で表現できる」「同一の数式を様々に解釈できる」という問題があるため，数式について議論する際は，2 つの観点を明確に区別しなければならない．その 2 つの観点とは，表示および意味である．表示とは，紙に印刷された数式や, ディスプレイ上に映し出された数式について考える立場である．表示において，数式は，用いられる記号の種類や，記号の配置（ $\frac{1}{2}$ やx^2など），記号のフォントや色といった情報で表現される．意味は，数式に表現されている数学的な演算や関係を考える立場である．意味の立場からは，数式は，演算の種別（加算，べき乗，絶対値など），演算の対象，変数の型（自然数，実数など）といった情報で表される．

表示と意味という言葉を使うと，「同一の内容を様々な数式で表現できる」という問題は

「1 つの意味が複数の表示に対応する」と言い換えられ,「同一の数式を様々に解釈できる」という問題は「1 つの表示が複数の意味に対応する」と捉えられる．すなわち，数式には，表示と意味が一対一対応しない（多対多対応する）という問題がある．

1.2. 計算機上での数式利用

数式を用いるアプリケーションには様々なものがある．それらは，数式を表示と意味のいずれの観点で捉えているかに応じて大別できる．

表示に関する数式利用の代表的なものは，web ページや電子書籍などの，電子的な文書への数式の記述である．文書中に数式を記述するためには，数式の記号の種別や配置など，表示の情報が必要となる．表示に関連するアプリケーションとしては，数式の文字認識[3, 4]や，数式をブラウザ上で描画するためのライブラリ[5]が挙げられる．

意味に関する数式利用には，数式の変形やグラフの描画が挙げられる．これらの処理を実現するためには，数式中で用いられている演算などの情報がなければならない．変形やグラフの描画を行うアプリケーションは数式処理ソフトウェア[6, 7]と呼ばれる．数式処理ソフトウェアを利用して，学習者の解答を自動採点するような教育用ソフトウェアも提案されており[8, 9]，これも数式の意味の側面に着目したアプリケーションの一例である．

数式データの作成のためのエディタは，表示，意味の両面から開発されており，例えば[10, 11]などが挙げられる．文書編集ソフトウェアや数式処理ソフトウェアの中に，数式データを作成する機能を持つものも多い．

数式の表示と意味の多対多対応，特に，1 つの表示が複数の意味に解釈されるという対応関係は，表示に関するアプリケーションと意味に関するアプリケーションの連携を困難にし，計算機上での柔軟な数式を妨げる．例えば，電子書籍中に記述されている関数を選択し，その関数を変形したり，グラフを描画したりすることができれば，高い利便性が得られる．特に，電子教科書においてそういったアプリケーションを利用できれば，高い教育効果をもたらすであろう．しかしながら，電子書籍中に記述されている数式は表示に関する情報しか持たず，意味に関する情報を必要とするような処理を適用できない．表示と意味を横断するようなアプリケーションを実現するための最も直接的な方法は，表示と意味の両方のデータを併記する形で数式を記述するというものである．この方法で記述された数式を大量に集めたデータセットとして，Wolfram Functions Site[12]が知られている．しかし，表示と意味の両方の情報を作成することはコンテンツの製作者にとって負担が大きく，両方の情報が併記されることは少ない．

1.3. 数式のデータ形式

計算機上で数式を扱うためには，数式をデータとして表現する方法が必要となる．数式のためのデータ形式として，いくつかのものが提案されている．数式を用いるアプリケーションと同様に，数式のデータ形式もまた，表示と意味の観点から二分できる．

1.3.1. 表示

TeX [13]

TeX は，研究者を中心として普及している組版ソフトウェアであり，学術的な文書を作成する目的で利用されることが多い．TeX では，数式を記述するための簡潔な記法が用意されている．このため，数式エディタを用いることなく，マークアップを文字列として直接入力して容易に数式を作成することができる．数式データを描画するための環境も整備されており，学術論文や書籍の執筆において広く利用されている．

TeX による数式は，静的なコンテンツ上に数式を記述するには十分な能力を持つが，動的なコンテンツの作成には適さない．例えば，数式にマウスカーソルを乗せることでその数式に関する情報をポップアップで表示する，といった機能を実現するには不向きである．

MathML Presentation Markup [14]

MathML は，World Wide Web Consortium によって策定されているデータ形式である．MathML には 2 つのタグセットがあり，そのうちの 1 つが Presentation Markup である．Presentation Markup は特に，web ページにおける数式の記述を得意としている．これは，MathML が XML と呼ばれる汎用的なデータ形式に基づいており，XML に基づく他のデータ形式と連携したり，XML に関する記述を利用したりすることが容易なためである．具体的には，文書を構造化して記述するためのデータ形式であり web ページにおける標準となっている HTML（XHTML）と併用して文書中に数式を記述する，ベクタグラフィックスのためのデータ形式 SVG と併用して画像中に数式を埋め込む，といった例が挙げられる．また，web ブラウザ上で動作するプログラムを作成するためのプログラミング言語である JavaScript を用いれば，動的なコンテンツも容易に作成できる．

1.3.2. 意味

MathML Content Markup

MathML のもう 1 つのタグセットが，Content Markup である．Content Markup では，演算や関係，およびその対象を指定することで数式を記述する．数式処理ソフトウェア間で相互にデータを交換するような用途が想定されている．

MathML Content Markup もまた XML に基づいて定義されているため，XML 関連技術を用いることができる．Content Markup に対して特に有用な XML 関連技術は，XSLT である．XSLT は XML データ用のスタイルシートであり，XML データの変換に用いられる．XSLT を用いることで，MathML Content Markup のデータを MathML Presentation Markup のデータに変換することができる．1 つの数式の意味は様々な表示になり得るが，XSLT によって，どの表示を用いるかを比較的容易に指定できる．例えば，MathML Content Markup で記述された虚数単位を含む数式を web ページ中に記載する場合を考える．その web ページが複素

関数論に関するものであれば，虚数単位という意味をiという記号に変換するように XSLT を記述すれば良く．web ページが電磁気学に関するものならば，虚数単位をjという記号に変換するような XSLT を用意すれば良い．

OpenMath [15]

OpenMath の特徴は，Content Dictionaries と呼ばれる，数学的概念に関する情報を多数収録した辞書である．OpenMath は MathML Content Markup と協力的な関係にあり，MathML Content Markup において Content Dictionaries を参照する形でデータを記述することができる．また，MathML Content Markup には，OpenMath と互換性のある記法が用意されている．

1.4. 数式の検索

計算機上での数式の利用が進み，数式のデータが蓄積された結果，数式を検索するための技術の必要性が高まっている[16]．本節では，数式検索研究を，表示の情報を対象にした手法と意味の情報を対象とした手法に大別して概説する．なお，数式検索研究に関するサーベイとしては[17]や[18]が挙げられる．

1.4.1. 表示の情報に対する検索

表示の情報は意味の情報と異なり演算に関する情報を持たないが，何らかの方法で表示の情報から演算に関する情報を得た上で検索を行う手法が多い．それらの手法は，表示の情報から意味の情報への明示的な変換を伴うものとそうでないものとに大別できる．

表示から意味への変換を伴う研究は，数式の曖昧さの問題を意識しており，その影響を低減することを目指している．[19, 20, 21]は，同一の数式に対する表示の情報と意味の情報の組を大量に用意して機械学習を行うことにより，高い精度で表示から意味への変換を行うことを目指している．[22]は，検索とは独立して，表示から意味への変換を行うためのツールである．ルールベースの手法ではあるが，初等的な数式に対しては効果的に動作し，無償で利用できるツールとして実装されている．

明示的な変換を伴わない手法は，曖昧さの問題や対応可能な記号の多様さなどは意識せず，限られた記号に対してルールベースの変換を行う．[23]は，“+”という記号は演算を意味するといったことをルールベースで判定した上で，数式における項を単位として，数式における n-gram を定義してインデキシングを行ったり，演算の優先順位を判断したりしている．[24]は，MathML Presentation Markup のデータを演算の観点からルールベースで構造化した上で，ワイルドカードを用いた検索を行っている．

意味の側面には着目しない手法として，[25]では，MathML Presentation Markup のデータから得られる木構造のパスに着目することで，数式中の記号の位置構造をもとに数式を特徴付けている．[26]は，LaTeX で記述された数式に対して，マッチングによる検索を行う．ワイルドカードや論理和など，正規表現に類する機能を有しているが，記法は正規表現とは

大きく異なり，正規表現と比べて機能も乏しい．この手法は，数式中の記号を文字列にし，連結した上で，並べ替えを行うという，Youssef[27]の述べた典型的な数式検索手法のうちの 1 つであると言える．ActiveMath[28]もまた，数式を記号列にした上で検索を行う．全文検索のための Apache Lucene[29]を用いて実装しており，自然言語の文章の検索も同時に行う．

1.4.2. 意味の情報に対する検索

意味の情報には，数式中の演算とその対象が曖昧さなく記述されているため，演算に関する情報を利用して検索が行われる．意味の情報に対する検索が効果を発揮する場面は，数式データを多数収めたデータセットに対する検索である．これは，意味の情報が表示に関する情報を持たないという性質上，web ページ等に意味の情報が記述されることが少ない[30]ためである．データ形式については，MathML Content Markup が意味の情報を記述するために最も頻繁に用いられるものであるため，MathML Content Markup で記述された数式を検索対象として想定しているものがほとんどである．

数式から特徴を抽出し，特徴間の関連性に基づいて検索を行う手法としては，[31,32]が挙げられる．[31]は，MathML Content Markup のデータの木構造のパスで数式を特徴付け，パスの集合の間の類似度を元に検索を行う．[32]は，数式中の演算の情報をもとに格子を構築した上で，問い合わせを格子に配置し，格子中の数式間の距離をもとに検索を行う．MathWebSearch[33, 34]はマッチングに基づく検索手法であり，問い合わせの記述に XML を用いる．また，インデキシングに自動定理証明に基づく方法を利用する．

Wolfram|Alpha[35]は，問い合わせに関する知識を提供するという特殊な検索システムである．数式を問い合わせとして入力することも可能であり，その数式のグラフや，微分や積分を行った後の数式などを提示する．

1.5. 本論文の概要

本論文では，数式の表示の情報を対象とした検索手法を提案する．検索処理には，自然言語における文字列探索に類するパターンマッチングを用いる．

既存研究の多くは，問い合わせとして入力した数式と検索対象となる数式との関連性の強さをもとに検索を行う．このような手法を用いると，検索結果を順序付けて提示できる．この特徴は，検索エンジンを実現する上で特に有用である．一方，問い合わせの数式の位置を特定することができない．このため，「数式中の，問い合わせが記述されている箇所を探す」といった処理を実現できない．本研究で提案するパターンマッチング（文字列探索）に基づく数式の検索手法を用いると，問い合わせ（パターン）の存在する位置を特定することができるため，検索した数式のハイライト表示や，数式に対する置換処理といった応用を実現できる．

自然言語に対するハイライト表示は web ブラウザに搭載されている機能であり，数式に対するハイライト表示を行うことができれば，数式を含む文章を読む際に高い利便性が得

られる．例えば，さまざまな関数について解説している web ページにおいて，正弦関数を分母に持つ関数について調べたい場合，その関数をハイライト表示できれば，高い利便性が得られるであろう．置換処理は，テキストエディタや文書編集ソフトウェアに搭載されている機能であり，数式の置換処理は，数式を含む文書を編集している際に効果を発揮する．例えば，2 人の著者が共同で文書を執筆しており，微分の表記を 1 人は$\frac{\mathrm{d}}{\mathrm{d}x}f(x)$，もう 1 人は$f'(x)$としていた場合を考える．ここで，$\frac{\mathrm{d}}{\mathrm{d}x}f(x)$という表記を$f'(x)$に置換する機能があれば，表記を統一する作業の負担が軽減される．

本手法では，数式の意味の側面にはまったく着目せず，純粋に表示の情報のみを用いてパターンの記述およびパターンマッチングを行う．意味に着目すると，表示から意味への変換処理が必要となり，変換処理における誤りが，検索処理の正確さに影響してしまう．意味に着目しないことで，ユーザの入力したパターンを確実に検索するシステムを実現できる．ここで，表示の情報における意味の側面は，ただ何もしなければ無視できるというものではない．なぜなら，表示の情報のためのデータ形式が，暗黙的に意味の情報を持ってしまっているためである．本研究では，数式のパターンマッチングを実現する上で扱うべき表示の情報とは何であるかを厳密に定め，表示の情報のためのデータ形式を詳細に検討した上で，数式のデータに対して正規化と呼ぶ処理を施し，数式の表示が同一である場合にはデータが一意に定まるようにする[36]．正規化によって，表示の情報のパターンマッチングにおいて精度の低下の原因となるような意味の情報を取り除くこともできる．

本手法の特徴として，パターンの記述に正規表現を利用できるという点が挙げられる[37]．正規表現を用いることで，ワイルドカードや，記号の繰り返しを表現できる．数式の検索において特に有用な機能は，後方参照である．後方参照を用いることで，「同一の分母を持つ分数同士の加算」といったパターンを表現できるようになる．一方，正規表現を用いたパターンは複雑であり，入力が煩雑になり得るという問題がある．これを解決するために，パターンを作成するためのグラフィカルユーザインタフェース（GUI）も提案する[38]．

正規表現を用いたパターンマッチングにおいては，検索対象も表示の情報であり，パターンも表示の情報である．これにより，検索を正確に行うことができるようになるが，処理することのできる問い合わせの幅は狭くなる．これを解決するために，数式の意味の推定にも取り組む[39]．意味推定によって，表示の情報に対して意味のラベルを与えることができるようになる．これは，「微分」や「絶対値」といった，演算に着目した意味的な問い合わせを受け付ける検索システムのための礎となるものである．

図 1.1 に，表示の情報および意味の情報と，本論文で示す各処理の関係を示す．

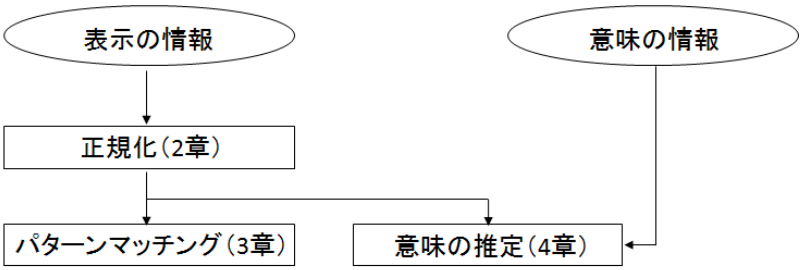


図 1.1 本論文における各処理の関係

正規化によって，表示の情報が一意なデータに定まる．その正規化されたデータに対してパターンマッチングを行うことで検索を実現する．正規化は，意味の推定の前処理にもなっている．意味推定によって，正規化された数式中の記号に対して意味のラベルを与えることができる．意味推定では学習という過程が必要となり，その過程において意味の情報を利用する．

2. 正規化

数式の検索手法を考える上で，計算機上で数式を表現するためのデータ構造は，検索の機能や機能の実装方法に非常に大きな影響を与える．このため，検索の目的に適うようにデータ構造を定義しなければならない．一方，1.3. で述べたように，数式を表現するためのデータ形式として広く知られたものがすでに存在している．検索を実際に使用することを考えると，検索手法はすでに普及したデータ形式に対応している必要がある．本研究では，広く普及している形式である MathML Presentation Markup（以下，MathML とは MathML Presentation Markup を指すものとする）のデータに対して正規化処理を適用することで，提案手法に適したデータ構造の実現と，普及したデータ形式への対応とを両立する．

本章ではまず，本研究における数式の表示とは何であるかを述べた上で，数式の表示に関する情報を表現するためのデータ構造を示す．次に，本研究で検索の対象とするデータ形式である MathML について述べる．MathML を対象とする理由とともに，MathML の仕様を概説する．その後，本研究における数式の表示のデータ構造と MathML との間との差異を示し，MathML データの正規化の必要性を述べ，最後に，正規化の手順を詳述する．

2.1. 本研究における数式の表示

本研究において，数式の表示の情報とは次の 2 つを指す．

- 記号の種類
- 記号の位置

以下の情報は，本研究では表示の情報として扱わない．

- 記号の書体
- 記号の色

ここで，記号の書体は，数式において重要な役割を持つ．例として，ボールド体について考える．自然言語におけるボールド体は，多くの場合，単に強調のために用いられる．一方，数式においては，ある識別子がベクトルや行列であるということなどを示すためにボールド体が用いられる．このため，$|\boldsymbol{a}|$ を検索したいが，$|a|$ は検索結果に含めたくはない，といった状況があり得る（この場合ユーザは，ベクトルのノルムを検索することを望み，数の絶対値を検索することは望まなかったのであろう）．しかしながら，本研究では記号の書体を表示の情報として扱わない．これは，書体の情報は正しく付与されないことが多いためである．数式データを作成するための主要な方法の 1 つとして，数式を認識する機能を持つ OCR ソフトウェアの利用が挙げられるが，OCR ソフトウェアは多くの場合，ボールド体を正しく認識しない．このため，表示の情報として書体を扱い書体を考慮した検索を行うと，データの誤りが原因でユーザの望まない検索結果が得られてしまう可能性がある．書体を区別した検索を行う機能を提供することで利便性を向上させるよりも，検索結果が不適切

になるという弊害を避ける方が望ましいと考えるため，本研究では書体を考慮しない．

2.2. 数式の表示の情報の表現方法

本研究では，数式の表示の情報を木構造として表現する．木の節は，構造節，統合節，記号節の 3 つに大別できる．構造節は，上付き文字や分数など，記号の位置構造を表現する．統合節は，記号および位置構造の横方向のまとまりを表す．記号節は，個々の記号を表現する．構造節の子となるのは統合節のみである．構造節は 9 種類存在し，構造節の種類ごとに，子となる統合節の数が決まっている．また，何番目の子であるかによって，構造節における役割が定まる．構造名，その構造節で表現できる数式の例，および構造節の子となる統合節の役割を，表 2.1 に示す．

