`sproof.sty`: Structural Markup for Proofs[*]

Michael Kohlhase
Jacobs University, Bremen
`http://kwarc.info/kohlhase`

July 20, 2010

Abstract

The `sproof` package is part of the STEX collection, a version of TEX/LATEX that allows to markup TEX/LATEX documents semantically without leaving the document format, essentially turning TEX/LATEX into a document format for mathematical knowledge management (MKM).

This package supplies macros and environment that allow to annotate the structure of mathematical proofs in STEX files. This structure can be used by MKM systems for added-value services, either directly from the STEX sources, or after translation.

---

[*]Version v0.9f (last revised 2010/06/25)

Contents

1 Introduction

The `sproof` (semantic proofs) package supplies macros and environment that allow to annotate the structure of mathematical proofs in SₜₑX files. This structure can be used by MKM systems for added-value services, either directly from the SₜₑX sources, or after translation. Even though it is part of the SₜₑX collection, it can be used independently, like it's sister package `statements`.

SₜₑX is a version of TₑX/LᴬTₑX that allows to markup TₑX/LᴬTₑX documents semantically without leaving the document format, essentially turning TₑX/LᴬTₑX into a document format for mathematical knowledge management (MKM).

```
% \begin{sproof}[id=simple-proof,for=sum-over-odds]
%    {We prove that $\sum_{i=1}^n{2i-1}=n^{2}$ by induction over $n$}
%  \begin{spfcases}{For the induction we have to consider the following cases:}
%    \begin{spfcase}{$n=1$}
%     \begin{spfstep}[display=flow] then we compute $1=1^2$\end{spfstep}
%    \end{spfcase}
%    \begin{spfcase}{$n=2$}
%       \begin{sproofcomment}[display=flow]
%         This case is not really necessary, but we do it for the
%         fun of it (and to get more intuition).
%       \end{sproofcomment}
%       \begin{spfstep}[display=flow] We compute $1+3=2^{2}=4$.\end{spfstep}
%    \end{spfcase}
%    \begin{spfcase}{$n>1$}
%       \begin{spfstep}[type=assumption,id=ind-hyp]
%         Now, we assume that the assertion is true for a certain $k\geq 1$,
%         i.e. $\sum_{i=1}^k{(2i-1)}=k^{2}$.
%       \end{spfstep}
%       \begin{sproofcomment}
%         We have to show that we can derive the assertion for $n=k+1$ from
%         this assumption, i.e. $\sum_{i=1}^{k+1}{(2i-1)}=(k+1)^{2}$.
%       \end{sproofcomment}
%       \begin{spfstep}
%         We obtain $\sum_{i=1}^{k+1}{2i-1}=\sum_{i=1}^k{2i-1}+2(k+1)-1$
%         \begin{justification}[method=arith:split-sum]
%           by splitting the sum.
%         \end{justification}
%       \end{spfstep}
%       \begin{spfstep}
%         Thus we have $\sum_{i=1}^{k+1}{(2i-1)}=k^2+2k+1$
%         \begin{justification}[method=fertilize] by inductive hypothesis.\end{justification}
%       \end{spfstep}
%       \begin{spfstep}[type=conclusion]
%         We can \begin{justification}[method=simplify]simplify\end{justification}
%         the right-hand side to ${k+1}^2$, which proves the assertion.
%       \end{spfstep}
%    \end{spfcase}
%     \begin{spfstep}[type=conclusion]
%       We have considered all the cases, so we have proven the assertion.
%     \end{spfstep}
%   \end{spfcases}
% \end{sproof}
%
```

Example 1: A very explicit proof, marked up semantically

We will go over the general intuition by way of our running example (see Figure 1 for the source and Figure 2 for the formatted result).[1]

EdNote(1)

---

[1]EᴅNᴏᴛᴇ: talk a bit more about proofs and their structure,... maybe copy from OMDoc spec.

