The knowledge package

Thomas Colcombet
thomas.colcombet@irif.fr

June 9, 2017

Abstract

The knowledge package offers automatic tools and commands for helping the writer of a (scientific) document to have all notions (hyper)linked to places where these are defined. Using the standard tools of LaTeX for doing that would be extremely tedious. At term, it is intended to use the package for producing scientific semantic aware documents.

Status of this version (June 9, 2017)

This is still an alpha version. A lot of functionalities are perfectly operational, and the syntax is close to stable. At any rate, if changes happen, these will only affect a few lines in the configuration of the document. Nevertheless, it is advised to put the package directly in the document's directory, rather than in the search tree of LaTeX.

For all questions, comments, bugs or requests of improvements, contact:

mailto:thomas.colcombet@irif.fr

Contents

1 Todo list

This is essentially a list for personal referencing of what has to be done.

- deal with repeated introductions in restatable
- remove the warnings about redefining commands
- give detail on where appear the multiple introduction of the same knowledge
- proper mathematics handling. Write a new `\NewDocumentCommand`-like macro
- Reorder the way directives are executed (currently a link directive is overriden by its target)
- proper description of 'notion' in the doc
- a uniform notation for refining display targetting: for instance `\kl {game@play}` and `\kl {music@play}` would point to two distinct knowledges for what is a 'play'. The text would be 'play'. Should it be linked with the notion of 'scope'. Probably yes. Before the 'at', this describes a scope for searching the knowledge.
- detect using some trick that the previous compilation failed, and deactivate the Kaux file in this case (for the moment, removing the aux file deactivates the kaux file. This is convenient but not sufficient.)
- proper code for `\nointro`.
- relax the definition of scope. Make this rigidity optional (should it be the case).
- Solve the problem of `\kl` (and similar commands) being unknown when creating the pdf index.
- improve the diagnose file

1. more information on the origin of errors (file/line)
2. help possible in it
3. short or long version of diagnose upon request

- improving the `\AP` and `\itemAP` commands. For the moment `\AP` silently fails inside the optional argument of `\item`, and probably elsewhere too. See how this can be improved.
- solve the `amsmath` problem that `\label` is deactivated in unnumbered environment, while knowledge would nevertheless like to point to the place...
- solving the problem of optional parameters inside optional parameters (e.f. `\kl` inside `\item`) This can possibly be done using other notations like `\kl`<key>{*text*}
- improve of scope, in particular, change the way `\knowledgeimport` is used.
- Properly code export and import features.
- Pointing inside an external pdf document.
- Properly treating the version of the expl package. In particular concerning the hack concerning `\c_job_name_tl`.
- Properly treat the issue of using quotes in combination with other packages, such as `xypic`.
- Would it be possible to link to labels as to knowledges. This would give something like `see the "main theorem@theorem:main"` .

2 Changes

2016-06-07 `\knowledgemacro` is now renamed to `\knowledgedirective`.

2017-01-13 `\AP` has been recoded, and is now more properly aligned in the margin. The `visible anchor points` option has also been made usable without the `xcolor` package.

2017-01-13 The package `scope option` can now be omitted. This in particular avoid clashes with the over-restriction on the structure of the document it entails. It should be improved to stop overloading the `\begin` command.

2017-01-14 The overloading of `\begin` and `\end` was done as protected commands, which should not be the case to be consistent with the behaviour of LaTeX (for instance, this was giving an extra line in the title in the conference mode of the class IEEEtran). Corrected: these commands are not protected anymore.

2017-01-15 A workaround for an incompatibility between the `hyperref package` and the two-column mode as been added in the macro `\knowledgeFixHyperrefTwocolumn` (thanks to Daniela Petrisan).

2017-01-15 Added the directive `synonym`.

2017-01-15 Added the noknowledge package for minimizing the effects of not having knowledge activated.

2017-01-17 Changed the way options are handled, decoupling the package options (options of `\usepackage`) from the configuration options (see `\knowledgeconfigure`).

2017-01-17 Proper treatment of 'final' option and `option` options.

2017-01-17 Added \IfKnowledgeFinalMode`[TF]` commands for the user.

2017-01-17 Added the option `fix hyperref twocolumn` as a shorthand for calling `\knowledgeFixHyperrefTwocolumn` (thanks to Daniela Petrisan and Luca Reggio).

2017-01-18 Added the configuration option `notion` that offers a basic configuration compatible with `xcolor` or not, and final and composition modes.

2017-01-19 Added `\phantomintro` and an explanation on how to deal with `align*`.

2017-02-20 Removed the warnings of latex for unknown labels in `autoref`.

2017-02-20 Removed nasty error making `\AP` not operative when anchor points were not visible.

2017-02-21 Added the `protect link` directive.

2017-02-21 Added the `hyperlinks=` configuration.

2017-02-27 `visible anchor points` is active by default now.

2017-02-27 A simple example is now included.

2017-02-28 Minor changes on the documentation.

2017-02-28 Added the `scope` environment.

2017-02-28 Added the `protect link` and `unprotect link` configuration directives.

2017-02-28 Added the `\knowledgescopingenvironment` command.

2017-03-03 Added the `breaklinks` faq (thanks to Luca Reggio for the request).

2017-03-10 Added the `"...` and `""..""` notations and the `quotation` mode (requested by Gabriele Puppis and Andreas Krebs).

2017-03-11 Added the `"...@..."` and `""...@...""` notations.

2017-03-13 Corrected for being compatible with version of expl3 posterior to Mars 2015 (`\c_job_name_tl` does not exist anymore). (Thanks to Jean-Éric Pin).

2017-03-14 Corrected that the `@` letter was left a letter after `\knowledgeFixHyperrefTwocolumn`.

2017-04-09 Internal change of code, for scope handling and for the quotation notation: slowly going toward an extended quotation notation that can make the scope of search explicit.

2017-04-09 Added the `protect quotation` configure option, that is given a list of environments, and deactivates automatically the quotation notation when in there environments. This is a simple code for the moment. Typically, one can use `\knowledgeconfigure{protect quotation=tikzcd}`. For the moment, it is not explained in the document.

2017-04-19 Changed the display code such hat nested knowledges behave properly: before, the introduction would be performed for the object and the subobjects.

2017-04-20 The electronic mode has been added, and the 'final mode' is now renamed into paper mode. The `\knowledgepackagemode` configuration variable is also available for easier scripting.

2017-06-06 FAQ on deactivating the quote in `Emacs`.

2017-06-08 Removed the noknowledge package and all references to it.

2017-06-08 Removed the `knowledgeutils.sty` and `scopearticle.sty` which are now integrated in the main file.

2017-06-08 The file `knowledge-example.sty` as been improved.

3 Quick start

The knowledge package offers several capabilities for handling colors, changing the display style, defining internal and external hyperlinks, producing an index, etc... All these possibilities arise from defining explicitly or implicitly knowledges associated to terms in plain english (or other languages).

We start by describing a certain number of problems/scenarii that a user is confronted to, and show how to solve them. The hyperlinks in this document have been generated using the knowledge package. In the subsequent questions, a more detailed account of how the package works and can be parameterized is given.

There is also a file `knowledge-example.tex` that can be used as a starting point.