表 2.1 構造節の一覧

構造名	数式の例	子の出現順序		
		1	2	3
下付き文字	$_1$	下付き文字		
上付き文字	^{2}	上付き文字		
上下付き文字	$_{-\infty}^{\infty}$	下付き文字	上付き文字	
下文字	$\lim_{n\to\infty}$	基底	下部の文字	
上文字	$\dot{x}$	基底	上部の文字	
上下文字	$\sum_{i=0}^{n}$	基底	下部の文字	上部の文字
分数	$\frac{1}{2}$	分子	分母	
平方根	$\sqrt{x}$	根号の内部		
冪根	$\sqrt[3]{z}$	根号の内部	根指数	

統合節は，1 つ以上の構造節と記号節を子に持つ．記号節は子を持たない．

例として，数式$\left(a+\frac{\sqrt{3}}{2}\right)x^n$を木構造で表現したものを図 2.1 に示す．

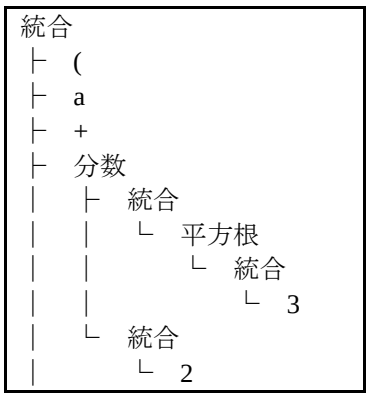


図 2.1 $\left(a+\frac{\sqrt{3}}{2}\right)x^n$の木構造による表現

図 2.1 において，構造節は分数，平方根，および上付きである．それぞれの統合節は，その位置に記述される数式を子として持っていることがわかる．

ここで，数式を表現するための木構造として，演算の優先順位を表す木構造が知られているが，本研究における木構造は数式中の文字の位置構造を表現したものであり，これは演算の優先順位を示す木構造とは全く異なるものである．そもそも，本研究において数式は表示の情報で表されるため，演算という概念が存在しない．なお，MathML Content Markup のデータから得られる木は，演算の優先順位を表現している．

2.3. MathML

MathML の概要と MathML の仕様の全体像を示した後，本研究において対象となる MathML の機能について述べる．また，本研究で MathML を対象とする理由も述べる．

2.3.1. MathML の概要

MathML では，タグを用いて記号の配置を指定する．例えば $x+y^2$ という数式は次の図 2.2 のように記述される．

```
<math>
    <mi>x</mi>
    <mo>+</mo>
    <msup>
        <mi>y</mi>
        <mn>2</mn>
    </msup>
</math>
```

図 2.2 MathML の例

このデータについて簡単に説明する．mi 要素は識別子，mo 要素は演算子，mn 要素は数を表すための要素である．msup 要素は，第一の子の上付き文字として第二の子を配置するという機能を持つ．すなわち，このデータは，識別子 x，演算子+，識別子 y の右上に数 2 が配置されたもの，この 3 つが並んだものであるということを表現している．

MathML データは入れ子構造を持っており，入れ子の外側を親，内側を子とすることで木構造として表現することができる．例として，$\left(a+\frac{\sqrt{3}}{2}\right)x^n$の MathML データと，それを木として表現したものを，図 2.3 に示す．

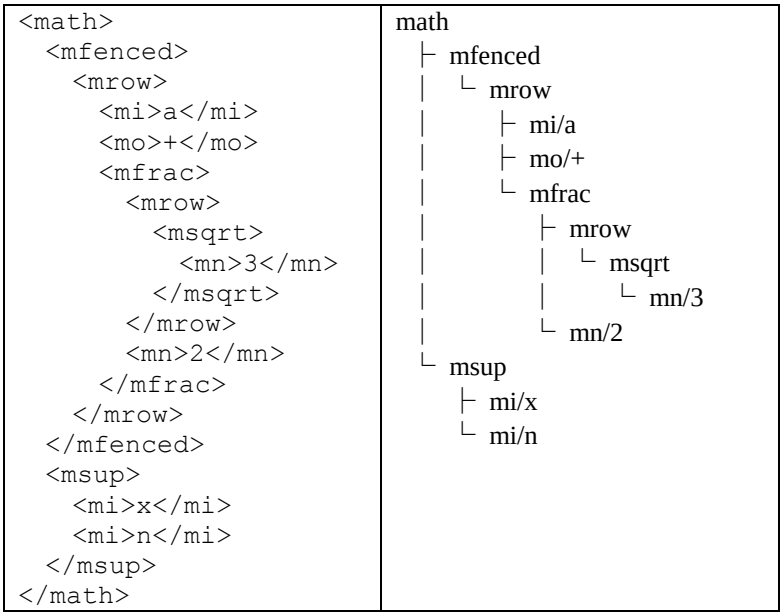


図 2.3 MathML のデータと木構造による表現

なお，mfenced 要素はその子を括弧記号で囲むための要素，mfrac 要素は第一の子を分子，第二の子を分母として記号を配置する要素である．

2.3.2. MathML の仕様

MathML 仕様の全体像を，MathML の仕様書（MathML3.0 2nd Edition, 3 Presentation Markup）に従いつつ概説する．3 Presentation Markup は 8 つの節からなる．1 節は概要，8 節は Content Markup との併記に関する説明であり，MathML の機能についての説明は 2 節から 7 節である．

3.2 Token Elements では，数式中の個々の記号について述べられている．3.2 において重要な要素は，識別子，数，および演算子を表す，mi 要素，mn 要素，mo 要素である．これら 3 つの要素は記号列（テキストノード）を子として持つ．ここで，記号列が unicode を用いて記述されるという点は特筆すべきである．unicode には，数式のための文字セットが定義されており[40]，∫や∂などの記号はもちろん，∰ や∇といった比較的高度な数学で用いられる記号も含まれている．また，広範な分野の記号を収めており，∈やא，∧やヨといった記号も含まれる．さらに，≧と≥なども別の記号として定義されている．3.2 では他に，unicode にない記号を自ら定義するための mglyph 要素，数式中に文を記述するための mtext 要素，空白を表示するための mspace 要素，文字列リテラルのための ms 要素について述べられている．

3.3 General Layout Schemata では 10 の要素について述べられている．mrow 要素は，数式の横方向のまとまりを明示するための要素である．mrow 要素が必要となる例を図 2.4 に示す．

```
<mfrac>
    <mrow>
```

```
        <mn>3</mn>
        <mi>x</mi>
    </mrow>
    <mn>2</mn>
</mfrac>
```

図 2.4 mrow 要素の使用例

mfrac 要素は分数を表現するためのものであり，第一の子を分子，第二の子を分母として記号を配置する．図において，mfrac 要素の第一の子の網掛けで示した箇所で mrow 要素が用いられている．ここでもし mrow 要素がなければ，3, x, 2 のうち，どの箇所が分子であり，どの箇所が分母であるのかが定まらない（厳密には，mfrac 要素が 3 つの子を持ってしまうため妥当な MathML と解釈されない）．mfrac 要素，msqrt 要素，mroot 要素，mfenced 要素は，位置構造を表現するための要素である．それぞれの要素を用いて記述することができる数式の例を，表 2.2 に示す．

表 2.2 「3.3 General Layout Schemata」における要素とその機能

要素名	数式の例
mfrac	$\frac{1}{2}$
msqrt	$\sqrt{x}$
mroot	$\sqrt[3]{z}$
mfenced	$(a+b)$

他に，数式中の記号のフォントサイズや色を指定するための mstyle 要素，囲み線を表示するための menclose 要素，記号の位置を揃えるなど体裁を整えるための mpadded 要素および mphantom 要素，MathML が妥当でない場合のエラー出力を指定するための merror 要素が定義されている．

3.4 Script and Limit Schemata は，記号の配置を指定するための要素に関する節である．表 2.3 に，要素名と，その要素を用いて記述できる数式の例を示す．

表 2.3 「3.4 Script and Limit Schemata」における要素とその機能

要素名	数式の例
msub	a_1
msup	x^2
msubsup	$\int_{-\infty}^{\infty}$
munder	$\lim_{n\to\infty}$
mover	$\dot{x}$
munderover	$\sum_{i=0}^{n}$

他に，添字を一括して配置するための mmultiscripts 要素，mprescripts 要素，none 要素が定義されている．

3.5 Tabular Math では，行列を記述するための機能について述べられている．3.6 Elementary

Math では，筆算の記法を表現するための要素が定義されている．3.7 Enlivening Expressions は動的なコンテンツを作成するための要素に関するものであり，数式をクリックした際の動作を指定する要素などについて述べられている．

2.3.3. 本研究で対象とする MathML の要素

本研究において, MathML の要素は, 2.2. で述べた表示の情報の木構造を表すためのもの, 検索対象にのみ含まれるもの, 正規化により除外されるもの, 本研究では対応しないものの 4 つに大別できる．

表示の情報の木構造を表すための MathML の要素は 11 ある．このうち 2 つは mi 要素および mrow 要素であり，それぞれ，記号節および統合節を表すために用いられる．残りの 9 つは位置構造を表すための要素である. これらは表 2.1 に示した 9 つの構造節に対応する．表 2.1 に，構造名に対応する MathML の要素名を加えたものを，表 2.4 に示す．

表 2.4 構造名と，それに対応する MathML の要素名

構造名	数式の例	何番目の子であるか			MathML の要素名
		1	2	3	
下付き文字	$_1$	下付き文字			msub
上付き文字	^{2}	上付き文字			msup
上下付き文字	$_{-\infty}^{\infty}$	下付き文字	上付き文字		msubsup
下文字	$\lim_{n\to\infty}$	基底	下部の文字		munder
上文字	$\dot{x}$	基底	上部の文字		mover
上下文字	$\sum_{i=0}^{n}$	基底	下部の文字	上部の文字	munderover
分数	$\frac{1}{2}$	分子	分母		mfrac
平方根	$\sqrt{x}$	根号の内部			msqrt
冪根	$\sqrt[3]{z}$	根号の内部	根指数		mroot

検索対象にのみ含まれる要素は，行列およびテーブルに関する要素（3.5 Tabular Math），動的な機能に関する要素（3.7 Enlivening Expressions）である．

正規化により除外される要素は，mn 要素，mo 要素，mglyph 要素，mtext 要素，mspace 要素，ms 要素，mfenced 要素，mstyle 要素，menclose 要素，mmultiscripts 要素，mprescripts 要素，none 要素，mpadded 要素，mphantom 要素，merror 要素，malignmark 要素，malignmark 要素である．これらを除外する理由および除外する方法は後述する．

本研究で対応しない要素は，筆算の記法に関する要素（3.6 Elementary Math）である．筆算に関する要素はまだ普及が進んでおらず, MathML の表示のための主要なライブラリである MathJax でも，ブラウザの中では最も MathML の対応が進んでいる Firefox でも完全な形では表示できない．

2.3.4. MathML を対象とする理由

本研究で MathML を対象とする理由は 3 つある．第一に，MathML は広く普及した形式であり[1], MathML は web ページにおける標準的な規格である HTML や電子書籍の規格 EPUB において，数式のためのデータ形式として採用されている．本研究で提案する検索手法は，web ページや電子書籍などの電子的な文書中に記述された数式を検索対象とすることを想定しているため，MathML は本手法の対象として適切である．第二に，他のデータ形式から MathML へと変換するツールが充実している．特に，TeX から MathML への変換を行うツール[41]が提案されているため，これを用いることで，TeX で記述された数式に対しても本手法を適用できる．第三に，MathML のデータは木構造として扱うことが容易である．本研究では数式の表示の情報を木構造として表現する. また, MathML は XML の一種であり, XML を木構造として扱うための DOM(Document Object Model)という技術が確立している．このため，MathML は本手法と親和性が高い．しかし，本研究における数式の表示の木構造と MathML の木構造との間には隔たりがある．この隔たりを埋めるための処理が正規化である．

2.4. 正規化の必要性

2.2. で述べた本研究における数式の木構造と 2.3.1. で述べた MathML の木構造とは，類似しているものの異なる点も多い．このため，この 2 つの木構造を同一のものとするために，データの正規化を行う．

正規化は，単に本研究における提案手法を実現するという目的のみならず，MathML を利用する研究一般において有用なものとなり得る．これは正規化によって「データのゆらぎの解消」が実現されるためである．

データのゆらぎとは，MathML において，同一の数式を表現するためのデータが一意に定まらないことを指す．データのゆらぎには，2 つの原因がある．1 つは，MathML による記号および記号の位置の表現方法である．例として，msup 要素が挙げられる．msup 要素は上付き文字を記述するための要素であり，第一の子が基底，第二の子が上付き文字となる．ここで，例えばab^2という数式は，abを基底とすることも，bを基底とすることも，空要素を基底とすることも可能である．つまり，図 2.5 のデータはすべて，ab^2という同一の数式を表している．なお，網掛け部が基底である．

```
<math>
    <msup>
        <mrow>
            <mi>a</mi>
            <mi>b</mi>
        </mrow>
        <mn>2</mn>
    </msup>
</math>
```

```
<math>
```

```
    <mi>a</mi>
    <msup>
        <mi>b</mi>
        <mn>2</mn>
    </msup>
</math>
```

```
<math>
    <mi>a</mi>
    <mi>b</mi>
    <msup>
        <mrow/>
        <mrow>
            <mi>2</mi>
        </mrow>
    </msup>
</math>
```

図 2.5 ab^2のデータのゆらぎ

このようなデータのゆらぎは, 問題を生じさせ得る. 例えば, MathML の木構造に着目して, 葉までのパスを活用するような手法[25]があるが, ab^2におけるbのパスは図 2.5 のうちいずれの形で表現されているかによって異なり, 図 2.5 の上のデータでは math/msup/mrow/mi/b, 中央のデータでは math/msup/mi/b, 下のデータでは math/mi/b となる. しかし, いずれのパスもab^2という数式におけるbであるため, これらが互いに異なることは不適切であろう.

もう 1 つのゆらぎの原因は, 不完全な意味の情報である. MathML Presentation Markup は, 記号の表示に関する情報を記述することが一義の目的であるが, 意味の情報と呼ぶべきものを記述する機能も持っている. 例えば, 記号が識別子, 数字, 演算子のいずれであるかを区別するために, mi 要素, mn 要素, mo 要素の 3 つの要素が用意されている. しかし, 識別子, 数字, 演算子といった記号の役割に関する情報は意味の情報であると言え, 少なくとも, 数式を表示するためには必要のない情報である. 例えば, <mi>a</mi>と<mo>a</mo>は, ともに妥当な MathML データであり, 同一の数式の表示aを表現するものであるが, データが異なったものとなってしまっている. つまり, mi 要素, mn 要素, mo 要素の区別は, 同一の数式が複数のデータで表現される原因になる（なお, 厳密には, mi 要素, mn 要素, mo 要素のいずれを用いるかによって, 記号間の間隔が微調整される場合がある）.

データのゆらぎは, 単に仕様上発生し得るというだけでなく, 実際に頻繁に発生する. ゆらぎの原因として, 第一に, データの作成ツールの違いがある. 数式データは, 数式エディタによる記述, 他形式から MathML への変換, OCR ソフトウェアによる数式認識など, 様々な方法で作成される. そのそれぞれの作成ツールの間で, MathML データにゆらぎがある. ツールの違いに起因するゆらぎとして最も顕著なものは, mrow 要素の位置である. mrow 要素は横方向のまとまりを表現するための要素であり, 表示に直接的には反映されない. また, mrow 要素は, 記述される箇所に関する制約がほとんどない. 例えば, 以下の図 2.6 中のデータはすべて, $\frac{1}{ab}$という数式を表す.

```
<math>
    <mfrac>
```

```
        <mn>1</mn>
        <mrow>
            <mi>a</mi>
            <mi>b</mi>
        </mrow>
    </mfrac>
</math>
```

```
<math>
    <mfrac>
        <mrow>
            <mn>1</mn>
        </mrow>
        <mrow>
            <mi>a</mi>
            <mi>b</mi>
        </mrow>
    </mfrac>
</math>
```

```
<math>
    <mrow>
        <mfrac>
            <mrow>
                <mn>1</mn>
            </mrow>
            <mrow>
                <mrow>
                    <mi>a</mi>
                </mrow>
                <mrow>
                    <mi>b</mi>
                </mrow>
            </mrow>
        </mfrac>
    </mrow>
</math>
```

図 2.6 mrow 要素に起因するデータのゆらぎ

上のデータは最小限の mrow 要素が用いられており，下のデータではかなり冗長に mrow 要素が記述されているが，これらは同一の数式を表す．mrow 要素が記述される箇所は，同一のツールにおいては一貫した規則に従っているが，異なるツールの間では差異が生じる．

第二に，データの作成手順の違いもゆらぎの原因となる．例として，Microsoft Word の数式エディタ機能を用いて数式ab^2の MathML データを作成する場合を考える．記述の手順として，上付き文字のテンプレートを入力して$\square^{\square}$とし，基底に相当する矩形に ab と入力して$ab^{\square}$とし，上付き文字の矩形に 2 と入力してab^2とした場合，図 2.7 の左側のデータが作成される．一方，記号aを入力した後，上付き文字のテンプレートを入力して$a\square^{\square}$とし，基底を入力して$ab^{\square}$，上付き文字を入力してab^2とした場合，図 2.7 の右側のデータとなる．

```
<math>
    <msup>
        <mrow>
            <mi>a</mi>
            <mi>b</mi>
        </mrow>
```

```
<math>
    <mi>a</mi>
    <msup>
        <mrow>
            <mi>b</mi>
        </mrow>
```

```
        <mrow>
            <mn>2</mn>
        </mrow>
    </msup>
</math>
```

```
        <mrow>
            <mn>2</mn>
        </mrow>
    </msup>
</math>
```

図 2.7 データの作成手順に起因するデータのゆらぎ

この 2 つの数式には見かけ上まったく違いがないため，このようなゆらぎは頻繁に発生すると考えられる．

既存の数式検索研究は，1 つのデータセットを対象として検索の評価を行っている．このため，ゆらぎを考慮しない手法を用いても，精度の高い検索を実現できているように見える．しかし現実には，前述のように，データのゆらぎは頻繁に起こる．このため，検索手法を実用化する上では，データのゆらぎへの対処は不可欠であると言える．本手法は，正規化によってデータのゆらぎの問題を解決することにより，様々な方法，様々な手順で作成された数式に対して正しく検索を行うことのできるような，頑健な検索を実現する．