3

2 The User Interface

2.1 Proofs and Proof steps

sproof   The `proof` environment is the main container for proofs. It takes an optional `KeyVal` argument that allows to specify the `id` (identifier) and `for` (for which assertion is this a proof) keys. The regular argument of the `proof` environment contains an introductory comment, that may be used to announce the proof style. The `proof` environment contains a sequence of `\step`, `proofcomment`, and `pfcases` environments that are used to markup the proof steps. The `proof` environment has a variant `Proof`, which does not use the proof end marker. This is convenient, if a proof ends in a case distinction, which brings it's own proof end marker with

sProof   it. The `Proof` environment is a variant of `proof` that does not mark the end of a proof with a little box; presumably, since one of the subproofs already has one and then a box supplied by the outer proof would generate an otherwise empty

\spfidea   line. The `\spfidea` macro allows to give a one-paragraph description of the proof idea.

spfsketch   For one-line proof sketches, we use the `\spfsketch` macro, which takes the `KeyVal` argument as `sproof` and another one: a natural language text that sketches the proof.

spfstep   Regular proof steps are marked up with the `step` environment, which takes an optional `KeyVal` argument for annotations. A proof step usually contains a local assertion (the text of the step) together with some kind of evidence that this can be derived from already established assertions.

     Note that both `\premise` and `\justarg` can be used with an empty second argument to mark up premises and arguments that are not explicitly mentioned in the text.

2.2 Justifications

justification   This evidence is marked up with the `justification` environment in the `sproof` package. This environment totally invisible to the formatted result; it wraps the text in the proof step that corresponds to the evidence. The environment takes an optional `KeyVal` argument, which can have the `method` key, whose value is the name of a proof method (this will only need to mean something to the application that consumes the semantic annotations). Furthermore, the justification can contain "premises" (specifications to assertions that were used justify the step) and "arguments" (other information taken into account by the proof method).

\premise   The `\premise` macro allows to mark up part of the text as reference to an assertion that is used in the argumentation. In the example in Figure 1 we have used the `\premise` macro to identify the inductive hypothesis.

\justarg   The `\justarg` macro is very similar to `\premise` with the difference that it is used to mark up arguments to the proof method. Therefore the content of the first argument is interpreted as a mathematical object rather than as an identifier as in the case of `\premise`. In our example, we specified that the simplification should take place on the right hand side of the equation. Other examples include

Proof: We prove that $\sum_{i=1}^{n} 2i - 1 = n^2$ by induction over n

P.1 For the induction we have to consider the following cases:

P.1.1 $n = 1$: then we compute $1 = 1^2$

P.1.2 $n = 2$: This case is not really necessary, but we do it for the fun of it (and to get more intuition). We compute $1 + 3 = 2^2 = 4$

P.1.3 $n > 1$:

P.1.3.1 Now, we assume that the assertion is true for a certain $k \geq 1$, i.e. $\sum_{i=1}^{k} (2i - 1) = k^2$.

P.1.3.2 We have to show that we can derive the assertion for $n = k + 1$ from this assumption, i.e. $\sum_{i=1}^{k+1} (2i - 1) = (k + 1)^2$.

P.1.3.3 We obtain $\sum_{i=1}^{k+1} (2i - 1) = \sum_{i=1}^{k} (2i - 1) + 2(k + 1) - 1$ by splitting the sum

P.1.3.4 Thus we have $\sum_{i=1}^{k+1} (2i - 1) = k^2 + 2k + 1$ by inductive hypothesis.

P.1.3.5 We can simplify the right-hand side to $k + 1^2$, which proves the assertion. □

P.1.4 We have considered all the cases, so we have proven the assertion. □

Example 2: The formatted result of the proof in Figure 1

proof methods that instantiate. Here we would indicate the substituted object in a `\justarg` macro.

