typst-theorems

sahasatvik
https://github.com/sahasatvik/typst-theorems

Contents

1. Introduction

The `typst-theorems` package provides `Typst` functions that help create numbered `theorem` environments. This is heavily inspired by the `\newtheorem` functionality of `LaTeX`.

A *theorem environment* lets you wrap content together with automatically updating *numbering* information. Such environments use internal `state` counters for this purpose. Environments can

- share the same counter (*Theorems* and *Lemmas* often do so)
- keep a global count, or be attached to
  - other environments (*Corollaries* are often numbered based upon the parent *Theorem*)
  - headings
- have a numbering level depth fixed (for instance, use only top level heading numbers)
- be referenced elsewhere in the document, via `labels`

2. Using `typst-theorems`

Import all functions provided by `typst-theorems` using

```
#import "theorems.typ": *
#show: thmrules
```

The second line is crucial for displaying `thmenvs` and references correctly!

The core of this module consists of `thmenv`. The functions `thmbox` and `thmplain` provide some simple defaults for the appearance of `thmenvs`.

3. Feature demonstration

Create box-like *theorem environments* using `thmbox`, a wrapper around `thmenv` which provides some simple defaults.

```
#let theorem = thmbox(
  "theorem",              // identifier
  "Theorem",              // head
  fill: rgb("#e8e8f8")
)
```

Such definitions are convenient to place in the preamble or a template; use the environment in your document via

```
#theorem("Euclid")[
  There are infinitely many primes.
] <euclid>
```

This produces the following.

Theorem 3.1 (Euclid): There are infinitely many primes.

Note that the `name` is optional. This `theorem` environment will be numbered based on its parent `heading` counter, with successive `theorems` automatically updating the final index.

The `<euclid>` label can be used to refer to this Theorem via the reference `@euclid`. Go to Section 3.4 to read more.

You can create another environment which uses the same counter, say for *Lemmas*, as follows.

```
#let lemma = thmbox(
  "theorem",              // identifier - same as that of theorem
  "Lemma",                // head
  fill: rgb("#efe6ff")
)

#lemma[
  If $n$ divides both $x$ and $y$, it also divides $x - y$.
]
```

Lemma 3.2: If n divides both x and y, it also divides $x - y$.

You can *attach* other environments to ones defined earlier. For instance, *Corollaries* can be created as follows.

```
#let corollary = thmbox(
  "corollary",            // identifier
  "Corollary",            // head
  base: "theorem",        // base - use the theorem counter
  fill: rgb("#f8e8e8")
)

#corollary(numbering: "1.1")[
```

```
  If $n$ divides two consecutive natural numbers, then $n = 1$.
]
```

Corollary 3.2.1: If n divides two consecutive natural numbers, then $n = 1$.

Note that we have provided a `numbering` string; this can be any valid numbering pattern as described in the numbering documentation.

3.1. Suppressing numbering

Supplying `numbering: none` to an environment suppresses numbering for that block, and prevents it from updating its counter.

```
#let example = thmplain(
  "example",
  "Example"
).with(numbering: none)

#example[
  The numbers $2$, $3$, and $17$ are prime.
]
```

Example: The numbers 2, 3, and 17 are prime.

Here, we have used the `thmplain` function, which is identical to `thmbox` but sets some plainer defaults. You can also write

```
#lemma(numbering: none)[
  The square of any even number is divisible by $4$.
]
#lemma[
  The square of any odd number is one more than a multiple of $4$.
]
```

Lemma: The square of any even number is divisible by 4.

Lemma 3.1.1: The square of any odd number is one more than a multiple of 4.

Note that the last *Lemma* is *not* numbered 3.1.2!

3.2. Limiting depth

You can limit the number of levels of the `base` numbering used as follows.

```
#let definition = thmbox(
  "definition",
  "Definition",
  base_level: 1,          // take only the first level from the base
  stroke: rgb("#68ff68") + 1pt
)

#definition("Prime numbers")[
```

```
  A natural number is called a _prime number_ if it is greater than $1$ and
  cannot be written as the product of two smaller natural numbers. <prime>
]
```

Definition 3.1 (Prime numbers): A natural number is called a *prime number* if it is greater than 1 and cannot be written as the product of two smaller natural numbers.

Note that this environment is *not* numbered 3.2.1!

```
#definition("Composite numbers")[
  A natural number is called a _composite number_ if it is greater than $1$
  and not prime.
]
```

Definition 3.2 (Composite numbers): A natural number is called a *composite number* if it is greater than 1 and not prime.

Setting a `base_level` higher than what `base` provides will introduce padded zeroes.

```
#example(base_level: 4, numbering: "1.1")[
  The numbers $4$, $6$, and $42$ are composite.
]
```

Example 3.2.0.0.1: The numbers 4, 6, and 42 are composite.

