`structview.sty`: Structures and Views in sTeX[*]

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

April 7, 2016

Abstract

The `structview` 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 infrastructure for OMDoc structures and views: complex semantic relations between modules/theories.

---

[*]Version v1.4 (last revised 2016/04/06)

Contents

1 Introduction

[1]

2 The User Interface

The main contributions of the modules package are the module environment, which allows for lexical scoping of semantic macros with inheritance and the \symdef macro for declaration of semantic macros that underly the module scoping.

2.1 Package Options

showmods

qualifiedimports

noauxreq

showmeta

The modules package takes two options: If we set showmods[2], then the views (see Section 2.3) are shown. If we set the qualifiedimports option, then qualified imports are enabled. Qualified imports give more flexibility in module inheritance, but consume more internal memory. As qualified imports are not fully implemented at the moment, they are turned off by default see Limitation **??**. The option noauxreq prohibits the registration of \@requiremodules commands in the aux file. They are necessary for preloading the module signatures so that entries in the table of contents can have semantic macros; but as they sometimes cause trouble the option allows to turn off preloading.

If the showmeta is set, then the metadata keys are shown (see [Koh15] for details and customization options).

2.2 Structures

importmodulevia

\vassign

\fassign

\tassign

The \importmodule macro has a variant \importmodulevia that allows the specification of a theory morphism to be applied. \importmodulevia{⟨thyid⟩}{⟨assignments⟩} specifies the "source theory" via its identifier ⟨thyid⟩ and the morphism by ⟨assignments⟩. There are four kinds:[3]

symbol assignments via \vassign{⟨sym⟩}{⟨exp⟩}, which defines the symbol ⟨sym⟩ introduced in the current theory by an expression ⟨exp⟩ in the source theory.

function assignments via \fassign{⟨bvars⟩}{⟨pat⟩}{⟨exp⟩}, is a variant which defines a function symbol ⟨sym⟩ introduced in the current theory by mapping a pattern expression ⟨pat⟩ (⟨sym⟩ applied to ⟨bvars⟩) to an expression ⟨exp⟩ in the source theory on bound variables ⟨bvars⟩.

term assignments via \tassign[⟨source-cd⟩]{⟨tname⟩}{⟨source-tname⟩}, which assigns to the term with name ⟨tname⟩ in the current theory a term with name ⟨source-tname⟩ in the theory ⟨source-cd⟩ whose default value is the source theory.

---

[1]EDNOTE: What are structures and views?

[2]EDNOTE: This mechanism does not work yet, since we cannot disable it when importing modules and that leads to unwanted boxes. What we need to do instead is to tweak the sms utility to use an internal version that never shows anything during sms reading.

[3]EDNOTE: MK: this needs to be consolidated and researched better.

\ttassign **term text assignments** via `\tassign{⟨tname⟩}{⟨text⟩}`, which defines a term with name ⟨tname⟩ in the current theory via a definitional text.

```
\begin{module}[id=ring]
\begin{importmodulevia}{monoid}
  \vassign{rbase}\magbase
  \fassign{a,b}{\rtimes{A}B}{\magmaop{a}b}
  \vassign{rone}\monunit
\end{importmodulevia}
\symdef{rbase}{G}
\symdef[name=rtimes]{rtimesOp}{\cdot}
\symdef{rtimes}[2]{\infix\rtimesOp{#1}{#2}}
\symdef{rone}{1}
\begin{importmodulevia}{cgroup}
  \vassign{rplus}\magmaop
  \vassign{rzero}\monunit
  \vassign{rinvOp}\cginvOp
\end{importmodulevia}
\symdef[name=rplus]{rplusOp}{+}
\symdef{rplus}[2]{\infix\rplusOp{#1}{#2}}
\symdef[name=rminus]{rminusOp}{-}
\symdef{rminus}[1]{\infix\rminusOp{#1}{#2}}
...
\end{module}
```

Example 1: A Module for Rings with inheritance from monoids and commutative groups

2.3   Views

A view is a mapping between modules, such that all model assumptions (axioms) of the source module are satisfied in the target module. [4]

3   Limitations & Extensions

In this section we will discuss limitations and possible extensions of the `modules` package. Any contributions and extension ideas are welcome; please discuss ideas, requests, fixes, etc on the sTEX TRAC [**sTeX:online**].

