`sproof.sty`: Structural Markup for Proofs*

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

February 18, 2014

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

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]

EdN:1

[1] EdNote: 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 [Koh13a] 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 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.

2.3 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.

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

\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 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 to, 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

\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{}.

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

EdN:2

Environment	configuration macro	value
`sproof`	`\spf@proof@kw`	Proof
`sketchproof`	`\spf@sketchproof@kw`	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	`\def\pst@make@label@long#1#2{\@for\@I:=#1\do{\@I.}#2}`
`angles`	⟩⟩⟩5	`\def\pst@make@label@angles#1#2 {\ensuremath{\@for\@I:=#1\do{\rangle}}#2}`
`short`	5	`\def\pst@make@label@short#1#2{#2}`
`empty`		`\def\pst@make@label@empty#1#2{}`

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], 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.

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

4 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

```
⟨*ltxml⟩
# -*- CPERL -*-
package LaTeXML::Package::Pool;
use strict;
use LaTeXML::Package;
RequirePackage('sref');
⟨/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]

EdN:3

```
⟨*package⟩
\DeclareOption{showmeta}{\PassOptionsToPackage{\CurrentOption}{metakeys}}
\ProcessOptions
⟨/package⟩
⟨*ltxml⟩
DeclareOption('showmeta','');
⟨/ltxml⟩
```

Then we make sure that the `sref` package is loaded [Koh13b].

```
⟨*package⟩
\RequirePackage{sref}
⟨/package⟩
```

4.2 Proofs

We first define some keys for the `proof` environment.

```
⟨*package⟩
\srefaddidkey{spf}
\addmetakey*{spf}{display}
\addmetakey{spf}{for}
\addmetakey{spf}{from}
\addmetakey*[\sproof@box]{spf}{proofend}
\addmetakey{spf}{type}
\addmetakey*{spf}{title}
\addmetakey{spf}{continues}
\addmetakey{spf}{functions}
```

[3] EdNote: need an implementation for LaTeXML

```
\addmetakey{spf}{method}
⟨/package⟩
```

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

```
⟨*package⟩
\def\spf@flow{flow}
⟨/package⟩
```

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 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.

```
⟨*package⟩
\newenvironment{pst@with@label}[1]%
{\edef\pst@label{#1}\advance\count10 by 1\count\count10=1}
{\advance\count10 by -1}
```

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

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

`\pstlabelstyle` `\pstlabelstyle` just sets the `\pst@make@label` macro according to the style.

```
\def\pst@make@label@long#1#2{\@for\@I:=#1\do{\@I.}#2}
\def\pst@make@label@angles#1#2{\ensuremath{\@for\@I:=#1\do{\rangle}}#2}
\def\pst@make@label@short#1#2{#2}
\def\pst@make@label@empty#1#2{}
\def\pstlabelstyle#1{\def\pst@make@label{\@nameuse{pst@make@label@#1}}}
\pstlabelstyle{long}
```

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

```
\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

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

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

spf@*@kw

```
⟨*package⟩
\def\spf@proofsketch@kw{Proof Sketch}
\def\spf@proof@kw{Proof}
\def\spf@step@kw{Step}
⟨/package⟩
```

spfsketch

```
⟨*package⟩
\newcommand\spfsketch[2][]{\metasetkeys{spf}{#1}\sref@target%
\ifx\spf@display\spf@flow\else%
{\stDMemph{\ifx\spf@type\@empty\spf@proofsketch@kw\else\spf@type\fi}:}\fi{ #2}%
\sref@label@id{this \ifx\spf@type\@empty\spf@proofsketch@kw\else\spf@type\fi}\sproofend}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\spfsketch OptionalKeyVals:pf{}',
       "<omdoc:proof "
                .  "?&defined(&GetKeyVal(#1,'id'))(xml:id='&GetKeyVal(#1,'id')')()>\n"
       . "?#2(<omdoc:omtext><omdoc:CMP>#2\n)()"
                . "</omdoc:proof>\n");
⟨/ltxml⟩
```

