数理解析研究所講究録
第1572巻 2007年 142-155

MathMLライブラリの開発と今後の展開について

黒田 拓
KURODA HIRAKU *
北海道大学大学院理学院数学専攻
DEPARTMENT OF MATHEMATICS, HOKKAIDO UNIVERSITY

1 MathMLライブラリの概要

本ライブラリはLaTeXの記法に従って記述された数式をMathML[3]に変換する処理を提供するものである。オブジェクト指向スクリプト言語Ruby[5]によって作成されており、変換処理そのものを提供する`math_ml.rb`と、ライブラリの使用を簡単にするための補助モジュールで構成されている。

1.1 math_ml.rb

1.1.1 math_ml.rbで定義されるクラス

`math_ml.rb`では、LaTeXの記法に従って記述された数式をあらわす文字列(以下、単に「LaTeX数式」とする)をMathMLに変換する為に、以下のクラスが`MathML`モジュール内に定義されている。

MathMLの各要素に対応するクラス

`MathML::XMLElement`クラスを基底クラスとして、`mn`要素に対応する`MathML::Number`クラス、`mi`要素に対応する`MathML::Identifier`クラスなどがある。LaTeX数式からの変換結果に使用される要素に対応するクラスだけが用意されており、全ての要素に対応するクラスが揃っているわけではない。MathMLのいくつかの要素がその内容として他の要素をとることが出来るように、これらの要素クラスのインスタンスも他の要素クラスのインスタンスを内容としてとることが出来る。

これらのクラスのインスタンスから、`MathML::XMLElement#to_s`メソッドによってMathMLの文字列を得ることが出来る。

以下は、これらのクラスによってMathMLを生成する例である。

```
#!/usr/bin/ruby
require "math_ml"

mm = MathML::Math.new(false)
n1 = MathML::Number.new << "1"
n2 = MathML::Number.new << "2"
frac = MathML::Frac.new(n1, n2)
```

*hiraku@math.sci.hokudai.ac.jp

```
mm << frac
puts mm.to_s
```

ここでは、MathML::Number クラスの 2 つのインスタンスによって分子の 1 と分母の 2 をあらわす要素を作り、それを MathML::Frac のパラメータとして与えることで mfrac 要素を作成している。更にそれを MathML::Math クラスのインスタンスに内容とすることで、最終的な MathML を作成している。

このスクリプトを実行すると次の MathML が得られる。

```
<math xmlns='http://www.w3.org/1998/Math/MathML' display='inline'><mfrac>
 <mn>1</mn>
 <mn>2</mn>
</mfrac></math>
```

実際の変換処理では、与えられた LaTeX 数式に従ってこれらの要素クラスのインスタンスの集まりを作成していく。

以下に挙げる LaTeX 数式から MathML への変換処理を行う為のクラスは、MathML モジュール内の MathML::LaTeX モジュール内で定義されている。

MathML::LaTeX::Scanner

LaTeX 数式の読み取りを補助するクラスであり、Ruby の標準添付ライブラリである StringScanner クラスのサブクラスである。処理中の文字列が LaTeX 数式を構成する要素 (英数字または記号 1 文字、\ で始まり複数の文字からなる命令、{...} で囲まれた複数の要素) のいずれであるかを判定したり、ひとまとめに取り出すメソッドが用意されている。

MathML::LaTeX::Macro

LaTeX のマクロである\newcommand 命令や\newenvironment 命令を処理するクラスである。マクロによって定義される新しい命令や環境の名前と、その内容とを保持し、与えられたパラメータと組み合わせて既存の LaTeX 数式に展開する処理を提供する。

MathML::LaTeX::Parser

LaTeX 数式から MathML への変換を実際に行う処理を提供するクラスである。このクラスのインスタンスは MathML::LaTeX::Macro クラスのインスタンスを 1 つ保持しており、これに必要に応じてマクロを登録した上でソースとなる LaTeX 数式を与えると、変換結果となる MathML を MathML::Math クラスのインスタンスとして返す。

具体的な使用方法は以下のようになる。

```
#!/usr/bin/ruby
require "math_ml"

lp = MathML::LaTeX::Parser.new
lp.macro.parse('\newcommand{\R}{\mathbb{R}}')
lp.macro.parse('\newcommand{\cbrt}[2][3]{\sqrt[#1]{#2}}')
mm = lp.parse('\cbrt{\frac12} \in \R')

puts mm.to_s
```

