reST and Sphinx

documenting formal abstracts

Jesse Michael Han[1]

University of Pittsburgh

June 6, 2018

[1]jmh288@pitt.edu

Outline

- reST

- Sphinx

- Lean and Sphinx

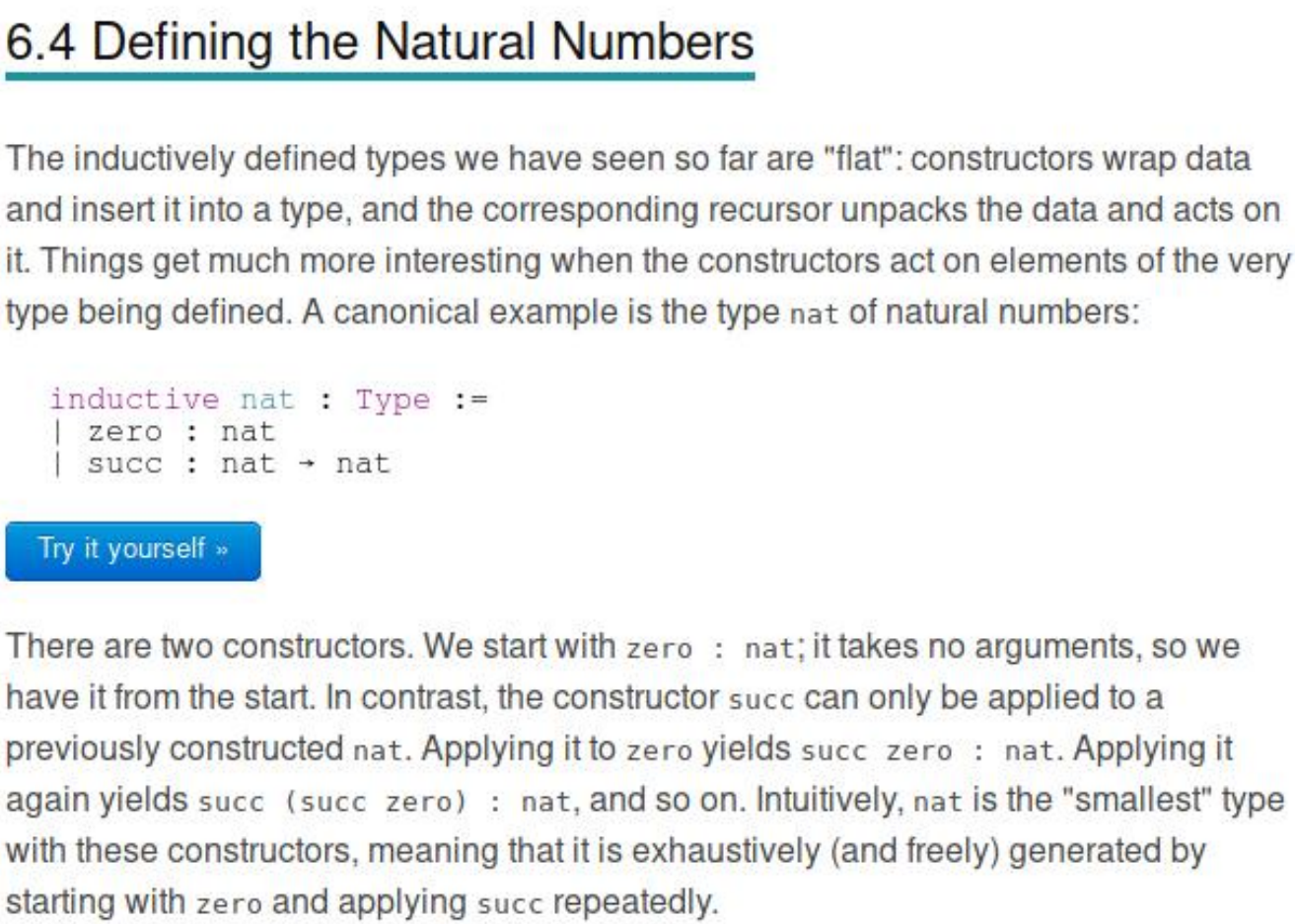
What we will learn:

What we will learn:

