

`sproof.sty`: Structural Markup for Proofs*

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

November 7, 2015

Abstract

The `sproof` package is part of the \LaTeX collection, a version of $\text{\TeX}/\text{\LaTeX}$ that allows to markup $\text{\TeX}/\text{\LaTeX}$ documents semantically without leaving the document format, essentially turning $\text{\TeX}/\text{\LaTeX}$ 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 \LaTeX files. This structure can be used by MKM systems for added-value services, either directly from the \LaTeX sources, or after translation.

*Version v1.1 (last revised 2013/09/29)

Contents

1	Introduction	3
2	The User Interface	4
2.1	Package Options	4
2.2	Proofs and Proof steps	4
2.3	Justifications	4
2.4	Proof Structure	6
2.5	Proof End Markers	6
2.6	Configuration of the Presentation	7
3	Limitations	7
4	The Implementation	8
4.1	Package Options	8
4.2	Proofs	8
4.3	Justifications	16
4.4	Providing IDs for OMDoc Elements	18
5	Finale	18

1 Introduction

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

\LaTeX 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

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

2 The User Interface

2.1 Package Options

`showmeta` The `sproof` package takes a single option: `showmeta`. If this is set, then the metadata keys are shown (see [Koh15a] for details and customization options).

2.2 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 its own proof end marker with 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 line. The `\spfidf` 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.

2.3 Justifications

`justification` This evidence is marked up with the `justification` environment in the `sproof` package. This environment is 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 to 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.

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. \square

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

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

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

2.4 Proof Structure

`subproof` The `pfcases` environment is used to mark up a subproof. 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).

`method` The `method` key can be used to give the name of the proof method executed to make this subproof.

`spfcases` The `pfcases` environment is used to mark up a proof by cases. Technically it is a variant of the `subproof` where the `method` is `by-cases`. Its contents are `spfcase` environments that mark up the cases one by one.

`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 description of the case under consideration. The content of a `pfcase` environment is the same as that of a `proof`, i.e. `steps`, `proofcomments`, and `pfcases` environments.

`\spfcasesketch` `\spfcasesketch` is a variant of the `spfcase` environment that takes the same arguments, but instead of the `spfsteps` in the body uses a third argument for a proof sketch.

`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 do, or what we have achieved so far. As such, it cannot be the target of a `\premise`.

2.5 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 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 `\sProofEndSymbol{}`.

2.6 Configuration of the Presentation

Finally, we provide configuration hooks in Figure 1 for the keywords in proofs. These are mainly intended for package authors building on **statements**, e.g. for multi-language support.² The proof step labels can be customized via

Environment	configuration macro	value
sproof	<code>\spf@proof@kw</code>	Proof
sketchproof	<code>\spf@sketchproof@kw</code>	Proof Sketch

Figure 1: Configuration Hooks for Semantic Proof Markup

`\pstlabelstyle` the `\pstlabelstyle` macro: `\pstlabelstyle{<style>}` sets the style; see Figure 2 for an overview of styles. Package writers can add additional styles by adding a macro `\pst@make@label@<style>` that takes two arguments: a comma-separated list of ordinals that make up the prefix and the current ordinal. Note that comma-separated lists can be conveniently iterated over by the \LaTeX `\@for...:=...\do{...}` macro; see Figure 2 for examples.

style	example	configuration macro
long	0.8.1.5	<code>\def\pst@make@label@long#1#2{\@for\@I:=#1\do{\@I.}\#2}</code>
angles	$\rangle\rangle\rangle 5$	<code>\def\pst@make@label@angles#1#2{\ensurermath{\@for\@I:=#1\do{\rangle}}}\#2}</code>
short	5	<code>\def\pst@make@label@short#1#2{\#2}</code>
empty		<code>\def\pst@make@label@empty#1#2{}</code>

Figure 2: Configuration Proof Step Label Styles

3 Limitations

In this section we document known limitations. If you want to help alleviate them, please feel free to contact the package author. Some of them are currently discussed in the TRAC.

1. The numbering scheme of proofs cannot be changed. It is more geared for teaching proof structures (the author’s main use case) and not for writing papers. (reported by Tobias Pfeiffer; see **sTeX:online**, issue 1658) (fixed)
2. currently proof steps are formatted by the \LaTeX `description` environment. We would like to configure this, e.g. to use the `inparaenum` environment for more condensed proofs. I am just not sure what the best user interface would be I can imagine redefining an internal environment `spf@proofstep@list` or adding a key `prooflistenv` to the `proof` environment that allows to specify the environment directly. Maybe we should do both.