MathML::LaTeX::Parser クラスのインスタンスを作成し、マクロ \newcommand{\R}{\mathbb{R}}と \newcommand{\cbrt}[2][3]{\sqrt[#1]{#2}}を登録した上で、数式 $\sqrt[3]{\frac{1}{2}}$ をあらわす LaTeX 数式 \cbrt{\frac12} \in \R の変換を行っている。\cbrt 命令では、オプションパラメータを使用している。

このスクリプトは以下の MathML を出力する。

```
<math xmlns='http://www.w3.org/1998/Math/MathML' display='inline'>
 <mroot>
  <mrow><mfrac>
   <mn>1</mn>
   <mn>2</mn>
  </mfrac></mrow>
  <mn>3</mn>
 </mroot>
 <mo>&in;</mo>
 <mrow><mrow><mi>&Ropf;</mi></mrow></mrow>
</math>
```

その他

他に、エラー処理の為の例外クラスや、処理に使用する定数を定義したモジュールなどが MathML モジュール内で定義されている。

1.1.2 MathML::LaTeX::Parser クラスが扱う LaTeX 数式

math_ml.rb が提供する変換処理では、以下の LaTeX 数式を扱うことが出来る。

数式構造をあらわす命令、記法 \sum_a^b や\int_a^b といった上付き、下付きの記法、分数を表す\frac 命令、根号をあらわす\sqrt、行列などに用いられる array 環境や matrix 環境などを扱うことが出来る。array 環境では罫線なども LaTeX と同様の記法で問題なく使用できる。

フォント切り替え \mathit 命令、\mathrm 命令、\mathbf 命令、\mathbb 命令を使うことでフォントの切り替えを行うことが出来る。

各種記号 \alpha などのギリシャ文字や\sum などの演算記号などを始めとして、LaTeX 標準の記号命令の他に ams symb パッケージと ams fonts パッケージで定義されている記号命令を使うことが出来る。これらの命令は LaTeX のソースファイルからスクリプトによって自動的に抽出され、math_ml.rb に組み込まれている。

1.1.3 変換処理の拡張

本ライブラリでは前述した通り MathML::LaTeX::Parser#macro メソッドによって参照できる MathML::LaTeX::Macro クラスのインスタンスに、\newcommand 命令や\newenvironment 命令によるマクロを登録して使用することが出来る。LaTeX と同様にオプションを使用することも出来る。

また、既存の命令や環境の組み合わせで表現できない数式を記述して MathML に変換するために、独自の命令や環境の変換処理そのものを Ruby で作成して MathML:LaTeX::Parser の変換処理に追加することが出来る。

次の例では立方根 $\sqrt[3]{\cdots}$ をあらわす\cbrt 命令の変換処理を直接作成し、MathML::LaTeX::Parser のインスタンスに登録して使用している。\cbrt 命令自体は、\newcommand{\cbrt}[1]{\sqrt[3]{#1}}といったマクロで定義できるが、これを Ruby によって直接変換するためのスクリプトは、ライブラリが提供するメソッドによって以下のように記述できる。

```
#!/usr/bin/ruby
require "math_ml"

module CubeRoot
   def cmd_cbrt
      cb = MathML::Number.new
      cb << "3"

      r = MathML::Root.new(cb, parse_any)
   end
end

lp = MathML::LaTeX::Parser.new
lp.add_plugin(CubeRoot)
lp.add_commands("cbrt")

puts lp.parse('\cbrt{x}').to_s
```

r = MathML::Root.new(cb, parse_any) では、mroot 要素の 1 つ目の子要素 (何乗根であるかを表す数式) として<mn>3</mn>を指定し、2 つ目の子要素 (根号の中の数式) に MathM::LaTeX::parser#parse_any メソッドで得られた数式を指定している。MathM::LaTeX::parser#parse_any メソッドは、変換元となる LaTeX 数式の処理中の命令の後に続く LaTeX 数式の任意の要素を 1 つ MathML に変換したものを返すメソッドである。

これを実行すると、次の出力が得られる。

```
<math xmlns='http://www.w3.org/1998/Math/MathML' display='inline'><mroot>
 <mrow><mi>x</mi></mrow>
 <mn>3</mn>
</mroot></math>
```

1.2 補助モジュール

本ライブラリには、変換処理の呼び出しを補助するために、以下の 2 つのモジュールが用意されている。

1.2.1 MathML::String モジュールによる String クラスの拡張

MathML::String モジュールは、Ruby の組み込み文字列クラスである String クラスを拡張し、LaTeX 数式を直接 MathML へ変換する機能を提供する。require "math_ml/string"としてモジュールの読み込みを行うだけで、Ruby の String クラスに to_mathml メソッドが追加され、LaTeX 数式の文字列を直接 MathML に変換できるようになる。以下がその使用例である。

```
#!/usr/bin/ruby
require "math_ml/string"

puts '\frac12'.to_mathml.to_s
```

