MathML における表現形式から意味形式へのコンバータツールの開発

Development of the converter tool from presentation markup into contents markup

荒川玲佳*1・浅本紀子*2・大武信之*3
Email: arakawa.reika@is.ocha.ac.jp

*1: お茶の水女子大学 人間文化創成科学研究科 理学専攻 情報科学コース
*2: お茶の水女子大学 自然科学系 基幹研究院
*3: 筑波技術大学

◎Key Words　　MathML，表現形式，意味形式

1. はじめに

これまで，ウェブブラウザ上で数式を表現する方法として，画像を埋め込む，もしくは PDF 文書として表示していた．そして近年，新たな方法として MathML という XML ベースの数式記述用マークアップ言語が登場した．数式を画像ではなく，データとして扱うことができるようになった．MathML には，二つの書式があり，一つは数式の見た目の位置情報だけでの記述で，数式の意味構造を持たない表現形式である．もう一つは数式の意味構造を持つ意味形式である．書式の例を以下の図 1 に示す．

【表現形式】

```
<mn>4<mn>
<mi> x </mi>
<mo>+</mo>
<mn>1</mn>
```

【意味形式】

```
<apply><plus/>
        <apply><times/>
                <cn>4</cn>
                <ci> x</ci>
        </apply>
        <cn>1</cn>
</apply>
```

図 1

表現形式は，見えない掛け算が省略されていても，ブラウザに表現することができ，閲覧する上では問題ない．一方の意味形式は，厳密に数式の意味構造と演算子情報を持つため，数式処理が可能である．多くのブラウザでは意味形式の表示がサポートされていないこともあり，数式の書かれたウェブページの多くは表現形式で記述されている．しかし，それらは表示されているだけで，計算処理ができない状態である．

表現形式を意味形式に自動変換して，ウェブページの数式を計算処理が可能な状態にすることで，数式の再利用性が大きく高まる．例えば，数学教育ツールの開発，数式検索の実用化，技術データベースの作成の実現，AI アプリケーションへの応用などが可能となる．

1.1 関連研究

MathML を扱う先行研究では，数式検索技術や LaTeX から表現形式への変換などがある．前者は，数式を DOM ツリーと呼ばれるツリー構造をもとに表現したパス(XPath)をデータベースに登録しておき，ユーザーの入力に対して類似した結果を出力するものである．これらの研究は MathML の意味形式を前提としている．後者は， LaTeX から表現形式に変換する際に，数式の表示属性をより厳密にし，数式の読み上げを正確にする研究などが行われている．また，本研究との類似テーマとして半自動変換を提案する手法がある．これは変換途中で，人が数式構造のパターンを選択し，半自動変換するというものであるが，実際の実装手法や出力結果が確認できないものであった．

1.2 研究目的

本研究では，表現形式から意味形式へ自動変換を行うコンバータツールの開発を行う．ツールの作成が実現されると，関連研究での数式検索技術の実用化はもちろん，数式処理ソフトの Mathematica に MathML の意味形式ソースをインポートすることで複雑な数式を簡単にグラフ化することができる．

また，このテーマは実現が難しいと考えられてきた．それは表現形式が持つ特徴として，ユーザが数式を見た目の位置情報で表現できることから，数式の意味構造が不完全な場合が多いことにある．さらに，中置記法と前置記法の属性が混在した数式順序から，前置記法順で統一された意味形式への数式順序にすることも難しさの原因であると考える．本研究では，それらを克服する手法の提案と実装を行う．

1.3 MathML について

MathML(Mathematical Markup Language) [1] は，XML アプリケーションの一つで，数式を記述するためのマークアップ言語である．W3C (World Wide Web Consortium)[2] によって 1999 年に MathML 規格バージョン 1.0 が勧告され，2010 年にバージョン 3.0 が勧告された．文書として利用するには，HTML 文書に埋め込んで利用する．数学的表記における意味と外観の両方の情報がコード化することができる．表現形式と意味形式の二つの書式があり，表現形式は数式の視覚的二次元構造についての情報をコード化する．そして，意味形式は数式の論理的意味をコード化する．

2. 提案手法と実装

2.1 手法の流れ

意味形式のコードを出力するには，構文木が必要である．表現形式から意味形式に書き換える際，数式情報が完全でない場合は，構文木を一意に作成できなくなるために意味形式への変換が難しくなる．そこで，