EdN:4
EdN:5

spfeq This is very similar to \spfsketch, but uses a computation array[4][5]

```
⟨*package⟩
\newenvironment{spfeq}[2][]%
{\metasetkeys{spf}{#1}\sref@target%
\ifx\spf@display\spf@flow\else%
{\stDMemph{\ifx\spf@type\@empty\spf@proof@kw\else\spf@type\fi}:} #2%
\fi% display=flow
\begin{displaymath}\begin{array}{rcll}}
{\end{array}\end{displaymath}}
⟨/package⟩
⟨*ltxml⟩
RawTeX('
\newenvironment{spfeq}[2][]%
{\begin{sproof}[#1]{#2}\begin{displaymath}\begin{array}{rcll}}
```

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

[5] EdNote: document above

```
{\end{array}\end{displaymath}\end{sproof}}
');
⟨/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.

```
⟨*package⟩
\newenvironment{spf@proof}[2][]{\metasetkeys{spf}{#1}\sref@target%
\count10=10%
\par\noindent%
\ifx\spf@display\spf@flow\else%
\stDMemph{\ifx\spf@type\@empty\spf@proof@kw\else\spf@type\fi}:\fi{ #2}%
\sref@label@id{this \ifx\spf@type\@empty\spf@proof@kw\else\spf@type\fi}%
\def\pst@label{}\newcount\pst@count% initialize the labeling mechanism
\begin{description}\begin{pst@with@label}{P}}
{\end{pst@with@label}\end{description}}
\newenvironment{sproof}[2][]{\begin{spf@proof}[#1]{#2}}{\sproofend\end{spf@proof}}
\newenvironment{sProof}[2][]{\begin{spf@proof}[#1]{#2}}{\end{spf@proof}}
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{sproof} OptionalKeyVals:pf{}',
       "<omdoc:proof "
               . "?&GetKeyVal(#1,'for')(for='&hash_wrapper(&GetKeyVal(#1,'for'))')()"
               .  "?&defined(&GetKeyVal(#1,'id'))(xml:id='&GetKeyVal(#1,'id')')()>\n"
       . "?#2(<omdoc:omtext>"
              .       "<omdoc:CMP>#2</omdoc:CMP>"
               .     "</omdoc:omtext>\n)()"
               . "#body"
             . "</omdoc:proof>\n");
DefMacro('\sProof','\sproof');
DefMacro('\endsProof','\endsproof');
⟨/ltxml⟩
```

spfidea

```
⟨*package⟩
\newcommand\spfidea[2][]{\metasetkeys{spf}{#1}%
\stDMemph{\ifx\spf@type\@empty{Proof Idea}\else\spf@type\fi:} #2\sproofend}
⟨/package⟩
⟨*ltxml⟩
DefConstructor('\spfidea OptionalKeyVals:pf {}',
       "<omdoc:proof "
             .      "?&defined(&GetKeyVal(#1,'id'))(xml:id='&GetKeyVal(#1,'id')')()"
             . "?&GetKeyVal(#1,'for')(for='&hash_wrapper(&GetKeyVal(#1,'for'))')()>\n"
     .   "<omdoc:omtext><omdoc:CMP>#2</omdoc:omtext>\n"
     . "</omdoc:proof>\n");
⟨/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]

```
⟨*package⟩
\newenvironment{spfstep}[1][]{\metasetkeys{spf}{#1}%
\ifx\spf@display\spf@flow\else\item[\the@pst@label]\fi%
\ifx\spf@title\@empty\else{(\stDMemph{\spf@title})}\fi%
\sref@label@id{\pst@label}\ignorespaces}%
{\next@pst@label\ignorespaces}
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{spfstep} OptionalKeyVals:pf',
             "<omdoc:derive "
                 .   "?&defined(&GetKeyVal(#1,'id'))(xml:id='&GetKeyVal(#1,'id')')()>"
        .  "<omdoc:CMP>#body</omdoc:derive>\n",
         beforeConstruct=>sub {
          $_[0]->maybeCloseElement('omdoc:CMP');
         });#$
⟨/ltxml⟩
```

`sproofcomment`

```
⟨*package⟩
\newenvironment{sproofcomment}[1][]{\metasetkeys{spf}{#1}%
\ifx\spf@display\spf@flow\else\item[\the@pst@label]\fi}
{\next@pst@label}
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{sproofcomment} OptionalKeyVals:pf',
       "<omdoc:omtext "
               .   "?&defined(&GetKeyVal(#1,'id'))(xml:id='&GetKeyVal(#1,'id')')()>"
       . "<omdoc:CMP>#body</omdoc:CMP>"
     . "</omdoc:omtext>");
⟨/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.