2.3 Proof Structure

spfcases   The `pfcases` environment is used to mark up a proof by cases. This environment takes an optional `KeyVal` argument for semantic annotations and a second argument that allows to specify an introductory comment (just like in the `proof` environment).

spfcase   The content of a `pfcases` environment are a sequence of case proofs marked up in the `pfcase` environment, which takes an optional `KeyVal` argument for semantic annotations. The second argument is used to specify the the description of the case under consideration. The content of a `pfcase` environment is the same as that of a `proof`, i.e. `step`s, `proofcomment`s, and `pfcases` environments.

sproofcomment   The `proofcomment` environment is much like a `step`, only that it does not have an object-level assertion of its own. Rather than asserting some fact that is relevant for the proof, it is used to explain where the proof is going, what we are attempting to to, or what we have achieved so far. As such, it cannot be the target of a `\premise`.

2.4 Proof End Markers

Traditionally, the end of a mathematical proof is marked with a little box at the end of the last line of the proof (if there is space and on the end of the next line if there isn't), like so: □

\sproofend   The `sproof` package provides the `\sproofend` macro for this. If a different symbol for the proof end is to be used (e.g. *q.e.d*), then this can be obtained by

\sProofEndSymbol   specifying it using the `\sProofEndSymbol` configuration macro (e.g. by specifying `\sProofEndSymbol{q.e.d}`).

Some of the proof structuring macros above will insert proof end symbols for sub-proofs, in most cases, this is desirable to make the proof structure explicit, but sometimes this wastes space (especially, if a proof ends in a case analysis which will supply its own proof end marker). To suppress it locally, just set `proofend={}` in them or use use `\sProofEndSymbol{}`.

6

3 The Implementation

The sproof package generates to files: the LaTeX package (all the code between ⟨*package⟩ and ⟨/package⟩) and the LaTeXML bindings (between ⟨*ltxml⟩ and ⟨/ltxml⟩). We keep the corresponding code fragments together, since the documentation applies to both of them and to prevent them from getting out of sync.

We first set up the Perl Packages for LaTeXML

```
1 ⟨∗ltxml⟩
2 # -*- CPERL -*-
3 package LaTeXML::Package::Pool;
4 use strict;
5 use LaTeXML::Package;
6 RequirePackage('sref');
7 ⟨/ltxml⟩
```

Then we make sure that the STEX sref package is loaded [Koh].

```
8 ⟨∗package⟩
9 \RequirePackage{sref}
10 ⟨/package⟩
```

3.1 Proofs

We first define some keys for the proof environment.

```
11 ⟨∗package⟩
12 \srefaddidkey{spf}
13 \omdaddkey{spf}{display}
14 \omdaddkey{spf}{for}
15 \omdaddkey{spf}{from}
16 \omdaddkey[\sproof@box]{spf}{proofend}
17 \omdaddkey{spf}{type}
18 \omdaddkey{spf}{title}
19 \omdaddkey{spf}{continues}
20 ⟨/package⟩
```

\spf@flow   We define this macro, so that we can test whether the display key has the value flow

```
21 ⟨package⟩\def\spf@flow{flow}
```

For proofs, we will have to have deeply nested structures of enumerated list-like environments. However, LaTeX only allows enumerate environments up to nesting depth 4 and general list environments up to listing depth 6. This is not enough for us. Therefore we have decided to go along the route proposed by Leslie Lamport to use a single top-level list with dotted sequences of numbers to identify the position in the proof tree. Unfortunately, we could not use his pf.sty package directly, since it does not do automatic numbering, and we have to add keyword arguments all over the place, to accomodate semantic information.

pst@with@label   This environment manages[1] the path labeling of the proof steps in the description

