TIKZSD VERSION 1.0.0 USER MANUAL

Contents

1. Introduction

This is a manual for version 1.0.0 of `tikzsd`, a Haskell program which generates code for inclusion into a LaTeX file. The generated code is a `tikzpicture` environment.

The program works by defining categorical objects, and then specifying a natural transformation using those categorical objects, with notation for how it is to be drawn.

For example,

```
define category C "$C$"
define category D "$D$"
define functor F "$F$"
    source : C
    target : D
define functor G "$G$"
    source : D
    target : C
define natural transformation unit "$\eta$"
    source : C \\
    target : F & G \\
    shape : utriangle
define natural transformation counit "$\varepsilon$"
    source : G & F \\
```

```
    target : D \\
    shape : btriangle
```

will define the categorical objects needed to specify an adjunction. The notation for these statements is explained in section .

If one then writes

```
draw adjunction-eq-1.tex
    f : & & F \\
    n : unit & \\
    f : F & G & F \\
    n : & counit \\
    f : F & & \\
```

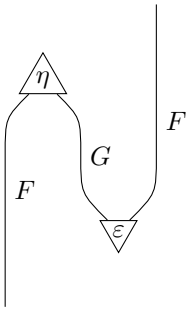
the program will write code to the file `adjunction-eq-1.tex`. This code will be a `tikzpicture` environment, which can be included in LaTeX code. For example, if one writes

```
\input adjunction-eq-1
```

in a LaTeX file which contains the lines

```
\usepackage{tikz}
\usetikzlibrary{shapes}
\tikzset{utriangle/.style={regular polygon, regular polygon sides=3,
    inner sep=0.5pt}}
\tikzset{btriangle/.style={regular polygon, regular polygon sides=3,
    shape border rotate=180, inner sep=0.5pt}}
```

in the preamble and is in the same folder as `adjunction-eq-1.tex`, then the following string diagram



will be created at the corresponding place in the document.

2. Usage

Once installed, one can invoke the program by typing in a terminal

```
$ tikzsd file1 ...  filen
```

where `file1 ...  filen` is a list of one or more text files containing code to be processed by the program. The program processes the commands in these files in order.

In our examples, we have named our text files with code for processing by `tikzsd` with extension `.tzsd`, i.e. `file1.tzsd`. This is not necessary, and just our convention. In the rest of this document, by a `.tzsd` file we mean a text file with code to be processed by `tikzsd`.

One of the commands which can be given in a `.tzsd` file is a draw command telling the program to output TikZ code to a file (see section 4). This file can then be `\input` into a `.tex` file where you want the corresponding diagram.

We recommend looking at the example in `/doc/examples/benabou-roubaud/` of this repository for an example usage of this program.

3. Defining Categorical Objects

3.1. **Defining categories.** One defines a category using the following notation:

```
define category <id> "<latex-string>"
```

where

- `<id>` is a string of characters which does not contain a space character, or one of the characters `&`, `\`, `[`, or `]`. This `<id>` is used to refer to the category in the rest of the `.tzsd` file.
- `<latex-string>` is a string of characters which does not contain the character `"`. This string contains the LaTeX code used to label the category.

The spaces in the above code may be replaced with any nonempty sequence of space characters, such as spaces, tabs or new lines.

3.1.1. **Example.** The code

```
define category C "$C$"
```

will define a category, which is referred to by the id `C` in the rest of the `.tzsd` file, and which will be labeled using `$C$`.

3.1.2. *Remark.* In version 1.0.0 of `tikzsd`, the `<latex-string>` is never used. There is at least one planned future feature which will use it. This `<latex-string>` will likely be made optional in a future release.

3.2. **Defining functors.** One defines a functor using the following notation:

```
define functor <id> "<latex-string>" source:<s-id> target:<t-id>
```

where

- `<id>` is a string of characters which does not contain a space character, or one of the characters `&`, `\`, `[`, or `]`. This `<id>` is used to refer to the functor in the rest of the `.tzsd` file.
- `<latex-string>` is a string of characters which does not contain the character `"`. This string contains the LaTeX code used to label the functor.
- `<s-id>` is the id of the category which is the source of the functor.
- `<t-id>` is the id of the category which is the target of the functor.

The spaces in the above code may be replaced by any nonempty sequence of space characters, such as spaces, tabs or new lines. One may also insert space characters after `source` or `target` before the `:` character, or after the `:` character before the `<s-id>` or `<t-id>`.

3.2.1. **Example.** The code

```
define functor F "$F$"
    source : C
    target : D
```

will define a functor, which is referred to by the id `F` in the rest of the `.tzsd` file, and which will be labelled using `$F$`. This functor has source the category `C` and target the category `D`. The program will print an error if the category `C` or the category `D` have not been defined.

Due to how spacing works, one can also put all of the above code on one line. So

```
define functor F "$F$" source:C target:D
```

is equivalent to the code at the beginning of this example.