```
⟨*package⟩
\newenvironment{subproof}[2][]{\metasetkeys{spf}{#1}%
\def\@test{#2}\ifx\@test\empty\else%
\ifx\spf@display\spf@flow {#2}\else\item[\the@pst@label]{#2} \fi\fi%
\begin{pst@with@label}{\pst@label,\number\count\count10}}
{\end{pst@with@label}\next@pst@label}
⟨/package⟩
```

[6] EdNote: MK: labeling of steps does not work yet.

```
⟨*ltxml⟩
DefEnvironment('{subproof} OptionalKeyVals:pf {}',
       "<omdoc:derive "
              .     "?&defined(&GetKeyVal(#1,'id'))(xml:id='&GetKeyVal(#1,'id')')()>\n"
      .  "<omdoc:CMP>#2</omdoc:CMP>\n"
      .  "<omdoc:method ?&defined(&GetKeyVal(#1,'method'))(xref='&GetKeyVal(#1,'method')')>"
              .    "<omdoc:proof>#body</omdoc:proof>"
              .  "</omdoc:method>"
     . "</omdoc:derive>\n");
⟨/ltxml⟩
```

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

```
⟨*package⟩
\newenvironment{spfcases}[2][]{\def\@test{#1}%
\ifx\@test\empty\begin{subproof}[method=by-cases]{#2}%
\else\begin{subproof}[#1,method=by-cases]{#2}\fi}
{\end{subproof}}
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{spfcases} OptionalKeyVals:pf {}',
       "<omdoc:derive "
              .     "?&defined(&GetKeyVal(#1,'id'))(xml:id='&GetKeyVal(#1,'id')')()>\n"
      .  "<omdoc:CMP>#2</omdoc:CMP>\n"
      .  "<omdoc:method ?&defined(&GetKeyVal(#1,'method'))(xref='&GetKeyVal(#1,'method')')>"
              .    "#body"
              .  "</omdoc:method>"
     . "</omdoc:derive>\n");
⟨/ltxml⟩
```

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

```
⟨*package⟩
\newenvironment{spfcase}[2][]{\metasetkeys{spf}{#1}%
\ifx\spf@display\spf@flow\else\item[\the@pst@label]\fi%
\def\@test{#2}\ifx\@test\@empty\else{\stDMemph{#2}: }\fi% need blank here
\begin{pst@with@label}{\pst@label,\number\count\count10}}
{\ifx\spf@display\spf@flow\else\sproofend\fi\end{pst@with@label}\next@pst@label}
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{spfcase} OptionalKeyVals:pf{}',
       "<omdoc:proof "
              .  "?&defined(&GetKeyVal(#1,'id'))(xml:id='&GetKeyVal(#1,'id')')()>\n"
       .  "?#2(<omdoc:omtext>"
              .       "<omdoc:CMP>#2</omdoc:CMP>"
              .     "</omdoc:omtext>\n)()"
       . "#body"
     . "</omdoc:proof>\n");
⟨/ltxml⟩
```

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

```
\newcommand\spfcasesketch[3][]{\metasetkeys{spf}{#1}%
\ifx\spf@display\spf@flow\else\item[\the@pst@label]\fi%
\def\@test{#2}\ifx\@test\@empty\else{\stDMemph{#2}: }\fi#3%
\next@pst@label}
```

```
DefConstructor('\spfcasesketch OptionalKeyVals:pf{}{}',
       "<omdoc:proof "
                .  "?&defined(&GetKeyVal(#1,'id'))(xml:id='&GetKeyVal(#1,'id')')()>\n"
       .  "?#2(<omdoc:omtext>"
              .        "<omdoc:CMP>#2</omdoc:CMP>"
               .      "</omdoc:omtext>\n)()"
       . "#3"
     . "</omdoc:proof>\n");
```

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.

