Editing MathML (ab)using TEX syntax

Luca Padovani

Department of Computer Science
University of Bologna
http://www.cs.unibo.it/~lpadovan

W3C Math Interest Group
http://www.w3.org/Math

Writing MathML? No, thanks

It is a fact that MathML cannot be edited by hand.

$$\int \frac{ax+b}{x^2+px+q}dx = \frac{a}{2}\ln(x^2+px+q) + \frac{2b-ap}{\sqrt{4q-p^2}}\operatorname{arctg}\frac{2x+p}{\sqrt{4q-p^2}} + c$$

Writing MathML? No, thanks

It is a fact that MathML cannot be edited by hand.

$$\int \frac{ax+b}{x^2+px+q}dx = \frac{a}{2}\ln(x^2+px+q) + \frac{2b-ap}{\sqrt{4q-p^2}}\operatorname{arctg}\frac{2x+p}{\sqrt{4q-p^2}} + c$$

```
<math> <mrow> <mo>&#x222b;</mo> <mo>&#x2061;</mo> <mfrac> <mrow> <mrow> <mi>a</mi>
<mo>&#x2062;</mo> <mi>x</mi> </mrow> <mo>+</mo> <mi>b</mi> </mrow> <mrow> <msup> <mi>x</mi>
<mn>2</mn> </msup> <mo>+</mo> <mrow> <mi>p</mi> <mo>&#x2062;</mo> <mi>x</mi> </mrow>
<mo>+</mo> <mi>q</mi> </mrow> </mfrac> </mrow> <mo mathvariant="italic">d</mo> <mi>x</mi>
<mo>=</mo> <mrow> <mrow> <mfrac><mi>a</mi> <mn>2</mn></mfrac> <mo>&#x2062;</mo> <mrow>
<mi>ln</mi> <mo>&#x2061;</mo> <mrow> <mo>(</mo> <mrow> <msup><mi>x</mi><mn>2</mn></msup>
<mo>+</mo> <mrow> <mi>p</mi> <mo>&#x2062;</mo> <mi>x</mi> </mrow> <mo>+</mo> <mi>q</mi>
</mrow> <mo>)</mo> </mrow> </mrow> </mrow> <mo>+</mo> <mrow> <mfrac> <mrow> <mrow> <mn>2</mn>
<mo>&#x2062;</mo> <mi>b</mi> </mrow> <mo>-</mo> <mrow> <mi>a</mi> <mo>&#x2062;</mo>
<mi>p</mi> </mrow> </mrow> <msqrt> <mrow> <mrow> <mn>4</mn> <mo>&#x2062;</mo> <mi>q</mi>
</mrow> <mo>-</mo> <msup> <mi>p</mi> <mn>2</mn> </msup> </mrow> </msqrt> </mfrac>
<mo>&#x2062;</mo> <mrow> <mi>arctg</mi> <mo>&#x2061;</mo> <mfrac> <mrow> <mrow> <mn>2</mn>
<mo>&#x2062;</mo> <mi>x</mi> </mrow> <mo>+</mo> <mi>p</mi> </mrow> <msqrt> <mrow> <mrow>
<mn>4</mn> <mo>&#x2062;</mo> <mi>q</mi> </mrow> <mo>-</mo> <msup> <mi>p</mi> <mn>2</mn>
</msup> </mrow> </msqrt> </mfrac> </mrow> </mrow> <mo>+</mo> <mi>c</mi> </mrow> </math>
```

Writing MathML? No, thanks

It is a fact that MathML cannot be edited by hand.

$$\int \frac{ax+b}{x^2+px+q}dx = \frac{a}{2}\ln(x^2+px+q) + \frac{2b-ap}{\sqrt{4q-p^2}}\mathrm{arctg}\frac{2x+p}{\sqrt{4q-p^2}} + c$$

```
\int{ax+b\over x^2+px+q}dx={a\over2}\ln(x^2+px+q)+{2b-ap\over\sqrt{4q-p^2}}
\mathrm{arctg}{2x+p\over\sqrt{4q-p^2}}+c
```

Writing MathML? No, thanks

It is a fact that MathML cannot be edited by hand.

$$\int \frac{ax+b}{x^2+px+q}dx = \frac{a}{2}\ln(x^2+px+q) + \frac{2b-ap}{\sqrt{4q-p^2}}\mathrm{arctg}\frac{2x+p}{\sqrt{4q-p^2}} + c$$

```
\int{ax+b\over x^2+px+q}dx={a\over2}\ln(x^2+px+q)+{2b-ap\over\sqrt{4q-p^2}}
\mathrm{arctg}{2x+p\over\sqrt{4q-p^2}}+c
```

Need assistance? Yes, please

WYSIWYG editors

- fully interactive
- menus and palettes for symbols and constructs
- direct visual feedback