MathML::LaTeX::Parser のインスタンス作成などを行うことなく、LaTeX 数式の文字列から直接 MathML への変換を行っている。出力は以下の通り。

```
<math xmlns='http://www.w3.org/1998/Math/MathML' display='inline'><mfrac>
 <mn>1</mn>
 <mn>2</mn>
</mfrac></math>
```

String#to_mathml メソッドでの変換に使用する MathML::LaTeX::Parser クラスのインスタンスは、MathML::String#mathml_latex_parser メソッドにより参照できる。例えば、マクロを登録して String#to_mathml の変換でそのマクロを使うためには、以下のように処理する。

```
#!/usr/bin/ruby
require "math_ml/string"
MathML::String.mathml_latex_parser.macro.parse('\newcommand{\R}{\mathbb{R}}')
puts 'x\in\R'.to_mathml
```

MathML::String.mathml_latex_parser メソッドにより、String#to_mathml での変換に使用する MathML::LaTeX::Parser クラスのインスタンスを参照し、マクロを登録している。出力は以下の通り。

```
<math xmlns='http://www.w3.org/1998/Math/MathML' display='inline'>
 <mi>x</mi>
 <mo>&in;</mo>
 <mrow><mrow><mi>&Ropf;</mi></mrow></mrow>
</math>
```

1.2.2 MathML::Util::SimpleLaTeX クラス

MathML::LaTeX::Parser クラスが提供する変換処理は LaTeX 数式のみからなる文字列を MathML に変換するものである。しかし、実際のプログラムでは LaTeX 数式を含む文書が対象となることが多く、その場合は変換に際して以下のような手順を踏む必要がある。

1. 文中から LaTeX 数式を抽出
2. 抽出した数式を MathML に変換
3. 変換結果である MathML を元の文に戻す

また、元の文書自体に何らかの加工を行う場合も多く、その際は文書の加工に LaTeX 数式や変換後の MathML を巻き込まないように配慮する必要がある。

このような LaTeX 数式を含む文書に対して、数式は MathML へ、本文は他の形式へ変換する場合に、数式と本文とを切り分けて処理するためのクラスが MathML::Util::SimpleLaTeX である。

このクラスに関しては後述の HikiDoc との併用で解説する。

2 MathMLライブラリの応用

2.1 ワンライナーでの利用

MathML::String による String クラスの拡張を使うことで、LaTeX 数式から MathML を得るプログラムを1行で作成することが出来る。

例えば、コンソールで次の通りに入力することで、標準入力から入力された LaTeX 数式を1行ずつ MathML に変換するプログラムとして動作する。

```
~/libmathml/$ ruby -r math_ml/string -e "$<.each{|s|puts s.to_mathml}"
f(x)=ax^2+bx+c
<math xmlns='http://www.w3.org/1998/Math/MathML' display='inline'>
 <mi>f</mi>
 <mo>(</mo>
 <mi>x</mi>
 <mo>)</mo>
 <mo>=</mo>
 <mi>a</mi>
 <msup>
  <mi>x</mi>
  <mn>2</mn>
 </msup>
 <mo>+</mo>
 <mi>b</mi>
 <mi>x</mi>
 <mo>+</mo>
 <mi>c</mi>
</math>
```

2行目はプログラムに入力する LaTeX 数式であり、その次の行は入力した LaTeX 数式を MathML に変換したものである。

2.2 HikiDoc との併用

HikiDoc[4] はいわゆる Wiki の記法に従って記述された文書を HTML や XHTML に変換する整形プログラムであり、"かずひこ" 氏によって作成された。例えば、以下のようなスクリプトで使用する。

```
#!/usr/bin/ruby
require "math_ml/util"
require "hikidoc"

txt = $<.read
puts HikiDoc.to_html(txt)
```

これは起動時に指定したファイルからソースとなるテキストを読み込み、それを HTML に変換するスクリプトである。ソーステキストには次のような内容を適当な名前のファイル名をつけて作成する。

```
!HikiDoc のサンプル
文章は空行で区切ることで段落が改められる。

箇条書きは、次のように行頭に*をつける事で記述できる。
*item1
*item2
**item2-1
```