---

[1] This gets the labeling right but only works 8 levels deep

environment of the outermost `proof` environment. The argument is the label prefix up to now; which we cache in `\pst@label` (we need evaluate it first, since are in the right place now!). Then we increment the proof depth which is stored in `\count10` (lower counters are used by TeX for page numbering) and initialize the next level counter `\count\count10` with 1. In the end call for this environment, we just decrease the proof depth counter by 1 again.

```
22 ⟨∗package⟩
23 \newenvironment{pst@with@label}[1]{\edef\pst@label{#1}\advance\count10 by 1\count\count10=1}
24 {\advance\count10 by -1}
```

`\the@pst@label`   `\the@pst@label` evaluates to the current step label.

```
25 \def\the@pst@label{\pst@label.\number\count\count10}
```

`\next@pst@label`   `\next@pst@label` increments the step label at the current level.

```
26 \def\next@pst@label{\global\advance\count\count10 by 1}
```

`\sproofend`   This macro places a little box at the end of the line if there is space, or at the end of the next line if there isn't

```
27 \def\sproof@box{\hbox{\vrule\vbox{\hrule width 6 pt\vskip 6pt\hrule}\vrule}}
28 \def\spf@proofend{\sproof@box}
29 \def\sproofend{\ifx\spf@proofend\@empty\else\hfil\null\nobreak\hfill\spf@proofend\par\smallskip
30 \def\sProofEndSymbol#1{\def\sproof@box{#1}}
31 ⟨/package⟩
32 ⟨ltxml⟩DefConstructor('\sproofend',"");
```

spfsketch

```
33 ⟨∗package⟩
34 \newcommand{\spfsketch}[2][]{\omdsetkeys{spf}{#1}\sref@target
35 \ifx\spf@display\spf@flow\else{\stDMemph{\ifx\spf@type\@empty{Proof}\else\spf@type\fi}:}\fi{ #2
36 \sref@label@id{this \ifx\spf@type\@empty{Proof}\else\spf@type\fi}}
37 ⟨/package⟩
38 ⟨∗ltxml⟩
39 DefConstructor('\spfsketch OptionalKeyVals:pf{}',
40         "<omdoc:proof "
41                  . "?&defined(&KeyVal(#1,'id'))(xml:id='&KeyVal(#1,'id')')()>\n"
42         . "?#2(<omdoc:omtext>"
43                  . "<omdoc:CMP><omdoc:p>#2</omdoc:p></omdoc:CMP>"
44                  . "</omdoc:omtext>\n)()"
45                  . "</omdoc:proof>\n");
46 DefConstructor('\sProofEndSymbol {}','');
47 ⟨/ltxml⟩
```

sproof   In this environment, we initialize the proof depth counter `\count10` to 10, and set up the description environment that will take the proof steps. At the end of the proof, we position the proof end into the last line.

```
48 ⟨∗package⟩
49 \newenvironment{@proof}[2][]{\omdsetkeys{spf}{#1}\sref@target
50 \count10=10
```

8

```
51 \ifx\spf@display\spf@flow\else{\stDMemph{\ifx\spf@type\@empty{Proof}\else\spf@type\fi}:}\fi{ #2
52 \sref@label@id{this \ifx\spf@type\@empty{Proof}\else\spf@type\fi}
53 \def\pst@label{}\newcount\pst@count% initialize the labeling mechanism
54 \begin{description}\begin{pst@with@label}{P}}
55 {\end{pst@with@label}\end{description}}
56 \newenvironment{sproof}[2][]{\begin{@proof}[#1]{#2}}{\sproofend\end{@proof}}
57 \newenvironment{sProof}[2][]{\begin{@proof}[#1]{#2}}{\end{@proof}}
58 ⟨/package⟩
59 ⟨∗ltxml⟩
60 DefCMPEnvironment('{sproof} OptionalKeyVals:pf{}',
61      "<omdoc:proof "
62           . "?&KeyVal(#1,'for')(for='&hash_wrapper(&KeyVal(#1,'for'))')()"
63           . "?&defined(&KeyVal(#1,'id'))(xml:id='&KeyVal(#1,'id')')()>\n"
64      . "?#2(<omdoc:omtext>"
65           . "<omdoc:CMP><omdoc:p>#2</omdoc:p></omdoc:CMP>"
66           . "</omdoc:omtext>\n)()"
67           . "#body"
68      . "</omdoc:proof>\n");
69 DefMacro('\sProof','\sproof');
70 DefMacro('\endsProof','\endsproof');
71 ⟨/ltxml⟩
```