Need assistance? Yes, please

WYSIWYG editors

- fully interactive
- menus and palettes for symbols and constructs
- direct visual feedback

Translators (from TEX)

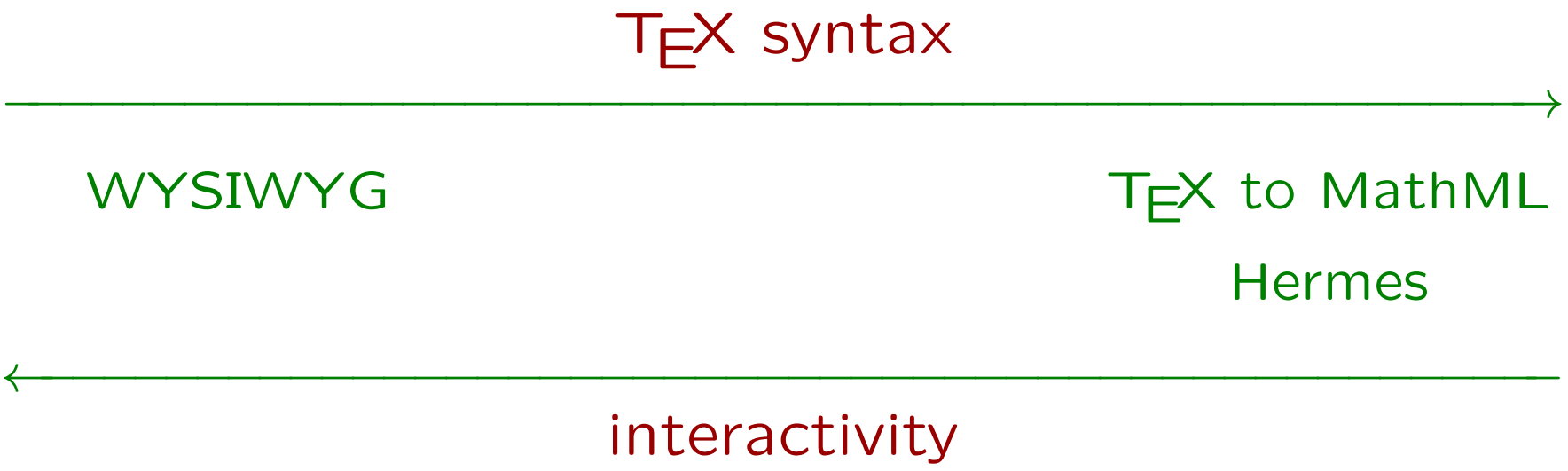
- batch processing
- require knowledge of TEX syntax
- separate editing window and view

WYSIWYG editors drawbacks

- slow editing, especially for complex markup
- no possibility of improving
- limited extensibility
- besides, they have usability issues

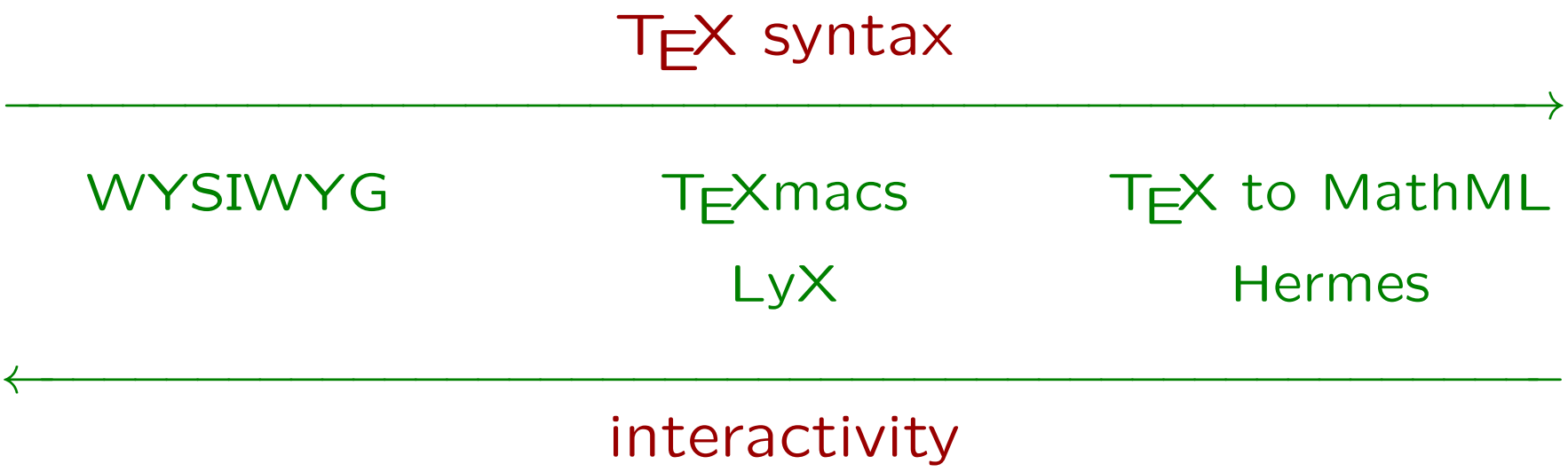
WYSIWYG editors drawbacks

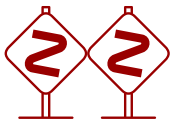
- slow editing, especially for complex markup
- no possibility of improving
- limited extensibility
- besides, they have usability issues



