Vol.2019-AAC-11 No.7
2019/12/14

MathML3.0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

荒川 玲佳[1,a)]　浅本 紀子[2]　大武 信之[3]

概要：近年、デジタル数式データの利活用の需要が高まっている．数式をウェブ表記する方法として XML 言語の MathML があり，二つの記述形式が存在する．そのうち，数式が書かれたページで使用されている形式は，閲覧を目的とした形式で記述されている．しかし，この形式では計算処理ができない．本研究では，それをもう一方の計算処理が可能な形式へ自動変換を行うツールを作成し，数式データを再利用可能な状態にすることを目的とする．この自動変換は，各形式の特徴から困難だと考えられており，先行研究の事例が殆どない．本研究ではその課題を克服する手法を提案し実装を行う．数式表現の多様な記述パターンに対応したデータ格納形式を使用し，欠落情報を付加した上で，構文木を作成する．そして順次コードを書出すことで実現した．このツールにより，関連研究の実用化へのハードルを下げることを可能にする．

キーワード：　MathML，数式処理，パーサー，コンバーター

Development tool to convert presentation format to contents format of MathML

Abstract: Recently, demand of using digital numerical formula data is increasing. MathML is a markup language for describing mathematical notation and capturing both its structure and content.There are two formats. The Web pages which are described mathematical formulas are used the format for only projection. This format, however, have the data of layout information of formulas. Therefore, it is not able to be calculated. In this study, in order to enable the formula data to be calculable by converting the format to the other.This automatic converting is considered to be difficult, because of properties of each formats. This study proposes methods: system of storing data correspond to various patterns of formulas projected, automatic appending operation. That enables to realize automatic conversion.

1. はじめに

1.1 研究背景

近年，デジタルの数式データの利活用の需要が高まっている．具体的には，教育の ICT 化により電子黒板やタブレット端末による数式の表示及び計算処理，機械学習で使用される数式技術データベース，センター試験の複数回実施案から想定されるペーパー方式から CBT 方式への切り替えなどといった需要が予想される．これらは数式データが持つ情報が，計算処理が可能な状態であることが必要と考えることができる．

電子情報として数式を記述する方法では，TeX が有名である．また，ウェブ上で数式を表示する方法にも，TeX で書いた PDF をそのまま表示したり，画像として表示したり，XHTML 内部にレンダリング表示する MathJAX など

[1] お茶の水女子大学大学院
Ochanomizu University, Otsuka, Tokyo 112-8610, Japan
[2] お茶の水女子大学基幹研究院
[3] 筑波技術大学
Tsukuba, Ibaraki 305-8520, Japan
[†1] 現在，お茶の水女子大学大学院
Presently with Ochanomizu University
[a)] arakawareika@is.ocha.ac.jp

の形で使用される．しかし，このように表示された数式情報というのは，数式の見た目の位置情報のみしか持たない．従って，計算処理ができないデータである．

また 1998 年に他のウェブ表記方法として，XML 言語の MathML(*Mathematical Markup Language*)[1] が登場した．これは，XHTML 内部に記述することで，リフロー型[*1]の数式記述を可能にした．そして，MathML には二つの書式があり，一つは表現形式，もう一つが意味形式である．表現形式は，数式表示を目的とした形式であり，数式が記述されたウェブページの多くがこちらの形式で記述されている．そして，もう一方の意味形式で，数式の意味が厳密に定義され意味構造を持つ形式である．本研究では，表示と意味の両方の形式が存在する MathML に着目した．

1.2 MathML の記述形式

MathML を使用して，ウェブページで数式を表示する場合，html 形式ページソースで表示したい場所に，`<math xmlns="http://www.w3.org/1998/Math/MathML">` というインターフェース要素で書き始めて，数式情報を記述し終えた末尾に`</math>`で閉じることで表示ができる．

先ほど紹介した形式について詳しく説明する．一つ目の表現形式は，TeX のように数式の見た目である位置情報をもとに記述する方法をとり，数式の記述順序は演算子の種類によって，中置記法と前置記法が混在する．現在，数式が記述されたページのソースはこの形式である．[*2]表示された数式の内容が同じ解釈で理解されれば，正式な記述の仕方でなくても表示ができてしまう．そのため，記述の仕方の自由度が非常に高いという特徴を持つ．また，表示が目的であるため，数式が持つ情報は数式のレイアウト情報のみで，計算処理を行うことができない．数式データとしては，再利用性が低いデータなのである．