spfidea

```
72 ⟨∗package⟩
73 \newcommand{\spfidea}[2][]{\omdsetkeys{spf}{#1}%
74 \stDMemph{\ifx\spf@type\@empty{Proof Idea}\else\spf@type\fi:} #2\sproofend}
75 ⟨/package⟩
76 ⟨∗ltxml⟩
77 DefCMPConstructor('\spfidea OptionalKeyVals:pf {}',
78      "<omdoc:proof "
79           . "?&defined(&KeyVal(#1,'id'))(xml:id='&KeyVal(#1,'id')')()"
80           . "?&KeyVal(#1,'for')(for='&hash_wrapper(&KeyVal(#1,'for'))')()>\n"
81      . "<omdoc:omtext><omdoc:CMP><omdoc:p>#2</omdoc:p></omdoc:CMP></omdoc:omtext>\n"
82      . "</omdoc:proof>\n");
83 ⟨/ltxml⟩
```

The next two environments (proof steps) and comments, are mostly semantical, they take KeyVal arguments that specify their semantic role. In draft mode, they read these values and show them. If the surrounding proof had display=flow, then no new \item is generated, otherwise it is. In any case, the proof step number (at the current level) is incremented.

spfstep

```
84 ⟨∗package⟩
85 \newenvironment{spfstep}[1][]{\omdsetkeys{spf}{#1}
86 \ifx\spf@display\spf@flow\else\item[\the@pst@label]\fi
87 \ifx\spf@title\@empty\else{(\stDMemph{\spf@title})}\fi}
88 {\next@pst@label}
89 ⟨/package⟩
90 ⟨∗ltxml⟩
```

9

```
91 DefCMPEnvironment('{spfstep} OptionalKeyVals:pf',
92              "<omdoc:derive "
93                    . "?&defined(&KeyVal(#1,'id'))(xml:id='&KeyVal(#1,'id')')()>"
94          . "<omdoc:CMP><omdoc:p>#body</omdoc:derive>\n",
95        beforeConstruct=>sub {
96          $_[0]->maybeCloseElement('omdoc:p');
97          $_[0]->maybeCloseElement('omdoc:CMP');
98        });
99 ⟨/ltxml⟩
```

sproofcomment

```
100 ⟨∗package⟩
101 \newenvironment{sproofcomment}[1][]{\omdsetkeys{spf}{#1}
102 \ifx\spf@display\spf@flow\else\item[\the@pst@label]\fi}
103 {\next@pst@label}
104 ⟨/package⟩
105 ⟨∗ltxml⟩
106 DefCMPEnvironment('{sproofcomment} OptionalKeyVals:pf',
107      "<omdoc:omtext "
108                    . "?&defined(&KeyVal(#1,'id'))(xml:id='&KeyVal(#1,'id')')()>"
109      . "<omdoc:CMP><omdoc:p>#body</omdoc:p></omdoc:CMP>"
110      . "</omdoc:omtext>");
111 ⟨/ltxml⟩
```

The next two environments also take a KeyVal argument, but also a regular one, which contains a start text. Both environments start a new numbered proof level.