- How to produce documentation like *Theorem Proving In Lean*:

What we will learn:

- How to produce documentation like *Theorem Proving In Lean*:



6.4 Defining the Natural Numbers

The inductively defined types we have seen so far are "flat": constructors wrap data and insert it into a type, and the corresponding recursor unpacks the data and acts on it. Things get much more interesting when the constructors act on elements of the very type being defined. A canonical example is the type `nat` of natural numbers:

```
inductive nat : Type :=
| zero : nat
| succ : nat → nat
```

Try it yourself »

There are two constructors. We start with `zero : nat`; it takes no arguments, so we have it from the start. In contrast, the constructor `succ` can only be applied to a previously constructed `nat`. Applying it to `zero` yields `succ zero : nat`. Applying it again yields `succ (succ zero) : nat`, and so on. Intuitively, `nat` is the "smallest" type with these constructors, meaning that it is exhaustively (and freely) generated by starting with `zero` and applying `succ` repeatedly.

What we will learn:

What we will learn:

- A document like this is compiled into HTML by Sphinx from source files written in reST.

What we will learn:

- A document like this is compiled into HTML by Sphinx from source files written in reST.
- Today, we'll learn the basics of marking up text with reST, compiling with Sphinx, and documenting Lean code with Sphinx.

What is reST?

What is reST?

- reST stands for `reStructuredText`.

What is reST?

- reST stands for `reStructuredText`.
- `reStructuredText` is an easy-to-read, what-you-see-is-what-you-get plaintext markup syntax and parser system.

What is reST?

- reST stands for `reStructuredText`.
- `reStructuredText` is an easy-to-read, what-you-see-is-what-you-get plaintext markup syntax and parser system.
- Think: Markdown, but fancier and extensible.

What is reST?

- reST stands for `reStructuredText`.
- `reStructuredText` is an easy-to-read, what-you-see-is-what-you-get plaintext markup syntax and parser system.
- Think: Markdown, but fancier and extensible.
- Let's look at an example to see how reST's markup works.

reST syntax by example

reST syntax by example

Consider the following compiled document, adapted from the official documentation:

reST syntax by example

Consider the following compiled document, adapted from the official documentation:

Inline markup

The standard reST inline markup is quite simple: use

- one asterisk: `*text*` for *emphasis* (italics),
- two asterisks: `**text**` for **strong emphasis** (boldface), and
- backquotes: ``` ``text`` ``` for code samples.