²EdNOTE: we might want to develop an extension `sproof-babel` in the future.

4 The Implementation

The `sproof` package generates to files: the L^AT_EX package (all the code between `<*package>` and `</package>`) and the L^AT_EXML 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 L^AT_EXML

```
1 <*ltxml>
2 # -*- CPERL -*-
3 package LaTeXML::Package::Pool;
4 use strict;
5 use LaTeXML::Package;
6 </ltxml>
```

4.1 Package Options

We declare some switches which will modify the behavior according to the package options. Generally, an option `xxx` will just set the appropriate switches to true (otherwise they stay false).³

```
7 <*package>
8 \DeclareOption*{\PassOptionsToPackage{\CurrentOption}{sref}}
9 \ProcessOptions
10 </package>
11 <*ltxml>
12 DeclareOption(undef, sub {PassOptions('sref','sty',ToString(Digest(T_CS('\CurrentOption')))); }
13 ProcessOptions();
14 </ltxml>
```

Then we make sure that the `sref` package is loaded [Koh15b].

```
15 <*package>
16 \RequirePackage{sref}
17 \RequirePackage{etoolbox}
18 \RequirePackage[base]{babel}
19 </package>
20 <*ltxml>
21 RequirePackage('sref');
22 </ltxml>
```

4.2 Proofs

We first define some keys for the proof environment.

```
23 <*package>
24 \srefaddidkey{spf}
25 \addmetakey*{spf}{display}
26 \addmetakey{spf}{for}
27 \addmetakey{spf}{from}
```

³EdNOTE: need an implementation for L^AT_EXML


```

28 \addmetakey*[\sproof@box]{spf}{proofend}
29 \addmetakey{spf}{type}
30 \addmetakey*{spf}{title}
31 \addmetakey{spf}{continues}
32 \addmetakey{spf}{functions}
33 \addmetakey{spf}{method}
34 \endpackage

```

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

```

35 \beginpackage
36 \def\spf@flow{flow}
37 \endpackage

```

For proofs, we will have to have deeply nested structures of enumerated list-like environments. However, L^AT_EX 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 manages¹ the path labeling of the proof steps in the description 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 T_EX 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.

```

38 \beginpackage
39 \newenvironment{pst@with@label}[1]{%
40   \edef\pst@label{#1}%
41   \advance\count10 by 1%
42   \count\count10=1%
43 }{%
44   \advance\count10 by -1%
45 }%

```

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

```

46 \def\the@pst@label{%
47   \pst@make@label\pst@label{\number\count\count10}\pstlabel@postfix%
48 }%

```

`\setpstlabelstyle` `\setpstlabelstyle{metaKey-Val pairs}` makes the labeling style customizable. `\setpstlabelstyle{prefix=Pr,delimiter=-,postfix=\dag}` will change the

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

labeling style from **P.1.2.3** to **Pr-1-2-3†**. `\setpstlabelstyledefault` will set the labeling style back to default.

```

49 \addmetakey[P]{pstlabel}{prefix}[]
50 \addmetakey[.]{pstlabel}{delimiter}[]
51 \addmetakey[]{pstlabel}{postfix}[]
52 \metasetkeys{pstlabel}{}% initialization
53 \newrobustcmd\setpstlabelstyle[i]{%
54   \metasetkeys{pstlabel}{#1}%
55 }%
56 \newrobustcmd\setpstlabelstyledefault{%
57   \metasetkeys{pstlabel}{prefix=P,delimiter=.,postfix=}%
58 }%

\pstlabelstyle \pstlabelstyle just sets the \pst@make@label macro according to the style.
59 \def\pst@make@label@long#1#2{\@for\@I:=#1\do{\expandafter\@I\pstlabel@delimiter}#2}
60 \def\pst@make@label@angles#1#2{\ensuremath{\@for\@I:=#1\do{\rangle}}#2}
61 \def\pst@make@label@short#1#2{#2}
62 \def\pst@make@label@empty#1#2{}
63 \def\pstlabelstyle#1{%
64   \def\pst@make@label{\nameuse{pst@make@label@#1}}%
65 }%
66 \pstlabelstyle{long}%

\next@pst@label \next@pst@label increments the step label at the current level.
67 \def\next@pst@label{%
68   \global\advance\count\count10 by 1%
69 }%

\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
70 \def\sproof@box{%
71   \hbox{\vrule\vbox{\hrule width 6 pt\vskip 6pt\hrule}\vrule}%
72 }%
73 \def\spf@proofend{\sproof@box}%
74 \def\sproofend{%
75   \ifx\spf@proofend\empty%
76     \else%
77       \hfil\null\nobreak\hfill\spf@proofend\par\smallskip%
78     \fi%
79 }%
80 \def\sProofEndSymbol#1{\def\sproof@box{#1}}%
81 \</package>
82 \<*lxml>
83 DefConstructor('sproofend','');
84 DefConstructor('sProofEndSymbol{','}','');
85 \</lxml>

spf@*kw
86 \<*package>

