SMathML: 一种表达MathML的嵌入Scheme的DSL, 或一种网页上排版数学内容的新方法

MathML是一种基于XML的标记语言, 用于表达数学符号. 我们设计了SMathML, 一种用于表达MathML的嵌入Scheme的DSL. 它允许用户轻松地编写网页上的数学内容, 并轻松地利用Scheme对其进行扩展.

1 引入

MathML[2]是一种基于XML的标记语言, 用于表达数学符号. MathML被分为两个部分, Presentation MathML和Content MathML. 其中, Presentation MathML用于刻画数学公式的形状, 而Content MathML用于刻画数学公式的语义. 实际上, Content MathML十分接近于Lisp系语言. 按照MathML设计者最初的想法, 用户只需要刻画数学公式的语义, 而数学公式的形状可以由程序自动确定. 尽管这个想法是有道理的, 但是鉴于数学实践中数学符号及其语义之间交互的复杂性, 它从来没能成为现实.

MathML是HTML5标准的一部分. 尽管如此, 主流浏览器对于MathML的支持却难言乐观, 这也是推广MathML的主要阻力之一. 目前, 在网页上排版数学内容的主流方法是MathJax和KaTeX, 而难见MathML的使用. 幸运的是, Chrome自109版本[1]开始提供对于MathML Core[1]标准的支持, 至此主流浏览器 (Chrome, Safari, Firefox等) 均对于MathML有所支持.

我们设计了SMathML, 一种用于表达MathML的嵌入Scheme的DSL. 它允许用户轻松地编写网页上的数学内容, 并轻松地利用Scheme对其进行扩展. 这个DSL分为几个不同的层次, 我们将在之后几节对其逐一阐释. 在理解了这些层次之后, 就理解了SMathML本身, 对其进行扩展也就是简单的了.

读者应该注意的是, 本文使用的Scheme方言是Racket, 并不完全兼容其他Scheme实现. 不过, 读者在理解的基础上, 很容易将其移植到其他Scheme方言.

2 SXML: 嵌入Scheme的XML

SXML是XML的S-exp句法版本[2], 任何熟悉XML的读者都应该能够很容易看出它们之间的对应关系. 首先, 我们定义SXML的句法.

```
<sxml> ::= <string>
         |  (<symbol> (<attr>*) <sxml>*)
```

请读者观察一个非常简单的例子.
SXML表达式

```
(foo ((bar "baz")) "qux")
```

就相当于XML表达式

```
<foo bar="baz">qux</foo>
```

我们提供了一个将SXML转换为XML的简单过程`Sxml`.

```
(define (Attr* attr*)
  (for-each
   (lambda (attr)
     (printf " ~s=~s" (car attr) (cadr attr)))
   attr*))
(define (Sxml sxml)
  (match sxml
    ((,tag ,attr* . ,sxml*)
     (printf "<~s" tag)
     (Attr* attr*)
     (printf ">")
     (for-each Sxml sxml*)
     (printf "</~s>" tag))
    (,str
```

1. 于2023年1月10日发布.

2. 我们的SXML与Oleg Kiselyov的SXML[3]类似但略有不同, 也与Racket的xexpr (注意与W3C标准XEXPR区分开来) 类似但有所不同. 实际上, 我们认为大部分Lisper/Schemer都设计过差不多的东西.

```
    (guard (string? str))
    (printf "~a" str))))
```

其中match是一个简单的模式匹配宏, 见附录A.
Sxml的用法是相当直接的, 请读者观察以下例子.

```
> (Sxml '(foo ((bar "baz")) "qux"))
<foo bar="baz">qux</foo>
```

3 SHTML: 嵌入Scheme的HTML

读者应该注意到, 严格来说, 本节并不是SMathML的一部分. 尽管如此, 为了将MathML嵌入HTML之中以观察浏览器的实际渲染结果, 本节是必不可少的.
SHTML在SXML的基础之上, 提供了一组构造网页的原语, 让我们观察其中一个典型的过程.

```
(define (P #:attr* [attr* '()] . shtml*)
  `(p ,attr* . ,shtml*))
```

其中#:attr* [attr* '()]代表attr*是可选的关键词参数, 默认值为空表.
我们按照HTML标准编写了这一系列基本的原语, 读者可以根据这些原语提供更高层次的一些抽象, 也就是说, 读者可以对其进行扩展, 例如Prelude可以减少编写网页的一些前置冗余.

```
(define (Prelude #:title [title "index"] #:css [css #f] . body*)
  (Html
   (Head
    (Meta #:attr* '((charset "utf-8")))
    (Title title)
    (if css
        (Link #:attr* `((href ,css) (rel "stylesheet")))
        ""))
   (apply Body body*)))
```