2.5. 正規化の手順

正規化の処理は，ゆらぎの原因となる要素ごとに別個に行う．本節では，それぞれの処理について詳述する．

2.5.1. 記号の表現方法の一意化

MathML では，記号を表現するための要素が複数あり，それぞれの要素は記号の役割（識別子，数，演算子など）を表す．しかし，前述のように，この情報は表示に反映されず，データのゆらぎの原因となる．そのため，記号を表現するために用いる要素をただ 1 つだけに限る．また，記号を記述するための要素は，複数の文字を子に持つことも可能であり，例えば，<mi>sin</mi>というデータも<mi>s</mi><mi>i</mi><mi>n</mi>というデータもあり得る．複数の文字が子となる場合はフォントがノーマル体となるため，前者のデータからはsin，後者からは*sin*という数式が得られる．本研究では，記号のフォントには着目しないため，これらの数式を同一のものとみなすべきである．このため，記号を記述するための要素は，ただ 1 つの記号のみを子として持つようにする．なお，複数の文字を子に持つことができるという性質は，フォントも含めてまったく同一の数式が異なるデータで表現される原因ともなる．例えば，<mi>ab</mi><mi>cd</mi>と，<mi>abcd</mi>は，ともにノーマル体のabcdという数式を表す．

正規化の手順として，記号を表現する要素である mn 要素，mo 要素，mtext 要素，ms 要素を，すべて mi 要素へと置換する．その際，ms 要素にだけは特別な処理を行う．ms 要素は，その子がダブルクオーテーションで囲まれた形で描画される．例えば，<ms>abc</ms>は，"abc"と描画される．このため，ms 要素を mi 要素に置き換える際は，要素の前後に（兄弟として）<mi>"</mi>を挿入する．すなわち，<ms>abc</ms>は

<mi>"</mi><mi>abc</mi><mi>"</mi>に置き換えられる．置き換えで得られた mi 要素が複数の記号を子に持つ場合，その記号列を記号ごとに分割した上で，mi 要素を，1 つの記号を子に持つ mi 要素の列に置換する．例えば，<mi>sin</mi>というデータは，<mi>s</mi><mi>i</mi><mi>n</mi>となる．

2.5.2. 表示に反映されない演算子の除去

unicode における数式のための記号には，透明な記号が含まれている．具体的には，「関数の適用（ApplyFunction）」「透明な乗算（InvisibleTimes）」である．次の図 2.8 および図 2.9 のように用いられる．

```
<mi>f</mi>
<mo>&ApplyFunction;</mo>
<mfenced>
    <mi>x</mi>
</mfenced>
```

図 2.8 関数の適用の例

```
<mn>2</mn>
<mo>&InvisibleTimes;</mo>
<mi>x</mi>
```

図 2.9 透明な乗算の例

これらは，意味を表現するためのものであると言え，表示に反映されないため，除去する．すなわち，「透明な乗算（InvisibleTimes）」「関数の適用（ApplyFunction）」を子に持つ要素を削除する．

2.5.3. mrow 要素の位置の一意化

mrow 要素は，図 2.6 に示したように，ゆらぎの原因となる．mrow 要素によるゆらぎを解消するためには，mrow 要素を記述しなければならない可能性がある箇所にのみ，漏れなく mrow 要素を記述すれば良い．

具体的には，表 2.4 に示した要素の子はすべて mrow 要素となるようにする．行列およびテーブルに関する要素については，mtd 要素の子はすべて mrow 要素となるようにする．動的な機能に関する要素については, maction 要素の子はすべて mrow 要素となるようにする．

それ以外の箇所に出現する mrow 要素はすべて除去する．

2.5.4. msub 要素，msup 要素，msubsup 要素の基底の一意化

msub 要素，msup 要素，msubsup 要素は，基底となる記号に対し，上付き文字，下付き文字，上下付き文字を付加するための要素である．図 2.7 で msup 要素を例に述べたように，これらの要素は基底となる記号を任意に取ることができ，それがゆらぎを生じさせる原因となる．

これを解消するために，これら 3 つの要素の第一の子を，これらの要素の直前に移動し，第一の子には空の mrow 要素を挿入する．すなわち，図 2.7 の上および中央のデータはともに，下のデータへと一意化される．この処理によって，上付き文字，下付き文字，上下付き文字は，基底をもたなくなる．

2.5.5. mmultiscripts 要素，mprescripts 要素，none 要素の除去

mmultiscripts 要素，mprescripts 要素，none 要素は，上付き文字および下付き文字を一括して配置するという機能を持つ．しかし，これらの要素を用いた数式と同一の数式を，msub 要素，msup 要素，msubsup 要素を用いて作成できる．このゆらぎを解消するために，mmultiscripts 要素，mprescripts 要素，none 要素を削除し，msub 要素，msup 要素，msubsup 要素に置き換える．

mmultiscripts 要素の一般的な記法を，図 2.10 に示す．

```
<mmultiscripts>
    基底となる要素
    基底の右の下付き文字
    基底の右の上付き文字
    基底の右の右の下付き文字
    基底の右の右の上付き文字
    …
    <mprescripts/>
    基底の左の下付き文字
    基底の左の上付き文字
    …
</mmultiscripts>
```

図 2.10 mmultiscripts 要素の記法

例えば，${}_{0}R_{1_i}^{2\ j}$という数式は，次の図 2.11 のデータから得られる．

```
<mmultiscripts>
    <mi>R</mi>
    <mn>1</mn>      ┐
    <mn>2</mn>      ┘ ①
    <mi>i</mi>      ┐
    <none/>         ┘ ②
    <none/>         ┐
    <mi>j</mi>      ┘ ③
    <mprescripts/>
    <mn>0</mn>      ┐
    <none/>         ┘ ④
</mmultiscripts>
```

図 2.11 mmultiscripts 要素の例

①では，下付き文字も上付き文字も指定されており，この場合，上下付き文字が表示される．②では，上付き文字の部分に none 要素が用いられているため，下付き文字が表示される．③は，下付き文字部分が none 要素となっているため，上付き文字が表示される．④は，mprescripts 要素の後ろに記述されているため，基底である記号 R の左側に表示される．

mmultiscripts 要素，mprescripts 要素，none 要素を削除し，msub 要素，msup 要素，msubsup 要素に置き換える手順を，擬似コードとして図 2.12 に示す．

```
1  var children;                    //mmultiscripts 要素の子の配列
2  children[0]を mmultiscripts 要素の直前に移動する;
3  var i;
4  var preFlag = false;             //mprescripts 要素を走査したら true になる
5  for(i=1; i<mmultiscripts の子の数; i+=2)
```

```
6   {
7       if(children[i] == mprescripts)
8       {
9           preFlag = true;
10          i += 1;
11      }
12      else if(children[i] != none    AND    children[i+1] != none)
13      {
14          msubsup 要素を作る;
15          msubsup の第一の子を空の mrow，第二の子を children[i]，第三の子を children[i+1]とする;
16      }
17      else if(children[i] != none    AND    children[i+1] == none)
18      {
19          msub 要素を作る;
20          msubsup の第一の子を空の mrow，第二の子を children[i]とする;
21      }
22      else if(children[i] == none    AND    children[i+1] != none)
23      {
24          msup 要素を作る;
25          msubsup の第一の子を空の mrow，第二の子を children[i+1]とする;
26      }
27      if(preFlag)
28      {
29          作った要素を children[0]の直前に移動する;
30      }
31      else
32      {
33          作った要素を mmultiscript 要素の直前に移動する;
34      }
35  }
36  mmultiscripts 要素を除去する;
```

図 2.12 mmultiscripts 要素の除去の擬似コード

中心となる処理は，12-26 行目である．ここで，none 要素を考慮して適切に msubsup 要素，msub 要素，msup 要素を構築する．7-11 行目で mprescripts 要素をチェックする．チェックの結果に応じて，msubsup 要素，msub 要素，msup 要素を基底となる記号の前に配置するか後ろに配置するかを決定する処理が，27-34 行目で行われる．

図 2.11 を正規化したものを図 2.13 に示す．なお，mmultiscripts 要素の除去だけでなく，ここまでに述べた他の正規化処理も適用している．

```
<msub>
    <mrow/>
    <mrow>
        <mi>0</mi>
    </mrow>
</msub>
<mi>R</mi>
<msubsup>
    <mrow/>
    <mrow>
        <mi>1</mi>
    </mrow>
    <mrow>
        <mi>2</mi>
    </mrow>
```

```
</msubsup>
<msub>
    <mrow/>
    <mrow>
        <mi>i</mi>
    </mrow>
</msub>
<msup>
    <mrow/>
    <mrow>
        <mi>j</mi>
    </mrow>
</msup>
```

図 2.13 正規化された図 2.11 のデータ

2.5.6. mfenced 要素の除去（括弧記号の表現方法の一意化）

括弧記号を表すための要素として，mfenced 要素がある．一方，括弧記号は，mo 要素を用いて記述することもできる．これらの違いは表示には表れない（厳密には，括弧記号の大きさの最適化に関して，例えば$\left(\frac{1}{2}\right)$と$(\frac{1}{2})$のような違いが生じ得るが，これは本研究における表示の情報ではない）．すなわち，ゆらぎの原因となるため，括弧記号の表現方法を一意化する．

一意化の処理としては，mfenced 要素を mi 要素で囲まれた記号に置き換える．ここで，mfenced 要素には，子をセパレータで分割するという機能も持つ．例えば，次の図 2.14 は，$f(x,y)$を表すデータである．

```
<mi>f</mi>
<mfenced>
    <mi>x</mi>
    <mi>y</mi>
</mfenced>
```

図 2.14 mfenced 要素を用いたセパレータによる分割

さらに，括弧記号およびセパレータ記号の種類を，属性を用いて指定することも可能である．次の図 2.15 は，$\{3x|x \in N\}$を表す．{ ， } が括弧記号，| がセパレータ記号である．

```
<mfenced open="{"
         close="}"
         separators="|">
    <mrow>
        <mn>3</mn>
        <mi>x</mi>
    </mrow>
    <mrow>
        <mi>x</mi>
        <mo>∈</mo>
        <mi>N</mi>
    </mrow>
</mfenced>
```

図 2.15 mfenced 要素における括弧記号，セパレータ記号の指定

このため，mfenced 要素の除去を行う際は，属性と子の数を考慮し，mo 要素への置き換えを行う．図 2.16 に，図 2.15 に対して正規化を施した結果を示す．なお，mfenced 要素の除去だけでなく，ここまでに述べた他の正規化処理も適用している．

```
<mi>{</mi>
<mi>3</mi>
<mi>x</mi>
<mi>|</mi>
<mi>x</mi>
<mi>∈</mi>
<mi>N</mi>
<mi>}</mi>
```

図 2.16 正規化された図 2.15 のデータ

2.5.7. 要素の除去

本研究における表示の情報ではない情報を表すための要素をすべて除外する．対象となる要素は，mspace 要素，mstyle 要素，menclose 要素，mpadded 要素，mphantom 要素，merror 要素，malignmark 要素，malignmark 要素である．

子に作用する要素は，mstyle 要素，menclose 要素，および mpadded 要素である．mstyle 要素は子のフォントサイズや色を指定する，menclose 要素は子を線で囲む，mpadded 要素は子の位置を微調整する，という機能を持つ．これらの要素は，その要素のみを削除し，その子は残すという形で処理する．図 2.17 に，menclose 要素の正規化の例を示す．左が正規化前，右が正規化後である．

$\boxed{\frac{3x}{2}}$ $\frac{3x}{2}$

```
<mfrac>
  <mrow>
    <menclose notation="box">
      <mn>3</mn>
      <mi>x</mi>
    </menclose>
  </mrow>
  <mrow>
    <mn>2</mn>
  </mrow>
</mfrac>
```

```
<mfrac>
  <mrow>
    <mi>3</mi>
    <mi>x</mi>
  </mrow>
  <mrow>
    <mi>2</mi>
  </mrow>
</mfrac>
```

図 2.17 要素のみを削除し子を残す正規化

残る mspace 要素，mphantom 要素，merror 要素，malignmark 要素，malignmark 要素は，子も含めて削除する．mspace 要素は空白を表示する機能を持つ．mphantom 要素は，子となる要素を透明で表示するための要素であり，例えば $\frac{a}{a+b+c}$ のような数式を記述するために用いられる．merror 要素はエラー出力のために用いられ，通常，表示に反映されない．malignmark 要素，malignmark 要素は，行列やテーブルにおいて記号を揃えて表示するために用いられるものである．

2.6. 正規化の例

1 つの MathML データに対して正規化を行った場合の例を示す. $\left(a+\frac{\sqrt{3}}{2}\right)x^n$という数式を表す正規化されていない MathML データと，正規化後の MathML データを図 2.18 に示す.

```
<math>
  <mfenced>
    <mrow>
      <mi>a</mi>
      <mo>+</mo>
      <mfrac>
        <mrow>
          <msqrt>
            <mn>3</mn>
          </msqrt>
        </mrow>
        <mn>2</mn>
      </mfrac>
    </mrow>
  </mfenced>
  <msup>
    <mi>x</mi>
    <mi>n</mi>
  </msup>
</math>
```

```
<math>
  <mi>(</mi>
  <mi>a</mi>
  <mi>+</mi>
  <mfrac>
    <mrow>
      <msqrt>
        <mrow>
          <mi>3</mi>
        </mrow>
      </msqrt>
    </mrow>
    <mrow>
      <mi>2</mi>
    </mrow>
  </mfrac>
  <mi>)</mi>
  <mi>x</mi>
  <msup>
    <mrow/>
    <mrow>
      <mi>n</mi>
    </mrow>
  </msup>
</math>
```

図 2.18 $\left(a+\frac{\sqrt{3}}{2}\right)x^n$の正規化前と正規化後

これら 2 つのデータを木として表現したものと，2.2. で述べた表示の情報の木構造として$\left(a+\frac{\sqrt{3}}{2}\right)x^n$を表現したものを，図 2.19 に示す．左に正規化前の MathML の木構造，中央に正規化後の MathML の木構造，右に表示の情報の木構造（図 2.1 と同じもの）を示す.

```
math
  ├ mfenced
  │   └ mrow
  │     ├ mi/a
  │     ├ mo/+
  │     └ mfrac
  │       ├ mrow
  │       │   └ msqrt
  │       │     └ mn/3
  │       └ mn/2
  └ msup
    ├ mi/x
    └ mi/n
```

```
math
  ├ mi/(
  ├ mi/a
  ├ mi/+
  ├ mfrac
  │   ├ mrow
  │   │   └ msqrt
  │   │     └ mrow
  │   │       └ mi/3
  │   └ mrow
  │     └ mi/2
  ├ mi/)
  ├ mi/x
```

```
統合
 ├ (
 ├ a
 ├ +
 ├ 分数
 │  ├ 統合
 │  │  └ 平方根
 │  │    └ 統合
 │  │      └ 3
 │  └ 統合
 │    └ 2
 ├ )
 ├ x
```

	└ msup 　├ mrow 　└ mrow 　　└ mi/n	└ 上付き 　└ 統合 　　└ n

図 2.19 正規化前の木，正規化後の木，表示の情報の木

中央の木と右の木を見比べると，これらはまったく同一の形状となっている（正規化後の MathML データの木には msup 要素が子を 2 つ持つが，第一の子は必ず空の mrow 要素であるため無視することができる）．すなわち，正規化によって，MathML データを本研究における表示の情報の木構造と同一のものとすることができている．

2.7. 正規化の評価

正規化によってデータが一意に定まることを確認するために，評価実験を行う．同一の数式の表示を表す数式データを 3 つの異なる方法で作成し，それらのデータにゆらぎがあることを確認した上で，正規化によってデータを一意に定められることを示す．

数式データの元となる数式は，2007 年度の大学入試センター試験[42]における数学 I・A および数学 II・B の問題文および解答の選択肢に出現する数式である．センター試験に出現する数式には，数式の一部が解答欄となっているものがある．その場合は正答によって解答欄を補完する．math 要素を除いて 1 種類のタグしか利用されていないものは除外する．これは具体的には，xy や100 など，mi 要素および mn 要素しか用いられない数式である．また，文書を作成する際に数式データとして記述されると想定し難いもの（①などの数式番号や[1],(1)などの問題番号）も除外する．得られた数式は 156 件であった．その 156 件は，log やsinなどの初等関数を用いた数式や，積集合や和集合などの集合論に関する数式，幾何学における三角形記号△や角記号∠が用いられた数式など，多様な数式を含む．また，記号の位置構造という点でも，$1 < \frac{1}{\log_3 y} + \frac{\log_3\left(1-\frac{x}{2}\right)}{\log_3 y}$といった複雑な数式を含む．

データは，Mathematica による MathML の出力，Microsoft Word 数式エディタ機能による記述，および LaTeXMathML による TeX 形式の数式の MathML へのコンバートという 3 通りの方法で作成した．なお，TeX 形式の数式は LyX[43]を用いて作成した．

表 2.5 に，異なる方法で作成された数式データ同士を比較し，互いに一致した数を示す．

表 2.5 正規化をしていない数式データ同士の比較結果

比較するデータ作成方法		一致した件数
Mathematica	Word	1
Mathematica	LaTeXMathML	0
Word	LaTeXMathML	0

正規化を行わないと，ほぼすべての数式においてデータにゆらぎが生じていることがわかる．ゆらぎの例として，$\tan\theta$という数式において，Mathematica によるデータではtanとθに間に「見えない掛ける」が挿入される．Microsoft Word の数式エディタによるデータでは「関

数の適用」が挿入される．LaTeXMathML によるコンバートで得られたデータでは何も挿入されない．他にも，mrow 要素の存在箇所や，mi 要素の子となる記号の数など，多くの点でゆらぎが見られた．なお，Mathematica のデータと Word のデータが一致した数式は$\sqrt{y}$である．

表 2.6 に，正規化をした上で数式データを比較し，一致した数を示す．

表 2.6 正規化をした数式データ同士の比較結果

比較するデータ作成方法		一致した件数
Mathematica	Word	137
Mathematica	LaTeXMathML	126
Word	LaTeXMathML	133

正規化を行わない場合と比べて，一致した件数が非常に多くなっている．しかし，156 件の数式すべてを一致させることはできていない．正規化を行ってもデータが一致しない最も大きな原因は，記号の異なりである．unicode には，異なる文字であるが見かけ上同一に見えるようなものがある．例えば，$2(x-2)^2=|3x-5|$という数式において，“|”という記号は，Mathematica では文字コード#x2758 の文字，Word では#x007c，LaTeXMathML では#x2223 となる．このような記号の異なりは，正規化を用いても一意化できない．