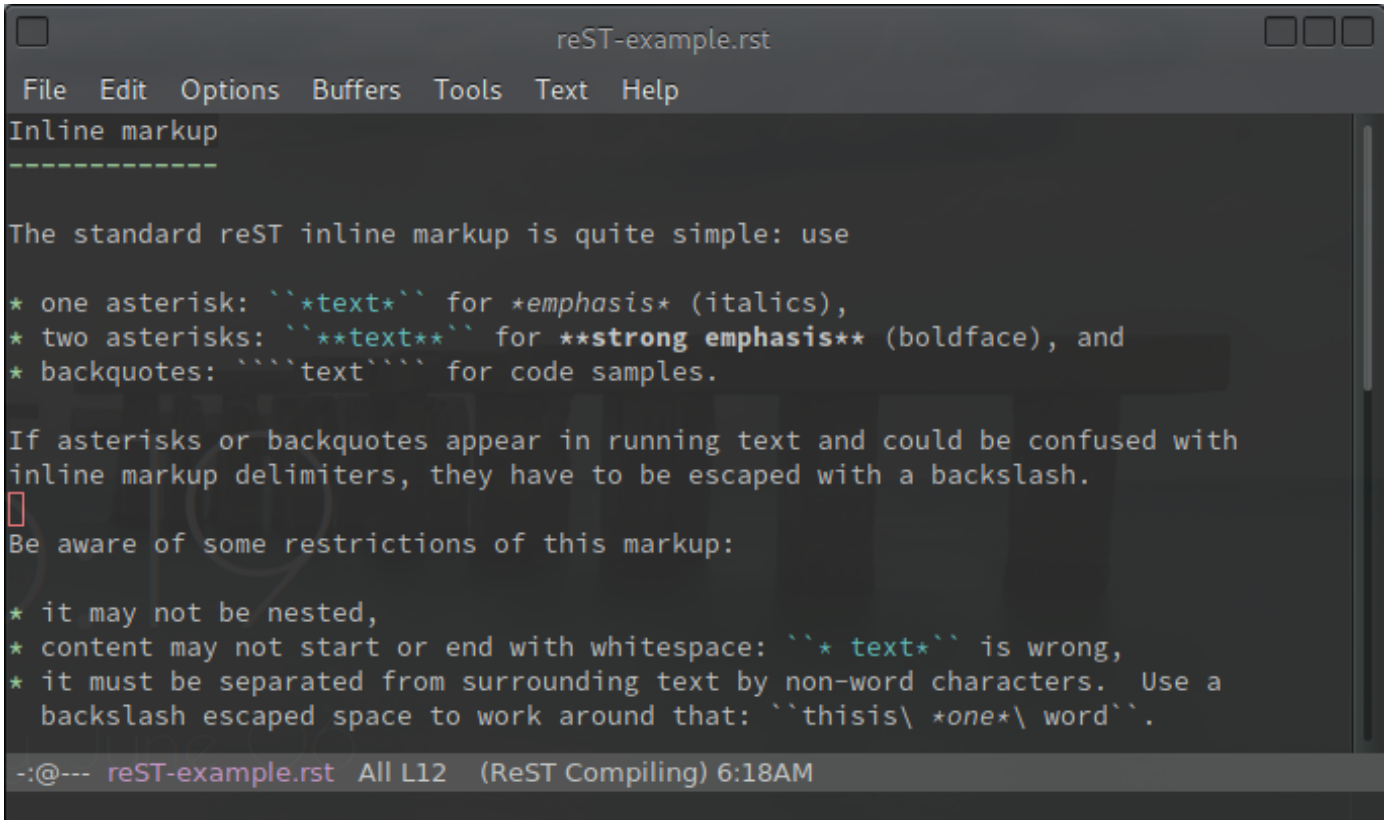
If asterisks or backquotes appear in running text and could be confused with inline markup delimiters, they have to be escaped with a backslash.

Be aware of some restrictions of this markup:

- it may not be nested,
- content may not start or end with whitespace: `* text*` is wrong,
- it must be separated from surrounding text by non-word characters. Use a backslash escaped space to work around that: `thisis\ *one*\ word`.

reST syntax by example: source code

reST syntax by example: source code



reST syntax by example: summing up

Inline markup

reST syntax by example: summing up

Inline markup

- To *italicize*, wrap with single asterisks.

reST syntax by example: summing up

Inline markup

- To *italicize*, wrap with single asterisks.

```
To *italicize*, wrap with single asterisks.
```

reST syntax by example: summing up

Inline markup

- To *italicize*, wrap with single asterisks.

```
To *italicize*, wrap with single asterisks.
```

- To **bold**, wrap with *double* asterisks.

reST syntax by example: summing up

Inline markup

- To *italicize*, wrap with single asterisks.

```
To *italicize*, wrap with single asterisks.
```

- To **bold**, wrap with *double* asterisks.

```
To **bold**, wrap with *double* asterisks.
```

reST syntax by example: summing up

Inline markup

- To *italicize*, wrap with single asterisks.

```
To *italicize*, wrap with single asterisks.
```

- To **bold**, wrap with *double* asterisks.

```
To **bold**, wrap with *double* asterisks.
```

- To delimit *in-line code*, wrap with ´´ double backticks ´´.

reST syntax by example: summing up

Inline markup

- To *italicize*, wrap with single asterisks.

```
To *italicize*, wrap with single asterisks.
```

- To **bold**, wrap with *double* asterisks.

```
To **bold**, wrap with *double* asterisks.
```

- To delimit *in-line code*, wrap with ´´ double backticks ´´.

```
To delimit *in-line code*, wrap with ´´ double
backticks ´´.
```

reST syntax by example: summing up

Whitespace

- Whitespace matters!
- Blocks of text are compiled to paragraphs depending on separating newlines.

reST syntax by example: summing up

Whitespace

- Whitespace matters!
- Blocks of text are compiled to paragraphs depending on separating newlines.