让我们观察一个简单的例子.

```
> (Sxml
   (Prelude
    #:title "gcd"
    #:css "styles.css"
    (H1 "Euclid算法")
    (P "以下简单的过程" (Code "gcd") "刻画了Euclid算法:")
    (Pre (Code "(define (gcd a b)
  (if (= b 0)
      a
      (gcd b (remainder a b))))"))))
<html><head><meta charset="utf-8"></meta><title>gcd</title><link href="styles.css"
rel="stylesheet"></link></head><body><h1>Euclid算法</h1><p>以下简单的过程<code>gcd</
code>刻画了Euclid算法:</p><pre><code>(define (gcd a b)
  (if (= b 0)
      a
      (gcd b (remainder a b))))</code></pre></body></html>
```

4 SMathML: 嵌入Scheme的MathML

4.1 SMathML原语

本小节与前一节类似, 刻画了基本的构造原语.
正如MathML表达式的根必须是<math>标签, SMathML也不例外.

```
(define (Math #:attr* [attr* '()] . smathml*)
  `(math ,attr* . ,smathml*))
```

的确, 这和前一节如出一辙. 目前, 我们只提供MathML Core里的原语, 以保持浏览器之间的兼容性. 也就是说, 我们提供了以下这些过程: `Math Merror Mfrac Mi Mmultiscripts Mn Mo Mover Mpadded Mphantom Mroot Mrow Ms Mspace Msqrt Mstyle Msub Msubsup Msup Mtable Mtd Mtext Mtr Munder Munderover`.

现在让我们观察一个简单的例子.

为了呈现欧拉公式

$$e^{i\pi}+1=0$$

我们需要编写以下的SMathML表达式

```
(Math #:attr* '((display "block"))
      (Msup (Mi "e") (Mrow (Mi "i") (Mo "&it;") (Mi "&pi;")))
      (Mo "+") (Mn "1") (Mo "=") (Mn "0"))
```

转换为MathML的形式也就是

```
<math display="block"><msup><mi>e</mi><mrow><mi>i</mi><mo>&it;</mo><mi>&pi;</mi></
mrow></msup><mo>+</mo><mn>1</mn><mo>=</mo><mn>0</mn></math>
```

尽管SMathML在结构上是清晰的, 但是它相较于MathML本身并没有多少简化. 不过, 这也正是抽象与组合的基本原则的用武之地, 我们将在本节的剩余部分呈现这些想法.

4.2 别名

命名是最基本的抽象手段. 一些常用的SMathML原语的名字比较冗长, 不免繁琐. 于是, 我们应该为其取一些较短的别名.

```
(define _ Msub)
(define ^ Msup)
(define _^ Msubsup)
(define ~ Mfrac)
```

名字本身是随意的, 用户可能有自己的喜好, 那么此时他就应该修改这些定义.

4.3 常用符号

一个数学公式总是由大量的基本的常用符号构成. 而且, 鉴于MathML区分数字, 标识符和运算符, 提供这些常用符号是自然的. 以下表格总结了我们目前提供的定义.

种类	示例定义	示例数学符号
小写拉丁字母	`(define $x (Mi "x"))`	x
大写拉丁字母	`(define $A (Mi "A"))`	A
小写希腊字母	`(define $epsilon (Mi "&epsilon;"))`	ε
大写希腊字母	`(define $Sigma (Mi "&Sigma;"))`	Σ
黑板粗体	`(define $RR (Mi "Ropf"))`	$\mathbb{R}$
数字0到9	`(define $7 (Mn "7"))`	7
括号	`(define $lc (Mo "{"))`	$\{$
运算符	`(define $c* (Mo "&times;"))`	$\times$
杂项	`(define $inf (Mi "&infin;"))`	∞

读者应该注意到, 这些符号的名字之前都加上了`$`以进行区分.

4.4 中缀运算符的抽象

与Scheme不同的是, 数学实践中大量运用到中缀运算符. 因此, 有必要单独为此设计一种抽象, 以允许我们像在Scheme中编写程序一样表达数学表达式.

```
(define ((make-op op n u) . x*)
  (cond
    ((null? x*) (n))
    ((null? (cdr x*)) (u (car x*)))
    (else
     (let iter ((x (car x*)) (x* (cdr x*)) (r '()))
```

```
      (if (null? x*)
          (apply Mrow (reverse (cons x r)))
          (iter (car x*)
                (cdr x*)
                (cons op (cons x r))))))))
```

于是, 我们设计了高阶过程make-op, 其接受三个参数op, n, u. op是中缀运算符字面上的符号, n和u给用户以选择在零元和幺元参数的情况下呈现何种行为.