3.3. **Specifying compositions of functors (and spacing).** A composition of functors is specified by a `&` separated list of elements which is terminated by the two characters `\\`, as below:

```
<elem-1> & <elem-2> & ... & <elem-n> \\
```

Here, `<elem-i>` corresponds to the ith column. Each `<elem-i>` is either

(1) The id of a functor.
(2) The id of a category, corresponding to the identity functor of that category.
(3) An empty string. This is a placeholder used for spacing purposes.

At least one of the elements is not of type (3). The spaces in the above code may be replaced with zero or more space characters, such as spaces, tabs and new lines.

In a composition of functors, we shall call elements of the first type *basic functors*. These will correspond to strings in a string diagram. The length of the composite will be the number of basic functors in the composite.

As is conventional for how string diagrams are drawn, the composition is from left to right. So to specify a composition $F \circ G$, one might write

```
G & F \\
```

which corresponds to a string for G in the first column and a string for F in the second column, or one might write

```
 & G & & F \\
```

which corresponds to a string for G in the second column, and a string for F in the fourth column.

3.3.1. *Remark.* Since identity functors are not drawn by the program, one can often omit elements of type (2) and use empty elements (3) instead.

For example, if $F : C \to D$ and $G : D \to C$ are functors, then

```
F & D & G \\
```

and

```
F & & G \\
```

both specify the same composite functor $G \circ F$ and the same spacing for the basic functors F and G.

The one exception to this is when specifying an identity functor by itself:

```
C \\
```

cannot be changed to

```
 \\
```

because an empty composition is not well-defined, so the meaning of a composition with only empty elements cannot be imputed.

3.4. **Defining natural transformations.** One defines a natural transformation using the following notation:

```
define natural transformation <id> "<latex-string>"
    source : <source-composition>
    target : <target-composition>
    shape : <shape>
```

where

(1) `<id>` is a string of characters which does not contain a space character, or one of the characters `&`, `\`, `[`, or `]`. This `<id>` is used to refer to the natural transformation in the rest of the `.tzsd` file.
(2) `<latex-string>` is a string of characters which does not contain the character `"`. This string contains the LaTeX code used to label the natural transformation.
(3) `<source-composition>` is code used to specify a composite functor which is the source of the natural transformation. Notation for this code is described in 3.3.
(4) `<target-composition>` is code used to specify a composite functor which is the target of the natural transformation. Notation for this code is described in 3.3.
(5) The line `shape:<shape>` is optional. If it exists, `<shape>` is a string of alphanumeric characters (no spaces or punctuation), and is LaTeX code used to specify the shape of the node representing the natural transformation. If it doesn't exist, no LaTeXcode specifying the shape is passed to the node, and the shape of the node will be the default shape in your `tikzpicture` environment.

In the current version of the program, the two composite functors specified by `<source-composition>` and `<target-composition>` cannot both be the identity functor of a category C. Conventionally, string diagrams do not draw identity functors, so the node corresponding to the natural transformation would be a very ambiguous node with no in or out strings.

3.4.1. *Remark.* In `<source-composition>` and `<target-composition>`, only the composite functor is desired, so the particular spacing given in these strings is ignored. In other words, for the purposes of specifying the source or target of a natural transformation,

```
    F & G \\
```

and

```
F & & G \\
```

are equivalent.

3.4.2. *Remark.* The string `<shape>` is just treated as LaTeX code which is passed to every node for the natural transformation. In particular, you may need to define the shape in your LaTeX file if it is not a shape given by the package `tikz`.

3.4.3. **Example.** The code

```
define natural transformation unit "$\eta$"
    source : C \\
    target : F & G \\
    shape : utriangle
```

defines a natural transformation, referred to by the id `unit` in the rest of the `.tzsd` file, and which will be labeled by using `$\eta$`. When represented by a tikz node, the program will pass the string `utriangle` as an option. If `C` is the id of a category C, and `F` and `G` are the ids of functors $C \to D$ and $D \to C$, the source of our defined natural transformation is the identity functor of C, and the target is the composition $G \circ F$.

4. Drawing String Diagrams

Once the categorical objects are defined, the program is ready to draw string diagrams. For us, a string diagram is a way of visualizing a vertical composition of horizontal compositions of natural transformations.

4.1. **Draw a natural transformation.** The notation for drawing a string diagram is as follows:

```
draw <output-file> [<options>]
    <line-1>
    <line-2>
    ...
    <line-n>
```

where

(1) `<output-file>` is a string of characters which does not include any space character. This is the name of the file the program will write the LaTeX code for a `tikzpicture` environment drawing the string diagram. The program will overwrite the contents of the file if it exists.
(2) `[<options>]` is optional. If it exists, `<options>` is a string of characters which does not include the characters `[` or `]`. This string is passed as options to the `tikzpicture` written to the output file. For example, one can write `[x=0.5cm,y=0.5cm]` to change the horizontal and vertical spacing.
(3) `<line-1>`, `<line-2>`, ..., `<line-n>` are lines used to specify the string diagram to be drawn. Each line is either a *functor line*, which starts with `f:`, or a *natural transformation line*, which starts with `n:`. Roughly speaking,

each natural transformation line specifies a horizontal composition of defined natural transformations, and the functor lines specify the spacing of the sources and targets of these natural transformations.