表 2.7 に，正規化をした上で，記号の異なりを無視して数式データを比較した場合に，一致した数を示す．なお，記号の異なりを無視するという処理は，mi 要素の子がすべて同一であるとみなすことで実現できる．

表 2.7 正規化をした数式データに対する記号の異なりを無視した比較結果

比較するデータ作成方法		一致した件数
Mathematica	Word	154
Mathematica	LaTeXMathML	146
Word	LaTeXMathML	148

Mathematica と Word の比較における 2 件の不一致は，角度を表す記号“°”に起因する．Mathematica では，“°”は単に mo 要素の子として記述されるが，Word では，msup 要素の第二の子，すなわち上付き文字として記述される．Word と LaTeXMathML の比較における 8 件の不一致は，LaTeXMathML が記号“≦”に対応していないことが原因である．“≦”は TeX（amsmath パッケージ）において`¥leqq`と表されるが，これを`¥leq`と q という 2 つの記号であると解釈し，$\leq q$に変換してしまったため，“≦”を含む数式はすべて不一致となった．Mathematica と LaTeXMathML の間の 10 件の不一致の内訳は，“°”に起因する 2 件と“≦”に起因する 8 件である．

この評価実験から，データの作成方法によって同一の数式を表すデータが一意に定まらないこと，および，正規化によってそれらのデータのほとんどを一意化できることがわかった．一方，unicode で別の文字として定義されているものの，見かけ上同一に見えるような記号に関しては，一意に定まらない．また，“°”のような記号そのものが上付き文字のよ

うに表現されている場合，これを単なる記号として表す場合と上付き文字として表す場合があり，これらを一意化することもできない．これらの問題にも対処するためには，unicode の文字の処理に取り組む必要がある．文字の処理については 5.2. で後述する．

3. パターンマッチング

本研究では，数式に対して文字列のパターンマッチング（文字列探索）に基づく検索を行う手法を提案する．本手法は，文書から特定の数式が記述された箇所を見つけ出す，文書内検索の機能に応用することができる．文書内検索を用いると，数百ページの書籍や web ページから，目的とする情報が記述されている位置を瞬時に見つけ出すことができる．

通常の言語におけるパターンマッチングでは，正規表現の機能が備えられていることも多い．正規表現を用いると，柔軟にパターンを記述することができるようになる．本研究では，数式に対する正規表現を用いた検索も実現する．これに類する研究として，数式検索にワイルドカードなどの機能を導入することが試みられているが[26, 44]，文字列の正規表現とは機能，記法ともに大きく異なるものとなっており，文字列の正規表現ほどパターンの記述能力を向上させるものではない．本研究では，数式の検索において，文字列の正規表現と同様の機能，記法の正規表現を実現する．本研究で提案する数式の正規表現は，後方参照の機能も備えている．これにより，文書内検索においてパターンを柔軟に記述できるのみならず，数式を含む文書を作成している際に数式の置換を行うといった際にも，本手法が高い利便性を発揮する．

3.1. 機能

提案手法における検索の機能を示すために，パターンの記法を説明する．また，提案手法を用いて実現することのできる，検索した数式のハイライト表示，数式の置換についても述べる．

3.1.1. パターンの記法

パターンは文字列として記述される．数式における記号の位置構造は，TeX の記法に類似した形式で記述できる．正規表現の記法は，標準的な正規表現の記法と同一である．このため，計算機上で数式を記述した経験を持ち，正規表現の記法を知っているユーザであれば，本手法を容易に利用することができる．また，パターンが文字列であるため，パターンを自動生成して本手法に対する入力とするといったパイプライン処理が可能であり，本手法をモジュールとしてシステムに組み入れることができる．一方，パターンを文字列として入力するという方法は，煩雑さや入力の誤りの原因となりうる．この問題を解決するために，パターンの文字列を作成するための GUI も用意している．GUI については，3.3. にて後述する．

3.1.1.1. 記号および構造

パターンにおいて記号を表現するために，unicode を使用する．記号をそのまま入力することも，文字実体参照を用いることもできる．文字実体参照とは，&と;の 2 つの記号でキーワードを囲むことで 1 つの記号を表現する記法である．例えば，∪という文字実体参照

は，∪という 1 つの記号である．

数式中の構造は，以下の形式で記述される．なお，角括弧記号（[,]）は「存在しない場合もある」ことを意味する．

¥keyword{contents1}[{contents2}[{contents3}]]

keyword の種別によって，keyword の後に続く contents の数は決まっている．また，contents は構造を含むことができる．つまり，構造の入れ子が許される．例えば，分数を記述するための keyword は frac であり，contents を 2 つとる．平方根の keyword は sqrt であり，contents は 1 つである．$\frac{1}{\sqrt{2}}$は，次のように表現される．

```
¥frac{1}{¥sqrt{2}}
```

keyword は 9 種類用意されている．9 種類のキーワードは，数式の木構造における 9 種類の構造節に対応する．表 2.1 を keyword および contents の観点から整理したものを，表 3.1 に示す．

表 3.1 keyword およびその contents

keyword	数式の例	contents		
		1	2	3
sub	$_{1}$	下付き文字		
sup	^{2}	上付き文字		
subsup	$_{-\infty}^{\infty}$	下付き文字	上付き文字	
under	$\lim_{n\to\infty}$	基底	下部の文字	
over	$\dot{x}$	基底	上部の文字	
underover	$\sum_{i=0}^{n}$	基底	下部の文字	上部の文字
frac	$\frac{1}{2}$	分子	分母	
sqrt	$\sqrt{x}$	根号の内部		
root	$\sqrt[3]{z}$	根号の内部	根指数	

マッチングにおいて，構造は 1 つの文字として扱われる．また，構造同士は，keyword が同一でかつすべての contents が同一である場合にマッチする．

マッチングは，構造の contents に対しても行われる．例えば，パターンが$\frac{1}{2}$，数式が$\sqrt{a+\frac{1}{2}}$の場合，sqrt の contents である$a+\frac{1}{2}$に対してもマッチングを行うため，$\sqrt{a+\frac{1}{2}}$のうちの$\frac{1}{2}$の部分がマッチする．

3.1.1.2. 正規表現

本システムで用いることができる正規表現の機能は，以下の通りである．

- ワイルドカード（任意の 1 文字または任意の 1 構造）
- 文字クラス，否定文字クラス
- 量化
- 選言
- 後方参照

それぞれの機能を解説する．

任意の 1 文字および任意の 1 構造は，メタ文字‘.’で表現される．例えば，`¥frac{.}{2}` は，$\frac{1}{2}$や$\frac{\sqrt{x}}{2}$にマッチする．

文字クラスおよび否定文字クラスは，メタ文字 ‘[’, ‘]’, ‘^’を用いて表現される．文字クラスは，括弧‘[’, ‘]’で囲んだ文字のうちのいずれかにマッチし，否定文字クラスは，括弧‘[^’, ‘]’で囲んだ文字以外の文字のうちのいずれかにマッチする．ハイフン記号を用いた略記も可能であり，例えば`[0-9]`は任意の数字，`[^stx-z]`は s, t, x, y, z 以外の文字を表す．文字クラスおよび否定文字クラスの括弧の中に構造および文字実体参照を含めることはできない．

量化は，直前に出現する表現の反復を指定する．量化のために，3 つのメタ文字が用意されている．

*	直前の表現が 0 個以上
+	直前の表現が 1 個以上
?	直前の表現が 0 個もしくは 1 個

例えば，`a×?b` は$a \times b$およびabにマッチする．

選言は，メタ文字‘|’を用いて記述され，複数の表現のうちのいずれかにマッチする．例えば，`x|y` は，xおよびyにマッチする．

量化および選言を用いる際，構造は単一の文字とみなされる．また，構造の contents において量化や選言を用いることもできる．例えば`¥sqrt{.+}+`というパターンは正しいパターンであり，$\sqrt{a}\sqrt{b}$，$\sqrt{5a^2-2a+2}$などにマッチする．

量化および選言のスコープは，丸括弧記号‘(’, ‘)’を用いて指定できる．例えば`(x¥sub{.+})+`は，$x_1x_2x_3x_4$やx_ix_jにマッチする．`2(¥sqrt{.+}|.+¥sup{¥frac{1}{2}})`は，$2\sqrt{x}$や$2z^{\frac{1}{2}}$にマッチする．

後方参照は，パターンの一部（サブパターン）を指定し，サブパターンにマッチした数式と同一の数式を後から呼び出す機能である．パターンの指定には，スコープの指定と同じく，丸括弧記号‘(’, ‘)’で囲むという方法を用いる．サブパターンは，開き丸括弧記号‘(’が最も左側のものを 1 として，順に番号付けされる．サブパターンにマッチした数式は，¥n で呼び出される．n には，サブパターンの番号を記述する．例えば，`¥frac{.+}{(.+)}-¥frac{.+}{¥1}`というパターンを考える．1 つ目の分数の分母にあたる部分は`.+`であり，これは任意の数式にマッチする．また，括弧で囲まれており，1 番目のサブパターンとして指定

されている．2 つ目の分数の分母を見ると，後方参照を用いて，1 番目のサブパターンにマッチした数式を参照している．つまり，このパターンは，分母が同一であるような分数の差を表現しており，$\frac{1}{2}-\frac{a+1}{2}$などの数式がマッチする．

ここまでに述べたメタ文字のうち，`.`, `*`, `+`, `?`, `|`, `(`, `)`, `[`, `]`, `{`, `}`, `^`, `&`, `;`, `¥`は，バックスラッシュ（本論文では円記号（¥）で示す）を用いてエスケープすることで，通常の記号とみなされる．なお，文字クラスおよび否定文字クラスの内部においてエスケープする必要のあるメタ文字は`[`, `]`, `^`, `-`, `¥`のみである．

3.1.2. ハイライト表示

ハイライト表示は，パターンにマッチした箇所を強調するための機能である．パターンと数式とのマッチング結果をハイライト表示した例を，表 3.2 ハイライト表示に示す．検索の意図と，それを実現するパターン，およびパターンにマッチした部分がハイライト表示された数式を，順に 4 つ示す．

表 3.2 ハイライト表示

aとbの乗算
`a[×･]?b`
$a \times b$ ， $a \cdot b$ ，ab
微分
`(¥frac{d.*}{d.+}|¥sup{'}|¥over{.+}{¯})`
$\frac{\mathrm{d}}{\mathrm{d}t}x(t)=v(t)$ ， $\left(af(x)\right)'=af'(x)$， $\ddot{x}(t)+\dot{x}(t)+t$
相加平均，相乗平均の関係
`¥frac{(.+)¥+(.+)}{2}≧¥sqrt{¥1[×･]?¥2}`
$\frac{a+b}{2}\geq\sqrt{ab}$ ， $\frac{x_1+x_2}{2}\geq\sqrt{x_1x_2}$
e^xのマクローリン級数
`e¥sup{x}=1¥+x¥+(¥frac{x¥sup{([0-9]+)}}{¥2!}¥+)+…`
$e^x=1+x+\frac{x^2}{2!}+\cdots$ ， $e^x=1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}+\frac{x^5}{5!}+\cdots$ （このパターンは $e^x=1+x+\frac{x^2}{2!}+\frac{x^2}{2!}+\frac{x^4}{4!}+\frac{x^3}{3!}+\cdots$ といった数式にもマッチする）

ハイライト表示は，web ページの閲覧の際に効果を発揮する．ハイライト表示を用いることで，目的とする記述が web ページ中に記載されている箇所を一目で把握することができる．主要な web ブラウザはすべて自然言語のためのハイライト表示機能を有していることからも，ハイライト表示が web ページ閲覧において必要とされていることがわかる．

3.1.3. 置換

置換は，パターンにマッチした部分を，別の数式に置き換える機能である．

置換後の数式の記法は，パターンの記法と同様である．しかし，置換後の数式は一意に定まる必要があるため，ワイルドカードや量化といった機能は使用できない．このため，置換後の数式では，.，*，+，?，|，(，)，[，]，^はエスケープしなくて良い．

置換後の数式における後方参照は，パターン中のサブパターンを参照する．例えば，パターンを¥sqrt{(.+)}，置換後の数式を¥1¥sup{¥frac{1}{2}}として，数式$\sqrt{x}$を置換する場合を考える．パターン中の，括弧で囲まれた（すなわち，サブパターンとして指定された）.+がxにマッチするため，置換後の数式の¥1 もまたxとなり，置換の結果は$x^{\frac{1}{2}}$となる．

置換は，文書編集において有用である．例えば，文書中に記述した数式中の記号xを，すべてθに書き換える必要が生じた際，置換を用いればすぐに書き換えを終えることができる．本手法では，置換後の数式中で後方参照を使用することが可能であるため，前述のような，平方根による表記を$\frac{1}{2}$ 乗による表記に置換するといった高度な処理にも対応できる．

3.2. 実装

本章ではまず，数式に対する正規表現を実現する上での問題とその対処方法を述べる．その後，数式データを文字列に内部的に変換する方法を示す．また，前章で示した記法で記述されたパターンを，実際にマッチングを行う正規表現エンジンである Onigmo[45]に入力するパターンへと内部的に変換する方法についても述べる．

3.2.1. 数式の正規表現における実装上の問題と対処方法

本研究では，数式の表示の情報を木構造として表現する．木構造は，括弧を用いることで，文字列として表現することができる．実際，本研究で提案する¥keyword{contents1}[{contents2}[{contents3}]]という記法から，一意に木を得ることができる．木が文字列に変換されると，その文字列に対して，正規表現を用いたパターンとのマッチングを行うことができるようになる．しかし，文字列に変換された木に対しては，正規表現を用いたパターンマッチングが正しく動作しない．これは，正規表現に括弧の対応関係を扱う能力がないためである（なお，ここで言う括弧とは，構造の contents を囲む括弧のことであり，$(a+b)^n$といった，数式における括弧のことではない．数式における括弧は，単なる 1 つの記号として扱われる）．この事実は形式言語理論において確かめられている．例を挙げると，「任意の文字を含む（1 つの）平方根」を意図して作成された¥sqrt{.+}というパターンは，2 つの平方根が並んだ¥sqrt{x}¥sqrt{y}にマッチしてしまう．これは，パターン中の「任意の文字の一度以上の反復」を意味する.+が，文字列化された数式中の x}¥sqrt{y にマッチしてしまうためである．これを避けるために，パターン中の.+を「{，}を除く任意の文字の一度以上の反復」を意味する[^{}]+に書き換えると，パターンが¥sqrt{x¥sup{2}}にマッチしなくなる．

正規表現を拡張して括弧の対応関係を取り扱うための方法の 1 つとして，木オートマトンを用いる方法がある[46]．しかし，木オートマトンを用いたマッチングには，後方参照の

機能を導入することが容易ではない.

別の方法として，パターンを再帰的に利用できるよう拡張が施された正規表現を利用するというものが挙げられる．このような正規表現を処理できる正規表現エンジンとして，Onigmo が知られている．Onigmo では後方参照を利用することもできる．また，文字列のマッチングで検索を実現できるため，XSLT[47]や XQuery[48]など，XML データを対象とした技術を用いるよりも高速に処理を行うことができる．このため，本研究では，数式データを文字列に変換し，かつシステムに入力された数式検索用のパターンを Onigmo で処理可能なパターンに変換した上で，パターンと文字列を Onigmo に入力し，マッチング処理は Onigmo が行うという方法をとる．処理の流れを概念的に表したものを，図 3.1 に示す.

