Arrows manual

Taichi Uemura

1. Introduction

The `arrows` package is a SATySFi package that provides a variety of arrows. It is intended to be used in packages for diagrams (like NCSq[*1]) and for extensible arrows (see Section 4).

An arrow consists of an arrow body, an arrow head, and an arrow tail. Arrow bodies, heads, and tails are designed in such a way that they can be used in arbitrary combination, so a small number of arrow bodies, heads, and tails produce a wide variety of arrows. Here are some examples of arrows drawn with this package.

2. Using arrows

2.1. The `Arrow` module

The `arrows/arrows` package provides the `Arrow` module .

```
type t
```

This is the arrow type.

```
val read : t -> context -> (|
  metadata : arrow-metadata;
  draw : point -> point -> graphics list;
|)
```

The `read` function is a destructor for the type `t`. Use `draw` to draw the arrow as a list of graphics. `metadata` contains metadata like the size of the arrow head and the interval of the arrow body, and is intended to be used for placing labels and for stretching the arrow, for example. See Section 2.5 for details of the type `arrow-metadata`.

1 `https://github.com/uemurax/satysfi-ncsq`

```
val solid : (|
  line-width : float;
  head-size : float;
|) -> t
```

For quick construction of arrows, the `Arrow` module provides some preconfigured arrows like `solid`, `dashed`, `tail`, `harpoon-l` and `mapsto`. See the source `arrows/arrows` for full details.

```
val of-bth : (|
  body : ArrowBody.t;
  tail : ArrowTail.t;
  head : ArrowHead.t;
|) -> t
```

The `of-bth` function constructs an arrow from an arrow body, an arrow tail, and an arrow head. Most arrow bodies, tails, and heads are designed such that they can be used in arbitrary combination. See Sections 2.2, 2.3 and 2.4 for details of these components.

2.2. The `ArrowBody` module

The `arrows/body` package provides the `ArrowBody` module.

```
type t
```

This is the arrow body type.

```
val read : t -> context -> (|
  metadata : arrow-body-metadata;
  draw : (|
    head : arrow-head-metadata;
    tail : arrow-tail-metadata;
  |) -> point -> point -> graphics list;
|)
```

The `read` function is a destructor for the type `t`. The value `metadata` contains metadata

like the interval and the minimum length of the arrow body. See Section 2.5 for details of the type `arrow-body-metadata`. The `draw` function draws the arrow body, but it requires metadata of an arrow head and an arrow tail. This is because the exact graphics of the body of an arrow depend on the head and the tail of the arrow. For example, consider the following two arrows.

Both arrows have the same arrow body (three lines). The left arrow has the empty arrow head, and the right arrow has the arrow head which looks like ">". The exact graphics of the arrow body are different in these arrows: in the left arrow, all the lines have the same length; in the right arrow, the top and bottom lines are shorter than the middle line.

```
val solid : (|
  width : float;
|) -> t
```

For quick construction of arrow bodies, the `ArrowBody` module provides some preconfigured arrow bodies like `solid` and `dashed`. See the source `arrows/body` for full details.

2.3. The `ArrowHead` module

The `arrows/head` package provides the `ArrowHead` module.

```
type t
```

This is the arrow head type.

```
val read : arrow-body-metadata -> t -> context -> (|
  metadata : arrow-head-metadata;
  draw : float -> graphics list;
|)
```

The `read` function is a destructor for the type `t`. The value `metadata` contains metadata like the size of the arrow head. See Section 2.5 for details of the type `arrow-head-metadata`. The `draw` function draws the arrow head at the origin `(0pt, 0pt)` towards the direction given by the first argument in radian. The `read` function requires metadata of an arrow body. This is because the arrow head in an arrow depends on the body of the

arrow. For example, consider the following two arrows.

Both arrows have the same arrow head (looks like ">"). The left arrow has the single line arrow body, and the right arrow has the five lines arrow body. In the right arrow, the arrow head is expanded to match the wide arrow body.

```
val vee : (|
  size : float;
  ext : float;
  line-width : float;
|) -> t
```

For quick construction of arrow heads, the `ArrowHead` module provides some preconfigured arrow heads like `vee`, `triangle` and `harpoon-l`. See the source `arrows/head` for full details.

2.4. The `ArrowTail` module

The package `arrows/tail` provides the `ArrowTail` module. Arrow tails are essentially the same as arrow heads. See the source `arrows/tail` for preconfigured arrow tails.