そしてもう一方が，意味形式である．これは，数式を構成する要素である演算子や引数の情報が，それぞれ厳密に定義されている．また，数式の記述順序は，計算機が理解しやすい前置記法で統一されており，演算子とそれ自身が持つ引数との関係を木構造で表現できる．この形式は，数式としての意味構造を持つ特徴がある．そのため，計算処理が可能なデータ形式で，再利用性の高いデータである．各形式を記述する際には，演算子や変数、定数の型を表すためのタグや要素が，MathDTD によって定められている．MathML の形式の登場は，構造を記述をするために意味形式が先に作られ，後に TeX ベースにした表現形式が登場した．**表 1** に，数式 $4x+1$ を例に形式の違いを載せる．

[*1] reflow 型：表示する端末画面サイズや文字サイズに合わせて，可変的に表示すること
[*2] 組版がサポートされていないブラウザもある

表 1 数式 $4x+1$ の形式ごとの記述比較
Table 1 Presentation format and Content format

表現形式	意味形式
`<mrow>`	`<apply>`
`<mn>4</mn>`	`<plus/>`
`<mo>&it;</mi>`	`<apply>`
`<mi>x</mi>`	`<times/>`
`<mo>+</mo>`	`<cn>4</cn>`
`<mn>1</mn>`	`<ci>x</ci>`
`</mrow>`	`</apply>`
	`<mn>1</mn>`
	`</apply>`

表現形式のタグや要素は．定数は`<mn>`定数`<\mn>`，変数は`<mi>`変数`</mi>`で表し，意味形式ではそれぞれ`<cn>`定数`<\cn>`，`<ci>`変数`</ci>`で表す．演算子は表現形式では，`<mo>`演算子の記号`</mo>`で表すものが，意味形式では，演算子によって個別にタグが定義されている．表 1 を参照．

また，表現形式において，mo タグで挟まれる演算子は中置記法，それ以外は前置記法となる．中置記法の演算子は加減乗．前置記法の演算子は，分数式，指数式および指数関数，根号である．さらに，前置記法の演算子の中でも引数を二つ取るものと，一つ取るもので分類される．

1.3 関連研究

MathML を使用した関連研究には，数式検索と数学学習支援システムの研究が主に行われている．特に数式検索技術の研究は数多く行われおり，数式の意味構造を利用して様々なアプローチで取り組まれている．しかし，これらの数式検索の研究は，検索の入力クエリの前提として，入力形式が意味形式，正確な表現形式，画像，検索用の新規に定義した言語を使うなどを前提とする大きな制約がある．すなわち，実際のブラウザソースへの適用が難しく，ユーザビリティの点で実用化に課題が残る．

先行研究として唯一，Web 上の数式表現の意味形式記述への半自動変換システム開発 [6] がある．変換方法として，タグの解析を行い，手作業による修正を行った後に解析木を生成し，意味形式出力する方法が提案されている．しかし，解析の処理手順や解析木の生成方法の詳細がないため，実装状況や物理的な構成を確認できるものではなかった．

1.4 研究目的

数式が書かれたウェブページのほとんどが，数式の位置情報による表示データのみで，計算処理ができない状態である．それを他方の計算処理が可能な形式へ変換することで，数式データを再利用可能な状態にすることを本研究の目的とする．

そして数式に関する関連研究において，技術構築の前提となる制約が，実用化を難しくしている要因であると分析

する．そこで，現状のウェブページに対応した形式の自動変換を可能にすることで，それらの制約のハードルが下がり，実用化への一助になると考える．したがって，数式データの再利用性を高めること，そして，関連研究技術の実用化へのハードルを下げることを目的とする．

2. 実験方法

2.1 提案手法

本節では，まず自動変換を行う上での課題を提示し，それから克服する手法を提案し説明する．これまで，MathMLの各形式の特徴から自動的な変換が難しいと考えられてきた．以下に示すような課題が大きく二つ挙げられる．

（課題 1） 数式に必要な演算子の欠落

表現形式の特徴として記述の自由度の高さに起因する．これは表示の見た目上は，明示的に演算子を記述しなくても解釈できてしまうことであった．すなわち，数式に必要な演算子の欠落が生じている記述と捉えることができる．

実際に，表現形式で記述されたソースを見てみると，表示の際に見た目が同じであれば，演算子の記述を省略しても表示可能な場合がある．**表 2** に実例として，ブラウザ上に数式 xy が表示されている場合を紹介する．（ここでは，変数 x と y は一変数とした乗算の式とする）