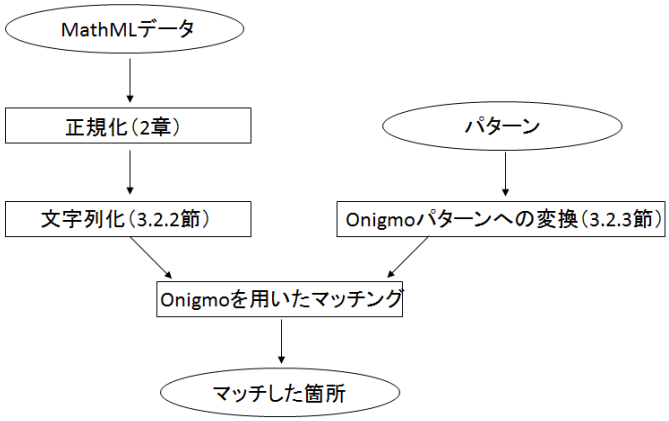


図 3.1 パターンマッチングの実装の概念図

3.2.2. 数式木から文字列への変換

2. で述べた方法で MathML データを正規化すると，構造節，統合節，記号節の 3 種類の節からなる木構造が得られる．この木構造を，葉から根に向けて再帰的に処理することで，文字列への変換を行う．構造節，統合節，記号節は，それぞれ，次のように変換される．

- 記号節は，1 つの文字に変換される．
- 統合節は，その子の文字列を順に並べた文字列に変換される．
- 構造節は，「/構造節名」という文字列の後ろに，その子の文字列を/{，および/}で囲んで順に並べた文字列に変換される．

例として，$\left(a+\frac{\sqrt{3}}{2}\right)x^n$（図 2.19 の元となる数式）を文字列に変換する過程を図 3.2 に示す．$\left(a+\frac{\sqrt{3}}{2}\right)x^n$の木構造における各節の横に，その節から変換して得られる文字列を示す.

math	**(a+/frac/{/sqrt/{3/}/}/{2/})x/sup/{n/}**
├ mi/(	**(**
├ mi/a	**a**

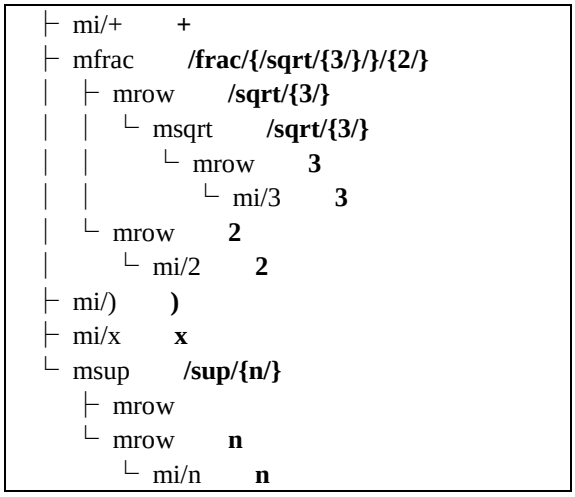


図 3.2 $\left(a+\frac{\sqrt{3}}{2}\right)x^n$の文字列への変換過程

結果として得られる文字列は，(a+/frac/{/sqrt/{3/}/}/{2/})x/sup/{n/}となる．

ここまでで得られた文字列をそのまま正規表現エンジンに入力すると，パターンが構造名や括弧記号にマッチしてしまう可能性がある（例えばパターンが「a」の時，構造名 frac の a にマッチする）．このため，構造名を別名に置き換える．構造名と別名との対応を表 3.3 に示す．

表 3.3 構造名と別名の対応

構造名	別名
underover	:::::::::
subsup	::::::::
under	:::::::
over	::::::
root	:::::
frac	::::
sqrt	:::
sub	::
sup	:

$\left(a+\frac{\sqrt{3}}{2}\right)x^n$から得られる文字列に対して別名への置き換えを行ったものを以下に示す．

(a+/::::/{/:::/{3/}/}/{2/})x/:/{n/}

なお，検索対象となる数式には，表 3.3 に示した 9 種類の構造の他に，行列およびテーブルに関する要素，動的な機能に関する要素が含まれる．それら要素名と別名の対応を表 3.4 に示す．

表 3.4 検索対象にのみ含まれる要素名と別名の対応

要素名	別名

maction	::::::::::::::
mtable	::::::::::::::
mtr	::::::::::::
mlabeledtr	::::::::::::
mtd	::::::::::

記号としての{, }, :, ¥は，それぞれ，¥{, ¥}, ¥:, ¥¥に置換する．ここで，統合節の内容を囲む括弧の直前にスラッシュ（/）が挿入されている理由を述べる．スラッシュを挿入しないと，例えば数式が√¥である場合，これを変換して得られる文字列/:::{¥¥}が¥}という文字列を含んでしまうため，パターンが「}」の場合，構造節を表現するための括弧にマッチしてしまう．スラッシュを挿入しておくことで，これを避けられる．

3.2.3. ユーザが作成したパターンから Onigmo のパターンへの変換

ユーザが作成したパターン（ユーザパターン）を，Onigmo に入力するパターン（Onigmo パターン）に変換する方法を述べる．パターンの変換は，前節で述べた数式から文字列への変換に対応する変換，括弧の対応関係を扱うための変換，後方参照のための変換の 3 つに大別できる．それぞれについて述べる．

3.2.3.1. 数式の文字列化に対応するための変換

ユーザパターン中の構造に対しては，前節で述べた，構造節の変換と同様の変換を行う．すなわち，¥keyword{contents1}[{contents2}[{contents3}]]は，以下のように変換される．なお，alias は表 3.3 に示した別名である．

/alias/{contents1/}[/{contents2/}[/{contents3/}]]

さらに，置換後の構造はすべて丸括弧で囲む．これは，量化のスコープを適切に設定するためである．ユーザパターンにおいては，利便性のため，例えば¥sqrt{2}+という記法を許しているが，このパターンにおける‘+’は，Onigmo パターンにおいては「}が 1 回以上連続する」ことを表現している．構造全体を括弧で囲むことで，‘+’が「構造全体が 1 回以上連続する」ことを表すようになる．

構造を表現するための文字である，:, {, }, /の処理について述べる．ユーザパターンが文字:を含む場合，これを¥¥:に置換する．また，¥{, ¥}は，¥¥{, ¥¥}に変換する．ユーザパターンが文字/を含む場合，これを/(?![{}:])に置換する．このパターンは，否定的先読みと呼ばれる機能を用いており，「直後の文字が，{, }, :のいずれでもないような/」を表現している．この変換によって，文字として記述された/は，構造を表現するための文字の接頭辞としての/にはマッチしなくなる．

3.2.3.2. 括弧の対応関係を扱うための変換

まず，括弧の対応関係を扱うための変換に用いる「名前付き捕獲式集合」および「名前指定呼び出し」について説明する．名前付き捕獲式集合とは，サブパターンに名前を割り当てる機能であり，名前指定呼び出しとは，名前を割り当てたサブパターンを呼び出す機能であ

る．サブパターンを再帰的に呼び出すことも可能であり，これによって括弧の対応関係を扱うことができる．

そもそも，¥(.+¥)が(a)(b)のような文字列にマッチしてしまうのは，任意の 1 文字を表す‘.’が括弧記号にマッチしてしまうためであった．上述の機能を用いて‘.’を書き換えることで，括弧の対応関係を考慮した形で，任意の 1 文字または 1 構造にマッチさせることができるようになる．そこで，‘.’を，次のパターンに変換する．可読性のためにインデントをつけて示す．

```
(?<a>
    (?<ac>[^{}:/¥¥]|¥¥{|¥¥}|¥¥:|¥¥¥¥|/(?![{}:]))
    |
    (?<as>/:+(/{(¥g<ac>|¥g<as>)*/})*)
)
```

(?<name>p)という記法は，パターン p に名前 name を割り当てることを意味する．

このパターンについて説明する．ac（arbitrary character）と名付けられたパターンは，

- 構造に関わる 5 つの文字（{, }, :, /, ¥）以外の文字
- ¥{
- ¥}
- ¥:
- ¥¥
- 直後が{, }, :でない/

のいずれかにマッチする．構造に関わる 5 つの文字が，単なる文字として数式中に出現する場合，数式の木を文字列に変換する処理において，直前にバックスラッシュが付与された形に変換されている．このため，このパターンは，数式における任意の 1 文字にマッチする．

as（arbitrary structure）と名付けられたパターンを見ると，まず:+は，keyword の別名にマッチする．/{(¥g<ac>|¥g<as>)*/}は，contents にマッチする．ここで，¥g<name>という記法は，name と名付けられたパターンを呼び出すためのものである．つまり，「(contents を意味する）括弧の内部は，0 つ以上の任意の文字（ac）または任意の構造（as)」であることを意味する．このパターンにおいて，パターン as の内部で as を再帰的に呼び出している．これにより，構造が入れ子になっている場合でも，括弧の対応関係を適切に処理することができる．さらに，contents に相当する部分を括弧で囲み‘*’を付しているため，contents が複数存在する場合にも as は構造に適切にマッチする．

上記のパターンを一度記述すれば，名前指定呼び出しでパターンを呼び出せば良いため，同一のパターン中に‘.’が複数出現した場合，2 番目以降の‘.’は¥g<a>に変換される．

3.2.3.3. 後方参照のための変換

まず Onigmo パターンにおける後方参照の記法を説明する．サブパターンの指定には，前述の(?<name>p)の記法を用いる．サブパターンにマッチした部分の呼び出しには，「名前

指定参照」と呼ばれる機能を用いる．名前指定参照の記法は¥k<name>である．ここで，前述の¥g<name>はサブパターンそのものを呼び出すが，¥k<name>はサブパターンにマッチした部分を呼び出す．例えば，(?<s>.)¥g<s>，および(?<s>.)¥k<s>という 2 つのパターンと文字列 ab がある時，前者は ab にマッチするが，後者はマッチしない．なぜなら，前者のパターンにおける¥g<s>は‘.’（つまり任意の文字）となるが，後者のパターンにおける¥k<s>は‘.’にマッチした文字すなわち a となるためである．

ユーザパターンから Onigmo パターンへの変換においては，ユーザパターン中の丸括弧記号のすべてに，名前を割り当てる．つまり，(*pattern*) というパターンを，(?<br*n*>*pattern*) に変換する．なお，*n* は数字であり，最も左側に開き括弧がある括弧を 1 として，右に向かって順に数字を割り当てる．ユーザパターンにおける，サブパターンにマッチした部分の呼び出し¥*n* は，¥k<br*n*>に置き換える．

後方参照で参照されるサブパターンすべてに特別な名前を割り当て，サブパターンにマッチした部分を名前指定参照で呼び出すという方法を用いることで，名前を割り当てられない括弧は純粋にスコープを指定するためのものとなり，サブパターンの指定という役割を持たなくなる．このため，ユーザパターンから Onigmo パターンへの変換処理で追加されたスコープを指定するための括弧を，後方参照の実現において考慮する必要はない．

3.2.4. マッチング結果を用いたハイライト表示と置換

ここまでに述べた方法で，数式の木を表現する文字列と Onigmo パターンが得られた．数式の文字列に対して Onigmo パターンによるマッチングを行うことで，ハイライト表示および置換を実現する方法を述べる．

3.2.4.1. ハイライト表示

ハイライト表示には，MathML の mstyle 要素を用いる．mstyle 要素は，数式中の記号の背景色やフォントカラーなどを指定するための要素である． mstyle 要素の子孫にあたる部分に，mstyle 要素で指定した装飾が施される．今回は，マッチした箇所の背景色を黄色にするため，mathbackground 属性の値を yellow に指定した mstyle 要素を用いる．

ハイライト表示を行うためには，ハイライト表示したい部分の親として mstyle 要素を挿入すれば良い．これはすなわち，数式木において構造節を新たに導入し，ハイライトされるべき要素をその構造節の子とするということである．以後，ハイライトのために導入した構造節を highlight と呼ぶ．

パターンにマッチした箇所を highlight の子とするために，正規表現エンジンの置換機能を用いる．まず，Onigmo パターン全体をサブパターンとして指定する．サブパターンの名前は br0 とする．すなわち，Onigmo パターンを p とすると，新たなパターンは(?<br0>p)となる．置換後の文字列は/:::::::::::/{¥k<br0>/}とする．置換後の文字列に名前指定参照を用いているため，¥k<br0>はもとの Onigmo パターンにマッチした部分となる．このようにして得られたパターンと置換後の文字列を用いて置換処理を行うと，Onigmo パタ

ーンにマッチしたすべての部分が/:::::::::::/{,/}で囲まれる．なお，:::::::::::はhighlight の別名とする．これによって，パターンにマッチした部分を構造節 highlight の子とすることができる．

この置換の結果として得られた文字列に対して，3.2.3.1. で述べた木から文字列への変換の，逆方向の変換を行う．数式と文字列とは一対一に対応するため，この逆変換は可能である．逆変換を行う際，highlight は mathbackground 属性の値を yellow に指定した mstyle 要素に変換する．この処理によって，マッチした部分がハイライトされた数式（mstyle 要素が挿入された MathML データ）を得ることができる．

例として，$\left(a+\frac{\sqrt{3}}{2}\right)x^n$という数式において，$\sqrt{3}$をハイライトする場合を考える．この時，文字列化された数式は(a+/::::/{/:::/{3/}/}/{2/})x/:/{n/}，Onigmo パターンは/:::/{3/}である．この Onigmo パターンを，br0 という名前のサブパターンとする．すなわち，(?<br0>/:::/{3/}) とする．そうした上で，置換後の文字列を/:::::::::::/{¥k<br0>/}とし，Onigmo を用いて置換処理を行う．結果，(a+/::::/{/:::/{3/}/}/{2/})x/:/{n/} のうちの下線部が置換され，(a+/::::/{/:::::::::::/{/:::/{3/}/}/}/{2/})x/:/{n/}となる．これを木構造に変換すると，次の図 3.3 のようになる．

```
math      (a+/::::/{/:::::::::::/{/:::/{3/}/}/}/{2/})x/:/{n/}
  ├ mi/(      (
  ├ mi/a      a
  ├ mi/+      +
  ├ mfrac     /::::/{/:::::::::::/{/:::/{3/}/}/}/{2/}
  │  ├ mrow     /::::::::::/{/:::/{3/}/}
  │  │  └ mstyle     /::::::::::/{/:::/{3/}/}
  │  │     └ mrow     /:::/{3/}
  │  │        └ msqrt     /:::/{3/}
  │  │           └ mrow     3
  │  │              └ mi/3     3
  │  └ mrow     2
  │     └ mi/2     2
  ├ mi/)      )
  ├ mi/x      x
  └ msup      /:/{n/}
     ├ mrow
     └ mrow     n
        └ mi/n     n
```

図 3.3 ハイライト表示のための文字列から得られる木

ハイライト表示を行う際，MathML データの正規化は，文書の読み込みが完了した時点で一度だけ行う．検索をするたびに正規化をするのではなく，読み込み時に正規化を行っておくことで，正規化を余分に行うことを避けられる．

3.2.4.2. 置換

置換を行うために，まず，ユーザが入力した置換後の数式を変換する．構造節およびメタ文字は，3.2.2. で述べた方法と同様の方法で変換する．なお，構造節全体を括弧記号 (,) で囲む処理，および/を/(?![{}:])に変換する処理は行わない．後方参照は，3.2.3.3. と同様の方法で変換する．

このようにして得られた置換後の文字列と Onigmo パターンを用いて，置換処理を行う．置換後の文字列に対して，ハイライト表示の場合と同様，3.2.2. で述べた変換の逆変換を行う．逆変換によって，置換された数式を得ることができる．

置換において，MathML データの正規化はまず，文書の読み込みが完了した時点で行われる．その後，文書中の MathML データが編集されるたびに，その MathML データを正規化する．このようにすることで，動的に変化する文書（すなわち，編集中の文書）に対して，正しく置換処理を行うことができる．

3.3. GUI

ここまで，パターンを文字列として表現してきた．しかし，文字列でパターンを記述すると，パターンの内容を直観的に把握することが難しい．これは，数式が，分数やべき乗などの二次元的な構造を持つためである．また，正規表現でメタ文字として用いられる，+や，(，)といった文字は，数式では通常の記号として頻出するが，これらのメタ文字を通常の文字として記述するには，バックスラッシュを用いてエスケープしなければならないため，エスケープ記号を何度も入力する必要が生じる．これらは，パターンの誤りの原因になり得る．また，初心者に利用の困難さを感じさせる可能性もある．これを解決するために，パターンを入力するための GUI を提案する．

数式を入力するための GUI は，すでに研究が進んでいる[8]．我々の GUI では，数式だけでなく，数式における正規表現の直観的な入力を実現することを目指す．正規表現のための GUI を作成する上で，メタ文字を使用せずに正規表現の機能を表示すること，いろいろなパターンの作成手順に対応できるよう入力の機構を用意することの 2 つの点を重視する．

パターンは，自然言語の文字入力のように，キャレットの位置に文字を挿入することを繰り返すという方法で作成される．さらに，数式を入力するために，キーボードで入力できない記号，および構造を入力する機能を用意する．また，正規表現を入力する機能も提供する．

3.3.1. 数式

英数字や+, -, (,)などは，キーボードを利用して入力することができる．キーボードで直接入力することのできない記号，例えばギリシャ文字（α，β）や，数式に特有の記号（∂，$\int$）などは，格子状に配置されたボタンをクリックすることで入力する．ボタンを図 3.4 に示す．

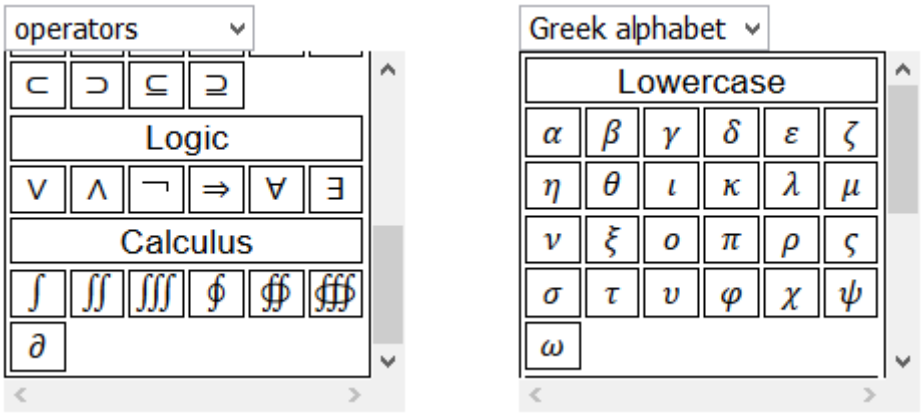


図 3.4 記号入力のためのボタン

構造は，ボタンをクリックすることで入力する．ボタンをクリックすると，空の構造が表示される．構造の引数にあたる箇所，つまり文字を入力することができる箇所は，二重線の矩形で示されている．二重線の矩形にキャレットを移動させると，その箇所に文字を入力することができるようになる．空の構造を図 3.5 に示す．

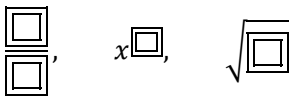


図 3.5 構造の入力

キャレットの移動は，クリックまたはカーソルキーの押下によって行われる．構造の直前あるいは構造の引数にあるカーソルキーを押下した場合，キャレットは，contents1，contents2，contents3，構造の外，の順に移動する．

構造を削除する場合は，ドラッグによる矩形選択を用いて構造全体を選択した上で，del キーを押下する．二重線の矩形にキャレットがある状態で del キーを押下した場合にも，構造全体が範囲選択される．

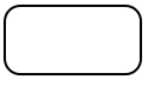
3.3.2. 正規表現

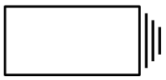
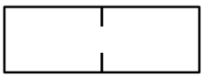
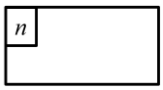
GUI における正規表現の表現方法について述べた後，正規表現の入力方法および編集方法を説明する．

3.3.2.1. 正規表現の図形

正規表現は，メタ文字ではなく図形として表現される．図形は，記号と混同されないような形にしている．正規表現の機能と図形との対応を，表 3.5 に示す．

表 3.5 正規表現の図形

機能	図形
ワイルドカード	
文字クラス	

否定文字クラス	
+	
*	
?	
選言	
サブパターンの指定	
後方参照呼び出し	

図形の特徴として，スコープを括弧でなく矩形で表現するため，二次元的に記号が配置される数式においてもスコープを直観的に把握することができる．また，通常の正規表現において，括弧はスコープの指定とサブパターンの指定という 2 重の意味を持つが，GUI では，スコープは矩形，サブパターンの指定は特別な図形で表現されるため，これらを明確に区別することができる．なお，サブパターンの番号（表 3.5 中の n）は，サブパターンを指定した順に付与される．