spfcases   In the pfcases environment, the start text is displayed as the first comment of the proof.

```
112 ⟨∗package⟩
113 \newenvironment{spfcases}[2][]{\omdsetkeys{spf}{#1}
114 \def\@test{#2}\ifx\@test\empty\else
115 \ifx\spf@display\spf@flow {#2}\else\item[\the@pst@label]{#2} \fi\fi
116 \begin{pst@with@label}{\pst@label.\number\count\count10}}
117 {\end{pst@with@label}\next@pst@label}
118 ⟨/package⟩
119 ⟨∗ltxml⟩
120 DefEnvironment('{spfcases} OptionalKeyVals:pf {}',
121      "<omdoc:derive "
122                    . "?&defined(&KeyVal(#1,'id'))(xml:id='&KeyVal(#1,'id')')()>\n"
123      . "<omdoc:CMP><omdoc:p>#2</omdoc:p></omdoc:CMP>\n"
124      . "<omdoc:method xref='#proof-by-cases'>#body</omdoc:method>"
125      . "</omdoc:derive>\n");
126 ⟨/ltxml⟩
```

spfcase   In the pfcase environment, the start text is displayed specification of the case after the \item

```
127 ⟨∗package⟩
```

10

```
128 \newenvironment{spfcase}[2][]{\omdsetkeys{spf}{#1}
129 \ifx\spf@display\spf@flow\else\item[\the@pst@label]\fi
130 \def\@test{#2}\ifx\@test\@empty\else{\stDMemph{#2}:}\fi
131 \begin{pst@with@label}{\pst@label.\number\count\count10}}
132 {\ifx\spf@display\spf@flow\else\sproofend\fi\end{pst@with@label}\next@pst@label}
133 ⟨/package⟩
134 ⟨∗ltxml⟩
135 DefEnvironment('{spfcase} OptionalKeyVals:pf{}',
136      "<omdoc:proof "
137            . "?&defined(&KeyVal(#1,'id'))(xml:id='&KeyVal(#1,'id')')()>\n"
138      . "?#2(<omdoc:omtext>"
139            . "<omdoc:CMP><omdoc:p>#2</omdoc:p></omdoc:CMP>"
140            . "</omdoc:omtext>\n)()"
141      . "#body"
142      . "</omdoc:proof>\n");
143 ⟨/ltxml⟩
```

EdNote(2)   **subproof** In the `subproof` environment, a new (lower-level) proof environment is started.[2]

```
144 ⟨∗package⟩
145 \newenvironment{subproof}{\begin{pst@with@label}{\pst@label.\number\count\count10}}
146 {\ifx\spf@display\spf@flow\else\sproofend\fi\end{pst@with@label}}
147 ⟨/package⟩
148 ⟨∗ltxml⟩
149 DefEnvironment('{subproof}',
150      "<omdoc:proof>\n #body\n</omdoc:proof>\n");
151 ⟨/ltxml⟩
```

3.2 Justifications

We define the actions that are undertaken, when the keys for justifications are
encountered. Here this is very simple, we just define an internal macro with the
value, so that we can use it later.

```
152 ⟨∗package⟩
153 \srefaddidkey{just}
154 \omdaddkey{just}{method}
155 \omdaddkey{just}{premises}
156 \omdaddkey{just}{args}
157 ⟨/package⟩
158 ⟨∗ltxml⟩
159 DefKeyVal('just','id','Semiverbatim');
160 DefKeyVal('just','method','Semiverbatim');
161 DefKeyVal('just','premises','Semiverbatim');
162 DefKeyVal('just','args','Semiverbatim');
163 ⟨/ltxml⟩
```

The next three environments and macros are purely semantic, so we ignore the
EdNote(3)   keyval arguments for now and only display the content.[3]

---

[2]EDNOTE: document this above
[3]EDNOTE: need to do something about the premise in draft mode.