表 2 数式 xy の多様な記述例
Table 2 xy pattern

例 1	例 2
<mi>x</mi> <mi>y</mi>	<mi>xy</mi>

例 1 および例 2 は，記述された数式が持つ情報として，変数が並んで表示されているだけである．このような数式として不完全な記述でも，数式 xy と表示されてしまうのである．本来の正しい記述の仕方は，以下のように見えない掛け算の情報を記述しなければならない

```
<mi>x</mi>
<mo>&InvisibleTimes;</mo>
<mi>y</mi>
```

（課題 2） 中置記法と前置記法の混在

表現形式では一つの数式を表現するのに，演算子によって中置記法と前置記法が混在する．一方の意味形式では，計算機が理解しやすい前置記法で統一されている．形式変換には，混在した記法から前置記法の変換を行う必要がある．

そもそも，中置記法から前置記法への記述順序の変換は，left-to-right を基本とする計算機の字句解析において難しい問題となる．電卓計算で入力されるような簡単な数式でも，一段階処理で変換を行うことは難しとされる．
次にこれらの課題を克服する方法の提示と各処理を示す．

（課題 1 の対処） 演算子の欠落箇所に自動付加する

演算子情報の欠落箇所を特定するために，数式データをリスト状に繋ぐ方法をとる．これにより，欠落した並びの状態で保持することができる．そして，データの前後の並びで型チェックを行い，欠落が発生している箇所を特定し，見えない掛け算の情報を自動的に付加する．

（課題 2 の対処） 後置記法に変換してから前置記法にする

まず課題 1 の対処のリストを作成する際に，演算子によって線型と階層構造の異なる組み合わせを使って作成する方法を取り入れる．これは数式全体として見かけ上は，中置記法として統一されたように構成するためである．そして，スタックを使ってリストを構成するデータを後置記法に並べ替える処理を行い，スタックに保持する．スタックの後方からデータを取り出しながら木構造を構成し，書き出すという複数段階の処理を踏むことで，意味形式の前置記法の順序にすることが可能となる．

自動変換の各処理

step 0. 変換元ファイル（表現形式）の入力
step 1. 字句解析および意味解析によるデータの取り出し
step 2. リスト連結
step 3. 型チェックと欠落情報の自動付加
step 4. 構文解析と構文木の作成
step 5. 意味形式コードへの書換えと出力

step2 から step4 の処理は，中間表現である．

2.2 字句解析と意味解析によるデータの取り出し

入力は，数式が表現形式で書かれたソースコードファイル（文脈自由文法記述文章含む html 形式）である．字句解析で，MathML の開始を表すインターフェース要素を認識したら意味解析の処理に移り，表現形式のタグや要素を解析して型づけを行い，数式に直接関わるデータの取り出しと格納を始める．データの 1 つ分は，タグの開きと閉じのセットを基準とする（**図 1**）．

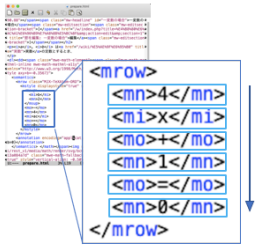

図 1 数式データの認識と取り出し処理
Fig. 1 Lexical Analysis

意味解析の際に，表現形式のタグと要素をそれぞれ解析して型付けを行う．定数や変数の記述では，タグを解析した時点で<mi>であれば変数型，<mn>であれば定数型と型付

けを行うことができる．演算子に関しては，前置記法の演算子はタグから型付けが可能であるが，中置記法の演算子（<mo>タグで挟まれるもの）の場合は，タグに挟まれる要素から型判別をしなくてはならない．要素部分の記述として，1 バイト文字の演算子の記号を使っている．しかし，これだけでは型判別ができない場合がある．実際のソースでは，例えば減算のマイナス記号（'-'）がハイフン記号だったりする．（具体的な例として，本来であれば<mo>-</mo>と表すところを，要素部分を&#x2212，&#x02D7，&#x002D，&#xFF0D，&#xFF63 のように記述している場合がある）．これらは，HTML の実態文字参照（&キーワード;），数値文字参照（&#x10 進又は 16 進コード;）に基づいている．読み込んだトークンを解析して型付けを行うために，これらの文字実体参照もトークンとして予め定義しておく．また，未定義の演算子のトークンは認識しない型である TO_ETC を定義する．

2.3 リスト連結

意味解析で型付けされたデータがオブジェクトデータとして格納された直後に，データをポインタで繋いでリスト状に連結する．それをリスト連結と呼ぶことにする．

表現形式の記法の混在した並びからリスト構造を作成するために，連結の仕方には線型と階層構造の二つの方法を組み合わせる．新たに生成されたオブジェクトデータをリスト連結する際には，リストの先頭からトレースして，該当の場所に連結するということを行なっている．次にそのリスト作成の方法についてそれぞれ説明する．