以下の図２に示すように，表現形式における演算子の欠落情報をチェックして数式情報を補った上で，補完された表現形式から意味形式への変換処理を行う．

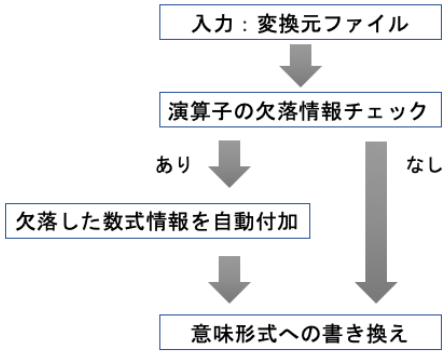

図2 提案手法

次に，この手法を実装するための処理ステップを，次の 2.2 の実装の各処理で詳しく説明する．

2.2 実装の各処理

表１は、変換処理の各処理ステップを表す．

表１　処理ステップ

処理ステップ	各処理説明
① 変換元ファイルの入力	数式が表現された表現形式ソースファイルを入力として受け取る．
② 字句解析及び線形リストの作成	字句解析を行い，数式に直接関わるデータを解析の順序（ファイルの先頭から末尾の流れ）で取り出しを行う．データを構造体形式に格納し，中置記法属性はその順番で双方向リンクリストを作成．構文木作成のためにリストが必要である．前置記法の属性は、双方向リンクリストに対して，枝分かれするようにリンクし，線形リストを作成する．
③ 欠落情報の自動付与	②の線形リストを先頭から，変数と演算子の属性の型の並びを調べ，欠落した場所を特定する．欠落した演算子情報を補う． また，変数が１文字より多く格納されている場合は，１変数ずつになるように分解し，同じように欠落情報を付与する．数式の完全なリストを作成する．
④ 構文木の作成	③の線形リストを，スタックを使って後置記法にリンクを繋ぎ直す．繋ぎ直したリストを後ろから，演算子をノード，その演算子が適用される引数を葉として構文木になるように，各構造体をリンクさせる．
⑤ 意味形式への書き出し	④の構文木を行きがけ順で，各ノードの表現形式の属性を意味形式の属性に置換しながら書き出しを行う．そのとき，演算子の適用範囲を示す属性<apply>と</apply>を付け加える．

ここでは，表現形式における<mn>や<mi>は属性とし、属性で挟まれたデータは要素とする．

コンバータツールは，C 言語で実装する．構成は，マルチモジュールで，表１の処理ステップ①から⑤をサブモジュールとし，それらをメインモジュールが呼びだす仕様である．実行は，ターミナルに実行コマンド Converter [変換したいファイル名.txt] を入力すると，変換結果を新規ファイルに出力する．

変換元ファイルに複数の数式が書かれていても，１つの数式に対して逐次変換処理を行い，一括して変換結果を出力することができる．また，テキスト文書と数式が混在するファイルを変換元としても，変換可能なコード設計である．

2.3 HTML 数値文字参照によるトークン認識

表現形式において，引き算は演算子属性を使って<mo>-</mo>を表すが，半角マイナス記号以外にも，見た目上マイナス記号に見える記号で表現されていることが多い．ハイフンとマイナスを表す記号，全角マイナス，ハイフン，−があり，それらは引き算記号としてトークンとして認識できるように，HTML 数値文字参照 16 進コードを使って，字句解析できるように実装している．また，数字属性<mn> .5</mn>は 0.5 として認識する．

2.4 表現形式において数式情報が不完全な例

2.2 の処理③において，欠落情報を付与すべき演算子情報の例を以下に示す．

- ♦ 見えない掛け算
  （演算子の属性：⁢, ⁢, ⁢）
- ♦ 関数を表す
  （演算子の属性:&ApplyFucntion;, ⁡, ⁡）

見えない掛け算および関数適用は，数式として演算子属性を記述する必要がある．下表 2，下表３で，左は省略された記述例，右は正式な記述の一例を示す．

表 2 見えない掛け算の例

数式の例	省略された記述例	正式な記述の一例
xyz	<mi>x</mi> <mi>y</mi> <mi>z</mi>	<mi>x</mi> <mo>⁢</mo> <mi>y</mi> <mo>⁢</mo> <mi>z</mi>
4ac	<mn>4</mn> <mi>ac</mi>	<mn>4</mn> <mo>⁢</mo> <mi>a</mi> <mo>⁢</mo> <mi>c</mi>

表 3　関数適用の例

数式の例	省略された記述例	正式な記述の一例
$logx$	<mi>log</mi> <mi>x</mi>	<mi>log</mi> <mo>⁡</mo> <mi>x</mi>
$f(x)$	<mn>f</mn> <mi>(x)</mi>	<mi>f</mi> <mo>&Applyfunction;</mo> <mfenced> <mi>x</mi> </mfenced>