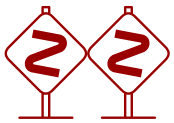
WYSIWYG editors drawbacks

- slow editing, especially for complex markup
- no possibility of improving
- limited extensibility
- besides, they have usability issues



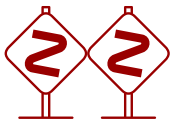


Combine the best (worst?) of both worlds: WYSIWYG editor controlled by TEX syntax.

 Combine the best (worst?) of both worlds: WYSIWYG editor controlled by TEX syntax.

Why TEX?

- popularity
- good support for macros
- relatively simple with good locality properties



Combine the best (worst?) of both worlds: WYSIWYG editor controlled by TEX syntax.

Why TEX?

- popularity
- good support for macros
- relatively simple with good locality properties

However the original TEX parser cannot be used:

- large
- not suitable for incremental processing

Architecture

Define a reasonable subset of TEX syntax and write our own incremental TEX parser.

Architecture

Define a reasonable subset of TEX syntax and write our own incremental TEX parser.

Split macro definitions:

`\def`⟨control sequence⟩	`\def\frac`
⟨parameter text⟩	`#1#2`
{⟨replacement text⟩}	`{{#1\over#2}}`

Architecture

Define a reasonable subset of TEX syntax and write our own incremental TEX parser.

Split macro definitions:

`\def`⟨control sequence⟩	`\def\frac`
⟨parameter text⟩	`#1#2`
{⟨replacement text⟩}	`{{#1\over#2}}`

⟨parameter text⟩ ⇒ document structure

Architecture

Define a reasonable subset of TEX syntax and write our own incremental TEX parser.

Split macro definitions:

`\def`⟨control sequence⟩	`\def\frac`
⟨parameter text⟩	`#1#2`
{⟨replacement text⟩}	`{{#1\over#2}}`

⟨parameter text⟩ ⇒ document structure

`\frac` defines a document fragment with 2 components

Architecture

Define a reasonable subset of TEX syntax and write our own incremental TEX parser.

Split macro definitions:

`\def`⟨control sequence⟩	`\def\frac`
⟨parameter text⟩	`#1#2`
{⟨replacement text⟩}	`{{#1\over#2}}`

⟨parameter text⟩ ⇒ document structure

`\frac` defines a document fragment with 2 components

⟨replacement text⟩ ⇒ document meaning

Architecture

Define a reasonable subset of TEX syntax and write our own incremental TEX parser.

Split macro definitions:

`\def`⟨control sequence⟩	`\def\frac`
⟨parameter text⟩	`#1#2`
{⟨replacement text⟩}	`{{#1\over#2}}`