```

```

87 \def\spf@proofsketch@kw{Proof Sketch}
88 \def\spf@proof@kw{Proof}
89 \def\spf@step@kw{Step}
90 \end{package}

```

For the other languages, we set up triggers

```

91 \begin{package}
92 \AfterBabelLanguage{ngerman}{\input{sproof-ngerman.1df}}
93 \end{package}

```

`spfsketch`

```

94 \begin{package}
95 \newrobustcmd\spfsketch[2] [] {%
96   \metasetkeys{spf}{#1}%
97   \sref@target%
98   \ifx\spf@display\spf@flow%
99   \else%
100    {\stDMemph{\ifx\spf@type\@empty\spf@proofsketch@kw\else\spf@type\fi}:}
101    \fi{ #2}%
102    \sref@label@id{this \ifx\spf@type\@empty\spf@proofsketch@kw\else\spf@type\fi}\sproofend%
103  }%
104 \end{package}
105 \begin{xml}
106 DefConstructor(' \spfsketch OptionalKeyVals:pf{ }',
107   "<omdoc:proof "
108     . " ?&defined(&GetKeyVal(#1,'id'))(xml:id='&GetKeyVal(#1,'id'))(> \n"
109     . " ?#2(<omdoc:omtext><omdoc:CMP>#2\n) ()"
110     . "</omdoc:proof> \n");
111 \end{xml}

```

`spfeq` This is very similar to `\spfsketch`, but uses a computation array⁴⁵

```

112 \begin{package}
113 \newenvironment{spfeq}[2] [] {%
114   \metasetkeys{spf}{#1}\sref@target%
115   \ifx\spf@display\spf@flow%
116   \else%
117     {\stDMemph{\ifx\spf@type\@empty\spf@proof@kw\else\spf@type\fi}:} #2%
118   \fi% display=flow
119   \begin{displaymath}\begin{array}{rcll}%
120   \end{array}\end{displaymath}%
121   \end{array}\end{displaymath}%
122 }%
123 \end{package}
124 \begin{xml}
125 RawTeX(
126 \newenvironment{spfeq}[2] [] {

```

⁴EDNOTE: This should really be more like a tabular with an ensuremath in it. or invoke text on the last column

⁵EDNOTE: document above

```

127 {\begin{sproof}[#1]{#2}\begin{displaymath}\begin{array}{rcll}}
128 {\end{array}\end{displaymath}\end{sproof}}
129 ');
130 \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.

```

131 \*package>
132 \newenvironment{spf@proof}[2] [] {%
133   \metasetkeys{spf}{#1}%
134   \sref@target%
135   \count10=10%
136   \par\noindent%
137   \ifx\spf@display\spf@flow%
138   \else%
139     \stDMemph{\ifx\spf@type\@empty\spf@proof@kw\else\spf@type\fi}:%
140     \fi{ #2}%
141     \sref@label@id{this \ifx\spf@type\@empty\spf@proof@kw\else\spf@type\fi}%
142     \def\pst@label{}%
143     \newcount\pst@count% initialize the labeling mechanism
144     \begin{description}\begin{pst@with@label}\{ \pstlabel@prefix}%
145     }{%
146     \end{pst@with@label}\end{description}%
147 }%
148 \newenvironment{sproof}[2] [] {\begin{spf@proof}[#1]{#2}}{\sproofend\end{spf@proof}}%
149 \newenvironment{sProof}[2] [] {\begin{spf@proof}[#1]{#2}}{\end{spf@proof}}%
150 \ltxml>
151 \*package>
152 DefEnvironment('sproof' OptionalKeyVals:pf{}',
153   "<omdoc:proof "
154     . "?&GetKeyVal(#1,'for')(for='&hash_wrapper(&GetKeyVal(#1,'for'))'())"
155     . "?&defined(&GetKeyVal(#1,'id'))(xml:id='&GetKeyVal(#1,'id'))(>)\n"
156     . "#2(<omdoc:omtext>"
157       . " <omdoc:CMP>#2</omdoc:CMP>"
158       . "</omdoc:omtext>\n)()"
159       . "#body"
160       . "</omdoc:proof>\n");
161 DefMacro('sProof','sproof');
162 DefMacro('endsProof','endsproof');
163 \ltxml>

```

spfidea

