REDUCE-MathML Interface

Luis Alvarez-Sobreviela
alvarez@zib.de

Konrad–Zuse–Zentrum für Informationstechnik Berlin
Takustraße 7
D-14195 Berlin-Dahlem

November 10, 1998

1 Introduction

MathML is intended to facilitate the use and re-use of mathematical and scientific content on the Web, and for other applications such as computer algebra systems. For this reason we believe MathML is an important step for the scientific community considering the widespread use of the Internet. We found necessary to include REDUCE in this trend, and developed the MathML-REDUCE interface.

The MathML interface for REDUCE provides an easy to use series of commands, allowing it to evaluate and output MathML. This manual is therefore intended to give the user a good guideline on how to make of it a proper use.

The principal features of this package can be resumed as:

- Evaluation of MathML code. Allows REDUCE to parse MathML expressions and evaluate them.
- Generation of MathML compliant code. Provides the printing of REDUCE expressions in MathML source code, to be used directly in web page production.

We assume that the reader is familiar with MathML. If not, the specification[1] is available at:

`http://www.w3.org/TR/WD-math/`

2 Getting Started

2.1 Loading

The MathML-REDUCE interface package is under the name **`mathml`**, and so is loaded by supplying `load mathml;`.

[1] This specification is subject to change, since it is not yet a final draft. During the two month period in which this package was developed, the specification changed, forcing a review of the code. This package is therefore based on the Nov 98 version.

2.2 Switches

There are two switches which can be used alternatively and incrementally. These are **mathml** and **both**. There use can be described as follows:

`mathml`**:** All output will be printed in MathML.

`both`**:** All output will be printed in both MathML and normal REDUCE.

`web`**:** All output will be printed within an HTML ¡embed¿ tag. This is for direct use in an HTML web page. Only works when `mathml` is on.

MathML has often been said to be too verbose. If **both** is on, an easy interpretation of the results is possible, improving MathML readability.

2.3 Entering MathML

The MathML-REDUCE interface gives the user various ways of providing input. This can be done via a file containing MathML, by writing MathML directly in the prompt, or by simply writing as usual in the prompt.

2.3.1 Reading MathML from a File: MML

When reading from a file the command **mml** is used. **mml** takes as argument the name of the file containing the MathML.

`mml`(FILE:*string*):*expression*

Example: As long as the file given contains valid MathML, no errors should be produced.

```
1:  mml "ex.mml";
```

2.3.2 Reading MathML from Prompt: PARSEML

By using the function **parseml** it is possible to introduce a series of valid mathml tokens to the prompt. **parseml** takes no arguments, although once it is called it will prompt you to enter mathml tags starting with <`mathml`> and ending with </`mathml`>. It then returns an expression resulting from evaluating the input.

Example: Here is an extract of a REDUCE session where `parseml()` is used:

```
2:  parseml();
2:  <math>
2:  <apply><plus/>
2:  <cn>3</cn>
2:  <cn>5</cn>
2:  </apply>
2:  </math>

        8

3:
```

If for some reason you do not want to continue typing in at the prompt, and cancel what has already been typed, it is possible to exit by calling control `C` or control `D`.

Although it is simpler to edit a file and then use `mml`, we still added **parseml** for completeness.

2.3.3 Reading Normal Input from the Prompt

If you wish to simply type in expressions in the normal fashion, without using MathML, then it is possible. Just use REDUCE as you normally would. If you wish to see the resulting expressions in MathML, then use the switches as described earlier in this text. In this way, you will be allowed to evaluate expressions and to generate MathML code, without needing to worry about how familiar you are with MathML. You can always switch **both** on to check if the MathML output corresponds to what you expect.

3 The Evaluation of MathML

MathML is evaluated by the interface always before outputting any results. Just in the same way as REDUCE normally does. The program works in the same way as the `algebraic` mode. Undefined variables remain undefined, and it is possible to use all normal REDUCE switches and packages.

Example: The following MathML:

```
<math>
   <relation><gt/>
      <ci>x</ci>
      <ci>y</ci>
   </relation>
</math>
```

will evaluate to `(gt x y)`. This only states that x is greater than y.
The interesting characteristic, is that we can set the values of x and y.
Suppose we enter the following:

```
   5:  x:=3; y:=2;

   x := 3

   y := 2
```

If we once again enter and evaluate the above piece of MathML we will have as result:

```
   t
```

because it is true that 3 is greater than 2. It is important to note that it is also possible to set only one of the two variables x or y above, say `y:=4`. The expression will then not evaluate completely, and we will have:

```
   (gt x 4)
```

When one of the switches is on, the MathML output will be:

```
<math>
   <relation><gt/>
     <ci>x</ci>
     <cn type="integer">4</cn>
   </relation>
</math>
```

Hence, it is possible when dealing with a MathML formula representation in which there are a set of undefined variables to set each variable to the desired value and evaluate it. Let us consider a second example to make sure everything is clear.

Example: Let the file `ex.mml` contain the following mathml:

```
<math>
  <apply><int/>
    <apply><fn><ci>F</ci><fn>
      <ci>x</ci>
    </apply>
    <bvar>
      <ci>x</ci>
    </bvar>
  </apply>
</math>
```

If we do the following:

```
   1:  mml "ex.mml";
```

This is what we get:

```
int(f(x),x);
```

It is clear that this has remained unevaluated. We can now set the function `f(x)` as follows:

```
   for all x let f(x)=2*x**2;
```

If we then enter '`mml "ex.mml"`' once again, we will have the following result:

$$\frac{2*x^3}{3}$$

Hence the MathML-REDUCE interface allows the user to set a value to a variable in order to manipulate the evaluation of the MathML, without needing to edit the MathML itself.