2.3.1 中置記法の演算子で構成される数式の場合

表現形式で記述された数式が単純に，中置記法しか存在しない場合であれば，新たに生成されたデータは先頭からトレースしてリストの末尾に順次連結するだけで済む．そして，**図 2** のように，リストは線型リスト一本で表すことができる．

数式例：$4x + 1 = 0$

図 2 線型リスト
Fig. 2 Infix notation list

2.3.2 前置記法の演算子を含む数式の場合

前置記法の演算子の場合は，それ自身のデータは線型に連結し，次に生成されるデータ（演算子の引数）を演算子データに対して，分岐した形で引数データを階層的に連結する．階層構造にして引数データを内包することで，数式全体を構成するデータの並びは，見かけ上は中置記法になる．階層構造リストの作成処理については，S 式の記述方法とリスト構造を紹介してから説明する．

S 式

S 式はプログラミング言語の LISP において，数式をリスト構造を直接表すものとして考案された記述方式である．数式例 a*x + b を (+,(*,a,x),b) のように表現され，前置記法である．括弧を使用することで，演算子がそれ自身が持つ引数への適用範囲と，構造的な関係を明示的に表現する．数式が S 式で表現できている状態は，意味構造持っていることと等価の意味を持つ．リスト構造は**図 3** のように表すことができる．

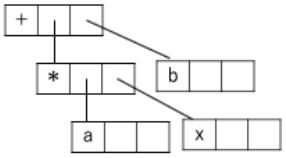

図 3 S 式をリスト構造で表現したもの
Fig. 3 List structure

階層構造リストの作成には，図 3 のようなリスト構造を参考にオブジェクトデータを簡略化した図の表現で行う．

数式例：$\dfrac{1}{x^2+1}$

この数式では，分数と二乗の部分が前置記法の演算子である．先にこの数式の最終的なリストの階層構造を**図 4** に示す．

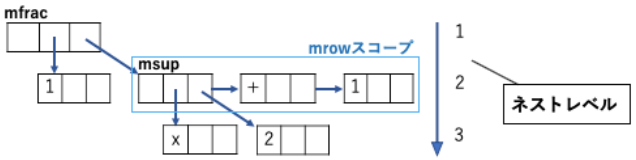

図 4 階層構造
Fig. 4 Prefix notation list

図 4 の mfrac は分数を表すタグで，意味解析の段階で一つのオブジェクトデータとして生成される．そこから分岐して二本の枝分かれとなるのは，分数の演算子が分子と分母の二つの引数を持つことによる．分子に定数 1 のデータが，そして分母に $x^2 + 1$ のデータの線型リストが連結する．この例では，x^2 部分が前置記法演算子の<msup>でさらに分岐する．このように階層を形成し，ネスト（入れ子）の階層の度合いを値にしたものをネストレベルとする．ネストレベルの大きいものは，深い階層を持つリストであることを示す．また，表現形式では引数が多項式の場合に，区切りを示すタグ<mrow>が定義されており，表現形式コードでは分母の式 $x^2 + 1$ を<mrow>タグで挟んで記述する．

このような階層構造のリストを形成する途中段階では，

意味解析の段階で新たに作成されたオブジェクトデータがどこに連結するかを特定するための情報が必要となる．この情報は，三つのスタックを同時に併用することで実現する．それは，前置記法のオブジェクトデータのアドレスを保持するスタック（ADDRESS_STACK），分岐した先が多項式かを判別するためのデータの型を保持するスタック（TYPE_STACK），引数が二つの場合にどちらの引数に連結している状態であるかを示す ON と OFF のスイッチの役割をするスタック（FLAG_STACK）である．スタックのサイズは，ネストレベルの最大定数値 PREFIX_STACK_SIZE で定める．各スタック操作とアルゴリズムの概要を以下に示す．

ADDRESS_STACK の操作

- 前置記法の演算子であれば，オブジェクトデータのアドレスを PUSH する．
- DELETE は，TYPE_STACK の DELETE 処理の際に，最上位のデータの型が mrow 以外の時に行う

TYPE_STACK の操作

- 意味解析で認識した型が，前置記法の演算子および mrow であれば，型を PUSH する．
- 最上位のデータの型に対応する閉じタグが来たら最上位を DELETE する