⟨parameter text⟩ ⇒ document structure

`\frac` defines a document fragment with 2 components

⟨replacement text⟩ ⇒ document meaning

`\frac#1#2` means `{#1\over#2}`

Architecture

lexer: group characters into tokens

dictionary: define known macros and their syntax

parser: build document structure

transformation engine: interpret the document

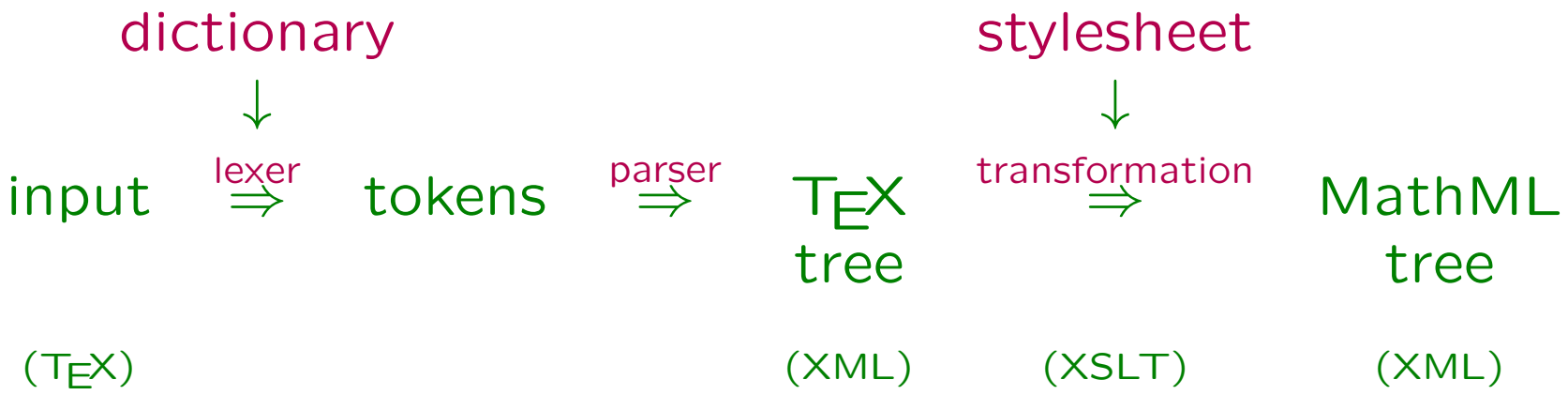
Architecture

lexer: group characters into tokens

dictionary: define known macros and their syntax

parser: build document structure

transformation engine: interpret the document



Revisiting TEX syntax

Plan:

- identify a subset of TEX syntax suitable for editing mathematical formulas
- have compatibility with the "full" syntax
- recognize the largest possible number of constructs
- have small and simple parser implementation

Revisiting TEX syntax: parameters

Classification of parameters:

`\frac12`	simple parameters
`\root n+1\of x`	delimited parameter
`\sqrt[n+1]x`	optional parameter
`{\rm 1+x}`	compound parameter
`a\sb b, a\sp b`	preceding simple parameters
`{1+x\over1+y}`	preceding compound parameter

Revisiting TEX syntax: parameters

Classification of parameters:

`\frac12`	simple parameters
`\root n+1\of x`	delimited parameter
`\sqrt[n+1]x`	optional parameter
`{\rm 1+x}`	compound parameter
`a\sb b, a\sp b`	preceding simple parameters
`{1+x\over1+y}`	preceding compound parameter

$type$ ::= simple | compound | optional | delimited(t)

Dictionary

The dictionary associates macro names with their signatures

It determines well-formed documents

Dictionary

The dictionary associates macro names with their signatures

It determines well-formed documents

Macro	Signature	
`sqrt`		[simple]
`bgroup`		[delimited(`\egroup`)]
`root`		[delimited(`\of`); simple]
`rm, bf, tt, it`		[compound]
`left`		[simple; delimited(`\right`); simple]
`sb, sp`	[simple]	[simple]
`over, choose`	[compound]	[compound]

Lexer

$$token ::= \mathsf{literal}(v) \mid \mathsf{v}_{\langle p_1, p_2 \rangle}$$

Lexer

$$token ::= \text{literal}(v) \mid \texttt{\textbackslash v}_{\langle p_1, p_2 \rangle}$$

- group characters into tokens ($\texttt{\textbackslash}, \texttt{s}, \texttt{p} \Rightarrow \texttt{\textbackslash sp}$)

Lexer

$$token ::= \text{literal}(v) \mid \texttt{\textbackslash v}_{\langle p_1, p_2 \rangle}$$

- group characters into tokens ($\texttt{\textbackslash}, \texttt{s}, \texttt{p} \Rightarrow \texttt{\textbackslash sp}$)
- map control sequences into literals ($\texttt{\textbackslash alpha} \Rightarrow \text{literal}(\alpha)$)

Lexer

$$token ::= \text{literal}(v) \mid \texttt{\textbackslash v}_{\langle p_1, p_2 \rangle}$$

- group characters into tokens ($\texttt{\textbackslash}, \texttt{s}, \texttt{p} \Rightarrow \texttt{\textbackslash sp}$)
- map control sequences into literals ($\texttt{\textbackslash alpha} \Rightarrow \text{literal}(\alpha)$)
- map characters into control sequences ($\texttt{\{} \Rightarrow \texttt{\textbackslash bgroup}$, $\texttt{\}} \Rightarrow \texttt{\textbackslash egroup}$, $\texttt{\_} \Rightarrow \texttt{\textbackslash sb}$)

Lexer

$$token ::= \text{literal}(v) \mid \texttt{\textbackslash v}_{\langle p_1, p_2 \rangle}$$

- group characters into tokens ($\texttt{\textbackslash}, \texttt{s}, \texttt{p} \Rightarrow \texttt{\textbackslash sp}$)
- map control sequences into literals ($\texttt{\textbackslash alpha} \Rightarrow \text{literal}(\alpha)$)
- map characters into control sequences ($\texttt{\{} \Rightarrow \texttt{\textbackslash bgroup}$, $\texttt{\}} \Rightarrow \texttt{\textbackslash egroup}$, $\texttt{\_} \Rightarrow \texttt{\textbackslash sb}$)
- annotate control sequences ($\texttt{\textbackslash bgroup}_{\langle [], \text{delimited}(\texttt{\textbackslash egroup}) \rangle}$)