このテキストファイルを test.txt という名前で保存し、スクリプト自体は hd.rb という名前で作成していた場合、コンソールから次のようにして変換を実行できる。

```
~/work/$ hd.rb test.txt
<h1>HikiDoc のサンプル</h1>
<p>文章は空行で区切ることで段落が改められる。</p>
<p>箇条書きは、次のように行頭に*をつける事で記述できる。</p>
<ul>
<li>item1</li>
<li>item2<ul>
<li>item2-1</li>
</ul></li>
</ul>
```

2 行目以下が変換結果の出力である。

このように作成したスクリプトを LaTeX 数式に対応させる場合、次のように変更して MathML::Util::SimpleLaTeX を使用する。

```
#!/usr/bin/ruby
require "math_ml/util"
require "hikidoc"

txt = $<.read

sl = MathML::Util::SimpleLaTeX.new(:delimiter=> "@")
encoded, data = sl.encode(txt)

html = HikiDoc.to_html(encoded)

puts sl.decode(html, data)
```

このスクリプトを使用して、次のテキストファイルを処理する場合を考える。

```
!数式混じりの文書
方程式$ax^2+bx+c=0$の解は公式
\[
x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}
\]
により得られる。
```

まず、MathML::Util::SimpleLaTeX#encode メソッドにより、数式が抽出されて次のように置き換えられる。

```
!数式混じりの文書
方程式@m0@の解は公式
@d0@
により得られる。
```

数式部分が@で囲まれた短い記号列に置換されており、これは変数 encoded に代入されている。また、抽出された LaTeX 数式を MathML に変換したものが変数 data に格納される。

元の文字列ではなくこの encoded を HikiDoc#to_html メソッドに与えて変換を行うと、次の XHTML が得られる。

```
<h1>数式混じりの文書</h1>
<p>方程式@m0@の解は公式
@d0@
により得られる。</p>
```

最後に、MathML::Util::SimpleLaTeX#decode メソッドによって MathML を戻すことで最終的な MathML を含む XHTML 文書が得られる。

```
<h1>数式混じりの文書</h1>
<p>方程式<math xmlns='http://www.w3.org/1998/Math/MathML' display='inline'>
 <mi>a</mi>
 <msup>
  <mi>x</mi>
  <mn>2</mn>
 </msup>
 <mo>+</mo>
 <mi>b</mi>
 <mi>x</mi>
 <mo>+</mo>
 <mi>c</mi>
 <mo>=</mo>
 <mn>0</mn>
</math>の解は公式
<math xmlns='http://www.w3.org/1998/Math/MathML' display='block'>
 <mi>x</mi>
 <mo>=</mo>
 <mfrac>
  <mrow>
   <mo>-</mo>
   <mi>b</mi>
   <mo>&pm;</mo>
   <msqrt><mrow>
    <msup>
```

```
      <mi>b</mi>
      <mn>2</mn>
     </msup>
     <mo>-</mo>
     <mn>4</mn>
     <mi>a</mi>
     <mi>c</mi>
    </mrow></msqrt>
   </mrow>
   <mrow>
    <mn>2</mn>
    <mi>a</mi>
   </mrow>
  </mfrac>
</math>
```

により得られる。</p>

2.3 RubyOnRails での利用

Ruby on Rails[1](以下、'Rails') は、リレーショナルデータベースと Ruby との組み合わせで Web アプリケーションを作成するために開発されたフレームワークである。

Rails が出力するページデータは XHTML であるので、簡単な修正で MathML を含む XHTML に変更できる。これに MathML ライブラリと組み合わせることで、数式を含む文書を MathML によって扱う Web アプリケーションを作成することが出来る。

本節では「数式メモ帳」とでも言うべき簡単なアプリケーションの作成を通して、Rails で MathML ライブラリの使用する際の注意点をまとめる。

2.3.1 準備

まず、作成するアプリケーションのプロジェクトを作成する。

```
~/$ rails math_memo
~/$ cd math_memo
```

プロジェクトを作成する際に使用するデータベースの種類などを指定する場合は、`rails` コマンドに必要に応じてパラメータを与えればよい。

次に、メモ帳のモデルを定義する。今回作成するモデルは、単純にメモの内容のみを持つものとする。`script/generate` コマンドで雛形を生成し、

```
~/math_memo$ ./script/generate model Memo
```

ついで、`db/migrate/001_create_memos.rb` を以下のように編集してモデルが持つデータを定義する。

```
class CreateMemos < ActiveRecord::Migration
  def self.up
```

```
    create_table :memos do |t|
      t.column :brief, :text
    end
  end

  def self.down
    drop_table :memos
  end
end
```

