Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 9 (2012) 439 – 448

Procedia Computer Science

International Conference on Computational Science, ICCS 2012

Literate Program Execution for Reproducible Research and Executable Papers

Sébastien Li-Thiao-Té[a,*]

[a]*Université Paris 13, CNRS, UMR 7539 LAGA*
99, avenue Jean-Baptiste Clément, F-93 430 Villetaneuse, France

Abstract

Lepton is an automaton for literate executable papers. It enables researchers to publish their work in the form of a script or program that can generate the research paper along with the corresponding source code, input data and output results. Lepton files do not contain pre-computed results, but the full set of instructions for reproducing the results presented in the manuscript.

Taking inspiration from literate programming for code review and code re-use, we have written Lepton to facilitate the review and re-use of computational methods. Lepton is designed to provide strong guarantees for the reproducibility of the results, many features for easily applying the methods to new data while remaining unobtrusive and easy to deploy in any environment.

Lepton is designed for writing reproducible technical reports during method development, and journal manuscripts when the research is polished. Developped independently from the Elsevier Executable Paper Grand Challenge, it addresses similar issues and objectives. This manuscript was written with Lepton.

Keywords: Executable papers, reproducible research, literate programming

1. Introduction

Communicating research results is about providing fellow scientists with the means to review and re-use. Reproducing the results of experiments has always been at the heart of the scientific method. Originally, reproducibility meant similar measurements and outcomes in experimental sciences such physics, chemistry, biology, etc. However, with the advent of the computer, a new type of reproducibility has emerged. Modern computational science is also about methods and results involved in extracting knowledge from the large datasets that are currently generated. These methods and their results must also be reproducible.

As expressed by the 2011 Elsevier Executable Paper Grand Challenge, fundamental requirements are not met by current publishing methodologies. In most research papers, essential elements of appreciation of the claims are absent: source code, implementation details, input data, input parameters, description of the computing environment, etc. Even when all these elements are present, reproducing the results in a paper can require high levels of technical expertise. Finally, research papers are intended to be archived and preserved in the long term. How can we ensure the reproducibility of methods and results when hardware and software evolve at an increasing pace?

*Corresponding author
Email address: `lithiao@math.univ-paris13.fr` (Sébastien Li-Thiao-Té)

1877-0509 © 2012 Published by Elsevier Ltd.
doi:10.1016/j.procs.2012.04.047

In this manuscript, we present Lepton[1], a tool developped independently from the Elsevier Executable Paper Grand Challenge but which aims to solve the same fundamental issues. Leptoninherits from literate programming [2] which is a paradigm for source code documentation and from experiments in programmatically generating PDF documents such as research reports, exam papers and source code documentation. This piece of software is stable, available for download, and was used to generate this manuscript.

As Lepton is a sophisticated yet very simple tool, we can provide a manual in Section 2. Lepton is a tool intended for doing research (see Section 3). It creates transparent and reproducible papers (see Section 4) that are suitable for publication and peer-review (see Section 5). In Section 6 we compare Lepton with other frameworks for reproducible research proposed during the Elsevier Grand Challenge, and discuss how to extend Lepton with some of their functionalities.

2. The Lepton manual

2.1. Tutorial

Lepton processes files written in LaTeX-like syntax. To write a "hello world" manuscript, the first step is to write a `hello.nw` file containing:

Code chunk 1: `<<hello.nw>>`
```
\documentclass[paper=a7]{scrartcl}
\usepackage[width=7cm,height=10cm]{geometry}
\input{lepton.sty}
\begin{document}
The code below sends "hello world" instructions to the \verb ocaml  interpreter.
<<hello_world -exec ocaml>>=
let msg = "Hello world.";;
print_string(msg); print_newline();;
@
\end{document}
```

The second step is to apply Lepton. This tool splits the file into documentation and source code, executes instructions where specified, and embeds the results. Lepton turns `hello.nw` into a legitimate LaTeX document `hello.tex`. When processing a file, Lepton outputs the name of each encountered code snippet and how it deals with it.