We rewrite the example from the introduction here for reference before describing the notation for functor lines and natural transformation lines.

```
draw adjunction-eq-1.tex
    f : & & F \\
    n : unit & \\
    f : F & G & F \\
    n : & counit \\
    f : F & & \\
```

4.2. **Functor lines.** A functor line is of the form

```
f : <composition>
```

where `<composition>` is code for specifying a composition of functors, with spacing, as described in 3.3.

For example,

```
f: F & G & F \\
```

specifies the composition $F \circ G \circ F$, and says that the strings for these functors should be placed in the first, second and third columns.

4.3. **Natural transformation lines.** Natural transformation lines are used to specify horizontal compositions of natural transformations, with empty spaces imputed to be the identity natural transformation.

A natural transformation line is of the form

```
n : <elem-1> & <elem-2> & ... & <elem-n> \\
```

where each `<elem-i>` is either

(1) The id of a natural transformation.
(2) The id of a functor, representing the identity natural transformation of that functor.
(3) The id of a category, representing the identity functor of the identity natural transformation of that category.
(4) An empty string. This tells the program use the expected source of the natural transformation specfied by this line of natural transformations (which comes from the preceding lines) to impute an identity functor at at this position.

The horizontal composition is from left to right, following the conventional layout of a string diagram in category theory.

4.4. **Default spacing.** One can specify the string diagram to be drawn by alternating functor and natural transformation lines, starting and ending with a functor line. The functor lines specify the spacing of the strings for the functors in horizontal cross-sections of the string diagram, while the natural transformation lines specify the natural transformations between these functor lines.

4.4.1. **Example.** For example, we have seen

```
f : & & F \\
n : unit & \\
f : F & G & F \\
n : & counit \\
f : F & & \\
```

which alternates between functor lines and natural transformation lines.

If one omits a functor line, the program can figure out which strings are expected at that cross section from the natural transformation lines, but it cannot figure out the spacing. If a functor line is omitted, the default spacing is used, where the leftmost string gets put in the first column, the second leftmost string in the second columns, ..., until the nth string gets put in the nth column. So to be less verbose, the above example is equivalent to

```
f : & & F \\
n : unit & \\
n : & counit \\
```

Here, we've omitted the second and third functor lines and let the program use the default spacing for these cross sections.

4.5. **Identity imputation.** If a natural transformation line contains a empty element, the program looks at the previous lines and imputes an identity natural transformation to put in the corresponding place.

4.5.1. **Example.** In the example from the introduction of this manual,

```
f : & & F \\
n : unit & \\
```

after processing the first line, the program sees that the current horizontal cross-section has one string, corresponding to the functor with id `F`. When processing the second line, the program sees that the natural transformation with id `unit` has 0 strings coming in and 2 strings coming out. It then imputes that the blank space should be the identity of the first string of the current cross-section. Thus, the program imputes `F` at the empty space, and processes the above as if it were

```
f : & & F \\
n : unit & F \\
```

Similarly,

```
f : F & G & F \\
n : & counit \\
```

is processed as if it were

```
f : F & G & F \\
n : F & counit \\
```

5. Future Roadmap

This is the user manual for version 1.0.0 of the `tikzsd` program. Here is a list of some features planned for future releases.

(1) Give a way to pass options to the labels of the strings. For example passing the option `swap` should put the label on the opposite side.
(2) Give a way to change the positioning of the the labels of the strings/ have multiple labels per string/change the default behavior in other ways.

    By default, the program attempts to put the label at approximately the vertical midpoint of the string. It should be possible to have a way of specifying something like `pos=20`, `pos=above` or `pos=below`. Since each string is actually several Bezier curves spliced together, one cannot just pass this as an option, and the program needs to compute the positioning itself.
(3) Give alternate, less verbose methods of defining functors and natural transformations. Instead of writing `source` and `target` every time, maybe just use an arrow `->` or `=>`.
(4) Give a way to define a functor as a composition of other functors. Then when drawing, give an option to either use one string to represent this functor or to `expand` the functor into its components. So a functor defined to be the composition of two other functors will be drawn as two strings when expanded, and just one string when not expanded. There needs a way to associate a default spacing when expanding this functor.

    One can view identity functors of categories as expanded versions of compositions of no functors.
(5) When drawing natural transformations, for functor lines, allow the specification of spacing without rewriting the names of the functors.
(6) Allow other ways of representing natural transformations. Currently, all natural transformations are represented by a TikZ node. We should be able to allow crossing over, crossing under, etc.
(7) Give notation to define common categorical constructs. For example, there can be simplified notation for defining adjunctions, monads, comonads, algebras for a monad, etc.
(8) Give a way to write comments.
(9) Give a way to change the base directory where the program writes its output.