3.1 Linking to outer documents/urls, and to labels

The problem 1 *I have a lot of external url's that I would like to [[very] often] have a link to, but I do not want to always type the full url. I do not want to remember weird labels/internal references/macro names either.*

A solution is as follows. One first loads the knowledge package with option `hyperref` using either:

```
\usepackage[hyperref,quotation]{knowledge}
```

or:

```
\usepackage[hidelinks[a]]{hyperref}
\usepackage[quotation[b]]{knowledge}
```

[a] The option `hidelinks` is advised, but not necessary.
[b] If you want to use the `"···"` notation.

Hint. You may use other options like `xcolor` for allowing debugging with colors (for undefined knowledges).

Then, in the preamble (or in an external file), one uses commands of the form:

```
\knowledge{latex}{url={https://en.wikipedia.org/wiki/LaTeX}}
```

This configures the text '`latex`' to be associated with the sole directive `url=`, which means an hyperreference to this address.

Finally in the body of the paper, the sole extra command `\kl` (or the `"`-symbol if the `quotation` option is activated) is used, with as parameter a text. This text is searched for, and the directives attached to it (here the url), are used for formatting its printing[1]. Hence:

```
This package has been written for use in \kl{latex}.
```

or, if the `quotation` option is activated,

```
This package has been written for use in "latex".
```

Hint. If the knowledge is not defined, this does not make the compilation fail. In fact, it is good practice to use many `\kl` commands while developing a text, and only resolve these questions at the end (see also the diagnose file).

[1] This resembles a lot a macro so far. It nevertheless differs in that: (a) if not defined, it does not make the compilation fail as a macro would, and thus does not interfere with the writing process, (b) any text can be used and not only alphabetic letters as in TEX, (c) you do not have to care about the space after, and (c) in fact the machinery for resolving the meaning of a knowledge is much more powerful than simple macro execution.

yields

This package has been written for use in latex.

Variation. But in fact, I would like 'latex' to also be properly typeset LaTeX, and in gray. This requires to load the package with the xcolor option (for being able to use colors, obviously), or by loading the package xcolor before, and then modify the \knowledge command using extra directives:

```
\knowledge{latex}{url=https://en.wikipedia.org/wiki/LaTeX,
text=\LaTeX, color=gray}
```

yields with the same code

This package has been written for use in LaTeX.

The directives text= and color= have quite obvious meaning. Directives can also control the style using emphasize, boldface, italic, typewriter and so on. See Section 6.3 for a complete list.

Variation. (synonyms) It happens very often that there are several ways to name a notion, because of capitalized letters, conjugacy, grammar. I would like this to be easily handled. By using a '|' separated list of texts in the optional parameter of \knowledge command, it is possible to add a list of 'synonyms', such as in:

```
\knowledge{latex}[LaTeX|Latex|LATEX]
{url=http://en.wikipedia.org/wiki/LaTeX, text=\LaTeX, color=gray}
```

This is interesting for people's name that can be displayed in various ways depending on the context. Hence

```
\knowledge{Donald Ervin Knuth}[Donald Knuth|Knuth]
{url=https://fr.wikipedia.org/wiki/Donald_Knuth}
```

would allow

\kl{Knuth} as well as \kl{Donald Knuth},
or simply "Knuth" as well as "Donald Knuth" and so on

to all point to the same web address. It is even more convenient to use it for nouns that are sometimes in plural form or at the beginning of a sentence. Hence:

```
\knowledge{group}[groups|Groups|group morphism|group
morphisms|Group morphisms]
{url=https://en.wikipedia.org/wiki/Group_(mathematics)}
```

makes it possible to use the notions in many contexts:

```
"Groups" form a category when equipped with "group morphisms".
```

Hint. This is a shorthand for a synonym (or link=) directive.
For instance, with the code

```
\knowledge{D. Knuth}
{link=Knuth}
```

then \kl{D. Knuth} would also point to the same url.

```
\knowledge{D. Knuth}
{synonym}
```

would also work if used just after the definition of Knuth.

Hint. Sometimes one wants to use a knowledge by explicitly mentioning it:

```
\kl[group]{This object}
is very important.
```

3.2 Linking inside a document

The problem 2 *I am writing a long scientific document with many notions tied together (typically, I have made all my best for clarifying but nevertheless it remains obscure, or it is a long survey involving many subfields, or a book, or a PhD thesis[2]). I would like all the notions be linked inside the document for being able in one click, whenever something is used, to jump to its definition. However, I do not want it to be a hassle when writing.*

A solution is as follows. First load the knowledge package in the preamble:

```
\usepackage[xcolor,hyperref,notion,quotation]{knowledge}
```

with suitable options: `hyperref` for links, and `xcolor` for colors (if required), and `notion` for automatic configuration.

Then write the document using `\intro` (or `""···""` if `quotation` is activated) when a notion is defined/introduced, and `\kl` (or `"···"` if `quotation` is activated)when it is used. For instance:

```
\AP A \intro{semigroup} is an ordered pair $(S,\cdot)$ where
$\cdot$ is an associative binary operator over $S$.
[...]
\AP A \intro{monoid} $(M,\cdot,1)$ is a \kl{semigroup}
$(M,\cdot)$ together with a neutral element $1$.
```

or

```
\AP A ""semigroup"" is an ordered pair $(S,\cdot)$ where $\cdot$
is an associative binary operator over $S$.
[...]
\AP A ""monoid"" $(M,\cdot,1)$ is a "semigroup" $(M,\cdot)$
together with a neutral element $1$.
```

This yields

> A semigroup is an ordered pair $(S,\cdot)$ where $\cdot$ is an associative binary operator over S.
> *[...]*
> A monoid $(M,\cdot,1)$ is a semigroup $(M,\cdot)$ together with a neutral element 1.

Hint. Using an `\AP` command is strongly advised, and allows to control more precisely where the target of hyperreferences is: at the beginning of a paragraph is better than the beginning of the section several pages before...

The `\AP` command is made visible thanks to a red corner.

Unknown knowledges are in brown (it is an important feature that the compilation does not fail: unknown knowledges should not interfere with the writing of the document, which is the purpose of all this). One can now see the list of such problems in the file 'filename.diagnose'. One can in particular find in the 'Undefined knowledges' section:

```
\knowledge{semigroup}{}
\knowledge{monoid}{}
```

[2] Reviewers should appreciate...

Which means that both 'monoid' and 'semigroup' are unknown knowledges.

To solve this, let us copy these two (or more) lines in the paper[3], adding the `notion` directive (which is a configured version of the `autoref` directive, meaning essentially that you want to use the features of the `\intro` command), i.e., in the preamble:

```
\knowledge{semigroup}{notion}
\knowledge{monoid}{notion}
```

The result is then (after two compilations):

> A *semigroup* is an ordered pair $(S, \cdot)$ where $\cdot$ is an associative binary operator over S. /.../ A *monoid* $(M, \cdot, 1)$ is a semigroup $(M, \cdot)$ together with a neutral element 1.

Clicking on 'semigroup' now jumps to the place it was introduced, and very precisely at the location of the red corned depicting the presence of the `\AP-`-command. If now one adds the option `paper` while loading the package, then the colors and the red corner disappear:

> A *semigroup* is an ordered pair $(S, \cdot)$ where $\cdot$ is an associative binary operator over S. /.../ A *monoid* $(M, \cdot, 1)$ is a semigroup $(M, \cdot)$ together with a neutral element 1.