省略された箇所の特定に関して，表２の数式 xyz のようにリストの属性型の並びで，変数型が連続している場合には，見えない掛け算が省略されたものと特定する．

また，表２の数式 $4ac$ は，リストに二段階の処理を行う．下図３のように最初の字句解析のデータの取り出しの段階で，acが構造体の要素の情報に保持される．本研究では，変数は１つとみなしているため，acを２つの構造体を分割し，見えない掛け算の演算子属性⁢の構造体を作成し付加する．

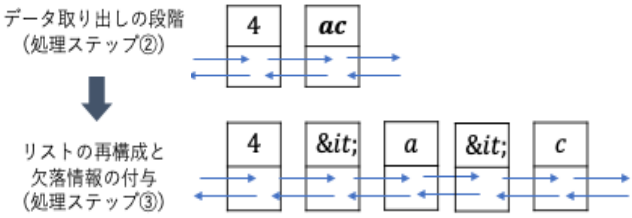

図 3 欠落情報を補う処理

2.5　前置記法属性のリスト構成

一般的に前置記法から後置記法に変換することは，計算機の基本データ構造の特性から難しいとされる．そこで，前置記法の属性の場合，その属性の適用範囲となる引数の式のリストを内包し，中置記法の線形リストに対して枝分かれするようにリストを作成する．全体としてあたかも中置記順に統一することで，構文木の作成及び書き出しを可能にしている．下図４

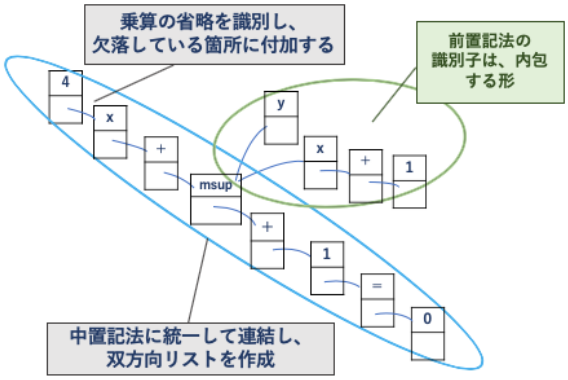

図 4 枝分かれリストの作成

2.6　実処理の様子

実際に数式 $4x + 1 = 0$ を表現形式から意味形式に変換する過程を図示する．(2.2 実装の各処理①から④に該当する．) この例では，中置記法の属性のみの単純な線形リストの例である．

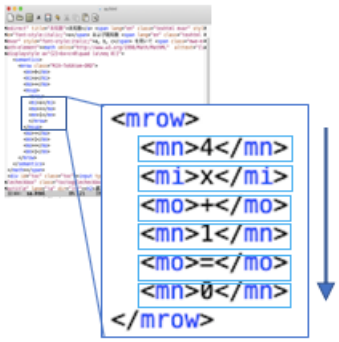

図 5　字句解析と線形リストの作成

リストを構成する各構造体が持つ情報は，表現形式の属性，属性が表す要素，リスト用のポインタ，構文木用のポインタを保持する．前置記法の属性に関しては，さらに多項式用のポインタを保持する．

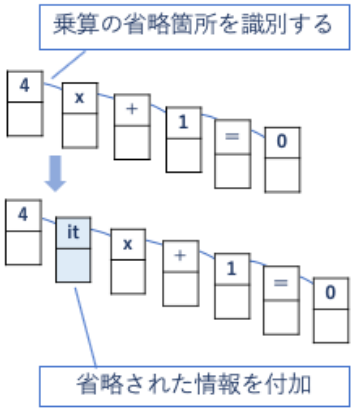

図 6　欠落した演算子情報の付加

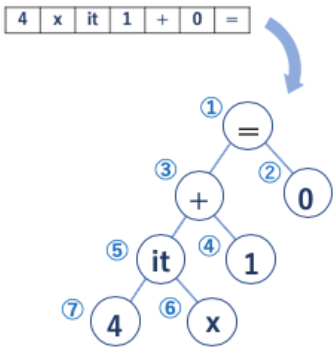

図 7　構文木の作成

中置記法順リストから、スタックを使って後置記法順リストに繋ぎ直し，構文木を作成する．構文木の根から行きがけ順で，表現形式の属性情報を逐次，意味形式の属性に書き換えることにより，意味形式への変換が完了する．

3. 実行結果

実際に数式 $4x + y^{x+1} + 1 = 0$ と記述された変換元ファイルの実行結果を下図に示す. この例では, $4x$のところで見えない掛け算が省略されており, y^{x+1} で <msup> という前置記法の属性が使用されているものである.