正規表現の機能のうち，量化，選言，サブパターンの指定は，スコープを持つ．同一のスコープに対して複数の機能が適用された場合，図形が複合する．複合は，次の規則に従う．破線の矩形は「出現しない可能性」，矩形に後置される三本線は「反復」，矩形中の中欠け線は「選言」，矩形の先頭の枠に囲まれた数字は「サブパターンの指定」を意味する．例えば，文字列によるパターン `(x|y|z)*`は，次の図 3.6 のようになる．

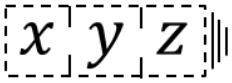


図 3.6 パターンの複合

3.3.2.2. 正規表現の入力

ワイルドカードは，キーボードで入力できない文字と同様に，ボタンをクリックすることで入力できる．文字クラス，否定文字クラス，後方参照の呼び出しは，構造と同様の方法で入力する．文字クラスおよび否定文字クラスは，ボタンをクリックすることで角丸の矩形を表示した後，角丸の矩形内に文字クラスの内容を入力する．後方参照の呼び出しは，ボタンをクリックして六角形を表示した後，参照するサブパターンの番号を入力する．

量化，選言，サブパターンの指定（すなわち，スコープを持つ正規表現）は，2 種類の方法で入力することができる．第一の方法は，構造の入力と同様のものである．ボタンをクリ

ックすることで空の矩形を表示し，矩形の内部にパターンを入力する．第二の方法は，矩形選択を用いてすでに記述された数式の一部を選択し，その部分をスコープとする正規表現の図形を，ボタンを用いて入力する方法である．

2 つの方法を用意することで，スコープを持つ正規表現を入力する際の 2 種類の手順に対応することができる．第一に，正規表現の機能を使うと決めてからパターンを作成し始める場合，すなわち，「特定の分母を共通に持つ分数同士の加算」として `¥frac{.+}{()}¥+¥frac{.+}{¥1}`と入力してから，空の括弧の中に具体的な分母の数式を入力するような場合は，空の矩形を表示するという方法を用いれば良い．第二に，数式部分を作成してから正規表現の機能を用いたいと考える場合，例えば，`sinx` というパターンを用いて検索をした後，検索結果を見て三角関数すべてを検索したくなったため，パターンを `(sin|cos|tan)x` に書き換える，というような場合には，数式を矩形選択して正規表現の機能を付与する方法を用いれば良い．

3.3.2.3. 正規表現の編集

編集も，ワイルドカードは文字と同様の方法で，文字クラス，否定文字クラス，後方参照の呼び出しは構造と同様の方法で行うことができる．

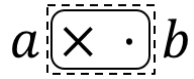
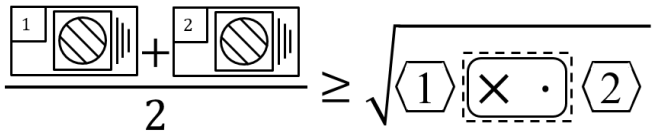
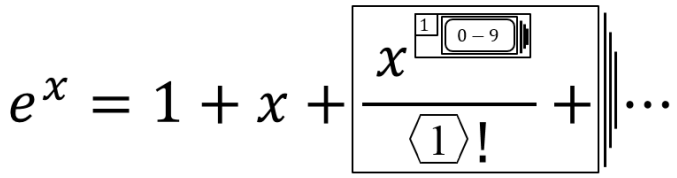
スコープを持つ正規表現を編集する場合，スコープをフォーカスする必要がある．スコープを表す矩形の直前（直後）にキャレットがある状態で→キー（←キー）を押す，またはこれらの矩形をクリックすると，スコープがフォーカスされる．スコープがフォーカスされた状態になると，キャレットが消え，矩形が点滅する．その状態でカーソルキーを押すか，数式中の文字や構造をクリックすると，スコープのフォーカスは解除され，適当な位置にキャレットが出現する．

スコープを持つ正規表現に対して可能な編集は，「機能の削除」「機能の追加（変更）」に分けられる．機能の削除は，スコープがフォーカスされた状態で del キーを押せば良い．選言の機能が用いられている場合，2 番目以降の候補は，機能を削除した際に矩形と一緒に消える．機能の追加（変更）は，ボタンを用いて行う．スコープがフォーカスされている状態では，量化および後方参照を入力するためのボタンがトグルとして動作する．つまり，フォーカスされているスコープにすでに付与されている機能のボタンは押下された状態となっており，付与されていない機能は押下されていない状態となっている．トグルを切り替えることで，ボタンに対応する機能がスコープに適用されているか否かを切り替えることができる．選言については，ボタンを押すことで，中欠けの縦棒が新たに挿入され，候補を追加することができる．

3.3.3. パターンの例

GUI によるパターンの視認性の向上を確認するために例を示す．表 3.6 に，検索の意図，文字列で記述されたパターン，GUI で作成されたパターン，およびパターンにマッチする数式の例を 3 つ示す．

表 3.6 GUI で作成されたパターンの例

aとbの乗算
a[×·]?b

$a \times b,\ a \cdot b, ab$
相加平均，相乗平均の関係
¥frac{(.+)¥+(.+)}{2}≥¥sqrt{¥1[×·]?¥2}

$\frac{a+b}{2} \geq \sqrt{ab},\quad \frac{x_1+x_2}{2} \geq \sqrt{x_1 x_2}$
e^xのマクローリン級数
e¥sup{x}=1¥+x¥+(¥frac{x¥sup{([0-9]+)}}{¥2!}¥+)+…

$e^x = 1 + x + \frac{x^2}{2!} + \cdots,\ e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \cdots$

3.3.4. GUI の評価

GUI によってパターンを効率的に入力できるようになることを確認するために，評価実験を行った.

情報学を専攻する 12 人の大学生および大学院生が，文字列によるパターンの作成と GUI によるパターンの作成を行い，パターンの完成までに要した時間を比較した．実験に先立って被験者はパターンの作成を練習した．被験者には，構造における keyword と contents およびその役割の一覧，正規表現の機能・メタ文字・GUI における図形の対応表，エスケープする文字の一覧を印刷したものを配布しており，実験中に自由に参照できることとした.

被験者は，3 つのパターンを，文字列による方法と GUI による方法の両方を用いて作成した．被験者は 2 つのグループに分けられ，A グループはまず GUI によるパターン作成を行い，次に文字列によってパターンを作成した．B グループは，文字列による作成を行った後，GUI を用いてパターンを作成した.

被験者がパターンを正しく作成できたことを確認する上で，作成したパターンを用いて実際に検索を行うという方法を用いた．具体的には，作成するパターンごとに「マッチする

数式」と「マッチしてはいけない数式」を用意しておき，作成したパターンで検索した結果として「マッチする数式」のすべての記号がハイライトされ，「マッチしてはいけない数式」のどの記号もハイライトされなければ，そのパターンの作成を終了して次のパターンの作成に進むようにした．パターンの作成および検索の処理は，web アプリケーションとして実装した．検索を実行した際に，その時点でのパターンと時刻を取得した．正しいパターンを作成して検索を実行した時刻と，web アプリケーションを起動した時刻との差を，パターンの作成に要した時間とした．

3.3.4.1. 複雑な構造を含むパターンの作成

構造を効率的に入力できることを確認するために，入れ子になった構造が含まれており，正規表現を利用しないパターンの作成について実験を行った．被験者が作成したパターンは次の数式である．

$$\frac{\mathrm{d}}{\mathrm{d}z}\sin^{-1} z = \pm\frac{1}{\sqrt{1-z^2}}$$

なお，文字列を用いてパターンを作成する際，“±”という記号は IME を用いて，「ぷらすまいなす」を変換することで入力できることをあらかじめ伝えた．

パターンを正しく作成できたことを確認するために検索対象となる数式として，「マッチする数式」は$\frac{\mathrm{d}}{\mathrm{d}z}\sin^{-1} z = \pm\frac{1}{\sqrt{1-z^2}}$，「マッチしてはいけない数式」は$a = b$とした．$a = b$を「マッチしてはいけない数式」としたのは，被験者が.*のような何にでもマッチするパターンを入力しそれを正しいパターンと判断してしまうことを避けるためである．

結果を表 3.7 に示す．「GUI」および「文字列」の列に，作成に要した時間を示す．

表 3.7 複雑な構造を含むパターンの作成に要した時間

グループ	被験者	GUI	文字列
A (先に GUI)	A_1	1 分 44 秒	6 分 38 秒
	A_2	1 分 20 秒	8 分 04 秒
	A_3	1 分 29 秒	1 分 53 秒
	A_4	2 分 39 秒	7 分 53 秒
	A_5	49 秒	2 分 03 秒
	A_6	2 分 12 秒	2 分 20 秒
B (先に 文字列)	B_1	1 分 03 秒	15 分 41 秒
	B_2	40 秒	3 分 58 秒
	B_3	30 秒	2 分 43 秒
	B_4	1 分 58 秒	4 分 23 秒
	B_5	45 秒	1 分 55 秒
	B_6	1 分 28 秒	3 分 46 秒

すべての被験者において，GUI の方が文字列よりも短時間でパターンを作成することが

できている．

文字列でパターンを作成した被験者が正しいパターンに至るまでにどのようなパターンで検索を行ったのかを観察すると，構造の keyword のスペルミス（被験者 A_1），構造の contents を囲む括弧の不適切な対応（被験者 B_1, B_4），構造の直前のバックスラッシュの入力忘れ（被験者 A_2, A_4, B_2, B_3）といった誤りが見られた．GUI を用いることで，構造が二次元的に表現されるため，こういった誤りは発生しなくなる．これは，GUI の方が短時間でパターンを作成できる一因になっていると考えられる．

また，上述のような誤りをしなかった被験者においても，GUI の方が文字列よりもパターンの作成に要した時間が短かった．このことから，誤りなくパターンを入力できたとしても，構造の入力においては GUI の方がより優れていることが分かる．

3.3.4.2. 正規表現を用いたパターンの作成

正規表現を用いたパターンとして，「1 より小さい小数の平方根」にマッチするパターンについて実験した．被験者への指示は次の通りである．

> 平方根の内容が，　“0”，　“.”，　0,1,2,3,4,5,6,7,8,9 のいずれかの 1 回以上の繰り返しであるようなパターンを作ってください．

このパターンの作成には，文字クラスを用いた被験者と選言を用いた被験者がいたが，いずれも正解としている．

「マッチする数式」は$\sqrt{0.1}$および$\sqrt{0.625}$，「マッチしてはいけない数式」は$\sqrt{xy}$および$\sqrt{0.1+0.1}$とした．

結果を表 3.8 に示す．「GUI」および「文字列」の列が「誤答」となっている箇所は，被験者が指示された通りにパターンを作成しなかったことを意味する．

表 3.8 正規表現を用いたパターンの作成に要した時間

グループ	被験者	GUI	文字列
A (先に GUI)	A_1	4 分 26 秒	58 秒
	A_2	1 分 29 秒	1 分 51 秒
	A_3	3 分 19 秒	1 分 12 秒
	A_4	10 分 26 秒	1 分 03 秒
	A_5	誤答	58 秒
	A_6	2 分 20 秒	35 秒
B (先に 文字列)	B_1	1 分 17 秒	11 分 03 秒
	B_2	36 秒	31 秒
	B_3	6 分 34 秒	誤答
	B_4	7 分 16 秒	4 分 20 秒
	B_5	35 秒	1 分 07 秒
	B_6	誤答	3 分 41 秒

正しくパターンを作成した 9 人の被験者のうち，GUI を用いた方が短時間で入力をでき

た被験者は 3 人しかおらず，「1 より小さい小数の平方根」にマッチするパターンの作成においては，GUI はあまり効果を発揮しなかった．

この原因は，「1 より小さい小数の平方根」のパターンの作成そのものが，被験者にとって難しかったということが挙げられる．そのため，先に GUI を用いてパターンを作成した被験者は GUI の方が長い時間を要し，先に文字列としてパターンを作成した被験者は文字列に長い時間を要するという傾向となった．

被験者 B_2 は，GUI でも文字列でも短時間でパターンの作成を終えており，パターンの作成に難しさを感じなかったと考えられるが，GUI でのパターン作成により長い時間を要している．これは「1 より小さい小数の平方根」が構造としては簡潔なパターンであったためであると考えられる．本研究の GUI は複雑な構造を持つパターンにおいて正規表現のスコープを直観的に提示し誤りを防ぐことを目指しているが，構造が簡潔であり文字列による表現でもスコープを容易に把握できる場合には，文字列を用いる方が素早くパターンを作成できる場合がある．

被験者 B_4 を見ると，先に文字列によってパターンを作成しているため正しいパターンを把握しているにも関わらず，GUI を用いた方が顕著に長い時間を要している．被験者 B_4 が正しいパターンに至るまでに検索に用いたパターンを観察したところ，文字クラスの誤入力を何度も繰り返しており，これが原因で GUI でのパターンの作成に時間がかかったものと思われる．

3.3.4.3. 後方参照を用いたパターンの作成

後方参照を用いたパターンとして，「和積の公式にマッチするパターン」について実験した．被験者への指示は次の通りである．

次の和積の公式$\sin x + \sin y = 2\sin\frac{x+y}{2}\cos\frac{x-y}{2}$にマッチするパターンを作ってください．

ここで，xおよびyが他の記号に置き換わってもマッチするようにしてください．

「マッチする数式」は$\sin x + \sin y = 2\sin\frac{x+y}{2}\cos\frac{x-y}{2}$および$\sin a + \sin b = 2\sin\frac{a+b}{2}\cos\frac{a-b}{2}$,

「マッチしてはいけない数式」は$\sin a + \sin b = 2\sin\frac{c+d}{2}\cos\frac{e-f}{2}$とした．

結果を表 3.9 に示す．

表 3.9 後方参照を用いたパターンの作成に要した時間

グループ	被験者	GUI	文字列
A (先に GUI)	A_1	2 分 51 秒	22 分 12 秒
	A_2	2 分 07 秒	2 分 36 秒
	A_3	2 分 08 秒	21 分 47 秒
	A_4	2 分 08 秒	16 分 14 秒
	A_5	1 分 36 秒	1 分 05 秒
	A_6	1 分 24 秒	16 分 20 秒
B	B_1	1 分 13 秒	3 分 51 秒

(先に文字列)	B_2	54 秒	1 分 57 秒
	B_3	1 分 40 秒	10 分 17 秒
	B_4	1 分 54 秒	12 分 48 秒
	B_5	1 分 18 秒	2 分 17 秒
	B_6	1 分 04 秒	5 分 25 秒

被験者 A_5 を除き，GUI を用いた方が短時間でパターンを作成できている．また，文字列を用いた場合にはパターンの作成に 10 分以上を要する場合もあるが，GUI を用いた場合には全員が 3 分以内に作業を終えることができている．

文字列でのパターンの作成に 10 分以上の時間を要した被験者（被験者 A_1, A_3, A_4, A_6, B_3, B_4）の検索の過程を観察すると，全員が正規表現のメタ文字“+”のエスケープを正しく行わずに検索しており，その誤りに気付くまでに長い時間を要していた．また，「和積の公式にマッチするパターン」の作成には記号“+”が 2 回出現するが，このうち 1 つしかエスケープせずに検索している場合もあった（被験者 A_2, A_3, A_4, A_6, B_2, B_6）．この誤りは，正規表現を理解しておりメタ文字をエスケープする必要があるとわかっていても，エスケープをし忘れてしまう場合があることを示している．これらから，数式において頻繁に用いられる記号が正規表現のメタ文字と衝突していることは，ユーザにとって非常に不便であることがわかる．GUI では，正規表現を図形として表示しメタ文字を用いる必要がないため，このような混乱を避けることができる．この特長が，検索に要する時間の短縮に反映されていると考えられる．

3.4. 議論

ここまでに述べた，正規表現を用いた数式のマッチングの特徴を，先行研究と比較しつつ述べる．

3.4.1. 数式中のパターンの位置の特定

本手法の特長として，数式中のパターンの位置を特定できるという点が挙げられる．例えば，xというパターンを入力したら，マッチングされる数式中のどの位置にxが存在するかを特定できる．これによって，ハイライト表示や，置換といった応用を実現することが可能となる．

一方，数式の間の類似性に着目した検索などの場合，問い合わせの数式全体と，検索対象の数式全体との間の関係を見るため，検索結果は数式全体であり，数式中のどの位置に問い合わせを含むか，といった情報は得られない．

3.4.2. 検索結果の順序付け

検索において重要な機能として，検索結果の順序付けがある．検索結果の順序付けは，問い合わせと検索対象との関連性の強さに基づくものと，問い合わせからは独立に検索対象のみから定まるものがある．

問い合わせと検索対象との関連性に基づいて検索結果を順序付けすることができれば，大量の数式から曖昧な記憶に基づいて数式を探し出したり，ユーザにとって有用な数式の発見を促したりすることが可能となる．しかし，本手法は，検索対象である数式の一部にパターン（問い合わせ）がマッチしたか否かを二値的に判定するものであるため，問い合わせと検索対象との関連性の強さに基づくような順序付けはできない．

問い合わせから独立に順序付けを行うことのできる手法を本手法と併用すれば，検索対象の順序付けは可能である．例えば，数式を含む web ページを検索し，検索結果を順序着けする場合，本手法を用いてパターンを含む web ページの集合を得た上で，PageRank[49]を用いて web ページの順序付けを行えば良い．

3.4.3. 数式における文字の数学的な性質の考慮

数式の検索を行う上で，数学的な性質が考慮される場合は多い．それらの処理のうち，複数の研究で扱われているものを挙げる．

3.4.3.1. 識別子の同一視

数式においては，識別子の記号の違いは特別な意味を持たない．例えば，$\sin x$と$\sin \theta$は，表層的には記号が異なっているが，数式の意味する内容は同一であると考えることもできる．この性質を考慮し，問い合わせに含まれるxを，aやθともマッチさせるという処理を行うような検索システムもある．

本研究においてこのようなマッチングを行いたい場合は，文字クラスを用いるという方法がある．`[a-zA-Zα-ωA-Ω]`と入力すれば，任意の英字およびギリシャ文字にマッチさせることができる．添字が付された文字も識別子であるとみなしたい場合は，`[a-zA-Zα-ωA-Ω]¥sub{.+}|[a-zA-Zα-ωA-Ω]`とすれば良い．