Parsing example

```
{\alpha^2\over2}
```

Parsing example

```
{\alpha^2\over2}
```

Token stream

- \bgroup ⟨[],[delimited(\egroup)]⟩
- literal(α)
- \sp ⟨simple,simple⟩
- literal(2)
- \over ⟨compound,compound⟩
- literal(2)
- \egroup

TEX tree

- \bgroup
  - \over
    - \sp
      - literal(α)
      - literal(2)
    - literal(2)

Parsing example (continued)

```
<t:tex xmlns:t="http://helm.cs.unibo.it/2002/TML">
 <t:macro id="I2" name="bgroup">
  <t:p>
   <t:macro id="I6" name="over">
    <t:p>
     <t:macro id="I4" name="sp">
      <t:p>
       <t:literal id="I3" name="alpha">&#x3B1;</t:literal>
      </t:p>
      <t:p> <t:literal id="I5">2</t:literal> </t:p>
     </t:macro>
    </t:p>
    <t:p> <t:literal id="I7">2</t:literal> </t:p>
   </t:macro>
  </t:p>
 </t:macro>
</t:tex>
```

Parsing example (continued)

```
<t:tex xmlns:t="http://helm.cs.unibo.it/2002/TML">
 <t:macro id="I2" name="bgroup">
  <t:p>
   <t:macro id="I6" name="over">
    <t:p>
     <t:macro id="I4" name="sp">
      <t:p>
       <t:literal id="I3" name="alpha">&#x3B1;</t:literal>
      </t:p>
      <t:p> <t:literal id="I5">2</t:literal> </t:p>
     </t:macro>
    </t:p>
    <t:p> <t:literal id="I7">2</t:literal> </t:p>
   </t:macro>
  </t:p>
 </t:macro>
</t:tex>
```

Parsing example (continued)

```
<t:tex xmlns:t="http://helm.cs.unibo.it/2002/TML">
 <t:macro id="I2" name="bgroup">
  <t:p>
   <t:macro id="I6" name="over">
    <t:p>
     <t:macro id="I4" name="sp">
      <t:p>
       <t:literal id="I3" name="alpha">&#x3B1;</t:literal>
      </t:p>
      <t:p> <t:literal id="I5">2</t:literal> </t:p>
     </t:macro>
    </t:p>
    <t:p> <t:literal id="I7">2</t:literal> </t:p>
   </t:macro>
  </t:p>
 </t:macro>
</t:tex>
```

Parsing example (continued)

```
<t:tex xmlns:t="http://helm.cs.unibo.it/2002/TML">
 <t:macro id="I2" name="bgroup">
  <t:p>
   <t:macro id="I6" name="over">
    <t:p>
     <t:macro id="I4" name="sp">
      <t:p>
       <t:literal id="I3" name="alpha">&#x3B1;</t:literal>
      </t:p>
      <t:p> <t:literal id="I5">2</t:literal> </t:p>
     </t:macro>
    </t:p>
    <t:p> <t:literal id="I7">2</t:literal> </t:p>
   </t:macro>
  </t:p>
 </t:macro>
</t:tex>
```

Error recovery: missing parameters

```
{1\over}
```

Error recovery: missing parameters

```
{1\over}
```

```
<t:tex xmlns:t="http://helm.cs.unibo.it/2002/TML">
 <t:macro id="I2" name="bgroup">
  <t:p>
   <t:macro id="I6" name="over">
    <t:p> <t:literal id="I3">1</t:literal> </t:p>
    <t:p> <t:empty/> </t:p>
   </t:macro>
  </t:p>
 </t:macro>
</t:tex>
```

Error recovery: ambiguity

```
x_1_2
```

Error recovery: ambiguity

x_1_2

```
<t:tex xmlns:t="http://helm.cs.unibo.it/2002/TML">
 <t:macro id="I7" name="sb">
  <t:p>
   <t:macro id="I5" name="sb">
    <t:p> <t:literal id="I4">x</t:literal> </t:p>
    <t:p> <t:literal id="I6">1</t:literal> </t:p>
   </t:macro>
  </t:p>
  <t:p>
   <t:literal id="I8">2</t:literal>
  </t:p>
 </t:macro>
</t:tex>
```

Error recovery: false ambiguity

Not allowed in TEX: `\sqrt\sqrt2`

Error recovery: false ambiguity

Not allowed in TEX: `\sqrt\sqrt2`

Parameters are captured by the innermost macro:

```
<t:tex xmlns:t="http://helm.cs.unibo.it/2002/TML">
 <t:macro id="I3" name="sqrt">
  <t:p>
   <t:macro id="I2" name="sqrt">
    <t:p> <t:literal id="I1">2</t:literal> </t:p>
   </t:macro>
  </t:p>
 </t:macro>
</t:tex>
```

Error recovery: false ambiguity

Not allowed in TEX: `\sqrt\sqrt2`

Parameters are captured by the innermost macro:

```
<t:tex xmlns:t="http://helm.cs.unibo.it/2002/TML">
 <t:macro id="I3" name="sqrt">
  <t:p>
   <t:macro id="I2" name="sqrt">
    <t:p> <t:literal id="I1">2</t:literal> </t:p>
   </t:macro>
  </t:p>
 </t:macro>
</t:tex>
```

Allowed in TEX: `\sqrt\bgroup2\egroup`

Uniform behavior $\Rightarrow$ simpler parser.