ADDRESS_FLAG の操作

- 初期状態は ON
- ADDRESS_STACK の最上位のデータの引数 1 の連結と TYPE_STACK の最上位の型を参照し，引数 1 の連結状態であれば ON のまま
- 引数 1 の連結が完了していたら，OFF にする

新たに生成されたオブジェクトデータが連結する分岐元の場所は，ADDRESS_STACK の最上位のポインタを参照し，分岐元データの連結先が引数 1 か引数 2 であるかの判別は，ADDRESS_FLAG を参照する．ON の場合であれば引数 1 であり，OFF であれば引数 2 に連結する．分岐した先が多項式の場合は，分岐先が線型リストになるが，その判別は TYPE_STACK の最上位に mrow の有無で行う．線型リストの生成途中であれば，その末尾に追加する．以上の操作により，階層ごとにリストの連結処理が行われ，階層構造リストが完成する．

前置記法の演算子でそれ自身の引数を階層構造として内包することで，ネストレベル 1 において見かけ上は中置記法であるかのようにすることができる．これにより，記法の混在の課題への対処として前処理の役割をする．

2.4 型チェックと欠落情報の自動付加

次に数式情報に演算子情報の欠落の有無を確認する．

2.4.1 隣接したオブジェクトデータで型が連続する場合

リストを構成するオブジェクトデータの前後の並びで，型を調べる．演算子の欠落が生じている場合は，連続してデータの型が引数のものが連続する形となる．例えば，変数同士の乗算で見えない掛け算が省略されている場合は，変数型であるオブジェクトデータが連続する．また，数字と変数の乗算なども省略されていると認識する．階層構造のリストに対しては，階層ごとに型チェックと

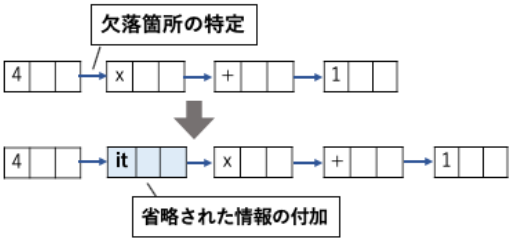

図 5 欠落情報の自動付加のパターン 1
Fig. 5 Append the lack of operational data

2.4.2 複数の変数がまとめて格納された場合

意味解析のデータの取り出しと格納の段階で，要素部分に複数変数が一つにまとまっている場合がある．この場合は変数を一文字と見なしてオブジェクトデータの再構成を行う．処理としては，分解前の変数文字列をコピーし，分解が必要な個数分の空のオブジェクトデータのリストを作成する．空のオブジェクトデータにそれぞれコピーした文字列を一文字ずつペーストする．そして，分解してできたオブジェクトデータ間に見えない掛け算のオブジェクトデータを挿入して，リストをつなぐことにより**図 6** のようにリストの再構成を行う．

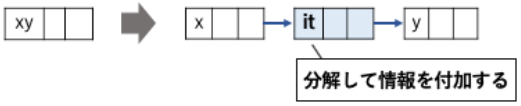

図 6 欠落情報の自動付加のパターン 2
Fig. 6 Append the lack of operational data

ここまでの処理で，意味形式の特徴である演算子の欠落のない完全な状態に変換する処理を行った．次に，数式として意味構造を持たせる構文解析の処理に移る．前処理で作成したリストの全体構造というのは，数式を構成する演算子の種類によって，一つのリストでも線型リストと階層構造を持ったリストが混在した構造で構成されていた．このリストに対し，演算子の優先順位をひゅう元するのに，各優先順位に対応する優先度因子を導入する．優先順位の高いものから PR1，PR2，PR3 とし，演算子の型を優先度因子を対応づける．前置記法の演算子は引数と同じ対応をするため，優先度に考慮しない（**表 3**）

表 3　演算子の結合優先度
Table 3　priority of operator

優先度	演算子
PR1	乗算
PR2	加算，減算
PR3	等号 その他の演算子

2.5　構文解析と構文木の作成

次に変換の中間コードとなる構文木作成のための準備を行う．リストで中置記法に統一されたリストのデータの並びを後置記法の順序へ変換していく．ここでは 2 つのスタックを用いる．演算子情報を保持するスタック（OUT スタック）と構文木で必要な出力用のスタック（OPE スタック）である．

後置記法への変換の基本操作

リストの先頭からデータを参照し，リストの末尾に到達するまで以下の処理を繰り返す．

(I) 参照したデータの型が定数型もしくは変数型であれば，OUT スタックの最上位に PUSH する
(II) 参照したデータの型が中置記法の演算子型であれば，OPE スタックの操作に移る
(III) リストの末尾に到達し参照するデータがないならば，OPE スタックのデータを上位から POP して OUT スタックに PUSH を繰り返す
(IV) OPE スタックが空になったら，処理を終了する