定義したデータ構造をデータベースに反映し、Web ページからデータを操作する為の雛形を作成する。

```
~/math_memo$ rake db:migrate
~/math_memo$ ./script/generate scaffold Memo Memo
```

ここまでで既に、テキストデータを作成・保存・編集する Web アプリケーションが出来上がっている。

```
~/math_memo$ ./script/server
```

とすることでテスト用の Web サーバーが起動するので、Web ブラウザで `http://192.168.11.100:3000/memo` にアクセスすれば (`192.168.11.100` はプロジェクトファイルを保存しテスト用のサーバーを起動しているホストの IP アドレス) メモの作成画面を見ることが出来る。

2.3.2 MathML ライブラリの追加

作成した Rails アプリケーションから MathML ライブラリを利用するためには、`math_ml.rb` および `math_ml/` ディレクトリ以下のファイルを `lib/` ディレクトリに配置すればよい。配置後は以下のようになる。

```
lib
|-- math_ml
|   \-- util.rb
\-- math_ml.rb
```

このように配置すると、rails からは `require "math_ml"` などの宣言無しに MathML ライブラリを使用することが出来る。

2.3.3 出力されるデータの形式

Rails が生成するページデータは基本的に XHTML であるが、ドキュメントタイプとして MathML が使えない XHTML のみのものが指定されており、HTTP のレスポンスヘッダで指定される Content-Type も `text/html` であるため、MathML をページ中に含めるためにこれらを変更する必要がある。

ドキュメントタイプは `app/views/layouts/` ディレクトリに保存される `.rhtml` ファイルで宣言される。今回は `app/views/layouts/memo.rhtml` を次のように修正し、XHTML に MathML を併用できるようにする。

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1 plus MathML 2.0//EN" "http://www.w3.org/Math/DTD/math
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ja">
<head>
  <meta http-equiv="content-type" content="text/html;charset=UTF-8" />
  <title>Memo: <%= controller.action_name %></title>
  <%= stylesheet_link_tag 'scaffold' %>
</head>
<body>

<p style="color: green"><%= flash[:notice] %></p>

<%= yield  %>

</body>
</html>
```

レスポンスヘッダの Content-Type は、ApplicationController.after_filter で指定する。具体的には、app/controller/application.rb に次のように ApplicationController#set_content_type メソッドを追加し、ApplicationController.after_filter メソッドで登録する。

```
class ApplicationController < ActionController::Base
  after_filter :set_content_type
  def set_content_type
    if /Gecko/ =~ request.env["HTTP_USER_AGENT"]
      headers["Content-Type"] = "application/xhtml+xml; charset=UTF-8"
    end
  end
end
```

今回は、ブラウザがその種類に応じてサーバーに送ってくる HTTP_USER_AGENT の情報を利用し、MathML に対応しているブラウザが (Firefox[2] など) である場合にのみ Content-Type を変更するようにしている。

2.3.4 メモの数式の MathML 化

メモの内容は、app/view/memo/show.rhtml などで、Memo#brief メソッドによって参照される。

そこで、与えられた文字列の数式部分を MathML に置き換えたものを返す ApplicationHelper#with_mathml メソッドを作成し、これを使用して Memo#brief の数式部分を MathML に変換したものを出力するようにする。

まず、app/helper/application.rb を次のように編集する。

```
# Methods added to this helper will be available to all templates in the application.
module ApplicationHelper
  def with_mathml(str)
    if /Gecko/ =~ request.env["HTTP_USER_AGENT"]
```

```
        sl = MathML::Util::SimpleLaTeX.new
        encoded, data = sl.encode(str)
        sl.decode(encoded, data)
      else
        str
      end
    end
  end
```

先ほどの Content-Type と同様、ブラウザが MathML に対応する場合にのみ数式部分を MathML に変換している。また文中の数式の抽出のために、`MathML::Util::SimpleLaTeX` を使用している。

次に、数式部分を変換した文書を出力するように、`app/view/memo/show.rhtml` を次のように編集する。

