

structview.sty: Structures and Views in \LaTeX^*

Michael Kohlhase
FAU Erlangen-Nürnberg
<http://kwarc.info/kohlhase>

October 20, 2020

Abstract

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

Contents

1	Introduction	2
2	The User Interface	2
2.1	Package Options	2
2.2	Theory Morphisms	2
2.3	Structures	3
2.4	Views	3
3	Limitations & Extensions	5
4	The Implementation	5
4.1	Package Options	5
4.2	Theory Morphisms by Assignments	6
4.3	Structures	6
4.4	Views	6

*Version v1.6 (last revised 2020/10/14)

1 Introduction

Structures and views constitute ways of defining and relating theories in a theory graph that considerably extend the “object-oriented inheritance” constituted by the imports relation given by the `STEX module` package.

Structures are like imports, only that they allow to define new theories via inheritance with renaming. Views relate pre-existing theories and model conceptual refinements, framing, and implementation relations, again via a mapping between the languages defined by the source and target theories; we call these mappings **theory morphisms**.

For details about theory morphisms we refer to [RK13], but hope to make the underlying concepts clear with examples.

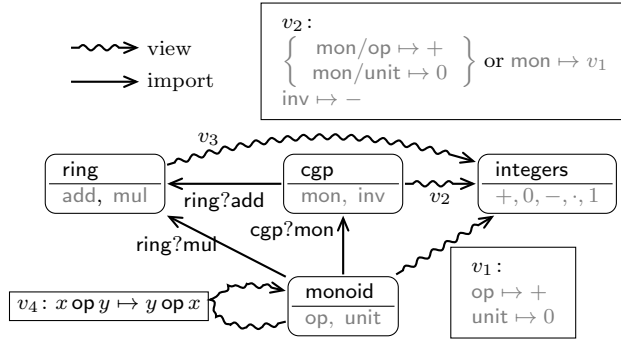


Figure 1: A Theory Graph with Structures and Views

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

`mh` The `mh` option turns on MathHub support; see [Koh20a].

2.2 Theory Morphisms

A theory morphism is a mapping between the languages of its source and target theory. This can be described mathematically using all the structures in the

¹EDNOTE: explain the contribution of structures and views to theory graphs and synchronize with Figure 1.

gTeX distribution. However, in many situations, the language transformation of a morphism can be given in form of **assignments** that map symbols of the source theory to expressions of the target theory.

EdN:2

There are three kinds assignments:²

`\vassign` **symbol assignments** via `\vassign{⟨sym⟩}{⟨exp⟩}`, which maps a symbol $\langle sym \rangle$ from source theory an expression $\langle exp \rangle$ in the target theory.

`\fassign` **function assignments** via `\fassign{⟨bvars⟩}{⟨pat⟩}{⟨exp⟩}`, is a variant which maps a function symbol $\langle sym \rangle$ by mapping a pattern expression $\langle pat \rangle$ ($\langle sym \rangle$ applied to $\langle bvars \rangle$) to an expression $\langle exp \rangle$ in the target theory on bound variables $\langle bvars \rangle$.

`\tassign` **term assignments** via `\tassign{⟨sym⟩}{⟨tname⟩}`, another special case, where the value is the symbol with name $\langle tname \rangle$ in the target theory.

EdN:3

Figure 1 shows a concrete example³

The assignments above can be seen as abbreviations for a simple, formal definitions, which define a symbol of the source theory by an expression in the target theory.

2.3 Structures

`structure` Structures are specified by the `sstructure`¹ environment:

`\begin{sstructure}[⟨keys⟩]{⟨name⟩}{⟨sthy⟩}{⟨morph⟩}\end{sstructure}`

gives the structure the name $\langle name \rangle$, specifies the “source theory” via its identifier $\langle sthy \rangle$, and the morphism $\langle morph \rangle$. The `structure` environment takes the same keys as the `\importmodule` macro, which it generalizes. The morphism $\langle morph \rangle$ in the body of the `structure` environment specifies the morphism (see 2.2 above). In a structure, we take the target theory to be the current theory.

2.4 Views

A view is a mapping between modules, such that all model assumptions (axioms) of the source module are satisfied in the target module. For marking up views

`view` the `structview` package supplies the `view` environment; see Figure 2 for the gTeX markup of view v_1 from Figure 1. The `view` environment takes one optional key/value argument followed by two mandatory ones: the names of the source and target modules. The `view` environment takes the following keys: `id` for a name, `title` and `display` for visual presentation, `loadfrom`, `loadto`, and `ext`⁴ for specifying the source files that supply the source and target modules, `creators`, `contributors`, `srccite` for document metadata, and `type`⁵.

EdN:4

EdN:5

²EdNOTE: MK: we need better macros here.

³EdNOTE: adapt when we fully understand this, and the implementation works.

¹The old `importmodulevia` environment is now deprecated.

⁴EdNOTE: MK: we probably need `toext` and `fromext` here, but this never came up yet.