Lists

reST syntax by example: summing up

Whitespace

- Whitespace matters!
- Blocks of text are compiled to paragraphs depending on separating newlines.

Lists

From the documentation:

reST syntax by example: summing up

Whitespace

- Whitespace matters!
- Blocks of text are compiled to paragraphs depending on separating newlines.

Lists

From the documentation:

> Lists must always start a new paragraph -- that is, they must appear after a blank line.
>
> **enumerated** lists (numbers, letters or roman numerals; quickref)
> Start a line off with a number or letter followed by a period ".", right bracket ")" or surrounded by brackets "( )" -- whatever you're comfortable with. All of the following forms are recognised:

```
1. numbers

A. upper-case letters
   and it goes over many lines

   with two paragraphs and all!

a. lower-case letters

   3. with a sub-list starting at a different number
   4. make sure the numbers are in the correct sequence though!
```

reST syntax by example: summing up

Nested lists

reST syntax by example: summing up

Nested lists

From the documentation:

reST syntax by example: summing up

Nested lists

From the documentation:

bulleted lists (quickref)

Just like enumerated lists, start the line off with a bullet point character - either "-", "+" or "*":

```
* a bullet point using "*"

  - a sub-list using "-"

    + yet another sub-list

  - another item
```

Results in:

- a bullet point using "*"
  - a sub-list using "-"
    - yet another sub-list
  - another item

reST syntax by example: summing up

Section and section titles

reST syntax by example: summing up

Section and section titles

- Sections are identified by their titles, which are marked up with *adornment*: single repeated punctutation symbols under (and optionally over) the title string.

reST syntax by example: summing up

Section and section titles

- Sections are identified by their titles, which are marked up with *adornment*: single repeated punctutation symbols under (and optionally over) the title string.
- Unlike vanilla Markdown, the *number* of punctuation symbols doesn't matter.

reST syntax by example: summing up

Section and section titles

- Sections are identified by their titles, which are marked up with *adornment*: single repeated punctutation symbols under (and optionally over) the title string.
- Unlike vanilla Markdown, the *number* of punctuation symbols doesn't matter.

Rather, the following convention is used:

reST syntax by example: summing up

Section and section titles

- Sections are identified by their titles, which are marked up with *adornment*: single repeated punctutation symbols under (and optionally over) the title string.
- Unlike vanilla Markdown, the *number* of punctuation symbols doesn't matter.

Rather, the following convention is used:

all sections marked with the same adornment style are deemed to be at the same level:

```
Chapter 1 Title
===============

Section 1.1 Title
-----------------

Subsection 1.1.1 Title
~~~~~~~~~~~~~~~~~~~~~~