Code chunk 2: `<<hello.tex>>`
```
lepton.bin hello.nw

hello_world (part 1): chunk as ocaml, exec with ocaml, output as text,
```

The last step is to compile using `pdflatex`. The `-shell-escape` option enables colorful pretty-printing with the `minted` LaTeX package. The resulting PDF file is displayed in Figure 1.

Code chunk 3: `<<hello.pdf>>`
```
pdflatex -interaction batchmode -shell-escape hello.tex

This is pdfTeX, Version 3.1415926-1.40.10 (TeX Live 2009/Debian)
 \write18 enabled.
entering extended mode
/usr/bin/pygmentize
```

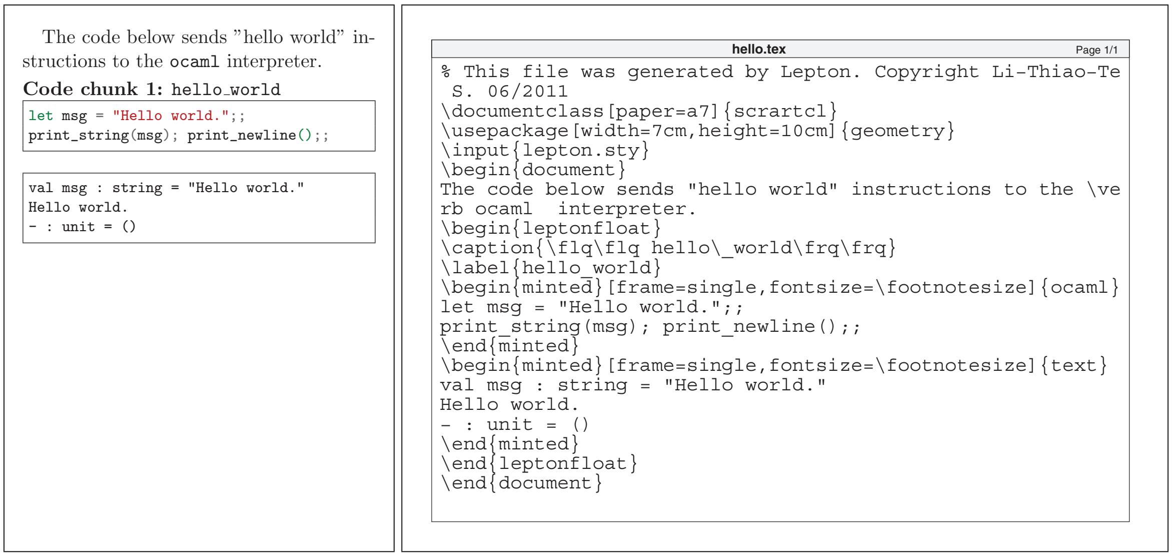
2.2. Usage and command-line options

```
lepton [-o texname] [-env envname] [filename]
```

Lepton uses `filename` as the input file name, or standard input when absent.

-o texname sets the name of the output LaTeX file.

-env envname uses `envname` instead of the default `minted` environment. See Section 2.5 for details.



The code below sends "hello world" instructions to the `ocaml` interpreter.

Code chunk 1: `hello_world`

```
let msg = "Hello world.";;
print_string(msg); print_newline();;
```

```
val msg : string = "Hello world."
Hello world.
- : unit = ()
```

hello.tex Page 1/1

```
% This file was generated by Lepton. Copyright Li-Thiao-Te
 S. 06/2011
\documentclass[paper=a7]{scrartcl}
\usepackage[width=7cm,height=10cm]{geometry}
\input{lepton.sty}
\begin{document}
The code below sends "hello world" instructions to the \ve
rb ocaml  interpreter.
\begin{leptonfloat}
\caption{\flq\flq hello\_world\frq\frq}
\label{hello_world}
\begin{minted}[frame=single,fontsize=\footnotesize]{ocaml}
let msg = "Hello world.";;
print_string(msg); print_newline();;
\end{minted}
\begin{minted}[frame=single,fontsize=\footnotesize]{text}
val msg : string = "Hello world."
Hello world.
- : unit = ()
\end{minted}
\end{leptonfloat}
\end{document}
```

Figure 1: PDF file (left) and LaTeX source (right, rendered by a2ps) produced from hello.nw by Lepton.