OPE スタックの操作

- スタックが空であれば PUSH する
- スタックが空でない
  - (i) スタックの最上位データの優先度が参照データの優先度よりも大きいまたは等しい場合は，最上位を POP して OUT スタックへ PUSH して参照データを OPE スタックに PUSH する
  - (ii) 最上位のデータの優先度の方が小さい場合は，参照データを OUT スタックへ PUSH する

上記は中置記法から後置記法への基本的なアルゴリズムで，階層構造のリストに対しては，ネストレベルの大きいリストから上記の基本操作を階層ごとに適用し，ネストレベルを一つずつ下げて記法の変換を繰り返す．そして最終的にネストレベルが一番小さいリストが OUT スタックに保持される．つのスタックを使って，演算子の結合力の優先度を反映して後置記法の順に並べる．（図 **7**）

2.6　意味形式のコード生成

構文木から意味形式の前置記法順序で書き出すためには，構文木のルート（根）からプレオーダー（行きがけ順）で，左の部分木と右の部分木を順にトラバースして再帰的な続きを行う．各部分木の節となるノードを訪問したら，ノードのデータの型と対応した意味形式の開きタグを出力する．この時，演算子の型であれば演算子のスコープをを表す<apply>をノードのデータをプリントする前に出力する．そして，左のノードに移り，同じ処理を葉に到達するまで繰り返す．

葉ノードは定数型または変数型の終端型であるので，型に対応する意味形式タグとデータに格納された要素をプリントしてから閉じタグをプリントする．単項演算子は，データにマイナスの記号と変数または定数がセットで格納されているので，マイナス記号があれば演算子のスコープタグと単項演算のマイナスであることを意味形式タグ<minus/>で要素を挟んでプリントする．これにより，単項演算子の結合の優先度の高さを考慮した出力ができる．右の部分木でも上記の処理を再帰的に行い，構文木を下から上向きにトラバースするときに，演算子のスコープの終了を示すタグ</apply>をプリントする．また，出力されたコードが視覚的に分かりやすくなるように，インデンテーションをつける．これは，ネストレベルに応じた数のタブを出力することにより実現している．

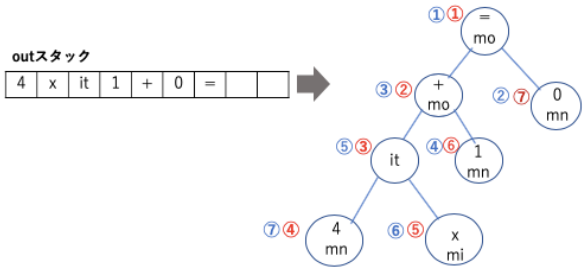

図 7　構文木の作成および意味形式コードの出力
Fig. 7　Making Tree and Print code contents markup format

図 7 において，青の数字が構文木の作成順序で，赤い数字が意味形式のプレオーダーで意味形式コードを出力する順番になる．また，この構文木は数式 $4x+1=0$ を表しており，意味形式に変換した結果のコードは**図 8** に示す．

3.　結果

3.1　コンバータツールの物理構成

本研究で実装したコンバータツールは，C 言語でマルチモジュールで構成している．プログラム構成は，メインモジュール 1 つと，サブモジュール 6 つである．メインモジュールは，変換したい元ファイル（表現形式で記述された html 形式ファイル）を入力として受け取り，方法で示した変換処理の step1 から 5 の各処理を，サブモジュールを呼び出して実行する．処理の過程は，一つの数式に対して一つのリストを作成し，変換して意味形式コードを出力したら，字句解析の処理を続ける．数式の数だけこれを繰

り返し処理する．ユーザは変換元ファイルだけを意識すればよいように実行コマンドは簡易な設計にしている．

コンパイルコマンド

```
make
```

実行コマンド

```
./Converter [変換元ファイル]
```

エイリアス設定をすれば

```
Converter [変換元ファイル]
```

意味形式に変換された結果のコードは，output ディレクトリ直下に出力ファイル out.html(html 形式) として作成される．変換元となる入力ファイルに，複数の数式が記述されていても変換が可能である．