Section 1.2 Title
-----------------

Chapter 2 Title
===============
```

reST syntax by example, 2

reST syntax by example, 2

Consider the following compiled document, adapted from text by the Grothendieck Circle:

reST syntax by example, 2

Consider the following compiled document, adapted from text by the Grothendieck Circle:

Grothendieck's lecture program [1] was prepared in agreement with his Vietnamese colleages. Ta Quang Buu, himself a mathematician and also the minister for higher and technological education, attended these lectures. The program, **occasionally interrupted by airstrikes**, consisted of the following events at the University of Hanoi:

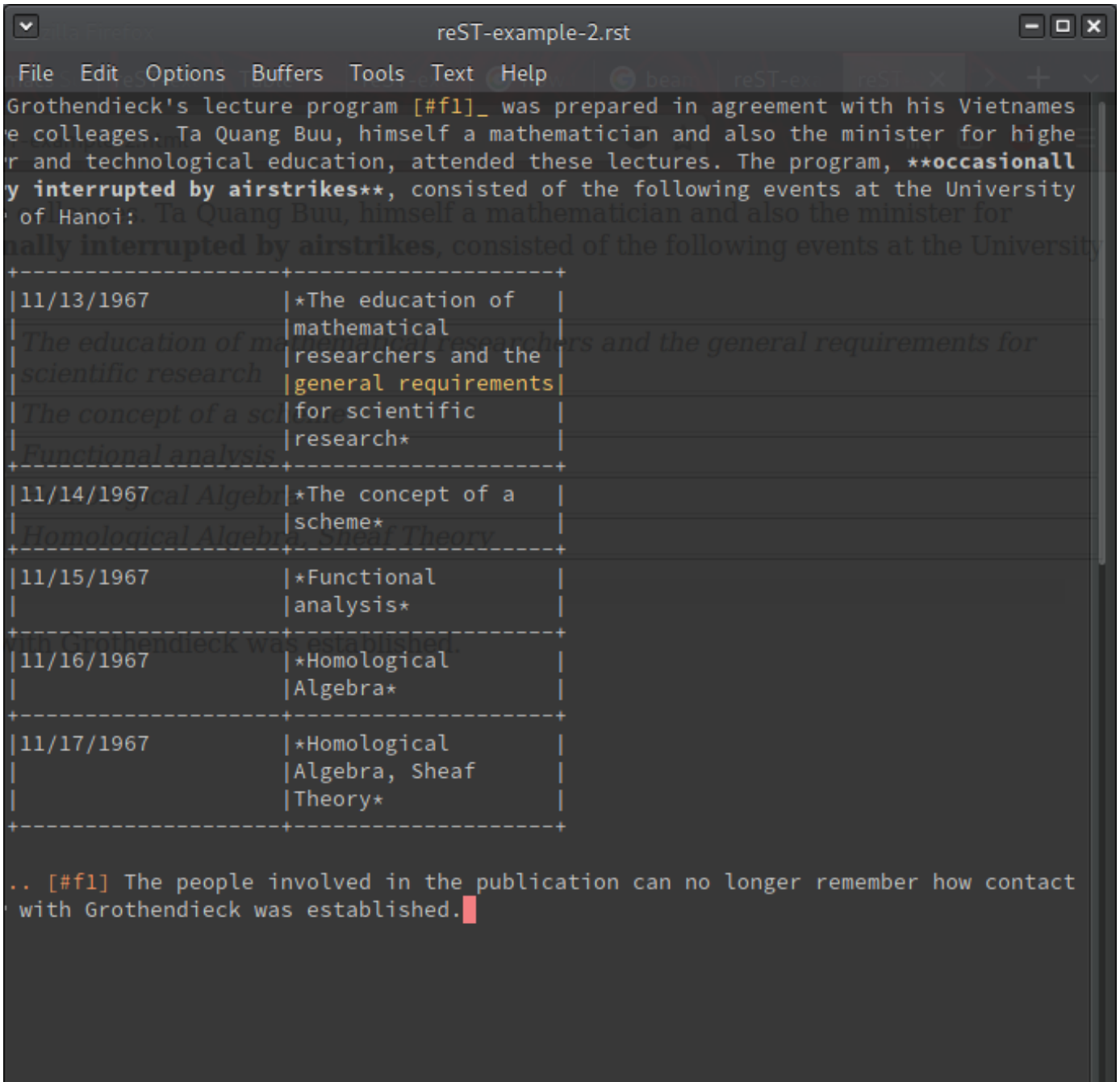
11/13/1967	*The education of mathematical researchers and the general requirements for scientific research*
11/14/1967	*The concept of a scheme*
11/15/1967	*Functional analysis*
11/16/1967	*Homological Algebra*
11/17/1967	*Homological Algebra, Sheaf Theory*

Footnotes

[1] The people involved in the publication can no longer remember how contact with Grothendieck was established.

reST syntax by example, 2: source code

reST syntax by example, 2: source code



reST syntax by example, 2: summing up

reST syntax by example, 2: summing up

Tables

reST syntax by example, 2: summing up

Tables

- One can create ASCII tables in reST documents that will be compiled to an HTML table.

reST syntax by example, 2: summing up

Tables

- One can create ASCII tables in reST documents that will be compiled to an HTML table.
- Your editor should have a macro for inserting them (e.g. Emacs' `M-x table-insert`)

reST syntax by example, 2: summing up

Tables

- One can create ASCII tables in reST documents that will be compiled to an HTML table.
- Your editor should have a macro for inserting them (e.g. Emacs' `M-x table-insert`)

Directives

reST syntax by example, 2: summing up

Tables

- One can create ASCII tables in reST documents that will be compiled to an HTML table.
- Your editor should have a macro for inserting them (e.g. Emacs' `M-x table-insert`)

Directives

- The footnote syntax in the previous source code is an example of a **directive**, which are blocks of explicit (as opposed to inline) markup in a reST document.

reST syntax by example, 2: summing up

Tables

- One can create ASCII tables in reST documents that will be compiled to an HTML table.
- Your editor should have a macro for inserting them (e.g. Emacs' `M-x table-insert`)

Directives

- The footnote syntax in the previous source code is an example of a **directive**, which are blocks of explicit (as opposed to inline) markup in a reST document.
- Footnotes, hyperlinks, and citations share similar syntax and share the same namespace for identifiers.

Labels and references

Note: the `:ref:` role is not included in vanilla reST, but is part of Sphinx.

Labels and references

Note: the `:ref:` role is not included in vanilla reST, but is part of Sphinx.

- Labels for **sections** are *created* by inserting the directive