---

[4]EDNOTE: Document and make Examples

4 The Implementation

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). The options we are not using, we pass on to the sref package we require next.

```
1 ⟨∗package⟩
2 \newif\if@structview@mh@\@structview@mh@false
3 \DeclareOption{mh}{\@structview@mh@true
4   \PassOptionsToPackage{\CurrentOption}{modules}}
5 \DeclareOption*{\PassOptionsToPackage{\CurrentOption}{modules}}
6 \ProcessOptions
```

The next measure is to ensure that the sref and xcomment packages are loaded (in the right version). For LaTeXML, we also initialize the package inclusions.

```
7 \if@structview@mh@\RequirePackage{structview-mh}\fi
8 \RequirePackage{modules}
```

4.2 Structures

\importmodulevia The importmodulevia environment just calls \importmodule, but to get around the group, we first define a local macro \@@doit, which does that and can be called with an \aftergroup to escape the environment grouping introduced by importmodulevia.

```
9 \newenvironment{importmodulevia}[2][]{%
10   \gdef\@@doit{\importmodule[#1]{#2}}%
11   \ifmod@show\par\noindent importing module #2 via \@@doit\fi%
12 }{%
13   \aftergroup\@@doit\ifmod@show end import\fi%
14 }%
```

*assign

```
15 \newrobustcmd\vassign[3][]{\ifmod@show\ensuremath{#2\mapsto #3}, \fi}%
16 \newrobustcmd\tassign[3][]{\ifmod@show #2\ensuremath{\mapsto} #3, \fi}%
17 \newrobustcmd\fassign[4][]{\ifmod@show \ensuremath{#3\mapsto #4}, \fi}%
18 \newrobustcmd\ttassign[3][]{\ifmod@show #2\ensuremath{\mapsto} ``#3'', \fi}%
```

4.3 Views

We first prepare the ground by defining the keys for the view environment.

```
19 \srefaddidkey{view}
20 \addmetakey*{view}{title}
21 \addmetakey{view}{display}
22 \addmetakey{view}{from}
23 \addmetakey{view}{to}
24 \addmetakey{view}{creators}
```

```
25 \addmetakey{view}{contributors}
26 \addmetakey{view}{srccite}
27 \addmetakey{view}{type}
28 \addmetakey[sms]{view}{ext}
```

\view@heading Then we make a convenience macro for the view heading. This can be customized.

```
29 \newcounter{view}[section]
30 \newrobustcmd\view@heading[4]{%
31   \if@importing%
32   \else%
33     \stepcounter{view}%
34     \edef\@display{#3}\edef\@title{#4}%
35     \noindent%
36       \ifx\@display\st@flow%
37       \else%
38         {\textbf{View} {\thesection.\theview} from \textsf{#1} to \textsf{#2}}%
39         \sref@label@id{View \thesection.\theview}%
40         \ifx\@title\@empty%
41           \quad%
42         \else%
43           \quad(\@title)%
44         \fi%
45         \par\noindent%
46       \fi%
47       \ignorespaces%
48   \fi%
49 }%ifmod@show
```

view The `view` environment relies on the `@view` environment (used also in the sTeX module signatures) for module bookkeeping and adds presentation (a heading and a box) if the `showmods` option is set.

```
50 \newenvironment{view}[3][]{%
51   \metasetkeys{view}{#1}%
52   \sref@target%
53   \begin{@view}{#2}{#3}%
54   \view@heading{#2}{#3}{\view@display}{\view@title}%
55 }{%
56   \end{@view}%
57   \ignorespaces%
58 }%
59 \ifmod@show\surroundwithmdframed{view}\fi%
```

@view The `@view` does the actual bookkeeping at the module level.

```
60 \newenvironment{@view}[2]{%from, to
61   \@importmodule[\view@from]{#1}{\view@ext}%
62   \@importmodule[\view@to]{#2}{\view@ext}%
63 }{}%
```

viewsketch The `viewsketch` environment behaves like `view`, but only has text contents.

```
64 \newenvironment{viewsketch}[3][]{%
65   \metasetkeys{view}{#1}%
66   \sref@target%
67   \begin{@view}{#2}{#3}%
68   \view@heading{#2}{#3}{\view@display}{\view@title}%
69 }{%
70   \end{@view}%
71 }%
72 \ifmod@show\surroundwithmdframed{viewsketch}\fi%
```

EdN:5 \qquad \obligation The \obligation element does not do anything yet on the latexml side.[5]

```
73 \newrobustcmd\obligation[3][]{%
74   \if@importing%
75   \else Axiom #2 is proven by \sref{#3}%
76   \fi%
77 }%
78 ⟨/package⟩
```

---

[5]EDNOTE: document above

7