3.4.3.2. 文字の意味の考慮

数式において，文字が特別な意味を持つ場合がある．例えば，+や∩は演算子，aやxは識別子を意味する場合が多い．また，$\sin x$における s は sin という関数の名称の一部であり，$s+t$における s は 1 つの識別子である．

本手法では，文字の意味は考慮しない．これは，意味を考慮するためには推測が必要となるためである．例えばsinhxという数式があった場合，三角関数$\sin hx$なのか，双曲線関数$\sinh x$なのか，あるいは s, i, n, h, x という 5 つの識別子の積なのか，数式の表示からは判別できない．ここで，記号のフォントは判別の手がかりとなり得るが，前述のとおり，そも

そも記号のフォントは正しく用いられない場合があるため，本研究ではフォントの情報は表示の情報として扱っていない．

検索システムによる意味の推測がユーザの判断と異なっている場合，ユーザが検索結果として得たい数式が得られないという事態が生じる．パターンマッチングに基づく検索において，この問題は利便性を大きく低下させると考えられる．ここで，もし高い精度で推測を行う手段があれば，利便性の低下を避けつつ，提案手法に新たな機能を追加できる可能性がある．これについては 4. で後述する．

なお，演算子と識別子を区別するようなパターンは，文字クラスを用いて擬似的に表現できる．例えば，「2 つの識別子の間の四則演算」を意図した問い合わせは，`[a-zA-Zα-ωA-Ω][+¥-÷×・]?[a-zA-Zα-ωA-Ω]`などとすれば良い．

3.4.3.3. 数学的な性質を考慮した置換

本手法の重要な応用として，置換が挙げられる．置換を行う際は，数学的な性質は考慮しない方が望ましい．例えば，識別子の同一視を行うと，x という問い合わせに対してaやθもマッチさせてしまうため，xだけをθに置換するといった処理ができない．文字の意味の考慮を行うと，sinhx という数式が実際は$\sin hx$を意図して記述されたものであるにも関わらず$\sinh x$であると推測されてしまった場合，sin が sinh にマッチしないため，sin を cos に置換するといった処理を正しく行うことができない．

4. 意味の推定

本章では，数式検索の利便性をさらに向上させることを目的として，数式の表示の情報をもとにその数式の意味を推定する方法を提案する．まず，ここまでに提案した数式検索手法の限界を示し，その限界に対する数式の意味の推定の意義，および機械学習を用いて意味推定を実現する方法の概要を述べる．次に，機械学習において必要となる MathML Content Markup について概説する．その後，意味推定の手法を詳述し，その評価を行う．

4.1. 数式検索と意味の推定

ここまでに述べた数式検索手法では対応できないような検索について述べた後，意味の推定を用いてそれを解決する方法を示す．また，これまでに述べた正規化およびパターンマッチングと，意味の推定（および意味の推定を用いた検索）との関係に触れる．最後に，意味の推定を実現するための方法を概説する．

4.1.1. 提案手法の問題とその解決方法

本論文ではここまで，表示の情報で表現された数式を検索する方法を提案した．本手法では，検索における問い合わせ（パターン）および検索対象が，ともに表示の情報で表現される．本手法では，正規表現を用いることができるため，柔軟にパターンを作成することができるが，ユーザの検索の意図をパターンとして表現することができない場合もある．それは，ユーザが数式の意味的な側面も考慮した検索を行いたいと考えた場合である．

例として，ユーザが，¥|.*¥| というパターンで検索を行ったとする（“|”はメタ文字であるが，ここでは記号として“|”を検索したいためエスケープしている）．その結果は，例えば次のようになるであろう．

$$|P \cup Q| = 3$$

$$|A| = ad - bc$$

$$|-5| = 5$$

この結果は，本手法のマッチング規則に照らせば，正しいものである．ここで，このユーザの検索の目的が，集合の濃度に関する数式を得ることであったとしよう．その場合，最初の数式だけがユーザの望む数式であり，2 番目の数式（行列式）および 3 番目の数式（絶対値）は，ユーザの望む数式ではないであろう．しかし本手法においては，“|”は単に 1 つの記号であり，それ以外の何ものでもない．このため，“|”の意味を考慮して，パターンにマッチさせるか否かを判定するような処理はできない．また，本手法では，¥|.*¥| というパターンで検索をした時，$\mathrm{card}\, A = 4$という数式のどの部分もハイライトしないが，card は集合の濃度を意味する記法であるため，このユーザは$\mathrm{card}\, A = 4$も検索結果として提示されることを望んだであろう．このように，本手法は，表示の情報ではない情報（意味的な情報）を扱う能力を持たないため，意味的な側面までも考慮した検索を行うという目的には利用できな

い.

この問題は，問い合わせとして入力できる情報の拡張，および数式の意味の推定を行うことで解決できる．問い合わせの拡張として，パターンだけでなく，自然言語によるキーワードの入力も許す．キーワードは，パターンに含まれる多義的な記号がとり得る意味のリストから選択するという方法でも入力できる．¥|.*¥| というパターンの場合，“|”が多義的であるため，「絶対値」「行列式」「集合の濃度」という意味のリストが提示され，そのリストから選択することでキーワードを入力できる．また，意味の推定を行うことで，数式中の記号に自動的に意味のラベルを割り当てる．上の例であれば，推定の結果として，最初の数式には集合の濃度というラベルが，2 番目および 3 番目の数式にはそれぞれ，行列式，および絶対値というラベルが付与される．ここで，例えばユーザが，パターン¥|.*¥| に加えてキーワードとして「集合の濃度」と入力したとすると，このキーワードをラベルとして含む数式は最初の数式のみであるため，最初の数式だけが検索結果として得られる．また，パターンを入力せずにキーワード「集合の濃度」だけを入力した場合，$\mathrm{card}\, A = 4$という数式にも集合の濃度というラベルが与えられるため，$|P \cup Q| = 3$という数式に加え，$\mathrm{card}\, A = 4$も検索結果として得ることができる．

ここで，意味の推定を用いた検索は，表示の情報のみで実現することができる．検索の対象は表示の情報で記述された数式であり，意味の推定に必要となるのも表示の情報のみである．つまり，数式の意味的な側面を考慮するものの，意味の情報は一切必要としないため，通常の web ページや電子書籍などに記述された数式も，検索の対象とすることができる．この点で，意味の情報を対象とした検索手法とは大きく異なるものである．

4.1.2. 正規化/パターンマッチングと意味の推定の関係

意味の推定を用いた検索と，正規化およびパターンマッチングとの関係について整理する．

正規化は，パターンマッチングを実現するための前処理として提案したものであるが，意味の推定を行う上でも有効に働く．これは，意味の推定を行う際にも，パターンマッチングと同様，数式の表示の情報を表す木構造に着目するためである．正規化を行えば，データのゆらぎが解消され 1 つの数式の表示が 1 つの木構造に定まるため，推定を誤ったり，推定の処理が過度に複雑になったりすることを避けられる．

パターンマッチングについては，意味の推定を併用した検索においても問い合わせとしてパターンを入力することを許すため，正規表現を用いてパターンを柔軟に記述できるという特長は依然として検索に効果を発揮する．パターンマッチングのみを用いた検索との違いとして，意味の推定を併用することで，数式中のパターンの位置を特定することができなくなり，代わりに，問い合わせを満たす数式を検索結果として提示するようになる．例えば，$|P \cup Q| = 3$という数式に対して，¥|.*¥| というパターンのみを用いて検索した場合，$|P \cup Q| = 3$のように，数式中のパターンの位置を特定しハイライト表示などの処理を行うこ

とができる．一方，意味の推定を併用し，問い合わせとしてパターン¥|.*¥|とキーワード「集合の濃度」を入力した場合，パターンにマッチする部分を含み，かつキーワードをラベルとして持つような数式そのものが結果として出力される．

4.1.3. 機械学習を用いた意味推定

本章では，意味の推定を併用した検索を実現するための第一歩として，意味の推定を行う手法を提案する．意味の推定は，数式中の個々の記号に対して行う．ここで，ある記号が複数の意味に解釈され得る場合にその意味を特定することは，その記号がどの意味に分類されるかを判定すること，つまり分類問題を解くことであると捉えることができる．そこで本研究では，機械学習における教師あり学習の手法を用いて，曖昧さを持つ記号ごとに分類器を構築することで，それぞれの記号に意味のラベルを与え，意味推定を実現する．

分類器を構築する際，記号をラベルに分類するための手がかりとなるような特徴が必要となる．本手法では，ある記号と同一の数式内に存在する他の記号が，記号の意味を特定する手がかりになるという考えの下，分類のための特徴を設計する．例えば，$|A|$という記号と∩という記号が同一の数式中に存在していれば，この“|”は集合の濃度という意味を持っている可能性が高いと言えよう．これは言い換えると，“|”という記号を「集合の濃度」というラベルに分類する上で，“|”と同一の数式に含まれる記号である∩が手がかりとして効果を発揮したということである．さらに本研究では，数式中の記号の順序関係も，意味の推定に有用な特徴であると考える．例えば“i”という記号が，“s”の右，“n”の左にあれば，これは正弦関数（の一部）という意味を持つであろう．記号の順序関係を表現するために，自然言語における n-gram に相当する概念を数式に導入する．

4.2. MathML Content Markup

数式の意味推定の対象となるのは，表示の情報，すなわち，MathML Presentation Markup で記述された数式である．しかし，推定を行うための分類器を構築する過程においては，意味の情報を記述するためのデータ形式である MathML Content Markup を利用する．本節では，Content Markup について説明する．

Content Markup では，演算およびその対象を記述することで数式を表現する．図 4.1 に，Content Markup のデータの例を示す．

```
<math>
    <apply>
        <power/>
        <ci>x</ci>
        <cn>2</cn>
    </apply>
</math>
```

図 4.1 MathML による記述の例

apply 要素は Content Markup の中心的な要素であり，「第一の子である演算を，第二以降の子に適用する」ことを表す．図 4.1 は，べき乗という演算を底 x および指数 2 に対して適用

するということを表しており，すなわち x の 2 乗を意味している．

本研究では，Content Markup で表現される情報のうち，演算と，特別な概念に着目する．演算とはすなわち，apply 要素の第一の子である．apply 要素の第一の子となり得る要素には，「等しい」ことを意味する eq 要素など，数学的には関係と呼ばれるものも含まれるが，本研究ではそれらもすべて演算と総称する．特別な概念は，Content Markup においては子を持たない要素として表される．具体的には，虚数単位を意味する imaginaryi 要素，ネイピア数を意味する exponentiale 要素, 実数の集合を意味する reals 要素などがある．なお, imaginaryi 要素および exponentiale 要素の要素名は，虚数単位およびネイピア数という概念が i や e といった記号で表現されると誤解させるようなものであるが，実際には Content Markup は，これらの概念がどのように表示されるかを規定しない．Content Markup で記述された数式を表示する際は，多くの場合，XSLT と呼ばれるスタイルシートを用いて Content Markup を Presentation Markup に変換するという方法をとる．ここで，imaginaryi 要素をiに変換するようなスタイルシートを用いれば，虚数単位という概念はiと表示され，imaginaryi 要素をjに変換するようなスタイルシートを（例えば電磁気学の文書のために）用いれば，虚数単位という概念はjと表示される．

演算および特別な概念の他に Content Markup で表現することのできる情報として，記号の型が挙げられる．型は付加的な情報であり，記述しなかったとしても Content Markup データとして妥当なものとなる．型を用いれば，例えば，x という記号が整数であるのか実数であるのか，といった情報を記述できる．しかし本研究では型には着目しない．これは，本研究において分類器を構築するための学習データに型の情報が記述されていないためである．

4.3. 分類

記号の意味推定は，個々の記号を意味のラベルに分類する問題であると捉えることができる．本研究では，教師あり学習を用いて記号ごとに分類器を構築することで，記号の曖昧さを解消する．

分類器を構築するためには，ラベルと特徴の組である学習データを用意する必要がある．本研究におけるラベルと特徴について述べる．その後，ラベルと特徴の具体例を示した後，分類器の構築について概説する．

4.3.1. ラベル

ラベルは，数式の表示の情報を表す木におけるそれぞれの節に対して付与される．すなわち，記号節だけでなく，構造節にも付与され得る．例えば，power（べき乗）というラベルをx^2という数式に付与する場合，「上付き文字」という構造そのものがべき乗を表しているため，上付き文字という構造節にラベルが付与される．

ラベルは，記号の意味である．これは具体的には，Content Markup の第一の子，あるいは

子を持たない要素の名前である．例えば，“|”という記号に付与され得るラベルは，abs（絶対値），card（集合の濃度），determinant（行列式）などのうちのいずれかである．

ラベルは，すべての節に付与されるわけではない．例えば，$x+y$という数式が，x と y の加算を意味している場合，+には plus というラベルが付与され，xおよびyにはラベルは付与されない．

4.3.2. 特徴

本研究では，「ある記号の意味は，その記号が出現する数式中の他の記号によって定まる」という立場を取る．したがって，数式の木に出現する節の頻度を分類のための特徴とする．なお，節の頻度であるため，記号節だけでなく構造節の頻度も数える．

さらに，記号の順序関係も，意味の推定に有効に働くと考えられる．順序関係を特徴として扱うために，自然言語における n-gram の概念を数式に導入する．n-gram とは，文中の n 連語のことを指す. 例えば, “This sentence is an example.”という文の 2-gram は，“this sentence,” “sentence is,” “is an,” “an example”である．しかし，数式の場合，記号が横一列に配置されないため，自然言語と同様の n-gram を用いることはできない．そこで，数式のための n-gram を定義する．数式の n-gram とは，数式の表示の情報を表す木において，同じ統合節を親に持つ連続する n つの節とする．例えば，図 4.2 の 2-gram は，“2 x”, “x 上付き”, “上付き, +”, “+ 3”, “- a”となる．

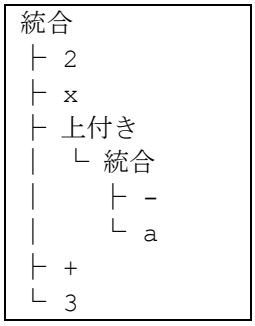


図 4.2 $2x^{-a}+3$の表示の情報の木構造

このようにして得られた n-gram の頻度も，分類のための特徴として用いる．

4.3.3. ラベルと特徴の例

1 つの数式に対する，学習データの作成例を示す．例として，$z=re^{i\theta}$という数式を考える．この数式を表す正規化後の Presentation Markup のデータと Content Markup のデータを，図 4.3 に示す．

```
<math>
  <mi>z</mi>
  <mi>=</mi>
  <mi>r</mi>
  <mi>e</mi>
  <msup>
    <mrow/>
```

```
<math>
  <apply><eq/>
    <ci>z</ci>
    <apply><times/>
      <ci>r</ci>
      <apply><power/>
        <exponentiale/>
```

```
    <mrow>
      <mi>i</mi>
      <mi>θ</mi>
    </mrow>
  </msup>
</math>
```

```
        <apply><times/>
          <imaginaryi/>
          <ci>θ</ci>
        </apply>
      </apply>
    </apply>
  </apply>
</math>
```

図 4.3 $z = re^{i\theta}$ の Presentation Markup および Content Markup のデータ

このうち，Content Markup のデータからラベルを得る．ラベルとなるのは，図において網掛けで示した 6 つの要素である．この 6 つのラベルを，Presentation Markup から得られる木構造の各節に対して割り当てる．この割り当ては手作業で行う．ラベルが割り当てられた木を図 4.4 に示す．

```
math
  ├ mi/z
  ├ mi/=      eq
  ├ mi/r
  ├ mi/e      exponentiale
  └ msup      power
      ├ mrow
      └ mrow
          ├ mi/i      imaginaryi
          └ mi/θ
```

図 4.4 ラベルが割り当てられた木

図を見るとわかるように，ラベルが割り当てられない節もある．また，節に割り当てられないラベルもある．具体的には，times はこの数式中では記号や位置構造として表現されていないため，ラベルとして割り当てられない．

次に，数式の表示の情報から得られる木構造をもとに特徴を得る．記号および n-gram（n は 2 および 3）の頻度を表 4.1 に示す．括弧で囲まれたものは n-gram である．

表 4.1 記号および n-gram の頻度

記号および n-gram	頻度
z	1
=	1
r	1
e	1
msup	1
i	1
θ	1
(z, =)	1
(=, r)	1
(r, e)	1

(e, msup)	1
(i, θ)	1
(z, =, r)	1
(=, r, e)	1
(r, e, msup)	1

こうして得られたラベルと特徴の組を学習データとする．本研究では，分類器は 1 つだけ構築されるのではなく，記号および位置構造ごとに構築される．この例では，“=”の分類器の構築のためにラベルが eq で特徴が表 4.1 であるような学習データが得られ，“e”の分類器のためにラベルが exponentiale で特徴が表 4.1 であるような学習データが得られ，同様に，msup 要素および“i”のための学習データが得られる．

この手順を多くの数式に対して行うことで，記号ごとの学習データが多数得られる．この多数の学習データをもとに学習を行うことで，分類器が構築される．得られた分類器に対して特徴を入力すると，分類器はラベルを出力する．これにより，記号ごとの意味推定が実現される．

4.4. 実験

提案した数式の意味推定手法の有効性を調べるために，簡単な実験を行った．

4.4.1. データ

実験のために，データセットを作成した．データセットの作成のために，Wolfram Functions Site から数式を収集した．Wolfram Functions Site では，同一の数式に対して，Presentation Markup および Content Markup の両方のデータを提供している．このうち，Content Markup のデータから，apply 要素の第一の子および子を持たない要素を取り出してラベルとし，正規化を施した Presentation Markup の節に対して手作業でラベルを付与した．この方法で，Wolfram Functions Site の Elementary Functions カテゴリの Sqrt に属する 220 の数式に対してラベル付けを行った．2 種類以上のラベルが付与され，かつ，30 回以上出現した節は 11 種類存在した．収集したデータに対して，4.3.2. で述べた方法で特徴ベクトルを得た．なお，特徴ベクトルを得るにあたって，n-gram の n は 3 とした．

なお，数式のためのデータセットとして，200 の数式にアノテートを行った[50]などがあるが，記号にラベルが付与された学習データとして利用することは難しく，今回は利用しなかった．

4.4.2. アルゴリズム

実験において，分類のためのアルゴリズムにはランダムフォレスト[51]を用いた．ランダムフォレストはアンサンブル学習の一種であり，分類のためのアルゴリズムとして広く知られている．実装には，python 上で動作する機械学習ライブラリである，scikit-learn[52]を

用いた. ランダムフォレストのパラメタは, scikit-learn 0.16.1 におけるデフォルト値とした.