justification

```
164 ⟨∗package⟩
165 \newenvironment{justification}[1][]{}{}
166 ⟨/package⟩
167 ⟨∗ltxml⟩
168 sub extractBodyText {
169   my ($box, $remove) = @_;
170   my $str = '';
171   my @boxes = $box->unlist;
172   foreach my $b(@boxes) {
173     my $s = '';
174     if ($b =~ /LaTeXML::Whatsit/) {
175       my $body = $b->getBody;
176       $s = $body ? extractBodyText($body, $remove) : '';
177     } elsif ($b =~ /LaTeXML::Box/) {
178       $s = $b->toString || '';
179       @{$b}[0] = '' if $remove; }
180     $str .= $s; }
181   $str =~ s/\s+/ /g;
182   $str; }
183
184 DefEnvironment('{justification} OptionalKeyVals:just', sub {
185   my ($doc, $keys, %props) = @_;
186   my $text = extractBodyText($props{body}, 1);
187   my $node = LookupValue('_LastSeenCMP');
188   #$node->appendText($text) if $node;
189   my $method = $keys ? $keys->getValue('method') : undef;
190   $doc->openElement("omdoc:method", $method ? (xref => $method) : ());
191   $doc->absorb($props{body}) if $props{body};
192   $doc->closeElement("omdoc:method");
193   return; });
194 ⟨/ltxml⟩
```

\premise

```
195 ⟨∗package⟩
196 \newcommand{\premise}[2][]{#2}
197 ⟨/package⟩
198 ⟨∗ltxml⟩
199 DefMacro('\premise[]{}', sub {
200   my ($xref, $text) = ($_[1], $_[2]);
201   my @res = (T_CS('\premise@content'));
202   push(@res, T_OTHER('['), $xref->unlist, T_OTHER(']')) if $xref;
203   push(@res, T_SPACE, $text->unlist) if $text;
204   @res; });
205 DefConstructor('\premise@content[]',
206                 "<omdoc:premise xref='#1'/>");
207 ⟨/ltxml⟩
```

\justarg   the \justarg macro is purely semantic, so we ignore the keyval arguments for
now and only display the content.

12

```
208 ⟨∗package⟩
209 \newcommand{\justarg}[2][]{#2}
210 ⟨/package⟩
211 ⟨∗ltxml⟩
212 DefMacro('\justarg[]{}', sub { (($_[1] ? $_[1]->unlist : ()),
213 T_SPACE, $_[2]->unlist, T_SPACE); });
214 Tag('omdoc:derive', afterClose=>sub {
215     my ($doc, $node) = @_;
216     my @children = grep($_->nodeType == XML_ELEMENT_NODE, $node->childNodes);
217     my $firstCMP = undef;
218     foreach my $child(@children) {
219       next unless ($child->localname || '') eq 'CMP';
220       if ($child->hasChildNodes()) {
221         next unless $#{$child->childNodes} == 0;
222         next unless $child->firstChild->nodeType == XML_TEXT_NODE; }
223
224       if ($firstCMP) {
225         $firstCMP->appendText($child->textContent);
226         $node->removeChild($child);
227       } else { $firstCMP = $child; }
228     }
229   });#$
230 ⟨/ltxml⟩
```

3.3 Providing IDs for OMDOC Elements

To provide default identifiers, we tag all OMDoc elements that allow `xml:id` attributes by executing the `numberIt` procedure from `omdoc.sty.ltxml`.

```
231 ⟨∗ltxml⟩
232 Tag('omdoc:proof',afterOpen=>\&numberIt,afterClose=>\&locateIt);
233 Tag('omdoc:derive',afterOpen=>\&numberIt,afterClose=>\&locateIt);
234 Tag('omdoc:method',afterOpen=>\&numberIt,afterClose=>\&locateIt);
235 Tag('omdoc:premise',afterOpen=>\&numberIt,afterClose=>\&locateIt);
236 Tag('omdoc:derive',afterOpen=>\&numberIt,afterClose=>\&locateIt);
237 ⟨/ltxml⟩
```

4 Finale

Finally, we need to terminate the file with a success mark for perl.

```
238 ⟨ltxml⟩1;
```

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

References

[Koh]   Tech. rep. Comprehensive TEX Archive Network (CTAN), URL: `http://www.ctan.org/tex-archive/macros/latex/contrib/stex/sref/sref.pdf`.