让我们来看一个简单的例子&cm.

```
(define &cm
  (make-op $cm
    (lambda () $)
    (lambda (x) x)))
```

其中$和$cm的定义如下.

```
(define $ (Mrow))
(define $cm (Mo ","))
```

注意到我们采用了这样的命名习惯, &代表构造的过程, 而$代表字面上的符号. &cm是这样的过程, 它将元素用逗号分隔开来. 在没有元素的情况下, 它就只是一个占位符; 在仅有一个元素的情况下, 也不需要分隔. 让我们看一点例子.

```
> (Sxml (&cm))
<mrow></mrow>
> (Sxml (&cm $a))
<mi>a</mi>
> (Sxml (&cm $a $b))
<mrow><mi>a</mi><mo>,</mo><mi>b</mi></mrow>
> (Sxml (&cm $a $b $c))
<mrow><mi>a</mi><mo>,</mo><mi>b</mi><mo>,</mo><mi>c</mi></mrow>
```

4.5 括号

数学实践总是使用括号将表达式划分成组, 以弥补运算符优先级和结合律带来的先天不足. MathML的一个复杂性在于它会自动将括号的大小自动拉伸以与括号内的表达式匹配, 然而很多时候这与数学实践是不一致的. 不过, MathML提供了调节括号大小的属性.

```
(define $lp (Mo "("))
(define $rp (Mo ")"))
(define $lp0
  (Mo "(" #:attr* '((stretchy "false"))))
(define $rp0
  (Mo ")" #:attr* '((stretchy "false"))))
```

因此像这样, 对于每种括号, 我们提供了两组, 一组是自动拉伸的, 一组是保持原有大小的. 然后, 我们也提供了两个括起来的过程.

```
(define (pare x)
  (Mrow $lp x $rp))
(define (par0 x)
  (Mrow $lp0 x $rp0))
```

括号可以与其他过程灵活地组合起来, 例如构造元组的过程不过就是&cm和pare或par0的复合.

```
(define (tup . x*)
  (pare (apply &cm x*)))
(define (tu0 . x*)
```

```
  (par0 (apply &cm x*)))
```

4.6 表格: 矩阵, 行列式, 选择等

MathML提供了<mtable>, <mtr>, <mtd>标签, 这可以类比于HTML的<table>, <tr>, <td>标签. 表格的机制是相当灵活的, 藉此我们可以表达矩阵, 行列式, 选择等.

```
(define (&table l)
  (apply Mtable
         (map (lambda (r)
                (apply Mtr (map Mtd r)))
              l)))
(define (mat l)
  (brac (&table l)))
(define (det l)
  (vert (&table l)))
(define (choice l)
  (Mrow $lc (&table l)))
```

读者可以看到, 矩阵不过就是表格之外用方括号括起来, 行列式不过就是用竖线括起来, 而选择不过就是左边加上一个花括号.

4.7 大运算符: 求和, 求积, 积分等

有些大运算符虽然和某些小运算符形状类似, 不过它们的渲染方式是不同的, 这些也值得单独抽象出来.

```
(define (sum u o e)
  (Mrow (Munderover (Mo "&sum;") u o) e))
(define (prod u o e)
  (Mrow (Munderover (Mo "&prod;") u o) e))
(define (integral a b e x)
  (Mrow (_^ $int a b) e
        (Mo "d" #:attr* '((rspace "0")))
        x))
```

5 一些实际例子

本节我们提供了一些例子, 并给出了Chrome和Firefox浏览器渲染的结果. 其中, Chrome的版本号是118.0.5993.71, Firefox的版本号是118.0.2.

5.1 二项式定理

以下是程序.

```
(MathB
 (&= (^ (par0 (&+ $x $y)) $n)
     (sum (&= $k $0) $n
          (&i* (comb $n $k)
               (^ $x (&- $n $k))
               (^ $y $k)))))
```

其中MathB的定义如下, MathB和Math类似, 但呈现方式是块而不是内联.

```
(define (MathB #:attr* [attr* '()] . smathml*)
  `(math ((display "block") . ,attr*) . ,smathml*))
```

以下是Chrome的渲染结果.

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$$

以下是Firefox的渲染结果.