3.2 ツールによる変換の結果

数式：$4x + 1 = 0$

変換結果を図 8 で示す．この例は，中置記法のみの演算子からなる線型リストを構成する数式で，<mn>4</mn><mi>x</mi> の部分で，欠落情報（見えない掛け算）が発生した表現形式のソースから変換したものになる．型チェックと欠落情報の自動付加によって，出力された意味形式コードでは，掛け算の演算子<times/>が入ってることが確認できる．そして，構文木の深さに対応したインデンテーションがされていることもわかる．

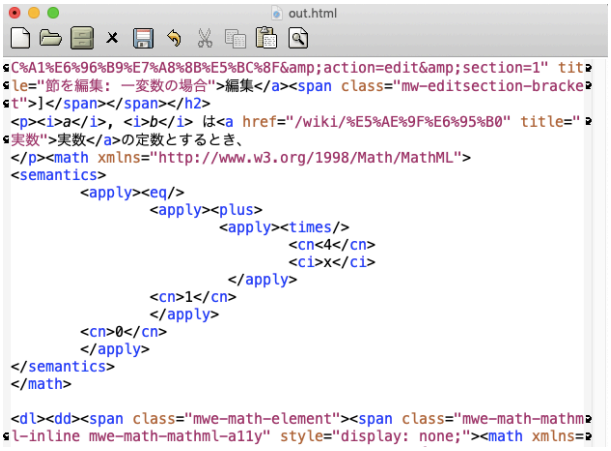

図 8 数式 $4x + 1 = 0$ の変換結果

Fig. 8 Content Markup Format1

数式：$b^2 - 4ac > 0$

変換結果を図 9 で示す．この例では，前置記法と中置記法の演算子が混在する数式例である．変換の過程で，ネストレベル 3 の階層構造を形成したリストができる例である．

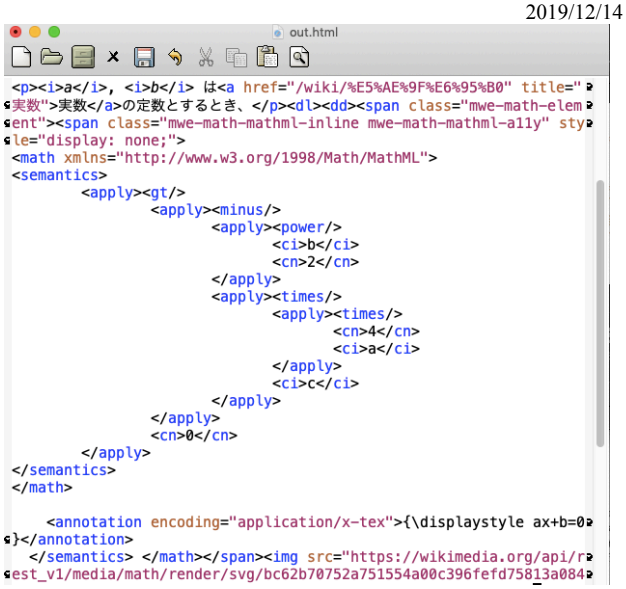

図 9 数式 $b^2 - 4ac > 0$ の変換結果

Fig. 9 Content Markup Format2

3.2.1 結果の分析

変換結果を表現形式の入力ソースと対応づけて，部分的に分析していく．表現形式における単項演算の記述は，青枠で囲っているような記述の仕方をする．矢印の先に示したコードは変換後のコードである．マイナス記号が<minus/>と解釈され，演算子の適用スコープのタグ<apply>も正しく出力されていることが確認できる．（図 10）

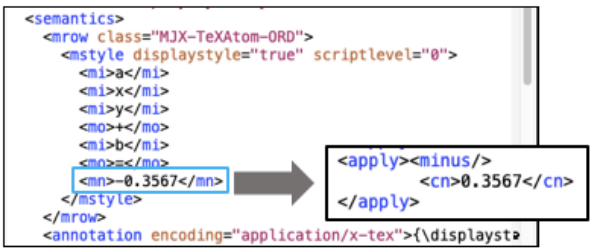

図 10 結果コードの分析 1

Fig. 10 analysis

次の例は，欠落情報の付加のところで，複数の変数がまとめて格納された場合の変換結果である．表現形式の入力ソースにおいて，<mi>xyz</mi>と 3 変数が一つにまとめて記述された場合の変換結果である．これは，3 つの変数がそれぞれ分解されて，変数の個数分のオブジェクトデータを新たに生成して，一変数ずつ格納した上で，欠落した掛け算の演算子を付加したものである．結果の出力では，各変数ごとに掛け算が適用されていることが確認できる．（図 11）