It is very often the case that for plain english (or other languages) some terms can be used in several forms; verbs can be conjugated; nouns can be plural, and so on. So usually the lines added to the file look more like:

Hint. The directive `synonym` can be also convenient.

```
\knowledge{semigroup}[semigroups|Semigroups]{notion}
\knowledge{monoid}[monoid|Monoids]{notion}
```

Now, using code like

```
"Monoids" and "semigroups" play the same role from now.
```

will properly be linked to the definition of a semigroup and a monoid.

3.3 Mathematics

The examples above show various techniques for using knowledges for enhancing the information associated to terms. In fact, these techniques are not incompatible with mathematics. Imagine, for instance that you would like each time a macro `\monoid` is met, to display $\mathcal{M}$, you would do for instance: .

Hint. Defining new macros is best done using `\newrobustcmd` (of the `etoolbox` package), rather than the usual `\newcommand` of LaTeX. This remark is general in LaTeX, unless you have very specific reason to have your macro expandable. This is even more true when using knowledge

```
\newrobustcmd\monoid{\mathcal M}
```

Imagine that furthermore, you would like to point hyperlink to the definition of a monoid. A standard `\kl` command does the job:

```
\newrobustcmd\monoid{\kl[monoid]{\mathcal M}}
What is $\monoid$ ?
```

[3] it is good practive to use a separate file, something like 'paper-knowledge.tex'.

would yield:

What is $\mathcal{M}$?

The problem 3 *But I want more. I want to be able to introduce variables. Even better, I would like to be able to have variables hyperlinking to the place of their introduction, knowing that the same variable name may mean different things depending on the lemma or proof we are in. Hence, I want to properly control the scope of knowledges.*

To be done, this requires to use scoping. The principle of scoping is that a knowledge can be attached to a particular context. This is particularly true when typesetting mathematics: a variable is meaningful inside a statement, and inside the proof of the statement. Furthermore, the same variable name may reappear elsewhere with a different meaning.

The following code gives an idea of what is possible using scoping:

```
\knowledgescopingenvironment{theorem,lemma,proof}
[...]
\begin{lemma}\label{theorem:main}
    \knowledge{n}{notion}
    For all number $\intro n$, [...]
\end{lemma}
[...]
Here $\kl n$ is an undefined knowledge.
[...]
\begin{proof}[Proof of theorem~\ref{theorem:main}]
    \knowledgeimport{theorem:main}
    Inside the proof, $\kl n$ is hyperlinked to the theorem...
\end{proof}
```

More on scoping can be found in Section 4.5.

4 Usage of the knowledge package

4.1 Options and configuration

Options are used to activate some capabilities. Some options have to be used when loading the knowledge package, while some others can also be used inside the document thanks to the use of `\knowledgeconfigure`. In this section, we review these options.

4.1.1 Options at package loading

The options that can be used in the optional parameter of `\usepackage` when loading the knowledge package belong to the following classes:

Other packages some of the options concern the loading and the use of other packages (`hyperref`, `xcolor`, ...). This is explained in Section 4.1.2.

Configuration options as used by the command `\knowledgeconfigure` can be used when loading the package.

Scoping The (experimental) `scope` activates the `scope option`. To be described.

Writing stage/mode The *paper*, *electronic* or *composition* modes are possible (composition is by default). These modes change several default rendering settings (for instance, the option visible anchor points or `notion` are sensitive to the modes). The meaning of these modes are:

- In paper mode, the paper is rendered as for printing: in particular, no informative colors are visible.
- In electronic mode, the document has some colors witnessing the existence of the links for the reader to know that clicking is available.
- In composition mode, the document has colors helping the writing: unknown knowledges appear explicitly, anchor points are displayed, and so on.

Activating the modes is obtained either at load time using one of:

```
\usepackage[paper]{knowledge}
```

or `\usepackage[electronic]{knowledge}`

or `\usepackage[composition]{knowledge}`

or by setting before loading the variable `\knowledgepackagemode` as in:

```
\def\knowledgepackagemode{paper}
```

The idea is that this can be used in automatic compilation scripts. For instance, launching in a terminal:

```
pdflatex "\def\knowledgepackagemode{electronic}\input{file.tex}"
```

would result in compiling '`file.tex`' using knowledge in electronic mode. The modes are easily accessible to the configuration of the user using:

```
\IfKnowledgePaperModeTF{true code}{falses code}
\ifKnowledgePaperMode true code [\else false code] \fi
\IfKnowledgeElectronicModeTF{true code}{false code}
\ifKnowledgeElectronicMode true code [\else false code] \fi
\IfKnowledgeCompositionModeTF{true code}{false code}
\ifKnowledgeCompositionMode true code [\else false code] \fi
```

4.1.2 Automatic loading of other packages

A certain number of functionalities coincide with the loading of other packages. For the moment, the packages that are concerned are `hyperref`, `xcolor`, and `makeidx` packages.

For activating these functionalities, it is sufficient, either to load the package *before* the knowledge package, or to name it explicitly as an option for knowledge. Loading separately the package is convenient for setting options for it. For instance, a typical preamble may look like:

```
\documentclass{article}
\usepackage[svgnames]{xcolor}
\usepackage[draft]{hyperref}
\usepackage[makeidx]{knowledge}
```

Such a sequence will activate the knowledge package using the features related to `xcolor` configured with `svgnames` option, to `hyperref` configured with `draft` option, and to `makeidx` with its standard configuration.

In fact, the syntax when a package is loaded as an option of knowledge is of the form 'package=choice' in which choice can take the following values:

`active` The package will be loaded, and all the capabilities that it triggers are activated. This is the implicit meaning when nothing more is specified.

`inactive` The package is not loaded, and no capabilities are activated (even if it had been loaded previously by another `\usepackage` command).

`compatibility` The package is not loaded. The directives it used do not cause any error, but have no effect.

`auto` If the package was loaded before, then the associated capabilities are activated. This is the default behavior when the package is not named while loading.

Currently, the packages that can be loaded are:

`hyperref` which activates all the (auto)referencing capabilities.

`xcolor` which activates coloring commands.

`makeidx` for making index automatically.

4.1.3 Configuring and \knowledgeconfigure

Some part of the configuration can be done outside of the \usepackage command that loads the knowledge package. This is done using the \knowledgeconfigure command:

```
\knowledgeconfigure{configuration directives}
```

Note that by default, the configuration directives used by \knowledgeconfigure can be used in the optional parameter of \usepackage when loading the knowledge package, but the converse is not true. *Configuration directives* consists of a comma separated list of elements that can take the following values:

`quotation` activates the quotation notation, which allows to use "···" and "···@···" instead of \kl commands and ""···"" and ""···@···"" instead of the \intro command.

`protect quotation=` is followed by a comma separated list of environments in which the quotation notation must be deactivated (surrounded by braces if more than one item in the list).

`visible anchor points` is an option that makes visible the anchor points of the \AP and \itemAP commands.

`fix hyperref twocolumn` is an option that solves a known problem that may occur when hyperref is used in two-columns mode.

`protect link` **and** `unprotect link` start and end an zone in which the knowledge package do not create hyperlinks. These can be nested. This is typically useful around, e.g. the table of contents.

4.2 What is a knowledge?

A *knowledge* is often informally used in this document. Essentially, it captures what is an elementary concepts in the document. Internally, a knowledge is identified by three components:

The *knowledge name* is a TEX string that has almost no limitation (but being well balanced, and containing no ♯). It is the text entered by the user for defining and using the knowledge.