```

164 \*package>
165 \newrobustcmd\spfidea[2] [] {%
166   \metasetkeys{spf}{#1}%
167   \stDMemph{\ifx\spf@type\@empty{Proof Idea}\else\spf@type\fi:} #2\sproofend%
168 }%
169 \ltxml>

```

```

170 <*ltxml>
171 DefConstructor('spfidea OptionalKeyVals:pf {}',
172     "<omdoc:proof "
173     . "??&defined(&GetKeyVal(#1,'id'))(xml:id='&GetKeyVal(#1,'id'))()"
174     . "??&GetKeyVal(#1,'for')(for='&hash_wrapper(&GetKeyVal(#1,'for'))')()>\n"
175     . "<omdoc:omtext><omdoc:OMP>#2</omdoc:omtext>\n"
176     . "</omdoc:proof>\n");
177 </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.

EdN:6

```

spfstep 6
178 <*package>
179 \newenvironment{spfstep}[1][{}%
180     \metasetkeys{spf}{#1}%
181     \@in@omtexttrue%
182     \ifx\spf@display\spf@flow%
183     \else%
184     \item[\the@pst@label]%
185     \fi%
186     \ifx\spf@title\empty\else{(\stDMemph{\spf@title})}\fi%
187     \sref@label@id{\pst@label}\ignorespaces%
188 }{}%
189 \next@pst@label\@in@omtextfalse\ignorespaces%
190 }%
191 </package>
192 <*ltxml>
193 DefEnvironment('{spfstep} OptionalKeyVals:pf',
194     "<omdoc:derive "
195     . "??&defined(&GetKeyVal(#1,'id'))(xml:id='&GetKeyVal(#1,'id'))()>"
196     . "<omdoc:OMP>#body</omdoc:derive>\n",
197     beforeConstruct=>sub {
198     $_[0]->maybeCloseElement('omdoc:OMP');
199     };#$
200 </ltxml>

```

sproofcomment

```

201 <*package>
202 \newenvironment{sproofcomment}[1][{}%
203     \metasetkeys{spf}{#1}%
204     \ifx\spf@display\spf@flow\else\item[\the@pst@label]\fi%
205 }{}%
206 \next@pst@label%
207 }%

```

⁶EDNOTE: MK: labeling of steps does not work yet.

```

208 </package>
209 <*ltxml>
210 DefEnvironment('{sproofcomment} OptionalKeyVals:pf',
211     "<omdoc:omtext "
212     .   "?&defined(&GetKeyVal(#1,'id'))(xml:id='&GetKeyVal(#1,'id'))>"
213     .   "<omdoc:OMP>#body</omdoc:OMP>"
214     .   "</omdoc:omtext>");
215 </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.

subproof In the `subproof` environment, a new (lower-level) proof environment is started.

```

216 <*package>
217 \newenvironment{subproof}[2][{}]{%
218   \metasetkeys{spf}{#1}%
219   \def\@test{#2}%
220   \ifx\@test\empty%
221   \else%
222     \ifx\spf@display\spf@flow {#2}%
223     \else%
224       \item[\the@pst@label]{#2} %
225     \fi%
226   \fi%
227   \begin{pst@with@label}{\pst@label,\number\count\count10}%
228 }{%
229   \end{pst@with@label}\next@pst@label%
230 }%
231 </package>
232 <*ltxml>
233 DefEnvironment('{subproof} OptionalKeyVals:pf {}',
234     "<omdoc:derive "
235     .   "?&defined(&GetKeyVal(#1,'id'))(xml:id='&GetKeyVal(#1,'id'))>\n"
236     .   "<omdoc:OMP>#2</omdoc:OMP>\n"
237     .   "<omdoc:method ?&defined(&GetKeyVal(#1,'method'))(xref='&GetKeyVal(#1,'method'))>"
238     .   "<omdoc:proof>#body</omdoc:proof>"
239     .   "</omdoc:method>"
240     .   "</omdoc:derive>\n");
241 </ltxml>

```

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