2.3. Syntax

The syntax used in Lepton is inspired by the syntax of Noweb files [3] because of its simplicity. Lepton files are plain-text LaTeX files which may contain special blocks called *code chunks*. In Lepton, code chunks start with a chunk header of the form `<<header>>=` at the beginning of the line, and end with `@` at the beginning of the line. The chunk header is parsed as a blank separated command line. The first word is the chunk name. The following words are interpreted as chunk options. These control the output and interpretation of the chunk contents. Code chunks can appear in any order.

Code chunks can contain references that are written as `<<chunkname>>`. The chunk reference is replaced by the concatenation of all chunks with the same name. The amount of whitespace before the chunk reference is used to set the indentation level: it is prepended to all lines when expanding the reference.

Code chunks can contain other code chunks. This allows embedding of Lepton files inside other Lepton files, such as the `hello.nw` example in Section 2.1.

Two directives in LaTeX syntax are interpreted by Lepton. We define a `\Lexpr{interpreter}{code}` macro for direct inclusion of results in the LaTeX documentation. We also define a `\Linput{filename}` directive for including Lepton files and interpreting their contents. The Lepton manual is included in this document with `\Linput`.

2.4. Interpretation of code snippets

The contents of code chunks are interpreted as specified by the options in the chunk header:

- `-write -nowrite` : write the chunk contents to disk and use the chunk name as file name. Default: `-nowrite`,
- `-expand -noexpand` : expand chunk references in the documentation. Default: `-noexpand`,
- `-exec interpreter` : execute the chunk contents in an external interpreter. Default: `none`, i.e. do not execute,
- `-chunk format -output format` : indicate the format of the chunk contents and the chunk output for pretty-printing. By default, the format is verbatim text. Special values are `verbatim` (the output is formatted LaTeX code intended for direct inclusion) and `hide` (the output is not included in the produced `tex` file).

Lepton interprets the source file sequentially. For each chunk, the references are recursively expanded, then the chunk contents are optionally written to disk, and the chunk contents are optionally sent to the external interpreter. In particular, written files and definitions sent to an interpreter are available for the subsequent code chunks.

The `interpreter` specified with `-exec` or `\Linput` is a session or process name. If it corresponds to a process already open by Lepton, the process will be reused. Otherwise, the interpreter name is matched (by prefix) to a list of known intepreters and a new instance is launched. Lepton currently supports the UNIX shell, OCaml, Python, R and there is preliminary support for Matlab. Several sessions of the same process can be open concurrently, e.g. `shell1, shell2, shellbis`.

Other programming languages, notably compiled languages such as C/C++, can be used in Lepton by writing the source code to disk and using the `shell` interpreter to compile and execute the programs. To use a makefile, put the text into a chunk, write the chunk to disk and execute with `shell`.

Options that are set for a code chunk are propagated to the following chunks of the same name. `lepton_options` is a reserved chunk name for setting default options. For example, `<<lepton_options -write -chunk ocaml>>=` sets the default behavior to writing all chunk contents to disk, and formatting the chunk contents as OCaml code. The chunk contents are ignored.

2.5. LATEX format and pretty-printing

Lepton relies on LATEX for formatting the documentation. Lepton wraps the chunk contents and its output in a LATEX environment called `leptonfloat`, which is based on the `float` package (see Figure 1). Consequently,

- a caption is automatically included based on the chunk name,
- labels and indexes are automatically defined, the `hyperref` package can be used to link to chunk definitions,
- for each chunk reference, Lepton automatically adds a hyperlink to the corresponding chunk definition.

A list of all code chunks can be generated with `\lelistoflistings` and an index of code chunks with `makeidx`.

The chunk contents and the chunk output are independently formatted according to their respective options. For pretty-printing, we use the `minted` package in combination with the Python `Pygments` beautifier [4] to provide colorful syntax highlighting for many languages (See the rendered `hello.pdf` in Section 2.1). When not available, the `minted` environment can be replaced with another LATEX environment via a command-line option to Lepton.

The current version of Lepton includes preliminary HTML output support.

2.6. Current implementation and availability

The current implementation is written as a Lepton file with source code in the OCaml programming language. The Lepton code can be compiled to native code for speed on many architectures, and requires no external libraries.