The *scope* which is a simple string identifying where the knowledge is usable. The scopes are generated by the system. For instance, internally, each section will be uniquely named '`section-1`', '`section-2`', and so on (this is invisible for the user). Each knowledge is primarily valid in exactly one such scope. Knowledges defined in the preamble are given the scope '`base`'.

The *namespace* is a simple string that is used for avoiding clashes. It is most of the time simply '`default`'. It is '`style`' for styles (that are internally as knowledges). It is a possibility available to a developer to, when developing a new set of functionalities, use a different namespace for avoiding clashes of names (for instance if one wants a french and an english set of knowledges that should not conflict, and would use separate sets of macros). Usually, a normal user does not see namespaces.

4.3 The \knowledge command and variations

In this section, we describe the main commands that create knowledges. The main one is \knowledge. It can also be used in combination with \knowledgedirective, \knowledgestyle and \knowledgedefault.

4.3.1 General description of the \knowledge command

The key command for introducing knowledges is \knowledge. The syntax is:

```
\knowledge{knowledge name}[synonyms]{directives}
```

The *knowledge name* is a string describing the knowledge. It may use any combination of symbols (well balanced with respect to brackets). This string will be used to fetch the knowledge. Note (and this is a standard TEX behavior) that several consecutive spaces is the same as one or a line feed. The *synonyms* are knowledge names given in a '|' separated list. This is a shorthand for writing 'link=' directives. (Note that there is another way to define synonyms using the synonym directive). The *directives* consists of 'key=value' statements in a comma separated list. There are many directives. A list can be found in Section 6.3. New ones can be created using the \knowledgedirective command.

The principle of the \knowledge command is to introduce a new knowledge, ready for being used. However, what it does exactly depends a lot on the situations. First, the directives (a comma separated list of 'key=value' commands) are parsed, and from it, the namespace and scope of the knowledge are determined, and it is decided if it will be executed immediately or postponed to the next compilation phase. Only then, either the knowledge is written in the kaux file for execution during the next compilation phase, or it is executed immediately. Finally, synonyms are parsed and linked to the knowledge.

4.3.2 Targeting and the corresponding directives

The \knowledge has to decide what to do when defining something. The basic behaviour is as follows.

- If the \knowledge command is used in the preamble, then the knowledge given as argument is defined immediately, and is accessible in the first compilation phase everywhere in the document (one extra phase is nevertheless required if autoref or ref= directives are used, for the hyperref package to do its job). This is the simplest way to use \knowledge.
- Import/Export *to be done*.
- Otherwise, the knowledge is written in an external file (the jobname.kaux file), and the knowledge will be really usable in the next compilation phase. This is particularly useful in conjunction with the scope option: the knowledge will have a scope depending on where it is introduced (for instance the document, or a theorem, or a lemma). The same knowledge name can then point to different knowledges depending on where it is used.

The *targeting directives* refine the above defined behaviour:

`scope=` (*not implemented*) When using a directive '`scope=`name', the scope can be modified. `\knowledge` will first check if there is an outer area of this name, that accepts knowledge. If this is the case, the knowledge will be associated to the corresponding instance. For instance, inside a theorem, by default, the scope is the theorem, but adding the directive '`scope=`document', the knowledge becomes globally available. Also in a section, by default, the scope of a knowledge is the entire document, but by specifying '`scope=`section', one can make it valid for the section only.

If no scope is found using the above search, a label of the given name is searched for, and if it exists, the corresponding scope is chosen.

`export=` (*not implemented*) When using this directive, the knowledge will be (furthermore) written in another file, ready for being used in another document. In particular, the knowledge (in the other document) will point to the present one. The details on how this is supposed to work is to be specified.

`namespace=` Allows to change the namespace. In itself, this is useless. It has to be used in conjunction with new forms of `\kl`-like commands.

`now` (*not implemented*) requires the knowledge to be defined immediately. This may save one compilation phase. The drawback is that the knowledge cannot be accessed before the `\knowledge` command that has been introduced. It may help for modularity considerations. (for instance a knowledge is used inside a proof, it has no sense to make it available elsewhere, and it is better style to locally define it).

4.3.3 General directives

We give here the list of *display directives* that are available without loading any sub packages. A certain number of Boolean directives are available without any options. These most of the time are used for typesetting the output. Each of these can be used as '`bool=true`' (or shortly just '`bool`'), '`bool=false`' or '`bool=default`' (that leaves it in the default state, or the one determined by surrounding knowledges). The general boolean directives are the following:

`emphasize` forces the text to be emphasized using '`\emph`',

`italic`/`up` forces/unforces italic (be it in math or text mode),

`boldface`/`md` forces/unforces boldface (be it in math or text mode),

`underline` forces the text to be emphasized using '`\underline`',

`fbox` puts a box around the text,

`typewriter` puts in typewriter font (be it in math or text mode),

`ensuretext` guarantees that text mode is used (using the '`\text`' macro, thus in a way consistent with the surrounding style),

`ensuremath` guarantees that math mode is used,

`mathord`, `mathop`, `mathbin`, `mathrel`, `mathopen`, `mathclose`, `mathpunct` yield the corresponding standard TEX spacing features in math mode,

- `mathord` for an ordinary mathematical object,
- `mathop` for a large operator (such as $\sum$, $\prod$, ...),
- `mathbin` for a binary operation (such as $+$, $-$, or $\otimes$, ...),
- `mathrel` for a binary relation (such as $=$, $<$, $\leq$, ...),
- `mathopen` for an opening bracket, parenthesis, ...
- `mathclose` for an closing bracket, parenthesis, ...
- `mathclose` for a punctuation symbol.

`lowercase` puts the content in lowercase,

`uppercase` puts the content in uppercase.

The non-boolean general directives are the following:

`text=`{text} will execute the LaTeX code 'text' instead of the key used for calling `\kl`. For instance, `\knowledge{latex}{text=\LaTeX}` will typeset 'LaTeX' properly when used. Surrounding braces can be omitted if there are no commas. Be careful when linking to such knowledges, since the substitution of meaning will happen for all the knowledges linking to it.

`link=`{knowledge} will continue searching the linked knowledge. Surrounding braces can be omitted if there are no commas. This directive is often bypassed by the use of the optional argument of `\knowledge` defining synonyms or the `synonym` directive.

`synonym` defines the knowledge as a link to the previously defined knowledge (in fact, the most recently defined that was not using `synonym`). For instance

```
\knowledge{Leslie Lamport}
     {ref={https://fr.wikipedia.org/wiki/Leslie_Lamport}}
\knowledge{L. Lamport}{synonym}
\knowledge{Lamport}{synonym}
```

results in the two subsequent knowledge names to point to the first one.

`style=`{knowledge style} will adopt the styling option of the knowledge style. Surrounding braces can be omitted if there are no commas.

`wrap=`\token will execute the macro '\token' with as argument the knowledge text before displaying it. For instance, `wrap=\robustdisplay`, (where `\robustdisplay` is a variant of `\detokenize` removing the trailing space) is used in this document for typesetting the commands.

4.3.4 Knowledge styles and the `\knowledgestyle` command

Styles are formatting information, as for knowledges, but that can be used by other knowledges. In some respect, this is very similar to macro directives (see below), but the difference lies in that styles are dynamically resolved, while macro directives are statically resolved. Styles in particular offer the access to some configuration features of the system. For instance, changing the `intro` style changes the way the `\intro` command is displayed. See below for some instances.