```

242 <*package>
243 \newenvironment{spfcases}[2][{}]{%
244   \def\@test{#1}%
245   \ifx\@test\empty%
246     \begin{subproof}[method=by-cases]{#2}%
247   \else%

```

```

248 \begin{subproof}[#1,method=by-cases]{#2}%
249 \fi%
250 }{%
251 \end{subproof}%
252 }%
253 \end{package}
254 \end{*ltxml}
255 DefEnvironment('{spfcases} OptionalKeyVals:pf {}',
256 " <omdoc:derive "
257 . " ?&defined(&GetKeyVal(#1,'id'))(xml:id='&GetKeyVal(#1,'id'))()>\n"
258 . " <omdoc:CMP>#2</omdoc:CMP>\n"
259 . " <omdoc:method ?&defined(&GetKeyVal(#1,'method'))(xref='&GetKeyVal(#1,'method'))>"
260 . " #body"
261 . " </omdoc:method>"
262 . " </omdoc:derive>\n");
263 \end{*ltxml}

```

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

```

264 \end{*package}
265 \newenvironment{spfcase}[2][]{%
266 \metasetkeys{spf}{#1}%
267 \ifx\spf@display\spf@flow\else\item[\the@pst@label]\fi%
268 \def\@test{#2}%
269 \ifx\@test\@empty%
270 \else%
271 {\stDMemph{#2}: }% need blank here
272 \fi%
273 \begin{pst@with@label}{\pst@label,\number\count\count10}
274 }{%
275 \ifx\spf@display\spf@flow%
276 \else%
277 \sproofend%
278 \fi%
279 \end{pst@with@label}%
280 \next@pst@label%
281 }%
282 \end{package}
283 \end{*ltxml}
284 DefEnvironment('{spfcase} OptionalKeyVals:pf{}',
285 " <omdoc:proof "
286 . " ?&defined(&GetKeyVal(#1,'id'))(xml:id='&GetKeyVal(#1,'id'))()>\n"
287 . " ?#2(<omdoc:omtext>"
288 . " <omdoc:CMP>#2</omdoc:CMP>"
289 . " </omdoc:omtext>\n)()"
290 . " #body"
291 . " </omdoc:proof>\n");
292 \end{*ltxml}

```

spfcase similar to **spfcase**, takes a third argument.

```

293 <*package>
294 \newrobustcmd\spfcasesketch[3][]{%
295   \metasetkeys{spf}{#1}%
296   \ifx\spf@display\spf@flow%
297   \else%
298     \item[\the@pst@label]%
299   \fi%
300   \def\@test{#2}%
301   \ifx\@test\@empty%
302   \else%
303     {\stDMemph{#2}: }%
304   \fi#3%
305   \next@pst@label%
306 }%
307 </package>
308 <*ltxml>
309 DefConstructor('spfcasesketch OptionalKeyVals:pf{}{}',
310   "<omdoc:proof "
311   . " ?&defined(&GetKeyVal(#1,'id'))(xml:id='&GetKeyVal(#1,'id'))()>\n"
312   . " ?#2(<omdoc:omtext>"
313   . " <omdoc:CMP>#2</omdoc:CMP>"
314   . " </omdoc:omtext>\n)()"
315   . "#3"
316   . "</omdoc:proof>\n");
317 </ltxml>

```

4.3 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.

```

318 <*package>
319 \srefaddidkey{just}
320 \addmetakey{just}{method}
321 \addmetakey{just}{premises}
322 \addmetakey{just}{args}
323 </package>
324 <*ltxml>
325 DefKeyVal('just','id','Semiverbatim');
326 DefKeyVal('just','method','Semiverbatim');
327 DefKeyVal('just','premises','Semiverbatim');
328 DefKeyVal('just','args','Semiverbatim');
329 </ltxml>

```

The next three environments and macros are purely semantic, so we ignore the keyval arguments for now and only display the content.⁷

EdN:7

justification

⁷EDNOTE: need to do something about the premise in draft mode.