3.1 Using Boolean Values

Boolean values are not defined in MathML, despite their importance in REDUCE. To get around this problem, we can set a variable's value to a boolean value, and when evaluating, the MathML-REDUCE interface will use the boolean value of the variable.

Suppose we want to evaluate the following expression:

```
and(t,nil);
```

Then all we do, is we create a file with the following MathML:

```
<mathml>
  <apply><and/>
    <ci> a </ci>
    <ci> b </ci>
  </apply>
</mathml>
```

And before evaluating it we set `a` to `true` and `b` to `nil`. When evaluating the MathML, it will produce the same result as the equivalent REDUCE expression.

4 Interpretation of Error Messages

The MathML-REDUCE interface has a set of error messages which aim to help the user understand and correct any invalid MathML. Because there can exist many different causes of errors, such error messages should be considered merely as advice. Here we shall consider the most important error messages.

Missing tag: Many MathML tags go by pairs, such as `<apply></apply>`, `<relation></relation>`, `<ci></ci>`, etc.... In the case where the ending tag is missed out, or misspelled, it is very probable that an error of this type will be thrown.

Ambiguous or Erroneous Use of `<apply>` This error message defines an undefined error. When this error message appears, it is not very clear to the interface where exactly lies the error. Probable causes are a misuse of the `<apply></apply>` tags, or a mispelling of the tag preceding the `<apply>`. However, other types of errors may cause this error message to appear.

Tags following an `<apply>` tag may be misspelled without causing an error message, but they will be considered as operators by REDUCE and therefore evaluate to some unexpected expression.

Syntax Errors It is possible that the input MathML is not syntactically correct. In such a situation, the error will be spotted, and in some cases a solution might be presented. There are a variety of syntax errors messages, but relying on their advice might not always be helpful.

Despite the verbose nature of the error messages and their recommended solutions, we do recommend that in most situations reference to the MathML specification is made.

5 Limitations of the Interface

Not all aspects of MathML have been perfectly fitted into the interface. There are still some problems unsolved in the present version of the interface:

- MathML Presentation Markup is not supported. The interface will treat every presentation tag as an unknown tag, or a REDUCE operator. We found presentation markup not prioritary when dealing with computer algebra systems, although in the future, parsing of presentation markup within content markup shall be supported.

- Certain MathML tags do not play an important role in the REDUCE environment. Such tags will not evaluate or affect in anyway the interface's behaviour. They will be parsed correctly, although their action will be ignored. These tags are:

  1. `<interval></interval>`
  2. `<inverse></inverse>`
  3. `<condition></condition>`
  4. `<compose></compose>`
  5. `<ident></ident>`
  6. `<forall/>`
  7. `<exists/>`

  Although `<condition></condition>` and `<interval></interval>` tags are supported when used within the following tags:

  1. `<int/>`
  2. `<limit/>`
  3. `<sum/>`
  4. `<product/>`

- The `<declare>` construct takes one or two arguments. It sets the first argument to the value of the second. In the case where the second argument is a vector or a matrix, an obscure error message is produced. It is clearly something which must be fixed in the future.

- The program throws an error when it encounters `nil` between `<ci></ci>` tags. It is not possible to use boolean values directly. Please refer to the above subsection treating this matter.

6 Examples

We would like to present a series of examples which will illustrate the possibilities of the interface.

Example 1 Type in the following and observe the resulting expression:

```
23: on mathml;

24: solve({z=x*a+1},{z,x});
```

Example 2 Have a file `ex2.mml` containing the following MathML source code:

```
<mathml>
  <apply><sum/>
    <apply><fn><ci>F</ci><fn>
      <ci>x</ci>
    </apply>
    <bvar>
      <ci>x</ci>
    </bvar>
  </apply>
</mathml>
```

and type:

```
mml "ex2.mml"
```

Example 3 This example illustrates how practical the switch **both** can be for interpreting verbose MathML. Introduce the following MathML source into a file, say `ex3.mml`

```
<mathml>
  <apply><int/>
    <apply><sin/>
      <apply><log/>
        <ci>x</ci>
      </apply>
    </apply>
    <bvar>
      <ci>x</ci>
    </bvar>
  </apply>
</mathml>
```

then do the following:

```
2: on both;

3: mml "ml";
```

7 An overview of how the Interface Works

The interface is primarily built in two parts. A first one which parses and evaluates MathML, and a second one which parses REDUCE's algebraic expressions and prints them out in MathML format. Both parts work by recursive parsing, using Top-Down Recursive Descent parsing with one token look ahead.

The BNF description of the MathML grammar is to be defined informally in APPENDIX E of the current MathML specification. It is with this document that we have developed the MathML parser. The MathML parser evaluates all that is possible and returns a valid REDUCE algebraic expression. When **mathml** or

both are on, this algebraic expression is fed into the second part of the program which parses these expressions and transforms them back into MathML.

The MathML generator parses through the algebraic expression produced by either REDUCE itself or the MathML parser. It works in a very similar way as the MathML parser. It is simpler, since no evaluation is involved. All the generated code is MathML compliant. It is important to note that the MathML code generator sometimes introduces Presentation Markup tags, and other tags which are not understood by the MathML parser of the interface[2].

[2]The set of tags not understood by the MathML parser are detailed in section **Limitation**.