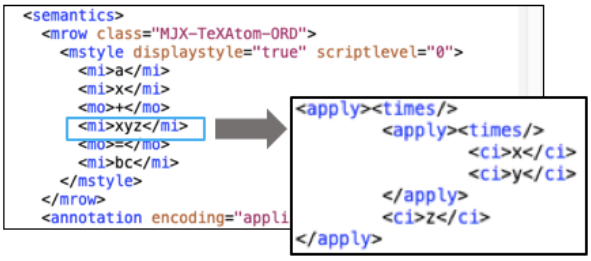

図 11　結果コードの分析 2
Fig. 11　analysis2

4. まとめ

4.1 変換の対象となる数式

本研究で実装した機能は，MathML の自動変換のカーネル部分である．自動変換が可能な数式は，次の演算子を組み合わせたものが可能になる．加減乗計算，分数式，指数式および指数関数，根号計算，不等号，等号関係，単項演算．リストのネストレベルは，PREFIX_STACK_SIZE で定義しており，数字を変えれば深い階層構造も可能となる．

4.2 研究の意義

表現形式によって表示された数式は計算処理ができない状態であった．それを他方の意味形式という数式に意味構造を持たせた記述が可能な意味形式へ自動変換することで，数式データの再利用性を高めることができた．

また，表現形式のソースにおいて，記述された内容が不完全な数式でも欠落情報を自動付加する方法を組み合わせることによって，自動変換を実現できたことの意義は大きいと考える．また，将来的にツールが改良され，ブラウザにプラグインすることで，関連研究の実用化を難しくしていた制約のハードルを下げる役割をすると予想する．

4.3 今後の課題

本研究のツールの前提として，変数は一文字である．入力ソースが場合によっては，複数文字で構成された一変数が定義されている可能性もあるが，これらは一文字ずつに分解して変換されてしまう．このような場合でも一意な変換を可能にする方法として，数式の前後に出てくる文章を文書ベクトル化し，パターン認識技術を取り入れて変数の文字数を認識させることが考えられる．これは，括弧を使用するベクトルや行列などの判別に有効であると考える．

表現形式で記述が省略されてしまう演算子には，帯分数と関数適用がある．これらは見えない掛け算同様に省略しても表示できてしまう演算子情報である．関数適用に関しては，例えば三角関数を例にすると，正しい表現形式では<mi>sin</mi><mo>⁡</mo><mi>x</mi>のように記述する．

しかし，実際のソースでは，<mo>ApplyFunction;</mo>を省略して記述されている場合が多い．これには，要素の sin を解析して sin 型を認識して階層構造（引数一つ）リストを構築すれば変換可能なのではないかと推測する．

5. 技術展望

- 数式処理ソフトでの計算処理
- 数式技術データベースの作成
- 数式検索技術の実用化
- 学習支援システムの実用化
- ユーザの入力に対して動的な解答を生成する数学学習支援ソフトの開発
- 視覚障がい者の数学学習における音声ソフトによる数式の正確な読み上げの実現

参考文献

[1] W3C Mathematical Markup Language，入手先 ⟨https://www.w3.org/TR/MathML3/⟩ (2019.11.18).
[2] Wolfram Language Documentation，入手先 ⟨https://reference.wolfram.com/language/XML/⟩ (2019.11.17).
[3] XML，入手先 ⟨http://www.w3.org.XML/⟩ (2019.11.13)
[4] 朝日新聞デジタル，「年複数回に疑問も センター後継試験」入手先 ⟨https://www.asahi.com/⟩ (2019.8.16)
[5] 等ロボは社会に何をもたらすのか，入手先 ⟨https://www.nii.ac.jp/about/upload/⟩ (2019.8.15)
[6] 大武信之，秋山富美子：Web 上における数式表現の意味形式記述への半自動変換システム．国際経営・文化研究，Vol7 No.1，January 2013.
[7] 石山寿子，高野文子，佐藤浩史，原俊介，大武信之：XML における数式の表現形式から意味形式への変換，電子情報処理学会研究技術研究報告．ET，教育工学，vol.101，No，506，pp.23-30
[8] 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，2007-DBS-142，pp.55-59，2007
[9] 宮崎佳典，林佳樹：MathML 記述の数式を対象とした木構造に基づく検索システムの提案・開発．数理解析研究所講究録，第 1780 巻，pp.130-140，2012.
[10] 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 (1990).
[11] 湯淺太一：コンパイラ，昭晃堂 (2004).
[12] Harold Abelson，Gerald Jay Sussman：計算機プログラムの構造と解釈，MIT 出版 (1985).