```
.. _label-id:
```

into the document directly before the section. The double periods must begin on a new line.

Labels and references

Note: the `:ref:` role is not included in vanilla reST, but is part of Sphinx.

- Labels for **sections** are *created* by inserting the directive

```
.. _label-id:
```

into the document directly before the section. The double periods must begin on a new line.

- Labels for sections can be *referenced* by inserting `:ref:`label-id`` later on in the document.

Labels and references

Note: the `:ref:` role is not included in vanilla reST, but is part of Sphinx.

- Labels for **sections** are *created* by inserting the directive

```
.. _label-id:
```

into the document directly before the section. The double periods must begin on a new line.

- Labels for sections can be *referenced* by inserting `:ref:`label-id`` later on in the document.
- Note that putting an underscore *before* an identifier wrapped in backticks indicates that it's the target of a reference.

Labels and references

Note: the `:ref:` role is not included in vanilla reST, but is part of Sphinx.

- Labels for **sections** are *created* by inserting the directive

```
.. _label-id:
```

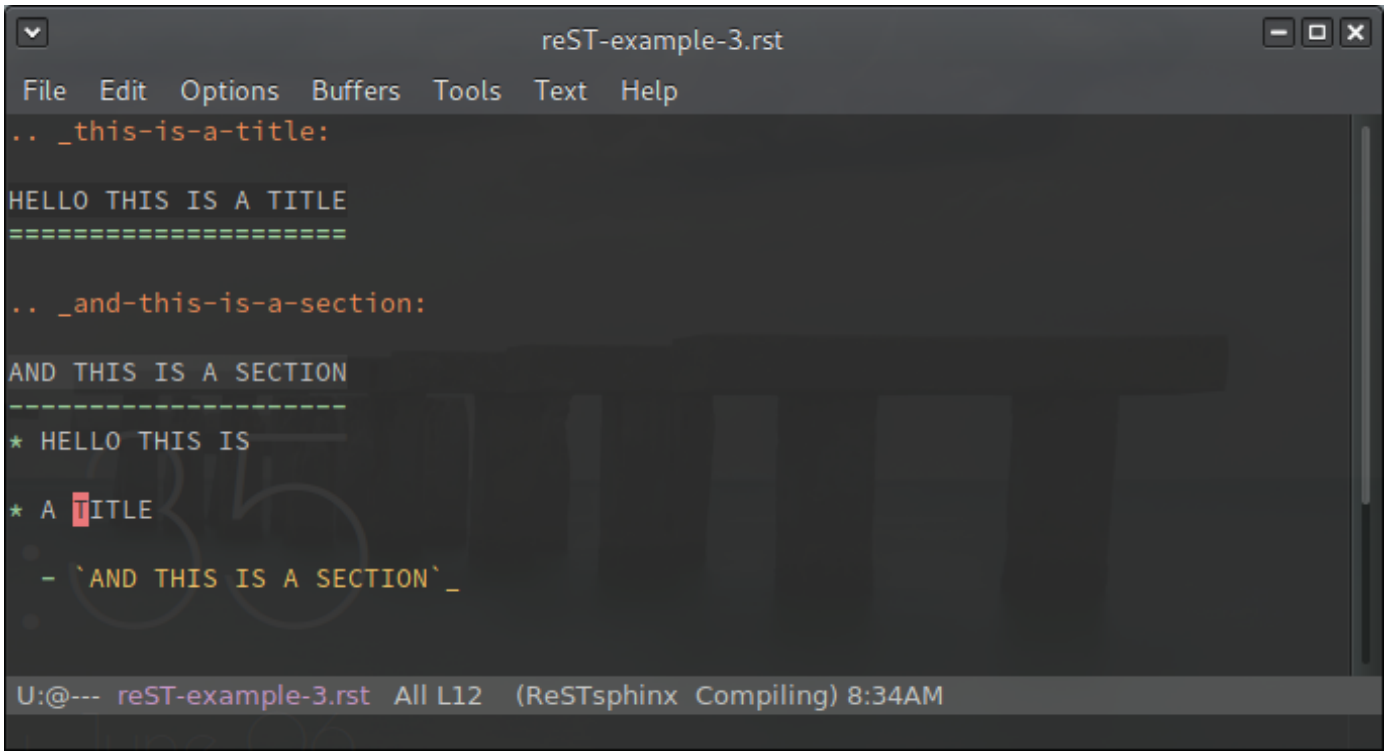