Standalone binaries are available for GNU/Linux 32-bit and 64-bit platforms and can be downloaded from `http://www.math.univ-paris13.fr/~lithiao/Lepton.html`. For other platforms such as Windows, the mechanism for external command execution has not been ported yet.

3. Using Lepton for research

Lepton is a tool primarily intended for doing research, before publishing a polished method and its results. As an example, we consider the sequence alignment problem in bioinformatics. The amino-sequences of two proteins are given in the `input.fasta` file in FASTA format. We want to evaluate the similarity of the character patterns because this is often related with similarity in the biological function.

Code chunk 4: `<<input.fasta>>`

```
cat input.fasta
```

```
>gi|263348|gb|AAB24881.1| zinc finger [Homo sapiens]
YECNQCGKAFAQHSSLKCHYRTHIGEKPYECNQCGKAFSKHSHLQCHKRTHTGEKPYECNQCGKAFSQHGLLQRHKRTH
TGEKPYMNVINMVKPLHNS
>gi|263346|gb|AAB24880.1| zinc finger [Homo sapiens]
TGEKPFACKGCKKAFDQKITLIQHEGVHTGEKPYECRRCGSPSAGVETSLCIRSHTLKRHPFKHRASHYQAHYT
```

3.1. *Literate programming for method implementation*

Proposed by D.E. Knuth [2, 5], literate programming is about writing source code as a *work of literature* that can be read both by computers and by humans. This involves writing documentation with embedded code rather than writing code with comments, as well as tools to extract the source code from the documentation.

In the literate programming paradigm, source code and its documentation can be organised regardless of the constraints of the programming language. Source code can be split into meaningful chunks and can appear inside the document in any order. Chunk references encourage code modularity. Code chunks can be pretty-printed to increase readability, indexed and referenced throughout the document with hypertext links. Documentation may include formulae, tables and graphics instead of plain-text comments.

When programming in particular and in computational science in general, the researcher cares about the correctness of source code. Compilation instructions in a Lepton file ensures syntactic correctness. Semantic correctness can be checked by executing the compiled programs on test cases to make sure that they behave according to the specification. Lepton automates these steps by including the compiler and test outputs and ensures that the documented code corresponds to what was compiled and executed.

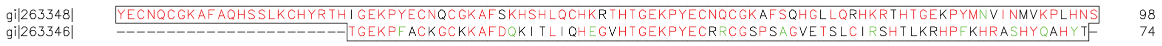
Due to size requirements, we will use ClustalW [6] instead of implementing a sequence alignment program in this manuscript. Source code and documentation are available at `www.clustal.org`.

3.2. *Report generation and analysis of results*

Similarly to Sweave [7], complete analyses of datasets can be written easily using Lepton. Code chunks can be used for including input data. Executable instructions and scripts can be directly seamlessly included in the LaTeX report; Lepton will automatically execute and embed the results. Code can be pretty-printed, hidden, or moved to the appendix. Output can be pretty-printed, hidden or included as legitimate LaTeX code.

Tables, plots, charts and other figures can be programmatically generated and inserted in the report. In particular, the instructions for generating these figures are available inside the documentation, next to the figure location. In the event of a parameter change, new data, or modifications to the analysis method, these instructions can be modified easily and the whole report will be updated. For example, in the following code chunk we compute the alignment, store the results in the file `output.aln` and render to PDF for inclusion in the current LaTeX article.

Code chunk 5: `<<draw_sequence_alignment>>`
```
clustalw -INFILE=input.fasta -OUTFILE=output.aln > /dev/null
prettyplot output.aln -residuesperline 100 -graph pdf
```



```
gi|263348|  YECNQCGKAFAQHSSLKCHYRTHIGEKPYECNQCGKAFSKHSHLQCHKRTHTGEKPYECNQCGKAFSQHGLLQRHKRTHTGEKPYMNVINMVKPLHNS  98
gi|263346|  -----------------------TGEKPFACKGCKKAFDQKITLIQHEGVHTGEKPYECRRCGSPSAGVETSLCIRSHTLKRHPFKHRASHYQAHYT-  74
```

Lepton supports collaborative research. As Lepton encourages researchers to write fully documented reports and provide the scripts for generating output, Lepton files automatically contain all the relevant information for discussing parameter values, method implementation, etc. and the file can be updated easily to follow the discussions. Being LaTeX files, Lepton files can be amended by several authors easily with any text editor.

