meditor, the mathemagical editor

Software demonstration

RAPHAËL JOLLY, Databeans, France

meditor is a text-based editor interface to the Java symbolic computing engine/library (JSCL). It operates as a magic board: expressions are evaluated in-place and rendered, either in text or MathML. Communication with the engine is made through the JSR-223 Scripting API, which makes it interchangeable. MathML results are integrated in the document as MathML islands. It is also possible to include graphs as SVG islands. With this technique, the file format is restricted to `text/plain`, for improved interoperability. There is however the option to export documents in XHTML and PDF. MathML fragments are translated to script using XSLT transformations, to be interpreted again by the engine. The symbolic engine can be either specialized for symbolic computation, or a general purpose scripting language, following the "libraries and scripting" approach [1], which will be showcased using the Beanshell scripting language [2].

CCS Concepts: • **Mathematics of computing** → Mathematical software; • **Computing methodologies** → Symbolic and algebraic manipulation; • **Applied computing** → Document management and text processing.

Additional Key Words and Phrases: typesetting mathematics

ACM Reference Format:
Raphaël Jolly. 2020. meditor, the mathemagical editor: Software demonstration. In . ACM, New York, NY, USA, 4 pages.

1 BASIC OPERATION : EVALUATION AND RENDERING

In the editor's text pane, an expression is selected and evaluated with the `Math->Evaluate` menu item (Ctrl+E). MathML rendering is enabled by selecting the `Rendering` checkbox in the option panel. The defining trait of symbolic computation, as opposed to numerical, is that one can manipulate symbols without assigning them first with a value:

```
x^2*y^2
```

= x^2y^2

2 JSCL EXAMPLES

Here are some example computations with the JSCL engine:

```
solve(c+b*x+a*x^2,x)
```

= $\text{root}_0(c, b, a)$

```
subst(1/x^2,x,a)
```

= $\frac{1}{a^2}$

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2020 Association for Computing Machinery.
Manuscript submitted to ACM

```
simplify(exp(sqrt(-1)*pi))
```

$= -1$

```
simplify(elementary(cos(x)^2+sin(x)^2))
```

$= 1$

```
numeric(exp(1))
```

$= 2.718281828459045$

```
d(cos(f(x)),x)
```

$= -(f\prime(x) \sin f(x))$

```
curl(grad(f(x,y,z),{x,y,z}),{x,y,z})
```

$$= \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

```
sum(d(exp(x),x,0,i)/i!*(x-0)^i,i,0,4)
```

$= 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4$

```
integral(1/(1+x^2),x)
```

$= \text{root}_0(1, 0, 4) \ln(1 - 2x\text{root}_0(1, 0, 4)) + \text{root}_1(1, 0, 4) \ln(1 - 2x\text{root}_1(1, 0, 4))$

A special `quote` operator is provided to suspend evaluation and render expression inputs:

```
quote(integral(1/(1+x^2),x))
```

$= \int \frac{1}{1+x^2}\, dx$

3 REVERSE INTERPRETATION OF MATHML EXPRESSIONS

Once computed and rendered, expressions can be re-interpreted by the script engine by means of XSLT transformation, in function of the considered language, as show in Figure 1.