into the document directly before the section. The double periods must begin on a new line.

- Labels for sections can be *referenced* by inserting `:ref:`label-id`` later on in the document.
- Note that putting an underscore *before* an identifier wrapped in backticks indicates that it's the target of a reference.
- Putting an underscore *after* an identifier wrapped in backticks indicates that it points *to* some target. This is reflected in the footnote syntax.

Labels and references.

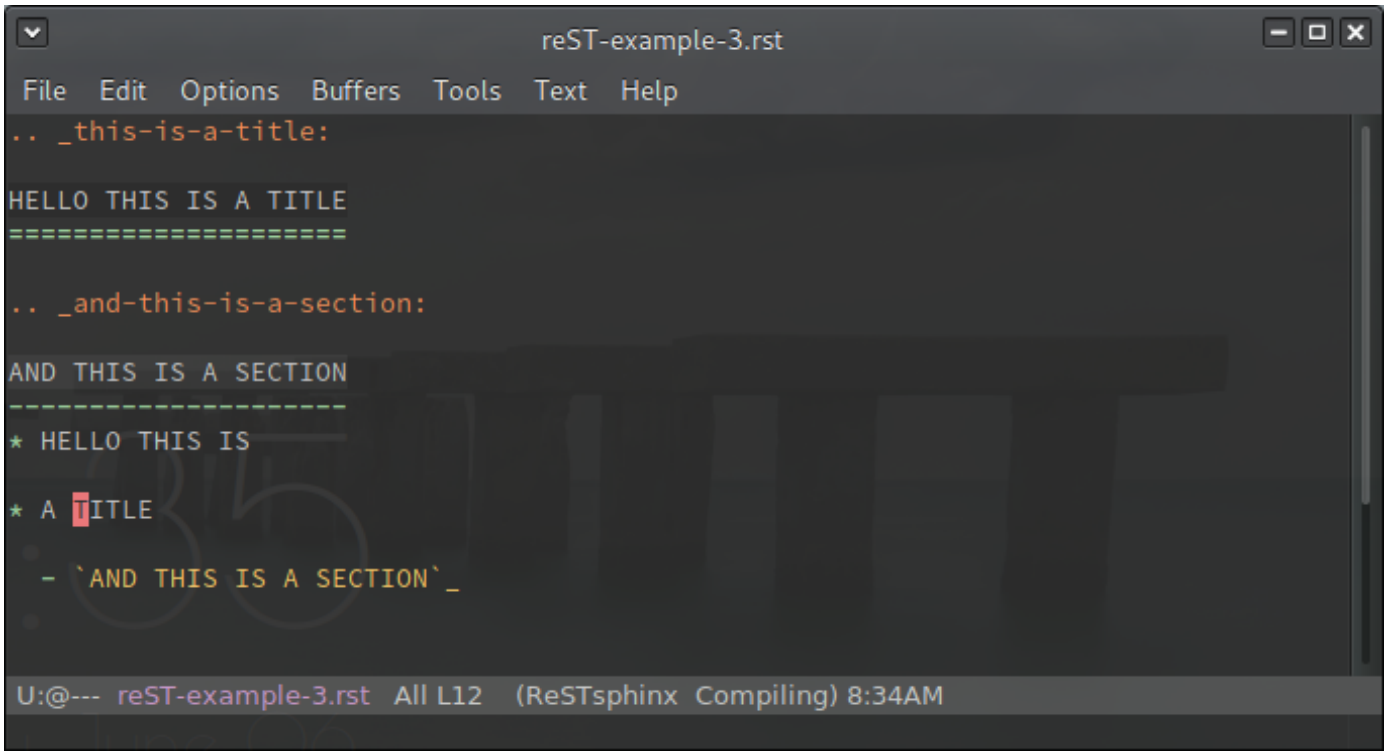
Labels and references.

- For example, the source code



Labels and references.

- For example, the source code



will compile to the following. . .

Labels and references

Labels and references

Labels and references

Labels and references

- You can *name* your references, and they don't need to refer to labels inside your document. You can point to a URL instead.

Labels and references

- You can *name* your references, and they don't need to refer to labels inside your document. You can point to a URL instead.

For example:

Labels and references

- You can *name* your references, and they don't need to refer to labels inside your document. You can point to a URL instead.

For example:

```
‘Lean prover <https://leanprover.github.io>‘_
```

What is Sphinx?

What is Sphinx?

- Sphinx is a **documentation generator** which converts reST source files to HTML and other formats.

What is Sphinx?

- Sphinx is a **documentation generator** which converts reST source files to HTML and other formats.
- Sphinx additionally extends reST with custom directives and roles.

What is Sphinx?

- Sphinx is a **documentation generator** which converts reST source files to HTML and other formats.
- Sphinx additionally extends reST with custom directives and roles.
- For example, the `:ref:` role covered in the previous section.

Installing Sphinx

Linux

Debian/Ubuntu

Linux

Debian/Ubuntu

Open a terminal and input:

```
sudo apt-get install python-sphinx
```

Linux

Debian/Ubuntu

Open a terminal and input:

```
sudo apt-get install python-sphinx
```

Arch

Linux

Debian/Ubuntu

Open a terminal and input:

```
sudo apt-get install python-sphinx
```

Arch

Open a terminal and input:

```
sudo pacman -S python-sphinx
```

Mac OSX

Either of these two will do:

Mac OSX

Either of these two will do:

- Using `homebrew`:

```
brew install sphinx-doc
```

Mac OSX

Either of these two will do:

- Using `homebrew`:

```
brew install sphinx-doc
```

- Using `port`:

```
sudo port install py27-sphinx
```

Windows

- Install Python 2.7+ from `https://www.python.org`.

Windows

- Install Python 2.7+ from `https://www.python.org`.
- Install pip from `https://bootstrap.pypa.io/get-pip.py`

Windows

- Install Python 2.7+ from `https://www.python.org`.
- Install pip from `https://bootstrap.pypa.io/get-pip.py`
- Now open a command prompt (`Ctrl + ´`), navigate to the directory containing `get-pip.py` and input:

Windows

- Install Python 2.7+ from `https://www.python.org`.
- Install pip from `https://bootstrap.pypa.io/get-pip.py`
- Now open a command prompt (`Ctrl + ´`), navigate to the directory containing `get-pip.py` and input:

```
python get-pip.py
pip install sphinx
```

How do we document Lean with Sphinx?

How do we document Lean with Sphinx?

- Jeremy Avigad has a tutorial about this (somewhere. . .)

How do we document Lean with Sphinx?

- Jeremy Avigad has a tutorial about this (somewhere. . .)
- But, for now let's just work in vanilla Sphinx. Clone a copy of:

`https://www.github.com/thalesant/formalabstracts`

How do we document Lean with Sphinx?

- Jeremy Avigad has a tutorial about this (somewhere. . .)
- But, for now let's just work in vanilla Sphinx. Clone a copy of:

`https://www.github.com/thalesant/formalabstracts`

- Now I'm going to walk us through adding our first contribution to the `formalabstracts` documentation.