ランダムフォレストを用いるには，特徴がベクトルとして表現されていなければならない．今回の実験では，特徴ベクトルの要素を，数式の木における各節の頻度，2-gram の頻度，および 3-gram の頻度とした.

4.4.3. 結果

学習データを 2 つに分割し，交差検定を行った．11 の節に対する分類の結果を，表 4.2 に示す．なお，正解率は，分類で得られたラベルが正しいラベルが同一であった割合であり，ベースラインは，それぞれの節において最も頻繁に出現したラベルの出現率である.

表 4.2 それぞれの節における分類の正解率およびベースライン

節	正解率(%)	ベースライン(%)
∞	97.1	97.0
+	99.4	98.9
-	98.1	97.7
/	73.0	74.6
;	100	98.0
a	99.1	94.8
g	90.8	77.9
m	76.3	69.7
msup	92.7	93.1
mfrac	92.0	96.4
msub	91.8	87.0

11 のうち 8 つの節で，分類の正解率がベースラインを超えた．正解率が最も低かったのは“/”で，73%あった（ベースラインは 74.6%）．Wolfram Functions Site では，“/;”という 2 つの記号で「これ以降の数式は条件である」という意味を表す特殊な記法があり，この場合 Content Markup のデータに Condition という演算が含まれているため，“/”には Condition というラベルを付与した．また，“/”が除算を表す場合には，divide というラベルを付与した．“/”の正解率が低かった原因は，“/;”が一般的な記法ではないため，“/;”と同一の数式中に出現する記号が，“/;”における“/”のラベル付けの手がかりとならなかったためであると考えられる．正解率がベースラインを下回った残る 2 つの節はともに構造節であり，msup 要素および mfrac 要素であった．msup 要素のラベルはほとんどが power で稀に arctan や partialdiff，mfrac 要素はほとんどが divide で稀に partialdiff であったため，最大出現率が正解率よりも高い値となった.

分類が特に有効に働いたのは，複数文字で 1 つの演算を構成する記号である．例えば“g”は，ベースラインは 77.8%であったが，正解率は 90.8%であった．“g”には，sign, arg, ln, functionName という 4 種類のラベルが付与された．なお，それぞれのラベルが付与された数

式の例を挙げると，$\mathrm{sgn}(\sqrt{z})$，$\arg(z)$，$\log(z)$，$\sqrt{g(z)}$である．

“g”の分類において，分類に有効に働いた特徴の上位 5 件をジニ係数の平均減分をもとに求めた結果を，表 4.3 に示す．なお，括弧で囲まれたものは n-gram である．

表 4.3 “g”の分類に有効に働いた特徴

“l”
(“2”, “g”, “(“)
(“a”, “r”)
(“a”, “r”, “g”)
(“=”, “c”, msub)

上位 5 件のなかに n-gram が多く含まれていることから，提案した数式の n-gram が，意味推定のための特徴として有効であったことがわかる．それぞれの特徴を見ると，まず“l”は，ラベルが ln かそれ以外かを区別する特徴として働いたものと思われる．(“2”, “g”, “(“)は functionName の分類に，(“a”, “r”)および(“a”, “r”, “g”)は arg の分類に有効に働いたと考えられる．(“=”, “c”, msub)が上位となった理由は不明である．

5. 結論

本章では，本論文を総括するとともに，今後の展望を示す．

5.1. 総括

本論文では，数式における表示と意味の 2 つの側面を明確に区別した上で，数式の表示の情報に対してパターンマッチングに基づく検索を行う手法を提案した．さらに，表示の情報に対して意味的な問い合わせを行うための礎として，表示の情報に対して意味のラベルを与える枠組みを示した．

表示の情報を正確に処理するために，表示の情報を表すためのデータ形式として広く普及している MathML Presentation Markup で記述されたデータを整形する正規化処理を提案した．正規化を行うことにより，同一の数式の表示を表すデータが一意に定まる．正規化は，検索処理および意味推定の両方にとって，精度の向上および処理の簡単化に有用なものである．また，正規化後のデータは MathML Presentation Markup のデータとして妥当なものとなるようにしているため，MathML Presentation Markup を対象とする処理全般に益するものとなる．

検索処理としては，パターンマッチングに基づく手法を提案した．パターンマッチングにより，パターンが存在する位置を特定することができるため，ハイライト表示や置換処理といった応用を実現することができる．本手法では，パターンを記述する際に正規表現を利用することができる．数式の表示の情報は木構造として表現され，通常の正規表現は本質的に木構造を扱うことができないが，パターンを再帰的に呼び出す機能を持つ正規表現エンジンを活用することにより，数式のためのパターンの記述に正規表現を利用することを可能とした．正規表現の機能のうち，数式の検索や置換においては，後方参照が特に効果を発揮する．後方参照によって，検索においては「同一の分母を持つ分数同士の加算」といったパターンを記述できるようになり，置換においては「平方根を$\frac{1}{2}$乗に置換する」といった高度な処理を実現できるようになる．

パターンマッチングにおいて問い合わせ（パターン）として入力できる情報は記号，記号の位置構造，および正規表現であるが，それだけでは，例えば「行列式を含む数式を検索する」といった意味的な検索要求を扱うことができない．表示の情報に対して，このような意味的な検索を行うことを可能とするために，数式の意味推定の枠組みを示した．意味推定は，数式中の記号および記号の位置構造を，意味のラベルへと分類する問題であると捉えることができる．この分類問題を解くために，機械学習を利用した．また，分類をする上での手がかりとして，意味を推定する記号と同一の数式中に存在する他の記号，および記号の順序が有効に働くと考え，これらを特徴として学習を行う方法を述べた．

5.2. 今後の展望

本論文で述べた手法には，さらなる発展の余地がある．

正規化に関しては，見かけ上同一の記号の処理を行うべきである．現在，数式中の記号とは，unicode で定義された 1 つの文字を指し，unicode において異なる文字であるならば，検索において異なる記号であると見なす．しかし unicode には，見かけ上同一に見えるが，異なる文字として定義されているものがある．例えばΩとΩは同一の文字に見えるが，前者はギリシャ文字の大文字のオメガ（コードは#x03a9），後者はオーム記号（#x2126）である．本研究は，見かけ上同一の数式はマッチさせるべきであるという立場であるため，Ωに対してΩがマッチしないという動作は混乱を招くであろう．これを解決するために，unicode 正規化という機構を用いることができる．unicode 正規化とは，unicode で定義されている等価という概念をもとに，文字の合成や分解を行う処理である．unicode 正規化における互換等価性に基づく分解と呼ばれる処理を行えば，ギリシャ文字のオメガとオーム記号はともにギリシャ文字の大文字のオメガに変換されるため，これらをマッチさせることができる．しかし，MathML で記述された数式において見かけ上同一の記号を同一視するためには，unicode 正規化だけでは不十分である．例えば，ṡとṡは同一に見えるが，前者はドットが上に置かれた小文字の s（#x1e61）という単一の記号であり，後者は MathML の mover 要素を用いて`<mover><mi>s</mi><mo>&#x307;</mo></mover>`と記述することで得られるものである．この 2 つを同一視するためには，unicode 正規化だけではなく，MathML のコードの整形を行う必要もある．こういった問題を精査した上で，記号に対しても正しく正規化を行い，見かけ上同一の数式を表現しているデータを真に一意に定めることができれば，検索の利便性はより向上するであろう．

パターンマッチングに関する課題として，利用可能な正規表現の機能をさらに拡充することが挙げられる．現在，正規表現の基本的な機能である，ワイルドカード，量化，選言を実装しており，後方参照も利用可能である．しかし，正規表現にはさらにいくつかの機能がある．そのうちの 1 つに，先読み/後読みと呼ばれる機能がある．例えば，パターンとして`(?<=b)a`と記述すると，これは「直前の文字が b であるような a」にマッチする．ここで，`(?<=p)`という箇所が後読みと呼ばれる機能のための記法であり，p の部分に任意のパターンの記述することで直前のパターンを指定できる．後読みを数式検索に用いると，「直前の文字が sin であるような x」といったパターンを記述できる．このパターンによって，正弦関数の引数であるようなxだけを一括でθに置換するといった処理を行うことができるようになる．後読みは Onigmo でも利用可能なため，ユーザパターンから Onigmo パターンへの変換処理を修正することで，数式検索において後読みを利用できるようになると考えられる．

意味推定は，数式に意味のラベルを付与するための枠組みを示した段階であり，今後取り組むべきことは多い．意味推定そのものについて言えば，学習のためのデータを増やすべき

である．本論文では，220 の数式をラベル付けし学習データとしたが，分類の精度は学習データの量に強く依存するため，データ量は多いほど良い．また，学習データに含まれていない記号については意味のラベルを付与することができないが，データ量を増やし，データに広範な記号が含まれるようになれば，意味のラベルを付与することのできる対象も拡大する．加えて，本研究では，数式に意味のラベルを与える枠組みの実装は行ったものの，意味のラベルを用いた検索は実装に至っていない．この実装に取り組むことも，今後の課題であると言える．

謝辞

本研究を進めるにあたり，長年に渡り懇切なる指導をくださった静岡大学学術院情報学領域准教授 宮崎佳典博士に，心より感謝いたします．また，本研究について有益な議論や温かな励ましをくださった立命館大学文学部教授 田中省作博士に，厚くお礼申し上げます．

お忙しいところ論文審査委員長をお引き受けいただいた静岡大学学術院情報学領域教授 中谷広正博士，副指導教員および論文審査委員をお引き受けいただき本論文を改善するための示唆を与えてくださった静岡大学学術院情報学領域教授 小西達裕博士，青木徹博士に，深く感謝いたします．

日本学術振興会による特別研究員への採用は，本研究の継続にあたって，精神的・経済的な支えとなりました．ここに感謝の意を示します．なお，本研究は，日本学術振興会特別研究員奨励費 26-2758 の助成を受けたものです．

参考文献

[1] Cole, T. W.: MathML in Practice: Issues and Promise, Data Science Journal, Vol.5, pp.209-218 (2006).

[2] Cajori, F.: A History of Mathematical Notations, Courier Corporation, Vol. 1 (1928).

[3] Suzuki, M., Tamari, F., Fukuda, R., Uchida, S., and Kanahori, T.: INFTY: An Integrated OCR System for Mathematical Documents, Proceedings of the 2003 ACM Symposium on Document Engineering, pp. 95-104 (2003).

[4] LaViola Jr, J. J., and Zeleznik, R. C.: Mathpad 2: A System for the Creation and Exploration of Mathematical Sketches, ACM Transactions on Graphics, Vol.23, No.3, pp.432-440 (2004).

[5] Cervone, D.: MathJax: A Platform for Mathematics on the Web, Notices of the AMS, Vol.59, No.2, pp.312-316 (2012).

[6] Wolfram, S.: Mathematica: A System for Doing Mathematics by Computer, Addison Wesley Longman Publishing Co., Inc. (1991).

[7] Maxima: http://maxima.sourceforge.net/

[8] Maple T.A.: http://www.maplesoft.com/products/mapleta/

[9] Sangwin, C.: Computer Aided Assessment of Mathematics, Oxford University Press (2013).

[10] Kovalchuk, A., Levitsky, V., Samolyuk, I., and Yanchuk, V.: The Formulator MathML Editor Project: User-friendly Authoring of Content Markup Documents, Intelligent Computer Mathematics, pp.385-397 (2010).

[11] Kawata, T., Kataoka, M., Kai, H., and Tamura, Y.: A MathML Content Markup Editor on the xfy, Applications for Computer Algebra (2008).

[12] Wolfram Functions Site: http://functions.wolfram.com/

[13] Knuth, D.: Mathematical Typography, Bulletin of the American Mathematical Society, Vol.1, No.2, pp.337-372 (1979).

[14] MathML: http://www.w3.org/TR/MathML3/

[15] Buswell, S., Caprotti, O., Carlisle, D. P., Dewar, M. C., Gaetano, M., and Kohlhase, M.: The Open Math Standard. Version 2.0, Technical report, The Open Math Society (2004).

[16] Aizawa, A., Kohlhase, M., and Ounis, I.: NTCIR-10 Math Pilot Task Overview, Proceedings of the 10th NTCIR Conference, pp.654-661 (2013).

[17] Zanibbi, R., and Blostein, D.: Recognition and Retrieval of Mathematical Expressions. International Journal on Document Analysis and Recognition, Vol.15, No.4, pp.331-357 (2012).

[18] Misutka, J.: Mathematical Search Engine, Doctoral Thesis, Charles University (2013).

[19] Nghiem, M. Q., Kristianto, G. Y., Topić, G., and Aizawa, A.: Which One Is Better: Presentation-based or Content-based Math Search?, Intelligent Computer Mathematics, pp. 200-212, (2014).

[20] Nghiem, M. Q., Kristianto, G. Y., Topić, G., and Aizawa, A.: A Hybrid Approach for Semantic Enrichment of MathML Mathematical Expressions, Intelligent Computer Mathematics, pp. 278-287 (2013).

[21] Nghiem, M.Q., Yoko, G.K., Matsubayashi, Y., and Aizawa, A.: Automatic Approach to Understanding Mathematical Expressions Using MathML Parallel Markup Corpora, The 26th Annual Conference of the Japanese Society for Artificial Intelligence, p.(1K2-IOS-1b-6) (2012).

[22] SnuggleTeX: http://www2.ph.ed.ac.uk/snuggletex/documentation/overview-and-features.html

[23] Miner, R., and Munavalli, R.: An Approach to Mathematical Search through Query Formulation and Data Normalization, Calculemus/MKM, pp.342-355 (2007).

[24] Watanabe, T., and Miyazaki, Y.: Development of IR Tool for Tree-Structured MathML-based Mathematical Descriptions, International Conference on Computers in Education, pp. 310-312 (2010).

[25] 橋本英樹，土方嘉徳，西田正吾：MathML を対象とした数式検索のためのインデックスに関する調査，情報処理学会研究報告．情報学基礎研究会報告，Vol.2007，No.54，pp.55-59（2007）．

[26] Miller, B. R., and Youssef, A.: Technical Aspects of the Digital Library of Mathematical Functions, Annals of Mathematics and Artificial Intelligence, Vol.38, No.1-3, pp.121-136 (2003).

[27] Youssef, A.: Search of Mathematical Contents: Issues and Methods, IASSE, pp.100-105 (2005).

[28] Libbrecht, P., and Melis, E.: Methods to Access and Retrieve Mathematical Content and Activemath, Mathematical Software - ICMS 2006, pp.331-342 (2006).

[29] Jakarta, A.: Apache Lucene - a High-performance, Full-featured Text Search Engine Library (2004).

[30] Kamali, S. and Tompa, F. W.: Improving Mathematics Retrieval, Towards a Digital Mathematics Library, pp.37-48 (2009).

[31] Yokoi, K., and Aizawa, A.: An Approach to Similarity Search for Mathematical Expressions Using MathML, Towards a Digital Mathematics Library, pp.27-35 (2009).

[32] Nguyen, T. T., Hui, S. C., and Chang, K.: A Lattice-based Approach for Mathematical Search Using Formal Concept Analysis, Expert Systems with Applications, Vol.39, No.5, pp.5820-5828 (2012).

[33] Kohlhase, M., and Sucan, I.: A Search Engine for Mathematical Formulae, Artificial Intelligence and Symbolic Computation, pp.241-253 (2006).

[34] Kohlhase, M., and Prodescu, C. C.: Mathwebsearch at NTCIR-10, National Institute of Informatics Testbeds and Community for Information Access Research Vol.10, pp.675-679 (2013).

[35] Wolfram|Alpha: http://www.wolframalpha.com/

[36] 渡部 孝幸，宮崎 佳典：二次元の位置構造に着目した数式のパターンマッチング手法，情報知識学会誌，Vol.22，No.3，pp.253-271（2012）．

[37] 渡部 孝幸，宮崎 佳典：正規表現を用いた数式検索手法の提案，情報処理学会論文誌，Vol.56，No.5，pp.1417-1427（2015）．

[38] Watabe, T., and Miyazaki, Y.: Graphical User Interface for Search of Mathematical Expressions with Regular Expressions, Human-Computer Interaction: Design and Evaluation, Springer International Publishing, LNCS, Vol.9169, pp.438-447 (2015).

[39] Watabe, T., and Miyazaki, Y.: Framework for Sense Disambiguation of Mathematical Expressions, JJAP Conference Proceedings of 14th International Conference on Global Research and Education "Inter-Academia". (accepted)

[40] STIX Fonts Project: http://www.stixfonts.org/

[41] LaTeXML: http://dlmf.nist.gov/LaTeXML/

[42] センター試験過去問研究 数学 I・A/II・B 2012 年版，教学社，2011.

[43] LyX: http://www.lyx.org/WebJa.Home

[44] Altamimi, M. E. and Youssef, A. S.: Wildcards in Math Search, Implementation Issues, CAINE, pp.90-96 (2007).

[45] Onigmo: https://github.com/k-takata/Onigmo

[46] Comon, H., Dauchet, M., Gilleron R., Jacquemard F., Lugiez D., Tison S. and Tommasi, M.: Tree Automata Techniques and Applications, http://www.grappa.univ-lille3.fr/tata (1997).

[47] XSLT: http://www.w3.org/TR/xslt20/

[48] XQuery: http://www.w3.org/TR/xpath-datamodel/

[49] Page, L., Brin, S., Motwani, R., and Winograd, T.: The PageRank Citation Ranking: Bringing Order to the Web, Technical Report, Stanford InfoLab (1999).

[50] Wolska, M., Grigore, M., and Kohlhase, M.: Using Discourse Context to Interpret Object-denoting Mathematical Expressions, Towards a Digital Mathematics Library, pp.85-101 (2011).

[51] Friedman, J., Hastie, T., and Tibshirani, R.: The Elements of Statistical Learning, Vol.1, (2001).

[52] Pedregosa, F., et al.: Scikit-learn: Machine Learning in Python, The Journal of Machine Learning Research, Vol.12, pp.2825-2830 (2011).