Incrementality

Any single user action changes the document:

- \+ an incremental parser reduces the overhead
- − an incremental parser is more complex

Incrementality

Any single user action changes the document:

+ an incremental parser reduces the overhead
- an incremental parser is more complex

We use groups `{...}` for identifying basic blocks:

- groups are black boxes
- groups give fairly good granularity
- limiting re-parsing at group granularity keeps the parser simple

Incrementality: example

```
{1\over2}
```

```
<t:tex xmlns:t="http://helm.cs.unibo.it/2002/TML">
 <t:macro id="I2" name="bgroup">
  <t:p>
   <t:macro id="I4" name="over">
    <t:p> <t:literal id="I3">1</t:literal> </t:p>
    <t:p> <t:literal id="I5">2</t:literal> </t:p>
   </t:macro>
  </t:p>
 </t:macro>
</t:tex>
```

Incrementality: example

```
{1\ove2}
```

```
<t:tex xmlns:t="http://helm.cs.unibo.it/2002/TML">
 <t:macro id="I2" name="bgroup">
  <t:p>
   <t:literal id="I3">1</t:literal>
   <t:macro id="I4" name="ove"/>
   <t:literal id="I5">2</t:literal>
  </t:p>
 </t:macro>
</t:tex>
```

Incrementality: example

```
{1\ove2}
```

```
<t:tex xmlns:t="http://helm.cs.unibo.it/2002/TML">
 <t:macro id="I2" name="bgroup">
  <t:p>
   <t:literal id="I3">1</t:literal>
   <t:macro id="I4" name="ove"/>
   <t:literal id="I5">2</t:literal>
  </t:p>
 </t:macro>
</t:tex>
```

Group components that are unaffected can be recycled.

Transformation

Generate an output document (MathML) according to the structure of the TEX tree.

Transformation

Generate an output document (MathML) according to the structure of the TEX tree.

Why XSLT?

- TEX tree is XML $\Rightarrow$ XSLT is a natural choice
- XSLT is expressive yet simple
- XSLT is easily extensible (template priority)
- XSLT is not restricted to XML output
- XSLT is standard (many implementations are available)

Transformation: example

```
<xsl:template match="t:macro[@name='over']">
 <m:mfrac>
  <xsl:if test="@id">
   <xsl:attribute name="xref">
    <xsl:value-of select="@id"/>
   </xsl:attribute>
  </xsl:if>
  <xsl:apply-templates select="t:p[1]"/>
  <xsl:apply-templates select="t:p[2]"/>
 </m:mfrac>
</xsl:template>
```

```
<t:macro id="I4" name="over">
 <t:p>
  <t:literal id="I3">1</t:literal>
 </t:p>
 <t:p>
  <t:literal id="I5">2</t:literal>
 </t:p>
</t:macro>
```

Transformation: example

```
<xsl:template match="t:macro[@name='over']">
 <m:mfrac>
  <xsl:if test="@id">
   <xsl:attribute name="xref">
    <xsl:value-of select="@id"/>
   </xsl:attribute>
  </xsl:if>
  <xsl:apply-templates select="t:p[1]"/>
  <xsl:apply-templates select="t:p[2]"/>
 </m:mfrac>
</xsl:template>
```

```
<t:macro id="I4" name="over">
 <t:p>
  <t:literal id="I3">1</t:literal>
 </t:p>
 <t:p>
  <t:literal id="I5">2</t:literal>
 </t:p>
</t:macro>
```

```
<m:mfrac>

</m:mfrac>
```

Transformation: example

```
<xsl:template match="t:macro[@name='over']">
 <m:mfrac>
  <xsl:if test="@id">
   <xsl:attribute name="xref">
    <xsl:value-of select="@id"/>
   </xsl:attribute>
  </xsl:if>
  <xsl:apply-templates select="t:p[1]"/>
  <xsl:apply-templates select="t:p[2]"/>
 </m:mfrac>
</xsl:template>
```

```
<t:macro id="I4" name="over">
 <t:p>
  <t:literal id="I3">1</t:literal>
 </t:p>
 <t:p>
  <t:literal id="I5">2</t:literal>
 </t:p>
</t:macro>
```

```
<m:mfrac>

</m:mfrac>
```