3.3. *Simplicity, flexibility, universality*

Inspired by Noweb and Sweave, Lepton attempts to bring the best features from these tools to the user in terms of simplicity. Lepton's manual fits in Section 2, and defines only four syntax elements. Lepton only requires a single invocation on the command-line to extract the source code, execute the instructions and produce the documentation instead of two separate steps called tangling and weaving in Noweb. As an echo to D. E. Knuth in [5], we say that the process of "tangle, compile, load, and go" has been reduced to "lepton, and tex".

Lepton can be used with any programming language, and is compatible with all sorts of tools. Although it serves as a full documentation for source code, it can be used with documentation systems which expect comments and generate API documentation. Lepton files are plain text files that can be used in version control systems. In particular, the `diff` utility can be used to track changes in the code and changes in the documentation at the same time.

Lepton produces fully compatible LaTeX files and a specific LaTeX package is not mandatory. Consequently, Lepton files are compatible with all LaTeX packages and styles. The current manuscript is an example of using Lepton for writing a journal submission, but Lepton can also be used for thesis reports or slides for conference presentations. Large portions of text and code can be shared or reused between those documents.

Lepton can be used in many different contexts besides research. This tool was originally designed for literate C programming and writing technical reports for our research in computational image analysis. We also use Lepton when teaching undergraduate statistics courses, producing randomly generated test subjects as well as the corresponding solutions.

4. Using Lepton for reproducible research

Reproducibility corresponds to two distinct properties. We say that a paper is executable when the computer program can be used as a black box to produce identical outputs[1]. Research is *literate* when it can be applied to other datasets, with other parameters, modified or reimplemented to produce original work. Executability corresponds to reproducibility for the computer whereas literacy corresponds to the point of view of the human reader.

4.1. Executable papers

In UNIX, executable files starting with `#!` followed by the path to a command interpreter are treated as scripts and sent to the command interpreter for execution. In particular, Lepton files starting with `#!/usr/bin/lepton` are treated as system commands. With this mechanism, typing `./paper124.nw` in a terminal reads the input.fasta file, executes the instructions and generates the results `prettyplot.pdf` and the documentation in `paper124.tex`.

As Lepton files contain input data, source code and the instructions for using this source code, it suffices to execute them in a compatible environment. Many tools have been developped to specify software requirements. For example, configure scripts can indicate both the minimal requirements and the version of locally installed software. They are used during installation but also to allow programmers to reproduce the conditions leading to a software bug. Their output can be included in a Lepton file by running shell commands such as:

Code chunk 6: `<<shell>>`
```
uname -a
COLUMNS=90 dpkg -l gcc clustalw emboss | tail -n 5    # Debian package system

Linux laptop 3.2.0-2-686-pae #1 SMP Tue Mar 20 19:48:26 UTC 2012 i686 GNU/Linux
||/ Name            Version          Description
+++-===============-================-============================================
ii  clustalw        2.1+lgpl-2       global multiple nucleotide or peptide sequence a
ii  emboss          6.3.1-6+b1       the european molecular biology open software sui
ii  gcc             4:4.6.2-4        GNU C compiler
```

By specifying the software requirements, Lepton allows authors to choose their software tools, and readers to reproduce the results on their own hardware. Lepton programs may even include commands for configuring the computing environment. In cases where specific hardware or large amounts of computational resources are required, we suggest that the method be executed on a toy example for illustration, and that intermediate results be used to generate the tables and figures in the manuscript.

4.2. Literate papers

For a thorough understanding, Lepton provides readers with documentation, instructions for executing the programs and the means to easily modify the research method. Input datasets can be easily replaced, in particular when the data set is supplied as an external file such as input.fasta. Running Lepton again will automatically update the results.