Converter samples/sample.html

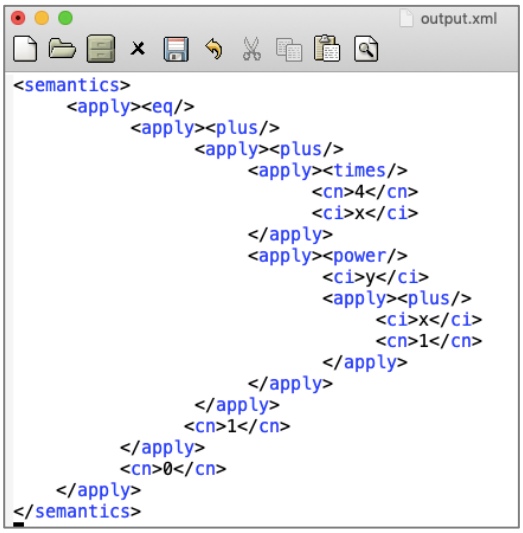

図 6　意味形式への変換結果

表 4 実行結果の検討

変換元（表現形式）	変換後（意味形式）
<mn>4</mn> <mi> x</mi>	<apply><times/> <cn>4</cn> <ci> x</ci> </apply> 欠落した演算子 ⁢ が, 意味形式において掛け算属性を表す times/に変換されている. 論理構造も反映されている.
<msup> <mi>y</mi> <mrow> <mi>x</mi> <mo>+</mo> <mn>1</mn> </mrow> </msup>	<apply><power/> <ci>y</ci> <apply><plus/> <ci>x</ci> <cn>1</cn> </apply> </apply> 前置属性の msup が, べき乗を表す power/に変換されており, 足し算の演算子も plus/に変換されている.

4. 応用技術への展望

表現形式で, ユーザが見た目の位置情報によって記述していた数式を, 論理的意味構造と厳密な属性をも持つことで、数式の再利用性が高まる. 本研究のコンバータツールによって, 以下のような応用技術への展望が考えられる.

- 数式処理ソフトでの処理が可能になるため, グラフ描画が素早くできる
- 先行研究にもある数式検索の実用化
- ユーザの入力に対して動的な正解を出力する数学教育支援ツールの開発
- ウェブページから数式コンテンツごとに分類した技術文書データベースの作成
- 一階述語論理を用いた自動証明
- AI アプリケーションにおける知識データ

5. まとめと今後の課題

今回, 現在の実装状況としてターミナルでコマンドを入力することによって変換結果を取得する形式になっている. よりユーザビリティを考慮してウェブアプリケーションの形で実装し, サイトの URL を入力して変換させると, 変換結果のファイルを生成することを目指す.

また, 構文木は二分木で作成しているため, 演算子一つに対して引数を二つ取る数式しか扱えていない. シグマ演算子や定積分などのような引数の多い演算子に対しての変換を, 二分木に帰着させるかもしくは, 構文木の枝を増やして表現するかの検討と実装を試みたい.

そして変数の扱いに関して, 変数は一文字として実装しているが, 前後の文脈によって変数が複数文字で構成される場合が考えられる. 今後の課題として, 前の文書を文書ベクトル化して, 変数の文字数をパターン認識して, 変数をより厳密に認識できるようにすることを目指す.

参考文献

(1) W3C Math Home http://www.w3.org/Math/ (2019/6/10 閲覧)
(2) W3C https://www.w3.org (2019/6/12 閲覧)
(3) 道廣勇司：“MathML 数式組版入門 ver1.1”, アンテナハウス株式会社 CAS 電子出版 (2017)
(4) 湯浅太一：“コンパイラ”, 昭晃堂 (2004)
(5) Michael Sipser, 太田和夫, 田中圭介：計算理論の基礎 原著第二版, 共立出版 (2009)
(6) MathML マニュアル http://toshichan.be.fukui-nct.ac.jp/tsujino/mathml/chaptch4.xhtml (2019/6/1 閲覧)
(7) 近藤嘉雪：“yacc による C コンパイラプログラミング”, ソフトバンク株式会社出版事業部 (1990)
(8) 中田育男：“コンパイラ”, 株式会社オーム社, (1995)
(9) 橋本英樹, 土方嘉徳, 西田正吾：MathML を対象とした数式検索エンジンの設計, 第 4 回情報プロフェッショナルシンポジウム予稿集, pp. 45-49
(10) 大武信之, 金掘利洋：XML 数式意味記述のための半自動変換, FIT(情報科学技術フォーラム), (2003)