```
\srefaddidkey{just}
\addmetakey{just}{method}
\addmetakey{just}{premises}
\addmetakey{just}{args}
```

```
DefKeyVal('just','id','Semiverbatim');
DefKeyVal('just','method','Semiverbatim');
DefKeyVal('just','premises','Semiverbatim');
DefKeyVal('just','args','Semiverbatim');
```

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`

```
\newenvironment{justification}[1][]{}{}
```

```
sub extractBodyText {
  my ($box, $remove) = @_;
  my $str = '';
  my @boxes = $box->unlist;
```

[7] EdNote: need to do something about the premise in draft mode.

```
  foreach my $b(@boxes) {
    my $s = '';
    if ($b =~ /LaTeXML::Whatsit/) {
      my $body = $b->getBody;
      $s = $body ? extractBodyText($body, $remove) : '';
    } elsif ($b =~ /LaTeXML::Box/) {
      $s = $b->toString || '';
      @{$b}[0] = '' if $remove; }
    $str .= $s; }
  $str =~ s/\s+/ /g;
  $str; }

DefEnvironment('{justification} OptionalKeyVals:just', sub {
 my ($doc, $keys, %props) = @_;
 my $text = extractBodyText($props{body}, 1);
 my $node = LookupValue('_LastSeenCMP');
 #$node->appendText($text) if $node;
 my $method = $keys ? $keys->getValue('method') : undef;
 $doc->openElement("omdoc:method", $method ? (xref => $method) : ());
 $doc->absorb($props{body}) if $props{body};
 $doc->closeElement("omdoc:method");
 return; });
⟨/ltxml⟩
```

\premise

```
⟨*package⟩
\newcommand\premise[2][]{#2}
⟨/package⟩
⟨*ltxml⟩
DefMacro('\premise[]{}', sub {
   my ($xref, $text) = ($_[1], $_[2]);
   my @res = (T_CS('\premise@content'));
   push(@res, T_OTHER('['), $xref->unlist, T_OTHER(']')) if $xref;
   push(@res, T_SPACE, $text->unlist) if $text;
   @res; });
DefConstructor('\premise@content[]',
               "<omdoc:premise xref='#1'/>");
⟨/ltxml⟩
```

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

```
⟨*package⟩
\newcommand\justarg[2][]{#2}
⟨/package⟩
⟨*ltxml⟩
DefMacro('\justarg[]{}', sub { (($_[1] ? $_[1]->unlist : ()),
T_SPACE, $_[2]->unlist, T_SPACE); });
Tag('omdoc:derive', afterClose=>sub {
      my ($doc, $node) = @_;
      my @children = grep($_->nodeType == XML_ELEMENT_NODE, $node->childNodes);
```

```
285       my $firstCMP = undef;
286       foreach my $child(@children) {
287         next unless ($child->localname || '') eq 'CMP';
288         if ($child->hasChildNodes()) {
289           next unless $#{$child->childNodes} == 0;
290           next unless $child->firstChild->nodeType == XML_TEXT_NODE; }
291
292         if ($firstCMP) {
293           $firstCMP->appendText($child->textContent);
294           $node->removeChild($child);
295         } else { $firstCMP = $child; }
296       }
297     });#$
298 ⟨/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`.

```
299 ⟨*ltxml⟩
300 Tag('omdoc:proof',afterOpen=>\&numberIt,afterClose=>\&locateIt);
301 Tag('omdoc:derive',afterOpen=>\&numberIt,afterClose=>\&locateIt);
302 Tag('omdoc:method',afterOpen=>\&numberIt,afterClose=>\&locateIt);
303 Tag('omdoc:premise',afterOpen=>\&numberIt,afterClose=>\&locateIt);
304 Tag('omdoc:derive',afterOpen=>\&numberIt,afterClose=>\&locateIt);
305 ⟨/ltxml⟩
```

5 Finale

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

```
306 ⟨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.