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$$

5.2 计算三阶行列式

以下是程序.

```
(MathB
 (deriv
  (det '((,$a ,$b ,$c)
         (,$d ,$e ,$f)
         (,$g ,$h ,$i)))
  (&+ (&- (&i* $a (det '((,$e ,$f)
                         (,$h ,$i))))
          (&i* $b (det '((,$d ,$f)
                         (,$g ,$i)))))
      (&i* $c (det '((,$d ,$e)
                     (,$g ,$h)))))
  (&- (&+ (&i* $a $e $i)
          (&i* $b $f $g)
          (&i* $c $d $h))
      (&i* $c $e $g)
      (&i* $a $f $h)
      (&i* $b $d $i))))
```

其中deriv的定义如下. deriv使用了<mtd>的一个属性，但是它并不在MathML Core之中，所以Chrome的渲染结果仍然是居中的.

```
(define (deriv x y . z*)
  (apply
   Mtable
   (cons (Mtr (Mtd x) (Mtd $=) (Mtd #:attr* '((columnalign "left")) y))
         (map (lambda (z)
                (Mtr (Mtd $) (Mtd $=)
                     (Mtd #:attr* '((columnalign "left")) z)))
              z*))))
```

以下是Chrome的渲染结果.

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix}$$
$$= aei + bfg + cdh - ceg - afh - bdi$$

以下是Firefox的渲染结果.

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix}$$
$$= aei + bfg + cdh - ceg - afh - bdi$$

5.3 线性方程组的一般形式

以下是程序.

```
(MathB
 (let ((A (lambda (i j) (_ $A (&cm i j)))))
```

```
      ($x_1 (_ $x $1)) ($x_2 (_ $x $2)) ($x_n (_ $x $n)))
  (choice
   (list
    (list (&i* (A $1 $1) $x_1) $+ (&i* (A $1 $2) $x_2)
          $+ $..c $+ (&i* (A $1 $n) $x_n) $= (_ $y $1))
    (list (&i* (A $2 $1) $x_1) $+ (&i* (A $2 $2) $x_2)
          $+ $..c $+ (&i* (A $2 $n) $x_n) $= (_ $y $2))
    (list $..v $ $..v $ $ $ $..v $ $..v)
    (list (&i* (A $m $1) $x_1) $+ (&i* (A $m $2) $x_2)
          $+ $..c $+ (&i* (A $m $n) $x_n) $= (_ $y $m))))))
```

其中`$..c`和`$..v`的定义如下.

```
(define $..c (Mo "&ctdot;"))
(define $..v (Mo "&vellip;"))
```

以下是Chrome的渲染结果.

$$\begin{cases} A_{1,1}x_1 & + & A_{1,2}x_2 & + & \cdots & + & A_{1,n}x_n & = & y_1 \\ A_{2,1}x_1 & + & A_{2,2}x_2 & + & \cdots & + & A_{2,n}x_n & = & y_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ A_{m,1}x_1 & + & A_{m,2}x_2 & + & \cdots & + & A_{m,n}x_n & = & y_m \end{cases}$$

以下是Firefox的渲染结果.

$$\begin{cases} A_{1,1}x_1 & + & A_{1,2}x_2 & + & \cdots & + & A_{1,n}x_n & = & y_1 \\ A_{2,1}x_1 & + & A_{2,2}x_2 & + & \cdots & + & A_{2,n}x_n & = & y_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ A_{m,1}x_1 & + & A_{m,2}x_2 & + & \cdots & + & A_{m,n}x_n & = & y_m \end{cases}$$

5.4 度量空间的定义

```
((definition)
 "给定集合" (Math $E) ", " (Math $E) "上的一个度量" (Math $d)
 "是一个类型为" (Math (&-> (&c* $E $E) $RR)) "的映射, 满足:"
 (let* ((d (lambda (x y) (ap $d (tu0 x y))))
        (dxy (d $x $y)) (dyx (d $y $x))
        (dyz (d $y $z)) (dxz (d $x $z)))
   (Ol (Li "对于任意的" (Math (&in (&cm $x $y) $E)) ", "
           (Math (&>= dxy $0)) ";")
       (Li "对于任意的" (Math (&in (&cm $x $y) $E)) ", "
           (Math (&<=> (&= dxy $0) (&= $x $y))) ";")
       (Li "对于任意的" (Math (&in (&cm $x $y) $E)) ", "
           (Math (&= dxy dyx)) ";")
       (Li "对于任意的" (Math (&in (&cm $x $y $z) $E)) ", "
           (Math (&>= (&+ dxy dyz) dxz)) "."))))
 "如果" (Math $d) "是" (Math $E) "上的一个度量, 那么称序对"
 (Math (tu0 $E $d)) "是一个度量空间.")
```

其中`definition`的定义如下. 这么定义的话, 读者可以使用CSS或者XSLT调整`definition`的样式.

```
(define ((definition #:n [n ""]) . x*)
  (keyword-apply
   Div '(#:attr*) '(((class "definition")))
   (B (format "定义~a." n)) " " x*))
```

以下是Chrome的渲染结果.