The central command is `\knowledgestyle`, that has the following syntax:

```
\knowledgestyle*{style name}{directives}
```

The optional star permits to overload an existing style (otherwise, this results in an error). The directives follow the same structure as for a normal `\knowledge` command. When defined, a style can be used in a `\knowledge` command using the directives '`style=`style name' (it will be used when a `\kl` command calls for the knowledge) or '`intro style=`style name' (that will be used by `\intro` commands).

A certain number of *default styles* are also offered, that in particular includes *warning styles*. The list is as follows:

`intro` is the default style for macros using `\intro`. It can be changed using the '`intro style=`' directive (after `autoref`).

`unknown` is the default style used for the first time an undefined knowledge is met.

`unknown (cont)` is the style adopted for the following occurrences of an undefined knowledge.

`autoref not introduced` is the style used the first time a knowledge has been used using the `autoref` directive, but there is no corresponding `\intro` (*not implemented*).

`autoref not introduced (cont)` is as above for the subsequent occurrences of the knowledge.

`autoref reintroduced` is the style used when a knowledge defined with the directive `autoref` has be found twice in an `\intro` command (*not implemented*)

`autoref reintroduced (cont)` (continued)

4.3.5 New directives: the `\knowledgedirective` command

When defining knowledges, it is often the case that the same sequence of directives are used. *Macro directives* are here for simplifying this situation (see also `\knowledgedefault` and `\knowledgestyle`). This is achieved using the `\knowledgedirective` directive:

Hint. This should not be confused with styles which offer another way to control the display.

```
\knowledgedirective{name}[optional parameter]{directives}
```

After such a command has been issued, 'name' becomes a directive usable in `\knowledge` commands, that amounts to execute the comma separated list 'directives'. The newly created directive may receive a value, that is accessible as `#1` in 'directives'. The 'optional parameter' gives a default value. For instance:

```
\knowledgedirective{highlight}[brown]{color={#1},emphasize,md}
[...]
\knowledge{notion A}{highlight}
\knowledge{notion B}{highlight}
\knowledge{notion C}{highlight}
\knowledge{important notion D}{highlight=red}
[...]
We shall now see \kl{notion A}, \kl{notion B}, \kl{notion C}, as
well as the \kl{important notion D}.
```

yields

We shall now see *notion A*, *notion B*, *notion C*, as well as the *important notion D*.

4.3.6 `\knowledgestyle` versus `\knowledgedirective`

The two commands `\knowledgestyle` and `\knowledgedirective` offer ways to systematize the writing of knowledges. These can seem redundant. This is not the case, and for understanding it, it is necessary to understand a bit the way the `\knowledge` command works.

In general when a `\knowledge` (or `\knowledgestyle`) command is found, the directives are parsed and a new internal form of the `\knowledge` command is written in the kaux file, that will be executed during the next compilation of the document. In this phase, some first operations are performed. For instance, in an `autoref` directive, an internal label name is constructed. Executing a knowledge macro is done at this moment.

The postponed command is then executed during the next compilation phase (or immediately if we are in the preamble, or if the `now` directive is used). The execution effectively stores the knowledge in the system. This is only at that moment that the knowledge becomes available to be used by `\kl` and similar commands.

When a `\kl` command (or similar) is met, it is 'executed', and display informations are considered, and in particular styles are called.

Somes consequences of this kind of this are as follows:

- `autoref` directives should not be used in the definition of a style, since this would mean that there would be one anchor point for all the knowledges that use this style. This is usually not the kind of behavior that we expect.
- configuring the default displays of the system (such as the `intro style=` in particular) has to be done through the style mechanism.
- styles are slightly more efficient than macros usually.

4.3.7 Default directives: the \knowledgedefault command

It is often the case that a long sequence of consecutive \knowledge commands need share the same list of directives. The macro directives can help solving this issue. The *default directives* also go in this direction, using the \knowledgedefault command:

```
\knowledgedefault*{directives}
```

When such a command is applied, then from that point, all \knowledge commands will use the given directives as default. This will stop when another \knowledgedefault command is met or the current group is closed. The optional star does not reset the default directives but simply add new ones.

4.4 The \kl command

4.4.1 The standard syntax

Hint. Note that the \kl- command can often be replaced by the "···" notation, activated by the quotation option.

The \kl command has the following syntax:

```
\kl[optional knowledge name]{knowledge name}
```

its semantic is to search for the knowledge name, or the optional knowledge name. The search process is as follows:

- the stack of search scopes is processed through (starting from the inner most) until a knowledge of this knowledge name, of namespace 'default' and this scope is found[4].

If the knowledge name has not been found, the style unknown is used, and the knowledge name displayed. Then a corresponding knowledge is added at the scope 'base', being defined to use the style unknown (cont).

- Otherwise, the knowledge is executed. If it is a link= or synonym defined knowledge, the link is followed, and the process continues.
- Finally, all the definition involved in the knowledge are processed, following a style= if defined, the knowledge is updated (essentially incrementing the counter of use), and the knowledge is displayed.

This general mechanism is used also by other commands that are variations arounf \kl such as in particular \intro.

4.4.2 The "···" and "···@···" notation using quotation

When activated, the quotation mode allows to use the shorthand "···" instead of \kl{···} (it also activates a similar ""···"" for use instead of \intro). The notation "···@···" is similar and allows to provide an alternate knowledge. Hence:

"Donald Knuth"	is equivalent to	\kl{Donald Knuth}
"the author@Knuth"	is equivalent to	\kl[Knuth]{the author}

[4] If the scope option is not activated, this simply means that the scope 'base' is searched for.

Activating this mode can be obtained using:

```
\knowledgeconfigure{quotation}
```

and deactivating it is obtained by using:

```
\knowledgeconfigure{quotation=false}
```

It can also be activated while loading the package.

It is sometimes the case that some package does use the quote symbol, usually in some environment (this is the case of the tickzcd environment). The knowledge package can be configured to deactivate always the quotation notation when entering the environment. This is obtained using the package directive `protect quotation=` followed by a list of environments to be protected:

```
\knowledgeconfigure{protect quotation={env1,env2,...}}
```

Note that the braces surrounding the list of environments can be omitted if the list contains only one item.

There are nevertheless some situation in which one would prefer to use the original `\kl` notation:

- When nesting of knowledges is involved, or the knowledge includes the symbol ",
- when `quotation` is deactivated (or not activated) because of a conflict
- in particular, this should be avoided in macros, in particular for the math mode, since these may be used one day or another in a `tikzcd` or similar environment for instance.

4.5 Scoping

Rapidly, when long documents are in construction, one wants knowledges to be isolated in some subparts. For instance, one may want that a temporary definition in a proof do not leak elsewhere in the document where the same term could be used with a different meaning. Some definitions may be only meaningful in, say, the current section/part. This is in particular true when one aims to track single variables: Clearly, a variable named x can occur in several lemmas, but each of them correspond to a distinct 'introduction' location. For handling this situation, the knowledge package posses some scoping features.

Hint. Note that in the current implementation, the use of scoping incurs one extra compilation phase.

4.5.1 What is the structure of scopes in a document

To start with, one needs to understand what are the possible scopes.

- By default, all the body of the document belongs to a scope called '`document`'. The user can open new scopes using the `scope` environment:

```
\begin{scope}
  \knowledge{local notion}{color=green}
  Here is a \kl{local notion} that appears in green.
\end{scope}
But this \kl{local notion} is undefined.
```

Note that scoping is independent from the the grouping mechanism of LaTeX.