Transformation: example

```
<xsl:template match="t:macro[@name='over']">
 <m:mfrac>
  <xsl:if test="@id">
   <xsl:attribute name="xref">
    <xsl:value-of select="@id"/>
   </xsl:attribute>
  </xsl:if>
  <xsl:apply-templates select="t:p[1]"/>
  <xsl:apply-templates select="t:p[2]"/>
 </m:mfrac>
</xsl:template>
```

```
<t:macro id="I4" name="over">
 <t:p>
  <t:literal id="I3">1</t:literal>
 </t:p>
 <t:p>
  <t:literal id="I5">2</t:literal>
 </t:p>
</t:macro>
```

```
<m:mfrac xref="I4">


</m:mfrac>
```

Transformation: example

```
<xsl:template match="t:macro[@name='over']">
 <m:mfrac>
  <xsl:if test="@id">
   <xsl:attribute name="xref">
    <xsl:value-of select="@id"/>
   </xsl:attribute>
  </xsl:if>
  <xsl:apply-templates select="t:p[1]"/>
  <xsl:apply-templates select="t:p[2]"/>
 </m:mfrac>
</xsl:template>
```

```
<t:macro id="I4" name="over">
 <t:p>
  <t:literal id="I3">1</t:literal>
 </t:p>
 <t:p>
  <t:literal id="I5">2</t:literal>
 </t:p>
</t:macro>
```

```
<m:mfrac xref="I4">

</m:mfrac>
```

Transformation: example

```
<xsl:template match="t:macro[@name='over']">
 <m:mfrac>
  <xsl:if test="@id">
   <xsl:attribute name="xref">
    <xsl:value-of select="@id"/>
   </xsl:attribute>
  </xsl:if>
  <xsl:apply-templates select="t:p[1]"/>
  <xsl:apply-templates select="t:p[2]"/>
 </m:mfrac>
</xsl:template>
```

```
<t:macro id="I4" name="over">
 <t:p>
  <t:literal id="I3">1</t:literal>
 </t:p>
 <t:p>
  <t:literal id="I5">2</t:literal>
 </t:p>
</t:macro>
```

```
<m:mfrac xref="I4">
 <m:mn xref="I3">1</m:mn>

</m:mfrac>
```

Transformation: example

```
<xsl:template match="t:macro[@name='over']">
 <m:mfrac>
  <xsl:if test="@id">
   <xsl:attribute name="xref">
    <xsl:value-of select="@id"/>
   </xsl:attribute>
  </xsl:if>
  <xsl:apply-templates select="t:p[1]"/>
  <xsl:apply-templates select="t:p[2]"/>
 </m:mfrac>
</xsl:template>
```

```
<t:macro id="I4" name="over">
 <t:p>
  <t:literal id="I3">1</t:literal>
 </t:p>
 <t:p>
  <t:literal id="I5">2</t:literal>
 </t:p>
</t:macro>
```

```
<m:mfrac xref="I4">
 <m:mn xref="I3">1</m:mn>
 <m:mn xref="I5">2</m:mn>
</m:mfrac>
```

Another template

`a^b_c` $\sb(\sp(a, b), c)$

Another template

a^b_c \sb(\sp(a, b), c)

```
<xsl:template
 match="macro[@name='sb'][p[1]/*[1][self::macro[@name='sp']]]">
 <m:msubsup>
  <xsl:if test="@id">
   <xsl:attribute name="xref">
    <xsl:value-of select="@id"/>
   </xsl:attribute>
  </xsl:if>
  <xsl:apply-templates select="p[1]/*/p[1]"/>
  <xsl:apply-templates select="p[2]"/>
  <xsl:apply-templates select="p[1]/*/p[2]"/>
 </m:msubsup>
</xsl:template>
```

Incremental transformation

Incremental transformation

Groups {...} define reasonable context-free fragments that can be transformed in isolation.

Incremental transformation

Groups `{...}` define reasonable context-free fragments that can be transformed in isolation.

Algorithm:

1. Identify the smallest group in the TEX tree to be transformed (look up the `id` attribute)

Incremental transformation

Groups `{...}` define reasonable context-free fragments that can be transformed in isolation.

Algorithm:

1. Identify the smallest group in the TEX tree to be transformed (look up the `id` attribute)
2. Transform that group

Incremental transformation

Groups {...} define reasonable context-free fragments that can be transformed in isolation.