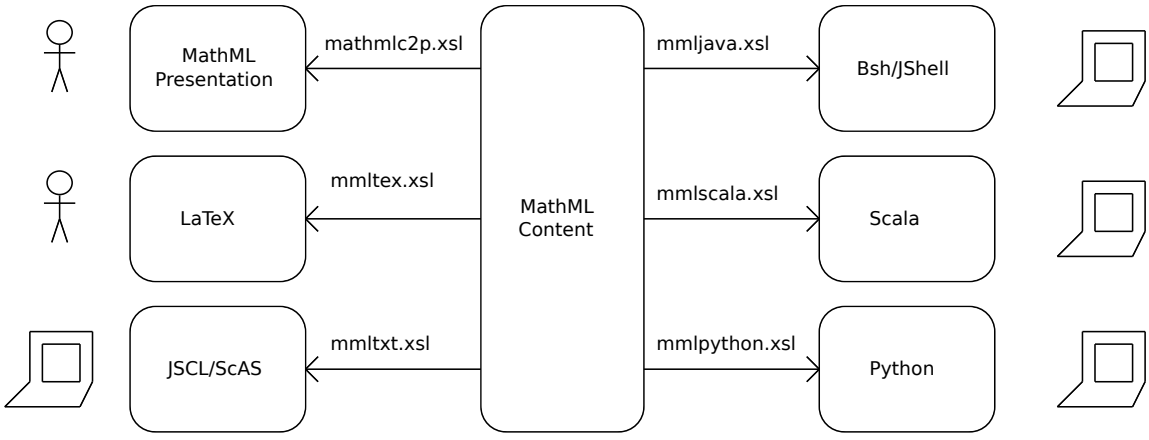

Fig. 1. Interpretation of MathML fragments through XSLT.

4 BEANSHELL SCRIPT EXAMPLE : POLYNOMIAL SYSTEM SOLVING

meditor can solve polynomial systems using Gröbner bases, which can be useful in a variety of domains, for example in geometry. Suppose we want to find the intersection of a line and a circle. We can write the corresponding system and apply the `groebner` operator, as shown below.

```
import jscl.math.Generic;
static import jscl.math.Predef.*;

x = variable("x");
y = variable("y");
```

$$\text{groebner}(\begin{pmatrix} 4-x^2-y^2 \\ 1-xy \end{pmatrix}, \begin{pmatrix} x \\ y \end{pmatrix}) = \begin{pmatrix} 1-4x^2+x^4 \\ 4x-x^3-y \end{pmatrix}$$

Input expressions are being first re-interpreted to Beanshell through XSLT. This is made explicit using the `Math->Copy to code` menu item (Ctrl-J):

```
vector(new Generic[] {integer("1").subtract(integer("4").multiply(x.pow(2))).add(x.pow(4)),
integer("4").multiply(x).subtract(x.pow(3)).subtract(y)})
```

Note that in contrast to the specialized case, some states need to be prepared beforehand by importing some classes and assigning symbolic values to some variables. Without it, we would be unable to interpret expressions as valid script fragments.

5 GRAPHING

Graphs are produced using the `graph` operator as shown in Figure 2.

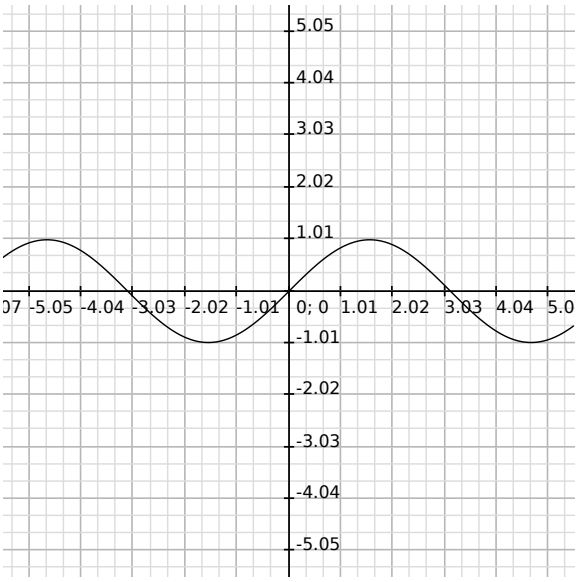

Fig. 2. `graph(sin(x),x)`

REFERENCES

[1] Raphael Jolly and Heinz Kredel. 2008. *How to turn a scripting language into a domain specific language for computer algebra.* Technical Report. http://arXiv.org/abs/0811.1061.

[2] Peter Niemeyer. 1997. *BeanShell: lightweight scripting for Java.* Technical Report. http://www.beanshell.org/.