The user can also declare an environment (existing or to exist) to behave like `scope` using the command `\knowledgescopingenvironment`:

```
\knowledgescopingenvironment{list of environments}
```

For instance:

```
\knowledgescopingenvironment{lemma,theorem,fact,proof}
```

Note that (in the current implementation) it is necessary to use the commands `\begin` and `\end`. Hence `\proof`...`\endproof` would not trigger a scoping environment while `\begin{proof}`...`\end{proof}` would.

- (Under unstable development) The use of the `scope option` configuration option reconstructs the structure of the document, and scopes will be created for sections, subsections, itemize, items, and so on. Be cautious.

4.5.2 How do we chose the scope of a knowledge

In general, when a `\knowledge` command is used, the system tries to figure out it what should be its scope:

- If the command occurs in the preamble, then the default scope will be 'document'.
- Otherwise, if nothing is specified, then the knowledge will be defined at the level of the innermost surrounding scope that 'attracts knowledges'. If the `scope option` is not activated, this is the inner most `scope` environment, or 'document' if the declaration is not in the scope. If the `scope option` is used, this will be the innermost lemma, proof, or theorem in the context.
- (*not implemented*) This default behaviour can be modified using the `scope=` directive. It makes really sense only when the `scope option` is activated. Then `scope=` can be followed with a scope level, such as 'section', 'subsection', 'chapter' or 'itemize', that we be looked for in the current context and will receive the knowledge. The directive can also be followed by a label name, and the active scope at the moment of this label will be used.

```
\section{First section}
\knowledge{one}{scope=section,color=green}
\knowledge{two}{scope=some label,color=green}

\begin{scope}\label{some label}
  Here \kl{one} and \kl{two} are defined.
\end{scope}
Here \kl{one} is defined but \kl{two} isn't.

\section{Second section}
Here neither \kl{one} nor \kl{two} is defined.
```

4.5.3 Accessing other scopes, the `\knowledgeimport` command

Something important is missing so far: one rapidly wants to access to knowledges that do not exist in the current scope. For instance, a notion is used in a section of a document, and one would like to refer to it in the introduction. Another instance is that of a notion or a mathematic variable that is introduced in the statement of a theorem, and should be accessible inside the proof. The `\knowledgeimport` command is here for that. Its syntax is:

```
\knowledgeimport{label}
```

The result is that the knowledges in the scope identified by the label are now accessible until the closure of the current scope.

For instance:

```
\knowledgescopingenvironment{theorem,proof}
[...]
\begin{theorem}\label{theorem:1}
  \knowledge\alpha{autoref,color=red}
  Let $\intro\alpha$ be an integer [...]
\end{theorem}
[...]
Here $\kl\alpha$ is unknown.
[...]
\begin{proof}
  \knowledgeimport{theorem:1}
  But now $\kl\alpha$ points to its definition.
\end{proof}
```

4.6 Error handling

By default, the knowledge package tries to not stop the compilation unless a serious problem has been found. In particular, it is possible to write an entire document using `\intro` and `\kl` commands without ever introducing a knowledge, and only in the end provide this information. This is a feature: as opposed to normal

macros, not defining a knowledge should not stop the real work, which is the writing of the document.

It happens very often that a *knowledge is not defined*. Such knowledges are then displayed using the `unknown` style the first time, and the `unknown (cont)` style the subsequent times. The detail of the problems are then gathered in the diagnose file.

4.7 Importing and exporting (*not implemented*)

4.8 Other packages

4.8.1 The `xcolor option`

The `xcolor option` is used if one wants to change colors. It is good to always load it since it also triggers coloring for debugging. It triggers colors in the warning styles that can be useful in debugging. It also offers two new directives:

`color=` where in '`color=`name', name is a color description following the syntax of the `xcolor package`.

`colorbox=` surrounds the text with a colorbox of given color (following the syntax of the `xcolor package`).

Loading the package before is necessary for changing the options of the `xcolor package` (for instance for using `svgnames`).

4.8.2 The `hyperref option`

Activating the `hyperref option` The `hyperref option` loads the `hyperref package` and triggers a certain number of link-related features. This is done either by the command:

```
\usepackage[hyperref]{knowledge}
```

or by loading the `hyperref package` before the knowledge package (suggestion: with the `hidelinks` option).

Hint. The `hyperref package` tends to surround links by boxes that do not help. A solution is to use the `hidelinks` option, i.e., load it with:

```
\usepackage
[hidelinks]{hyperref}
```

This is done by default when it is loaded by the knowledge package.

The directives activated by the package are:
`url=` for hyperlinking to an external document
`ref=` for hyperlinking inside document
`protect link` it a boolean for protecting from the creation of nested hyperlinks,
`autoref` for relating objects with their definition
`autorefhere` similar, and used implicitely for math
The package comes also with the configuration option `hyperlinks=` which is a boolean which can be used to deactivate or reactivate the links.

Functionnalities triggered by the `hyperref option`

`ref=` `{label}` puts an hyperlink pointing toward a label inside the document (the braces can be omitted when there is no comma).

`protect link` disables the inside hyperlinks,

`url=` `{url address}` puts an hyperlink to an (external) url (the braces can be omitted when there is no comma).

Hint. You may have to use \~ instead of ~ in url's addresses.

`autoref` activates the ability to introduce once, use several times an instance. This is very convenient when writing scientific documents with many notions. This is the basic directive activating the features of the `\intro` command.

`autorefhere` puts immediately a label at the location of the definition, and makes all `\kl` occurrences of this knowledge hyperlink to this location.

Hint. It is usually easier to use the '`notion`' directive than simply the `autoref` directive. Its use it already configured.

The `autoref` directive The `autoref` directive is among the most useful offered by the knowledge package. When set, the knowledge should be used with both `\intro` (exactly once) – or the `""...""` and `""...@...""` notations if `quotation` is active – and `\kl` (possibly several times) – or the `"..."` notation if `quotation` is active. The use of `\kl` will hyperlink to the location of the `\intro`. The syntax of `\intro` is the same as for `\kl`:

```
\intro[optional knowledge name]{knowledge name}
```

See `\AP` below for improving the result.

A typical use looks as follows:

Hint. Though the `\intro-` command can be used in the title of, e.g. sections, without any errors, this may cause a warning when a table of contents is used: the command is executed twice, once in the table of contents, and once in the document itself.

```
\knowledge{house}[Houses|houses]{autoref}
[...]
\begin{document}
[...]
In this document, we will see the very important notion of
"houses".
[...]
\AP
Let us define a ""house"" to be a building that functions as a
home.
[...]
\end{document}
```

yields

> *[...]*
> In this document, we will see the very important notion of houses.
> *[...]*
> Let us define a *house* to be a building that functions as a home.
> *[...]*

The variant `\intro*` makes the next `\kl` command behave like `\intro`. This is useful in particular in math mode:

```
\newcommand\monoid{\kl[\monoid]{\mathcalM}}
\knowledge\monoid{autoref}
[...]
\AP
Let now $\intro*\monoid$ be a monoid.
[...]
Remember now who is $\monoid$.
```

Hint. This does not work in `align*` and similar environments. Section 5.5 gives some solutions.

Let now $\mathcal{M}$ be a monoid.
[...]
Remember now who is $\mathcal{M}$.

The `\phantomintro` version:

```
\phantomintro{knowledge}
```

takes a knowledge, and introduces it at the current location, without displaying anything. This is behaves like an invisible intro, i.e., essentially an abbreviation for `\intro[knowledge]{}`. This can be used as a workaround in environment like `align*` that do not allow the use of labels (see Section 5.5).

