

Editing MathML (ab)using T_EX 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.

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

```
\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.

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

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

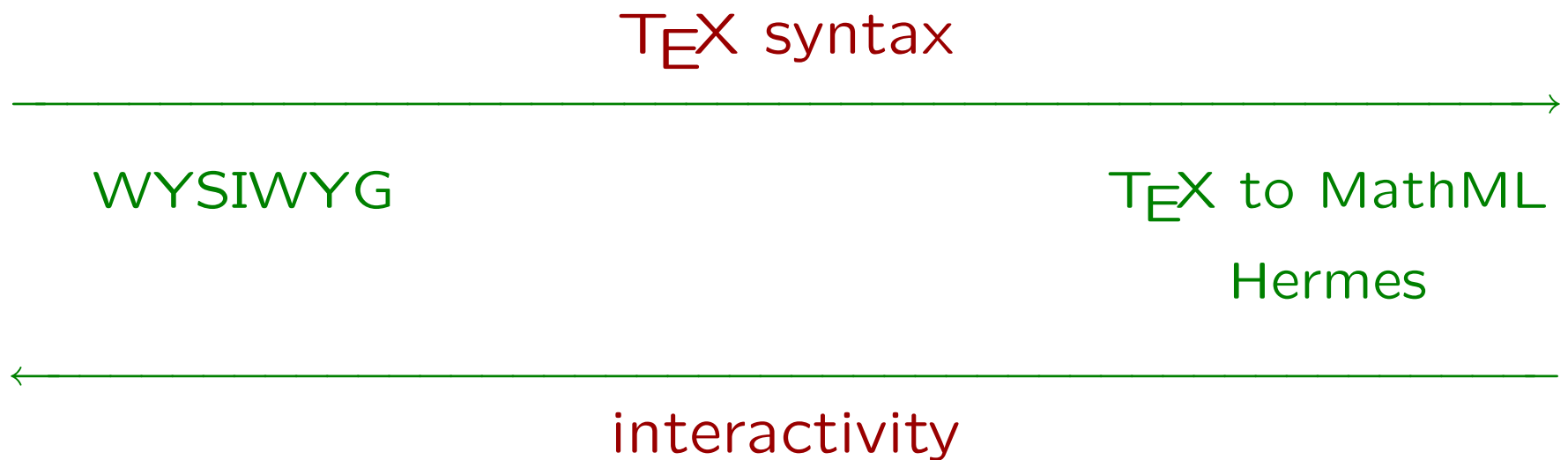
- batch processing
- require knowledge of T_EX syntax
- separate editing window and view

WYSIWYG editors drawbacks

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

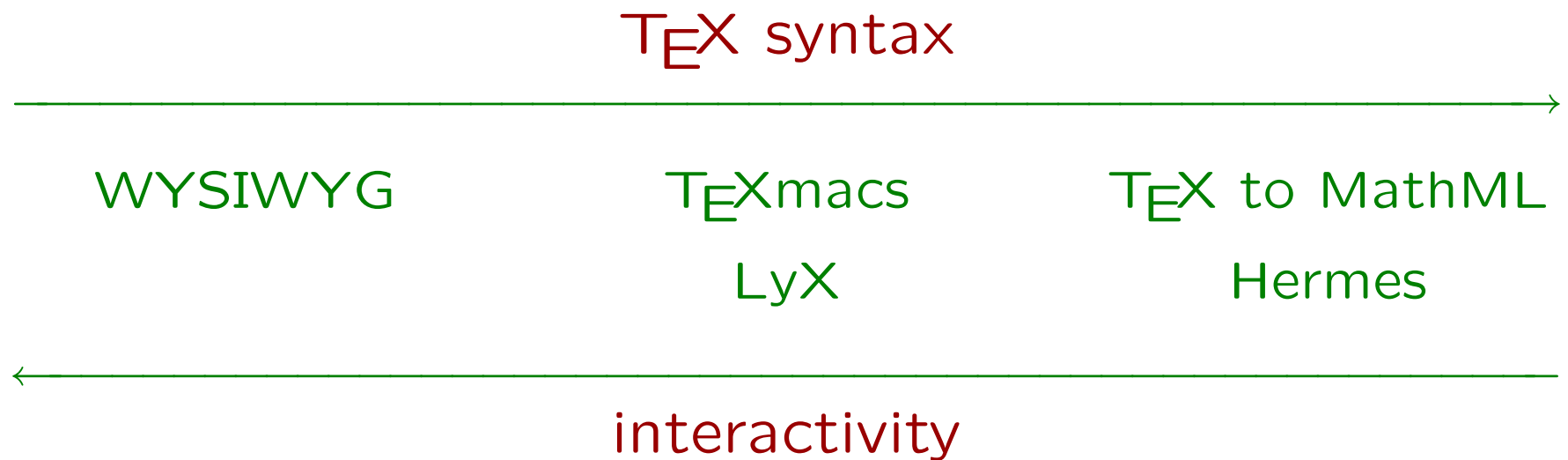
WYSIWYG editors drawbacks

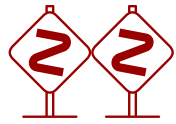
- slow editing, especially for complex markup
- no possibility of improving
- limited extensibility
- besides, they have usability issues



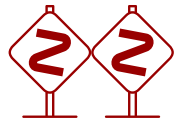
WYSIWYG editors drawbacks

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





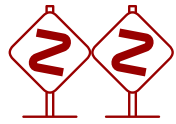
Combine the best (worst?) of both worlds: WYSIWYG editor controlled by T_EX syntax.



Combine the best (worst?) of both worlds: WYSIWYG editor controlled by $\text{T}_\text{E}\text{X}$ syntax.

Why $\text{T}_\text{E}\text{X}$?

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



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

Why T_EX ?

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

However the original T_EX parser cannot be used:

- **large**
- not suitable for **incremental processing**

Architecture

Define a reasonable subset of $\text{T}_{\text{E}}\text{X}$ syntax and write our own incremental $\text{T}_{\text{E}}\text{X}$ parser.

Architecture

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

Split macro definitions:

```
\def⟨control sequence⟩
```

```
⟨parameter text⟩
```

```
{⟨replacement text⟩}
```

```
\def\frac
```

```
#1#2
```

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

Architecture

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

Split macro definitions:

`\def` <control sequence>

<parameter text>

{<replacement text>}

`\def` \frac

#1#2

{{#1\over#2}}

<parameter text> \Rightarrow document structure

Architecture

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

Split macro definitions:

<code>\def</code>	<code><control sequence></code>	<code>\def</code>	<code>\frac</code>
	<code><parameter text></code>		<code>#1#2</code>
	<code>{<replacement text>}</code>		<code>{{#1\over#2}}</code>

`<parameter text>` \Rightarrow document structure

`\frac` defines a document fragment with 2 components

Architecture

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

Split macro definitions:

`\def` <control sequence>

<parameter text>

{<replacement text>}

`\def` \frac

#1#2

{{#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 T_EX syntax and write our own incremental T_EX parser.

Split macro definitions:

<code>\def</code>	<code><control sequence></code>	<code>\def</code>	<code>\frac</code>
	<code><parameter text></code>		<code>#1#2</code>
	<code>{<replacement text>}</code>		<code>{{#1\over#2}}</code>

`<parameter text>` \Rightarrow document structure

`\frac` defines a document fragment with 2 components

`<replacement text>` \Rightarrow 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

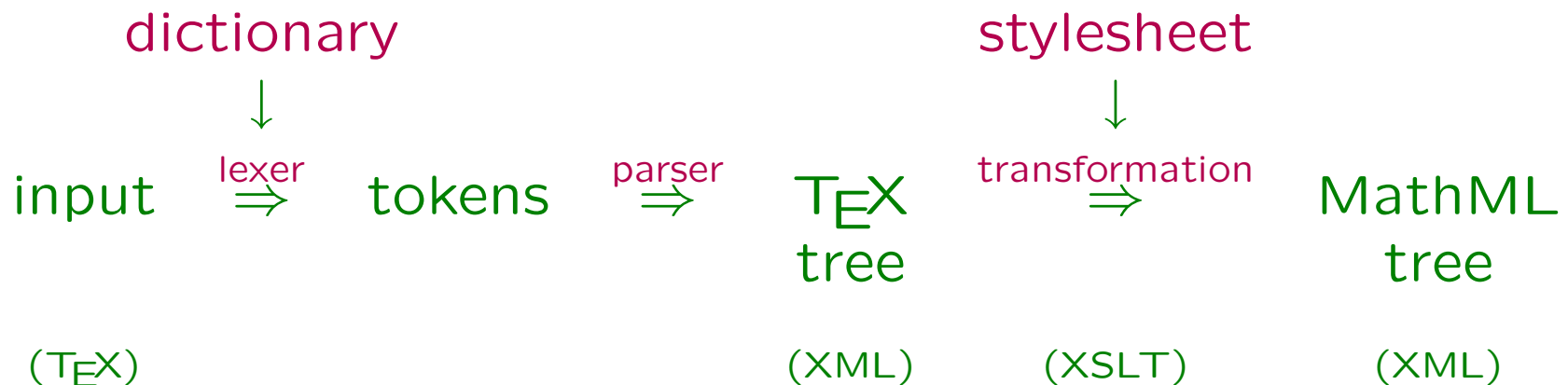
Architecture

lexer: group characters into tokens

dictionary: define known macros and their syntax

parser: build document structure

transformation engine: interpret the document



Revisiting T_EX syntax

Plan:

- identify a **subset** of T_EX 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 T_EX syntax: parameters

Classification of parameters:

<code>\frac</code> 12	simple parameters
<code>\root</code> n+1 <code>\of</code> x	delimited parameter
<code>\sqrt</code> [n+1]x	optional parameter
<code>{\rm</code> 1+x <code>}</code>	compound parameter
a <code>\sb</code> b, a <code>\sp</code> b	preceding simple parameters
<code>{</code> 1+x <code>\over</code> 1+y <code>}</code>	preceding compound parameter

Revisiting T_EX syntax: parameters

Classification of parameters:

<code>\frac</code> 12	simple parameters
<code>\root</code> n+1 <code>\of</code> x	delimited parameter
<code>\sqrt</code> [n+1]x	optional parameter
<code>{\rm</code> 1+x <code>}</code>	compound parameter
a <code>\sb</code> b, a <code>\sp</code> b	preceding simple parameters
<code>{</code> 1+x <code>\over</code> 1+y <code>}</code>	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

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

Lexer

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

- group characters into **tokens** ($\backslash, s, p \Rightarrow \backslash sp$)

Lexer