2.5. Metadata

Global types `arrow-metadata`, `arrow-body-metadata`, `arrow-head-metadata` and `arrow-tail-metadata` are defined in the `arrow/types` package.

```
type arrow-metadata = (|
  body : (|
    left : length;
    right : length;
    length : length;
  |);
  head : (|
    left : length;
    right : length;
```

```
    length : length;
  |);
  tail : (|
    left : length;
    right : length;
    length : length;
  |);
|)
```

`body#left` and `body#right` are the coordinates of the left side and the right side, respectively, of the body facing the direction of the arrow. `body#length` is the minimum length of the body. The other components are similar.

```
type arrow-body-metadata = (|
  mid : (|
    left : length;
    right : length;
  |);
  head : (|
    left : length;
    right : length;
  |);
  tail : (|
    left : length;
    right : length;
  |);
  length : length;
|)
```

Metadata of an arrow body consist of the coordinates of the left sides and the right sides of the middle, the head and the tail of the arrow body facing the direction of the arrow and the minimum length of the arrow body.

```
type arrow-head-tail-metadata = (|
  left : length;
```

```
  right : length;
  length : length;
  depth : length -> length;
|)
type arrow-head-metadata = arrow-head-tail-metadata
type arrow-tail-metadata = arrow-head-tail-metadata
```

Metadata of arrow heads and arrow tails are the same. `left` and `right` are the coordinates of the left side and the right side, respectively, of the arrow head or tail. `length` is the length of the arrow head or tail. The function `depth` tells an arrow body the depth of the arrow head or tail at a position facing the arrow direction. For example, the depth of the arrow head

at a position x is $|x|$, so outer part of an arrow body will be shorter than the center of the arrow body.

3. Defining arrows

```
Arrow.of-bth : (|
  body : ArrowBody.t;
  tail : ArrowTail.t;
  head : ArrowHead.t;
|) -> t
```

An arrow is usually built up out of an arrow body, an arrow tail and an arrow head by the function `Arrow.of-bth`.

```
val invert : t -> t
```

The `Arrow` module provides some constructions of arrows. `invert arr` constructs the arrow in the same style as `arr` but in the opposite direction.

```
val make : (context -> (|
  metadata : arrow-metadata;
```

```
  draw : point -> point -> graphics list;
|)) -> t
```

There is also a raw constructor for arrows, but the interface might be changed in future development.

3.1. Defining arrow bodies

```
val none : t
val union : t -> t -> t
val shift : float -> t -> t
val multiple : int -> float -> t -> t
```

The `ArrowBody` module provides basic constructions of arrow bodies. `multiple n s b` constructs the arrow body consisting of `n` copies of `b` with spacing `s`.

```
val cross : (|
  line-width : float;
  size : float;
  ext : float;
|) -> t -> t
```

`cross rec body` constructs the arrow body from `body` by adjoining a line orthogonal to the arrow direction.

The interface of `cross` might be changed in futuer development (e.g. introducing a type of body decorators).

```
val make : (context -> (|
  metadata : arrow-body-metadata;
  draw : (|
    head : arrow-head-metadata;
    tail : arrow-tail-metadata;
  |) -> point -> point -> graphics list;
```

```
|)) -> t
```

For more complex arrow bodies not realized by basic constructions, the `ArrowBody` module provides a constructor for arrow bodies. However, the interface of `make` might be changed in future development.

3.2. Defining arrow heads and tails

```
val none : t
val union : t -> t -> t
val shift : bool -> float -> t -> t
val duplicate : bool -> int -> float -> t -> t
```

Both the `ArrowHead` module and the `ArrowTail` module provide basic constructions of arrow heads and arrow tails, respectively. `duplicate b n s h` constructs the arrow head consisting of `n` copies of `h` with shift `s`. The boolean value `b` affects the depth of the arrow head constructed. Compare the following.


The left arrow is built widh `duplicate true`, and the right arrow is built with `duplicate false`.

```
val to-head : t -> ArrowHead.t
val from-head : ArrowHead.t -> t
```

The `ArrowTail` module provides head/tail converters. Note that the graphics of an arrow head or tail is rotated 180 degrees by these converters: if `tail` is the following arrow tail,

then `ArrowTail.to-head tail` is the following arrow head.


```
val make : (arrow-body-metadata -> context -> (|
  metadata : arrow-head-metadata;
  draw : float -> graphics list;
```

```
|)) -> t
```

Like the `ArrowBody` module, the `ArrowHead` module and the `ArrowTail` also provide raw constructors, but these interfaces might be changed in future development.

4. Math commands

The `ArrowCommands` module in the `arrows/commands` package provides some math commands for drawing arrows.

```
val \draw : [float?; float?; Arrow.t] math-cmd
```

`\draw!(arr)` draws an arrow `arr`. It accepts two optional arguments. The first one is the y-coordinate in ratio to the font size, and the default value is `0.25`. The second one is the length of the arrow in ratio to the font size , and the default value is `1.0`.

```
val \xrightarrow : [float?; float?; Arrow.t; float?; math] math-cmd
```

`\xrightarrow!(arr){label}` draws an arrow `arr` with label `label`. The arrow automatically extends when `label` is wide, like `\xrightarrow` of the amsmath LaTeX package[*2]. It accepts three optional arguments. The first one is the same as that of `\draw`. The second one is similar to that of `\draw` and specifies the minimum length of the arrow. The third one is the distance from the arrow to the label in ratio to the font size, and the default value is `0.2`.

```
val \xleftarrow : [float?; float?; Arrow.t; float?; math] math-cmd
```

Similar to `\xrightarrow`, but `\xleftarrow` draws an arrow from right to left.

2 `https://www.ctan.org/pkg/amsmath`