定义. 给定集合E, E上的一个度量d是一个类型为$E \times E \to \mathbb{R}$的映射, 满足:

1. 对于任意的$x, y \in E$, $d(x, y) \geq 0$;
2. 对于任意的$x, y \in E$, $d(x, y) = 0 \Leftrightarrow x = y$;
3. 对于任意的$x, y \in E$, $d(x, y) = d(y, x)$;
4. 对于任意的$x, y, z \in E$, $d(x, y) + d(y, z) \geq d(x, z)$.

如果d是E上的一个度量, 那么称序对(E, d)是一个度量空间.

以下是Firefox的渲染结果.

定义. 给定集合E, E上的一个度量d是一个类型为$E \times E \to \mathbb{R}$的映射, 满足:

1. 对于任意的$x, y \in E$, $d(x, y) \geq 0$;
2. 对于任意的$x, y \in E$, $d(x, y) = 0 \Leftrightarrow x = y$;
3. 对于任意的$x, y \in E$, $d(x, y) = d(y, x)$;
4. 对于任意的$x, y, z \in E$, $d(x, y) + d(y, z) \geq d(x, z)$.

如果d是E上的一个度量, 那么称序对(E, d)是一个度量空间.

6 总结

我们设计并实现了SMathML, 它在概念上是简单的, 理解上是清晰的, 扩展上是容易的. SMathML的一个优势在于它生成的是XML/HTML, 因而容易与现有的XML/HTML生态集成. 目前, SMathML是嵌入式的DSL, 它提供了一集花样繁多的构造过程. 有的时候, 我们会觉得这些过程的数量太多了, 以至于严重挤占了命名的空间, 在某些情况下这是难以接受的. 因此, 我们认为提供基于S-exp的外挂DSL作为替代也是有必要的. 例如, 读者可以期望编写

```
(MB '(= (^ (par0 (+ x y)) n)
        (sum (= k 0) n
             (i* (comb n k)
                 (^ x (- n k))
                 (^ y k)))))
```

而不是

```
(MathB
 (&= (^ (par0 (&+ $x $y)) $n)
     (sum (&= $k $0) $n
          (&i* (comb $n $k)
               (^ $x (&- $n $k))
               (^ $y $k)))))
```

除此之外, 我们还没有考虑到图像绘制的要求, 但是数学实践对于绘制函数图像和交换图表等对象的需求是迫切的. 未来我们或许会考虑实现一个类似于MetaPost的嵌入Scheme的DSL, 以SVG格式作为输出, 以方便地呈现在网页之中.

致谢

如果没有阅读Harold Abelson和Gerald Sussman的经典著作SICP, 我无法学会编程. 如果没有阅读Daniel P. Friedman的小人书系列和EoPL, 我无法获得对于编程语言的深刻理解. 这篇论文的许多想法都受到了Oleg Kiselyov的启发, 他的个人主页是关于编程语言的宝藏.

参考文献

[1] Mathml core. https://www.w3.org/TR/mathml-core/, May 2022.

[2] Mathematical markup language (mathml) version 4.0. https://www.w3.org/TR/2022/WD-mathml4-20220908/, September 2022.

[3] Xml and scheme. https://okmij.org/ftp/Scheme/xml.html, May 2014.

A `match`: 一个模式匹配宏

```
(define-syntax match
  (syntax-rules (guard)
    ((_ v) (error 'match "~s" v))
    ((_ v (pat (guard g ...) e ...) cs ...)
     (let ((fk (lambda () (match v cs ...))))
       (ppat v pat (if (and g ...) (let () e ...) (fk)) (fk))))
    ((_ v (pat e ...) cs ...)
     (let ((fk (lambda () (match v cs ...))))
       (ppat v pat (let () e ...) (fk))))))
(define-syntax ppat
  (syntax-rules (unquote)
    ((_ v () kt kf) (if (null? v) kt kf))
    ((_ v (unquote var) kt kf) (let ((var v)) kt))
    ((_ v (x . y) kt kf)
     (if (pair? v)
         (let ((vx (car v)) (vy (cdr v)))
           (ppat vx x (ppat vy y kt kf) kf))
         kf))
    ((_ v lit kt kf) (if (eqv? v (quote lit)) kt kf))))
```

`error`的部分可能需要修改, 以适应不同的Scheme实现.