3.3. Custom formatting

The `thmbox` function lets you specify rules for formatting the `title`, the `name`, and the `body` individually. Here, the `title` refers to the `head` and `number` together.

```
#let proof = thmplain(
  "proof",
  "Proof",
  base: "theorem",
  titlefmt: smallcaps,
  bodyfmt: body => [
    #body #h(1fr) $square$    // float a QED symbol to the right
  ]
).with(numbering: none)

#lemma[
  All even natural numbers greater than 2 are composite.
]
#proof[
  Every even natural number $n$ can be written as the product of the natural
  numbers $2$ and $n\/2$. When $n > 2$, both of these are smaller than $2$
  itself.
]
```

Lemma 3.3.1: All even natural numbers greater than 2 are composite.

Proof: Every even natural number n can be written as the product of the natural numbers 2 and $n/2$. When $n > 2$, both of these are smaller than 2 itself. □

You can go even further and use the `thmenv` function directly. It accepts an `identifier`, a `base`, a `base_level`, and a `fmt` function.

```
#let notation = thmenv(
  "notation",                 // identifier
  "Notation",                 // supplement
  none,                       // base - do not attach, count globally
  none,                       // base_level - use the base as-is
  (name, number, body) => [   // fmt - format content using the environment
                              // name, number, and body
    #h(1.2em) *Notation (#number) #name*:
    #h(0.2em)
    #body
    #v(0.5em)
  ]
).with(numbering: "I")        // use Roman numerals

#notation[
  The variable $p$ is reserved for prime numbers.
]
```

Notation (I) : The variable p is reserved for prime numbers.

3.4. Labels and references

You can place a `<label>` outside a theorem environment, and reference it later via `@` references! For example, go back to Theorem 3.1.

```
Recall that there are infinitely many prime numbers via @euclid.
```

Recall that there are infinitely many prime numbers via Theorem 3.1.

```
You can reference future environments too, like @oddprime[Cor.].
```

You can reference future environments too, like Cor. 3.3.1.1.

Caution: Links created by references to `thmenvs` will be styled according to `#show link:` rules.

3.5. Overriding base

```
#let remark = thmplain("remark", "Remark", base: "heading")
#remark[
  There are infinitely many composite numbers.
]

#corollary[
  All primes greater than $2$ are odd.
] <oddprime>
#remark(base: "corollary")[
  Two is a _lone prime_.
]
```

Remark 3.5.1: There are infinitely many composite numbers.

Corollary 3.3.1.1: All primes greater than 2 are odd.

Remark 3.3.1.1.1: Two is a *lone prime*.

This `remark` environment, which would normally be attached to the current *heading*, now uses the `corollary` as a base.

4. Function reference

4.1. `thmenv`

The `thmenv` function produces a *theorem environment*.

```
#let thmenv(
  identifier,             // environment counter name
  supplement,             // supplement used in references
  base,                   // base counter name, can be "heading" or none
  base_level,             // number of base number levels to use
  fmt                     // formatting function of the form
                          // (name, number, body) -> content
) = { ... }
```

A *theorem environment* is itself a map of the following form.

```
(
  ..name,                 // name, often used in the title
  body,                   // body content
  numbering: "1.1",       // numbering style, can be a function
  refnumbering: auto,     // numbering style used in references,
                          // defaults to "numbering"
  base: base,             // base counter name override
  base_level: base_level  // base_level override
) -> content
```

4.2. `thmbox` and `thmplain`

The `thmbox` wraps `thmenv`, supplying a box-like `fmt` function.

```
#let thmbox(
  identifier,             // identifier
  head,                   // head - common name, used in the title
  supplement: auto,       // supplement for references, defaults to "head"
  fill: none,             // box fill
  stroke: none,           // box stroke
  inset: 1.2em,           // box inset
  radius: 0.3em,          // box corner radius
  breakable: false,       // box breakability
  padding: (top: 0.5em, bottom: 0.5em),
                          // box padding, passed to #pad
  namefmt: x => [(#x)],   // formatting for name
  titlefmt: strong,       // formatting for title (head + number)
  bodyfmt: x => x,        // formatting for body
```

```
  separator: [#h(0.1em):#h(0.2em)],
                          // separator inserted between name and body
  base: "heading",        // base - defaults to using headings
  base_level: none,       // base_level - defaults to using base as-is
) = { ... }
```

The `thmplain` function is identical to `thmbox`, except with plainer defaults.

```
#let thmplain = thmbox.with(
  padding: (top: 0em, bottom: 0em),
  breakable: true,
  inset: (top: 0em, left: 1.2em, right: 1.2em),
  namefmt: name => emph([(#name)]),
  titlefmt: emph,
)
```

5. Acknowledgements

Thanks to MJHutchinson for suggesting and implementing the `base_level` and `base: none` features, rmolinari for suggesting and implementing the `separator: ...` feature, and to the awesome devs of typst.app for their support.