

structview.sty: Structures and Views in \S T E X^*

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

November 16, 2017

Abstract

The **structview** package is part of the \S T E X collection, a version of $\text{T E X}/\text{\La T E X}$ that allows to markup $\text{T E X}/\text{\La T E X}$ documents semantically without leaving the document format, essentially turning $\text{T E X}/\text{\La T E X}$ 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	5
4.3	Structures	5
4.4	Views	6

*Version v1.4 (last revised 2016/04/07)

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.

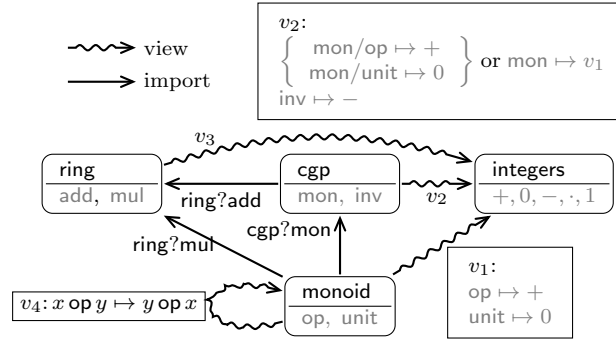


Figure 1: A Theory Graph with Structures and Views

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.

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

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].

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 \langle *package\rangle
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 Theory Morphisms by Assignments

```
6
9 \newrobustcmd\vassign[3] []{\ifmod@show\ensuremath{#2\mapsto #3}, \fi}%
10 \newrobustcmd\fassign[4] []{\ifmod@show \ensuremath{#3(#2)\mapsto #4}, \fi}%
11 \newrobustcmd\tassign[3] []{\ifmod@show \ensuremath{#2\mapsto} #3, \fi}%
```

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 }%
```

⁶EdNOTE: probably get rid of the optional argument

`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}
32 \addmetakey[sms]{view}{ext}

```

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

```

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

```

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

```

54 \newenvironment{view}[3] []{% keys, from, to
55  \metasetkeys{view}{#1}%

```

```

56 \sref@target%
57 \begin{@view}{#2}{#3}%
58 \view@heading{#2}{#3}{\view@display}{\view@title}%
59 }{%
60 \end{@view}%
61 \ignorespaces%
62 }%
63 \ifmod@show\surroundwithmdframed{view}\fi%

```

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

```

64 \newenvironment{@view}[2]{%from, to
65 \@importmodule[\view@loadfrom]{#1}{\view@ext}%
66 \@importmodule[\view@loadto]{#2}{\view@ext}%
67 }{}%

```

viewsketch The viewsketch environment is deprecated, we give an error

```

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

```

EdN:7 **\obligation** The \obligation element does not do anything yet on the latexml side.⁷

```

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

```

⁷EdNOTE: document above