```

330 <*package>
331 \newenvironment{justification}[1] [] {}{}
332 </package>
333 <*ltxml>
334 sub extractBodyText {
335   my ($box, $remove) = @_;
336   my $str = '';
337   my @boxes = $box->unlist;
338   foreach my $b(@boxes) {
339     my $s = '';
340     if ($b =~ /LaTeXML::Whatsit/) {
341       my $body = $b->getBody;
342       $s = $body ? extractBodyText($body, $remove) : '';
343     } elsif ($b =~ /LaTeXML::Box/) {
344       $s = $b->toString || '';
345       @{$b}[0] = '' if $remove; }
346     $str .= $s; }
347   $str =~ s/\s+/ /g;
348   $str; }
349
350 DefEnvironment('{justification} OptionalKeyVals:just', sub {
351   my ($doc, $keys, %props) = @_;
352   my $text = extractBodyText($props{body}, 1);
353   my $node = LookupValue('_LastSeenCMP');
354   # $node->appendText($text) if $node;
355   my $method = $keys ? $keys->getValue('method') : undef;
356   $doc->openElement("omdoc:method", $method ? (xref => $method) : ());
357   $doc->absorb($props{body}) if $props{body};
358   $doc->closeElement("omdoc:method");
359   return; });
360 </ltxml>

```

\premise

```

361 <*package>
362 \newrobustcmd\premise[2] [] {#2}
363 </package>
364 <*ltxml>
365 DefMacro('\premise[]{}', sub {
366   my ($xref, $text) = ($_[1], $_[2]);
367   my @res = (T_CS('\premise@content'));
368   push(@res, T_OTHER('[', $xref->unlist, T_OTHER(']')) if $xref;
369   push(@res, T_SPACE, $text->unlist) if $text;
370   @res; });
371 DefConstructor('\premise@content[]',
372   "<omdoc:premise xref='#1' />");
373 </ltxml>

```

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

```

374 <*package>
375 \newrobustcmd\justarg[2][\{#2}
376 </package>
377 <*ltxml>
378 DefMacro('\justarg[]{}', sub { (($_[1] ? $_[1]->unlist : ()),
379 T_SPACE, $_[2]->unlist, T_SPACE); });
380 Tag('omdoc:derive', afterClose=>sub {
381     my ($doc, $node) = @_;
382     my @children = grep($_->nodeType == XML_ELEMENT_NODE, $node->childNodes);
383     my $firstCMP = undef;
384     foreach my $child(@children) {
385         next unless ($child->localname || '') eq 'CMP';
386         if ($child->hasChildNodes()) {
387             next unless $#{$child->childNodes} == 0;
388             next unless $child->firstChild->nodeType == XML_TEXT_NODE; }
389
390         if ($firstCMP) {
391             $firstCMP->appendText($child->textContent);
392             $node->removeChild($child);
393         } else { $firstCMP = $child; }
394     }
395     };#$
396 </ltxml>

```

4.4 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`.

```

397 <*ltxml>
398 Tag('omdoc:proof',afterOpen=>\&numberIt,afterClose=>\&locateIt);
399 Tag('omdoc:derive',afterOpen=>\&numberIt,afterClose=>\&locateIt);
400 Tag('omdoc:method',afterOpen=>\&numberIt,afterClose=>\&locateIt);
401 Tag('omdoc:premise',afterOpen=>\&numberIt,afterClose=>\&locateIt);
402 Tag('omdoc:derive',afterOpen=>\&numberIt,afterClose=>\&locateIt);
403 </ltxml>

```

5 Finale

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

```

404 <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.

LATeXML,	8	proofs		semantic	
OMDoc,	18	semantic,	3	proofs,	3

References

- [Koh15a] Michael Kohlhase. *metakeys.sty: A generic framework for extensible Metadata in L^AT_EX*. Tech. rep. Comprehensive T_EX Archive Network (CTAN), 2015. URL: <http://www.ctan.org/tex-archive/macros/latex/contrib/stex/metakeys/metakeys.pdf>.
- [Koh15b] Michael Kohlhase. *sref.sty: Semantic Crossreferencing in L^AT_EX*. Tech. rep. Comprehensive T_EX Archive Network (CTAN), 2015. URL: <http://www.ctan.org/tex-archive/macros/latex/contrib/stex/sref/sref.pdf>.