The `\nointro` command:

```
\nointro{knowledge}
```

does not display anything and silently prevents the knowledge from issuing warnings because it is not introduced.

The `\reintro` command:

```
\reintro[optional knowledge]{knowledge}
```

is displayed as for `\intro`, but without being an anchor for hyperlinks, and without counting as a real `\intro`. It is used if there are for some reason several places that should look like an introduction (typically in the same paragraph), but count as a single target. There is a variant `\reintro*` that makes the next `\kl` command behave like a `\reintro` (similar to `\intro*` with respect to `\intro`).

Knowledges that use this directive can be parameterized by modifying the style `intro`.

For modifying the display of knowledges introduced by `\intro`, there is a new directive:

`intro style=` that takes the name of a style as argument. This style will be used when the knowledge is used in a `\intro` or `\reintro` command.

The `autorefhere` directive The `autorefhere` directive silently introduces an anchor point at the location of the `\knowledge` command invoking it. Uses of `\kl` commands will be hyperlinked to this location.

In some sense, an `autorefhere` directive can be understood as the sequence of a `autoref` directive that would be immediately followed by the corresponding `\intro` command. This is a bit better since using `autoref` is the body of the document requires three phases of compilation (two only if in the preamble). However, the `autorefhere` directive does only require two (as for normal labels).

In fact, this `autorefhere` directive is what is used underneath when introducing mathematical variables, and should be used for implementing similar behaviors.

Using anchor points The directives `autoref` and `autorefhere` use underneath the `hyperref package`. This means that it puts à label at the place of the `\intro` command, and then points to it. However, the semantics in this case, is to jump to the beginning of the surrounding 'region'. If the `\intro` happens in a 'section' (but not inside a theorem-like envionment) then the `\kl` command will point at the beginning of the section, possibly 10 pages above the definition itself.

The standard solution in the `hyperref` package is to use the `\phantomsection`. This means defining *anchor points* in the document that will be the target of hyperlinks.

We offer here new commands for helping using this feature:

`\AP` declares an anchor point at the left of the current column, at the height of the current line. If the configuration option `visible anchor points` is set (and this is the case by default), a mark will show the precise location of the target. It does not work in some situation, like for instance inside the optional argument of an `\item` command (but this is ok elsewhere in an itemize environment). In this particular case, one should use instead:

`\itemAP` Similar to `\AP`, but to be used instead of an `\item`.

Usually putting an `\AP` (a standard command of the `hyperref package`) at the beginning of every paragraph, and replacing `\item` by `\itemAP` in itemize-like environments is most of the time good and safe option.

For instance:

```
\AP
In order to describe what is a \kl{monoid}, let us us first define
a \intro{product} to be an associative binary operator, and a \int-
ro{unit} to be [...]

\begin{description}
\itemAP[A \intro{semigroup}] is a set equipped with a \kl{product}.
\itemAP[A \intro{monoid}] is a \kl{semigroup} that has a \kl{unit}.
\end{description}
```

yields

> In order to describe what is a monoid, let us us first define a *product* to be an associative binary operator, and a *unit* to be */.../*
>
> **A *semigroup*** is a set equipped with a product.
>
> **A *monoid*** is a semigroup that has a unit.

One can check that the different knowledges are properly hyperlinked, and that precise targets are the one described by `\AP` and `\itemAP`. For helping debugging the anchor points, these are by default made visible as (red) corners on output. When the knowledge package is loaded with the `paper` option these graphical helps disappear. This can also be desactivated using:

```
\knowledgeconfigure{visible anchor points=false}
```

4.8.3 The `makeidx option`

Activating the `makeidx option` The `makeidx option` loads the `makeidx package` and triggers a certain number of link-related features. This is done either by the command:

```
\usepackage[makeidx]{knowledge}
```

or by loading the `makeidx package` before the knowledge package.

Features When activated, it becomes possible to trigger the `\index` command when a `\kl` command is used. The following directives are use:

`index=` is the version that uses the standard syntax of the `\index` parameter.

`index key=` takes as argument the index key: a text that is used for identifying the index entry (usually an accent free version of it).

`index parent key=` makes the index entry be a subentry of the given main index entry.

`index style=` makes the index entry be a subentry of the given main index entry.

4.9 Dealing with math

This part is under development.

4.10 Fixes

In this section, we present some fixes that have been added to help the user solve problems.

Hyperref and twocolumn It happens that the `hyperref package` and twocolumn mode yields a fatal error. This happens when a link spans across the boundary between two pages. This is an issue which is not related to the knowledge package, but becomes severely more annoying when more links have to be used. A *workaround* can be tried by calling the command `\knowledgeFixHyperrefTwocolumn` in the preamble. I do not know to which extend it is compatible with various classes...

4.11 Predefined configuration

4.11.1 Science paper

The configuration option `notion` is activated using:

```
\knowledgeconfigure{notion}
```

It automatically configures a directive `notion` which is an `autoref` displayed properly:

- In paper mode, the `\intro` commands (not in math mode) are emphasized, while the `\kl` commands are displayed as normal. It has the aspect of a normal paper.
- In composition mode (with the `xcolor package`), notions are furthermore typeset in blue when introduced, and in dark blue when used. Without the `xcolor` package, underlining draw the attention to the knowledges (not in math mode).

A typical document using notion could start by the following commands:

```
\documentclass{article}
\usepackage{xcolor}
\usepackage[hidelinks]{hyperref}
\usepackage[paper]{knowledge}
\knowledgeconfigure{notion}
[...]
\knowledge{some text}{notion}
```

Then the paper is displayer in a colorful way. As soon as the `false` is replaced by `true`, the paper becomes black and seriously looking as it should.

5 Some questions and some answers

5.1 How to compile?

As usual with LaTeX, a certain number of compilation phases are necessary for reaching a document in final form. The problematic point is of course the use of labels, and in particular the `\intro` command. When it is used, and all the `\knowledge` commands are in the preamble, then two phases are necessary. When `\knowledge` commands are used in the body of the documents, then one extra phase is required (in particular when using scopes), meaning three with `autoref` definitions. It could be possible to improve this if the knowledge is always defined before the `\intro` command, but this is not done for the moment.

5.2 Problem with `\item` parameters

The use of `\AP` inside `\item` does not work. Do not use `\AP` inside the optional argument of `\item`, and rather use the command `\itemAP`.

Argument of `\kl` has an extra '}'. This is a problem of using optional parameters inside optional parameters such as in `\item[\kl[test]{Test}]`. You can surround the content of the optional parameter by two level of curly braces as in `\item[{{\kl[test]{Test}}}]`. The notation `"···"` does not have this issue.