```
<pre><%= with_mathml @memo.brief %></pre>

<%= link_to 'Edit', :action => 'edit', :id => @memo %> |
<%= link_to 'Back', :action => 'list' %>
```

ここでは pre 要素を使って、入力された改行位置で改行して表示されるようにしている。

これで我々が作成したメモ帳アプリケーションは、LaTeX 数式をメモに含むと表示の際には MathML を使った数式として描画されるようになった。図 1 は作成した数式メモ帳を使用している様子である。

以上をまとめると、Ruby on Rails で MathML による数式を扱うためには、標準的な Rails のアプリケーション作成作業に加えて以下の手順が必要になる。

- MathML ライブラリの配置
- ドキュメントタイプの変更
- Content-Type の変更
- LaTeX 数式部分の変換処理の追加

このようにして作成され実際に使用されている Web アプリケーションとして、Center for Nonlinear Science at Hokkaido University のサイト[1)]がある。ここでは、セミナー案内のアブストラクトに数式が含まれている場合、MathML に変換してブラウザに出力している。

3 おわりに

本 MathML ライブラリを使用することで、LaTeX 数式を MathML に変換するプログラムを作成したり、既存のプログラムに LaTeX 数式を MathML に変換して扱う機能を追加することが容易になる。

次の計画として、現在、数式を扱える総合的な CMS を開発中である。Matherial[2)] と名づけたこのシステムは Wiki を基本として複数人がブログやプレゼンの資料、論文の下書き等をオンラインで作成することが出来るものにする予定である。

---

1) http://nls.es.hokudai.ac.jp/

2) http://www.matherial.org/

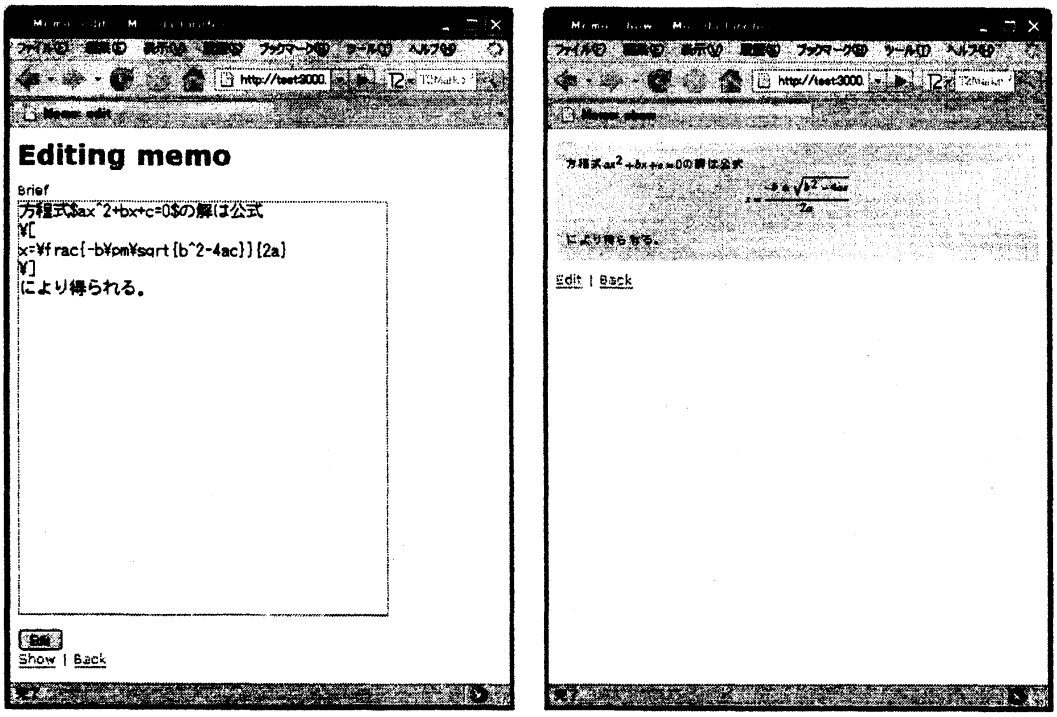

図 1: 数学メモの動作の様子

左:LaTeX 数式を含むメモを入力している。右:MathML に変換された数式が Firefox によって適切に描画されている。

参 考 文 献

[1] David Heinemeier Hansson. Ruby on rails.
http://www.rubyonrails.org/.

[2] Mozilla Japan. Firefox.
http://www.mozilla-japan.org/products/firefox/.

[3] W3C. Mathematical Markup Language (MathML) Version 2.0 (Second Edition).
http://www.w3.org/TR/2003/REC-MathML2-20031021/.

[4] かずひこ. Hikidoc.
http://projects.netlab.jp/hikidoc/?FrontPage.ja.

[5] まつもとゆきひろ. オブジェクト指向スクリプト言語 ruby.
http://www.ruby-lang.org/ja/.