Algorithm:

1. Identify the smallest group in the TEX tree to be transformed (look up the `id` attribute)
2. Transform that group
3. Substitute the resulting fragment in the MathML tree (search for the `xref` attribute)

Incremental transformation: example

```
\frac{1}2
```

```
<t:macro id="I1" name="frac">
 <t:p>
  <t:macro id="I2" name="bgroup">
   <t:p>
    <t:literal id="I3">1</t:literal>


   </t:p>
  </t:macro>
 </t:p>
 <t:p>
  <t:literal id="I4">2</t:literal>
 </t:p>
</t:macro>
```

```
<m:mfrac xref="I1">
 <m:mrow xref="I2">
  <m:mn xref="I3">1</m:mn>
 </m:mrow>
 <m:mn xref="I4">2</mn>
</m:mfrac>
```

Incremental transformation: example

\frac{1}2 $\Rightarrow$ \frac{1+x}2

```
<t:macro id="I1" name="frac">
 <t:p>
  <t:macro id="I2" name="bgroup">
   <t:p>
    <t:literal id="I3">1</t:literal>
    <t:literal id="I5">+</t:literal>
    <t:literal id="I6">x</t:literal>
   </t:p>
  </t:macro>
 </t:p>
 <t:p>
  <t:literal id="I4">2</t:literal>
 </t:p>
</t:macro>
```

```
<m:mfrac xref="I1">
 <m:mrow xref="I2">
  <m:mn xref="I3">1</m:mn>
 </m:mrow>
 <m:mn xref="I4">2</mn>
</m:mfrac>
```

Incremental transformation: example

\frac{1}2 $\Rightarrow$ \frac{1+x}2

```
<t:macro id="I1" name="frac">
 <t:p>
  <t:macro id="I2" name="bgroup">
   <t:p>
    <t:literal id="I3">1</t:literal>
    <t:literal id="I5">+</t:literal>
    <t:literal id="I6">x</t:literal>
   </t:p>
  </t:macro>
 </t:p>
 <t:p>
  <t:literal id="I4">2</t:literal>
 </t:p>
</t:macro>
```

```
<m:mfrac xref="I1">
 <m:mrow xref="I2">
  <m:mn xref="I3">1</m:mn>
 </m:mrow>
 <m:mn xref="I4">2</mn>
</m:mfrac>

<m:mrow xref="I2">
 <m:mn xref="I3">1</m:mn>
 <m:mo xref="I5">+</m:mo>
 <m:mi xref="I6">x</m:mi>
</m:mrow>
```

Incremental transformation: example

\frac{1}2 $\Rightarrow$ \frac{1+x}2

```
<t:macro id="I1" name="frac">
 <t:p>
  <t:macro id="I2" name="bgroup">
   <t:p>
    <t:literal id="I3">1</t:literal>
    <t:literal id="I5">+</t:literal>
    <t:literal id="I6">x</t:literal>
   </t:p>
  </t:macro>
 </t:p>
 <t:p>
  <t:literal id="I4">2</t:literal>
 </t:p>
</t:macro>
```

```
<m:mfrac xref="I1">
 <m:mrow xref="I2">
  <m:mn xref="I3">1</m:mn>
  <m:mo xref="I5">+</m:mo>
  <m:mi xref="I6">x</m:mi>
 </m:mrow>
 <m:mn xref="I4">2</mn>
</m:mfrac>
```

Custom lexers

Neat separation between lexer and parser

$\Rightarrow$ the lexer can be replaced too!

Custom lexers

Neat separation between lexer and parser

$\Rightarrow$ the lexer can be replaced too!

- have different rules regarding spaces
  (ignorable vs significant, see λ-calculus)

Custom lexers

Neat separation between lexer and parser

$\Rightarrow$ the lexer can be replaced too!

- have different rules regarding spaces
  (ignorable vs significant, see λ-calculus)
- have different rules regarding identifiers
  (one-letter, multi-letter, decorated, . . .)

Custom lexers

Neat separation between lexer and parser

$\Rightarrow$ the lexer can be replaced too!

- have different rules regarding spaces
  (ignorable vs significant, see λ-calculus)
- have different rules regarding identifiers
  (one-letter, multi-letter, decorated, . . .)
- force balanced fences
  (treat `(` like `\left(` and `)` like `\right)`)

Custom lexers

Neat separation between lexer and parser

$\Rightarrow$ the lexer can be replaced too!

- have different rules regarding spaces
  (ignorable vs significant, see λ-calculus)
- have different rules regarding identifiers
  (one-letter, multi-letter, decorated, . . .)
- force balanced fences
  (treat `(` like `\left(` and `)` like `\right)`)
- TAB expansion
  (`\Longr` + TAB $\Rightarrow$ `\Longrightarrow`)

Final remarks

We have

- identified an interesting, non-trivial subset of TEX syntax
- implemented a parser that is small, simple, formally defined, incremental
- written an MathML generator using XSLT
- obtained an interactive editor that can be used as a light-weight frontend

Final remarks

We have

- identified an interesting, non-trivial subset of TEX syntax
- implemented a parser that is small, simple, formally defined, incremental
- written an MathML generator using XSLT
- obtained an interactive editor that can be used as a light-weight frontend

Final remarks

We have

- identified an interesting, non-trivial subset of TEX syntax
- implemented a parser that is small, simple, formally defined, incremental
- written an MathML generator using XSLT
- obtained an interactive editor that can be used as a light-weight frontend

EdiTEX: http://helm.cs.unibo.it/editex