5.3 Knowledges and moving arguments (table of contents, ...).

The use of `\kl` does not work in (e.g.) the table of content. When the knowledge name contains expandable macros, or accentuated letters, then these are not copied in the table of content as the exact same text, but are expanded/translated. Thus, when the table of content is displayed, the `\kl` command complains of not knowing the knowledge. For instance[5]:

[5] with `\usepackage[utf8]{inputenc}` and, for instance `\usepackage[T1]{fontenc}` for the accents.

```
\newcommand\Ltwo{\ensuremath{L^2}}
\knowledge{\Ltwo-space}[\Ltwo-spaces]{autoref}
\knowledge{étale topology}[Étale topology]
       {url={https://en.wikipedia.org/wiki/Étale_topology}}
[...]
\begin{document}
\tableofcontents
\section{On \kl{\Ltwo-spaces}
[...]
\section{On the \kl{étale topology}}
[...]
\end{document}
```

will result in that both knowledges are considered unknown in the table of contents. For the first one, this is due to the expansion of `\Ltwo`. For the second, this is due to an implicit translation of the accentuated letter into an internal sequence of commands (for instance 'é' is translated into the internal sequence '`\IeC {\'e}`'). Some solutions are as follows:

- Make the macros non-expandable, for instance using `\newrobustcmd` (of the `etoolbox package`) or `\NewDocumentCommand` (of the `xparse package`, with a different handling of arguments) instead of `\newcommand`. Hence:

  ```
  \newrobustcmd\Ltwo{\ensuremath{L^2}}
  ```

  solves the first problem.

- Using an equivalent text that does not have the problem:

  ```
  \knowledge{\'etale topology}{link=étale topology}
  [...]
  \section{On the \kl{\'etale topology}}
  ```

- Both problems can be solved using synonyms/links that have no problem. For instance:

  ```
  \knowledge{Ltwo-space}{link=\Ltwo-space}
  \knowledge{etale topology}{link=étale topology}
  [...]
  \section{On \kl[Ltwo-space]{\Ltwo-spaces}}
  \section{On the \kl[etale topology]{étale topology}}
  ```

- Other solutions? None so far. I am trying to systematize the treatment of these problems.

Using \intro in a section title causes introducing the knowledge twice. Do not use \intro in titles, but rather \reintro. If you want the section to be the target of the knowledge, then put after the section a \nointro command.

```
\section{On \intro{topology}}
```

Problematic code

```
\section{On \reintro{topology}}
\phantomintro{topology}
```

A solution

5.4 Problems with tikzcd and other issues with the quotation notation

The package tikzcd uses (heavily) the quotes. Thus, it conflicts with the quotation notation. Some other packages may do the same. For solving this issue, the only thing to do is to temporarily deactivate the quotation notation.

To avoid this problem, it is sufficient to use before each figure:

```
\knowledgeconfigure{quotation=false}
```

and after the figure:

```
\knowledgeconfigure{quotation}
```

Another possibility is to force some environment to deactivate systematically the quotation notation when used. For instance

```
\knowledgeconfigure{protect quotation={tikzcd}}
```

will deactivate the quotation notation in all the tikzcd environments.

5.5 Problems with amsmath

The \intro command does not work in align* or similar environments It happens that in starred environment (i.e., unnumbered), the package amsmath deactivates the labels. As a consequence the command \intro, which internally uses \label (at least so far), does not work. For the moment, there is no real solution, but a workaround which consists in introducing the knowledge before the incriminated environment using \phantomsection, and then use \reintro inside the environment. Imagine for instance a command \SomeCommand, that inside uses \kl[\Somecommand], then:

does not work

```
\begin{align*}
\intro*\SomeCommand
\end{align*}
```

works

```
\phantomintro\SomeCommand
\begin{align*}
\reintro*\SomeCommand
\end{align*}
```

5.6 Hyperref complains

A fatal error occurs in twocolumn mode. A workaround is to use `\knowledgeconfigure{fix hyperref twocolumn}`.

5.7 Incorrect display

Incorrect breaking at the end of lines (in Arxiv for instance) It may happen that some hyperlinks generated by knowledge are not broken properly at the end of lines. This is an issue with the `hyperrref` package. This in particularly happened for files compiled by the Arxiv system while the file on the local computer was not having any problem.

A workaround is to use the `breaklinks` option of `hyperrref`. The preamble thus looks like:

```
[...]
\usepackage[breaklinks,hidelinks]{hyperref}
[...]
\usepackage{knowledge}
[...]
```

5.8 Editor

5.8.1 Emacs editor and quotes

The AucTex mode in Emacs binds the quote symbol to other characters. This is not convenient when using the knowledge package.

This behavior can be deactivated temporarily using:

```
M-x local-unset-key \".
```

or definitively using:

```
(defun my-hook () (local-unset-key "\""))
(add-hook 'LaTeX-mode-hook 'my-hook)
```

5.9 Others

If other kind of problems occur, report them to `thomas.colcombet@irif.fr`.

6 Resources

6.1 List of commands

`\intro` searches for a knowledge and put an anchor to it (to be used with the `autoref` directive).
`\kl` searches for a knowledge and displays it accordingly.
`\knowledge` defines new knowledges.
`\knowledgeconfigure` configures the package.
`\knowledgedefault` declares the default directives to be automatically used in `\knowledge` commands.
`\knowledgeimport` gives access to knowledges existing in other scopes.
`\knowledgedirective` defines a new directive.
`\knowledgestyle` defines a new style.
`\nointro` declares that the knowledge will never be introduced (does not work properly yet).
`\phantomintro` performs an invisible `\intro`.
`\reintro` uses the display style of `\intro` without introducing an anchor.

6.2 List of environments

`export` (*not implemented*) requires exportation of the content
`import` (*not implemented*) declares external resources
`scope` Defines a scope in which knowledges are internal.

6.3 List of directives

`autoref` Activates the `\intro` feature (requires the `hyperref package`).
`autorefhere` creates an anchor point that points to the `\knowledge` command (Requires the `hyperref option`).
`boldface` Displays the knowledge in boldface.
`color=` Displays the knowledge is the given color (uses `xcolor`).
`colorbox=` Displays the knowledge in a box of the given color (uses `xcolor`).
`emphasize` Emphasizes the displayed output.
`ensuretext` Guarantees that the output will be displayed in text mode.
`ensuremath` Guarantees that the output will be displayed in math mode.
`export=` (*not implemented*)
`fbox` Surround the text with a box.
`md` Removes boldface typesetting.
`notion`
`index=` Chooses the text to be displayed in the `index=`.
`index key=` the key used to choose the place in the index.
`index style=` the style to be used to display in the index.
`index parent key=` the parent key in the index.
`intro style=` Chooses the typesetting in case of an intro.
`italic` Typesets the output in italic.

`link=` Follow with the search the linked knowledge.
`lowercase` Put all letters of the output in lowercase.
`mathord`, `mathop`, `mathbin`, `mathrel`, `mathopen`, `mathclose`, `mathpunct` Selects a spacing behaviour in math mode.
`protect link` Disables the hyperlinks inside the link.
`ref=` Links to a label inside the document.
`scope=` Choose the scope of the definition.
`style=` Links to a style.
`synonym` Is a synonym of the lastly defined knowledge.
`text=` Changes the output text.
`typewriter` Typeset in as with `\texttt`.
`underline` Underlines the text.
`up` Removes italic typesetting.
`uppercase` Put all letters of the output in uppercase.
`url=` An url to point to (uses the `hyperref package`).
`wrap=` A macro used to process the displayed text.

6.4 List of configuration directives (to use with `\knowledgeconfigure`)

`hyperlinks=` activates or deactivates the hyperlinks
`quotation` activates or deactivates the `quotation` notation
`protect quotation=` declares a list of environment in which the quotation notation should be deactivated
`visible anchor points` makes the anchor points either visible or invisible
`notion` activates the `notion` directive
`paper` switches to paper mode,
electronic switches to electronic mode,
`composition` switches to composition mode,
`fix hyperref twocolumn` fixes a known problem between `hyperref package`hyperref and the two column mode.

List of default styles