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

- group characters into **tokens** ($\backslash, s, p \Rightarrow \backslash sp$)
- map control sequences into **literals**
($\backslash alpha \Rightarrow literal(\alpha)$)

Lexer

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

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

Lexer

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

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

Parsing example

`{\alpha^2\over2}`

Parsing example

`{\alpha^2\over2}`

Token stream

`\bgroup` $\langle [], [delimited(\egroup)] \rangle$

`literal`(α)

`\sp` $\langle simple, simple \rangle$

`literal`(2)

`\over` $\langle compound, compound \rangle$

`literal`(2)

`\egroup`

$\text{T}_{\text{E}}\text{X}$ 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 T_EX: `\sqrt\sqrt2`

Error recovery: false ambiguity

Not allowed in T_EX: `\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 T_EX: `\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 T_EX: `\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

$\{1\over r2\}$

```
<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 T_EX tree.

Transformation

Generate an output document (MathML) according to the structure of the $\text{T}_{\text{E}}\text{X}$ tree.

Why XSLT?

- $\text{T}_{\text{E}}\text{X}$ 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">                                     <m:mfrac>
  <t:p>
    <t:literal id="I3">1</t:literal>
  </t:p>                                                         </m:mfrac>
  <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">                                <m:mfrac>
  <t:p>
    <t:literal id="I3">1</t:literal>
  </t:p>                                                    </m:mfrac>
  <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 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 $\text{sb}(\text{sp}(a, b), c)$

```
<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 $\text{T}_{\text{E}}\text{X}$ 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 T_EX 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 T_EX 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:macro>
<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

⇒ the lexer can be replaced too!

Custom lexers

Neat separation between lexer and parser

⇒ 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

⇒ 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

⇒ 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

⇒ 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 ⇒ `\Longrightarrow`)

Final remarks

We have

- identified an interesting, non-trivial subset of $\text{T}_{\text{E}}\text{X}$ 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 $\text{T}_{\text{E}}\text{X}$ 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 $\text{T}_{\text{E}}\text{X}$ 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

Edi $\text{T}_{\text{E}}\text{X}$: <http://helm.cs.unibo.it/editex>