⁵EdNOTE: ????

```

\begin{module}[id=ring]
\symdef{rbase}{R}
\symdef{rtimes}[2]{\infix\cdot{#1}{#2}}
\symdef{rone}{1}
\begin{sstructure}{mul}{monoid}
\tassign{magbase}{rbase}
\fassign{a,b}{\magmaop{a}b}{\rtimes{a}b}
\tassign{monunit}{rone}
\end{sstructure}
\symdef{rplus}[2]{\infix+{#1}{#2}}
\symdef{rminus}[1]{\infix-{#1}{#2}}
\begin{sstructure}{add}{cgroup}
\fassign{a,b}{\magmaop{a}b}{\rplus{a}b}
\tassign{monunit}{rzero}
\tassign{cginvOp}{\rminus}
\end{sstructure}
...
\end{module}

```

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

```

\begin{view}{monoid}{integers}
\vassign{magbase}{base}
\fassign{a,b}{\magmaop{a}b}{\inttimes{a,b}}
\tassign{monunit}{\intzero}
\begin{assertion}
The Integers with addition form a monoid in the obvious way.
\end{assertion}
\end{view}

```

Example 2: A view from monoids to integers

Just as for other statements (see [Koh20b]), we have an inline version of views that can be embedded into other statements: `\inlineview`. Intuitively, **inline views** are like inline assertions, however, they can bring along concepts from the source module. Example 3 shows a typical situation: the directory structure in a hierarchical file system forms a tree (the view), and we inherit concepts from that.

```
\begin{definition}
  \inlineview{tree}{The \treffi[hfs]{directory} structure in a
    \treffiii[hfs]{hierarchical}{file}{system} induces a \treffi[tree]{tree}
    or \treffi[cycle]{DAG}, so we inherit the concepts of (file system)
    \drefi[tree?root]{root}, \drefi[tree?parent]{parent} (directory),
    \drefi[tree?child]{child} from there.}
  ...
\end{definition}
```

Example 3: An inline view that brings along concepts.

Note that `\inlineview` does not specify the target module, that is the current module. We provide the `inlineView` environment as a block-level alternative.

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 \LaTeX issue tracker at [sTeX].

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 \LaTeX ML, we also initialize the package inclusions.

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

4.2 Theory Morphisms by Assignments

`*assign` 6

```
9 \newrobustcmd\vassign[3] []{\ifmod@show\ensuremath{#2\mapsto #3}, \fi\ignorespacesandpars}%
10 \newrobustcmd\fassign[4] []{\ifmod@show \ensuremath{#3(#2)\mapsto #4}, \fi\ignorespacesandpars}%
11 \newrobustcmd\tassign[3] []{\ifmod@show \ensuremath{#2\mapsto} #3, \fi\ignorespacesandpars}%
```

4.3 Structures

`sstructure` The `structure` 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 `structure`.

```
12 \newenvironment{sstructure}[3] []{%
13   \gdef\@@doit{\importmodule[#1]{#3}}%
14   \ifmod@show\par\noindent importing module #3 via \@@doit\fi%
15 }{%
16   \aftergroup\@@doit\ifmod@show end import\fi%
17 }%
```

`importmodulevia` This is now deprecated, we give an error, but punt to `structure`.

```
18 \newenvironment{importmodulevia}[2] []%
19 {\PackageError{structview}%
20  {The {importmodulevia} environment is deprecated}{use the {sstructure} instead!}%
21  \begin{sstructure}[#1]{missing}{#2}}
22 {\end{sstructure}}
```

4.4 Views

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

```
23 \srefaddidkey{view}
24 \addmetakey*{view}{title}
25 \addmetakey{view}{display}
26 \addmetakey{view}{loadfrom}
27 \addmetakey{view}{loadto}
28 \addmetakey{view}{creators}
29 \addmetakey{view}{contributors}
30 \addmetakey{view}{srccite}
31 \addmetakey{view}{type}
```

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

```
32 \ifdef{\thesection}{\newcounter{view}[section]}{\newcounter{view}}
33 \newrobustcmd\view@heading[4]{%
34   \if@importing%
35   \else%
36     \stepcounter{view}%
37     \edef\@display{#3}\edef\@title{#4}%
```

⁶EdNOTE: probably get rid of the optional argument

```

38 \noindent%
39 \ifx\@display\st@flow%
40 \else%
41 {\textbf{View} {\thesection.\theview} from \textsf{#1} to \textsf{#2}}%
42 \sref@label@id{View \thesection.\theview}%
43 \ifx\@title\@empty%
44 \quad%
45 \else%
46 \quad(\@title)%
47 \fi%
48 \par\noindent%
49 \fi%
50 \ignorespacesandpars%
51 \fi%
52 }%ifmod@show

```

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

```

53 \newenvironment{view}[3][\keys, from, to
54 \metasetkeys{view}{#1}%
55 \sref@target%
56 \begin{@view}{#2}{#3}%
57 \view@heading{#2}{#3}{\view@display}{\view@title}%
58 }{%
59 \end{@view}%
60 \ignorespacesandpars%
61 }%
62 \ifmod@show\surroundwithmdframed{view}\fi%

```

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

```

63 \newenvironment{@view}[2]{\from, to
64 \@importmodule[\view@loadfrom]{#1}{export}%
65 \@importmodule[\view@loadto]{#2}{export}%
66 }{}%

```

viewsketch The `viewsketch` environment is deprecated, we give an error

```

67 \newenvironment{viewsketch}[3][\%
68 {\PackageError{structview}%
69 {The {viewsketch} environment is deprecated}{use the {view} environment instead!}%
70 \begin{view}[#1]{#2}{#3}}
71 {\end{view}}

```

inlineView We essentially do the same as for the `view` environment, but we are already in a module, hence we can use `\module@id` for the target module and do not have to load it. All presentational keys are ignored.

```

72 \newenvironment{inlineView}[2][\keys, source
73 \metasetkeys{view}{#1}\sref@target%
74 \@importmodule[\view@loadfrom]{#2}{export}%
75 \ignorespacesandpars}
76 {\ignorespacesandpars}

```

```

inlineview
77 \newcommand\inlineview[3] [] {\begin{inlineView}[#1]{#2}{\module@id}#3\end{inlineView}}

EdN:7 \obligation The \obligation element does not do anything yet on the latexml side.7
78 \newrobustcmd\obligation[3] [] {%
79   \if@importing%
80   \else Axiom #2 is proven by \sref{#3}%
81   \fi%
82 }%
83 \end{package}

```

⁷EdNOTE: document above