As the instructions for running the programs are provided, they can be altered. Lepton files are plain text so code chunks can be easily edited to change the values of parameters, input and output file names, etc. Enabling and

[1]In fact, exact reproduction may not be possible in some cases like random number generation, differences in compiler rounding policy and number representation, etc.

disabling parts of the analysis workflow is possible by modifying the source code, or by toggling the execution of the whole chunk.

Most importantly, Lepton encourages the author to provide a complete documentation for its programs and methods. This ensures reproducibility and understandability. With the proper documentation, source code and intermediate results generated by Lepton are available and can be reused in other approaches. If needed, users should be able to reimplement the method from scratch.

5. Using Lepton for publishing research

5.1. Submission

Technical reports written with Lepton can serve as drafts for manuscript submission. By using the LaTeX format, Lepton files are compatible with many existing publication workflows. This manuscript was written with Lepton using the Elsevier LaTeX package and sent to the online submission system as usual. The Lepton file is included as supplementary material, along with the produced LaTeX and figure files.

Lepton files are self-contained. If the research paper can be executed by the submission system, then it is not necessary to transmit the source code, result files and figures. As discussed in Section 4, we suggest that input data should be included as external files and read as input to the executable Lepton paper.

Lepton files are rendered as traditional research papers, with high-quality typesetting, colors and figures. The Lepton syntax is flexible. Writers are free to organise their ideas and chunk references can be used to adapt the execution order to the flow of the document. Lepton files are compatible with publishers' style and guidelines. By using chunk references source code can be hidden or moved to the appendix without modifying the order of execution.

Although Lepton is currently focused on LaTeX documentation, the tool is designed to be independent of the documentation format. Only 20 lines of code are specific to LaTeX; these are mostly related to the encapsulation of code chunks and their output in a LaTeX environment, labels and references. Lepton incorporates a template system and preliminary support for HTML documents and WiKi documents; this will accomodate future publishing formats.

5.2. Peer review

As already discussed, properly written Lepton files contain all the required information for reproducing the results in a research paper. We suggest the following review procedure to examine the information contained in those files.

Reviewers should first attempt to execute the Lepton file in a blank environment. When encountering a problem, Lepton does not generate a complete LaTeX file; successfully applying Lepton ensures that the submission is complete and free of errors. The computing environment may optionally be restricted by the publisher to a standard list of software or the publisher may provide authors and reviewers with a pre-configured environment.

Then, reviewers should examine the Lepton log and the list of supplementary files provided. This quickly indicates which elements of the research paper are generated and which are pre-computed by the authors. In the current version, Lepton writes to the terminal the name of each code chunk and how it interprets it (write to disk, execute with interpreter, show or hide, see the example in Section 2.1). We can alternatively supply a modified version of Lepton that inserts this information inside the draft PDF document.

Finally, reviewers should check that the algorithm description matches the source code implementation. In the literate programming paradigm, this should be a fairly easy task when the code is properly documented. Reviewers should also check that to the best of their knowledge the published source code does not contain malicious code.

5.3. Publication

When the paper is accepted, the Lepton source file and the computing environment can be provided as supplementary material, so that readers can reproduce the results in the research paper. Lepton does not make a distinction between authors, reviewers and readers; all can expect the same level of functionality.

When publishing a research paper online, LaTeX can be converted to HTML with already existing tools such as `latex2html` (See the list at `http://www.tex.ac.uk/cgi-bin/texfaq2html?label=LaTeX2HTML`).

Lepton does not deal with licensing issues. To facilitate review and re-use, there is no access control mechanism; Lepton files are completely transparent. Nevertheless, authors can provide intermediate results instead of the original input data and compiled executables instead of source code. The Lepton file may be withheld depending on the editorial policy or licensing issues.

5.4. Long-term preservation

As discussed in Section 4, the publisher must preserve both the executability of research papers and their readability. Preserving executability is a technical issue related to the rapid evolution of hardware (new cpus) and software architectures (programming languages, library versions, file formats).

Nowadays, complete computing environments can be preserved with virtual machines. The hardware is emulated with near native speed, and software is stored in a special file format. Virtual machines can easily be cloned and transferred. Publishers could provide a set of standard virtual machines corresponding typical environments (Windows, MacOS, UNIX/LINUX) with a standard suite of software (including Lepton), add new machines to this set to follow software updates and preserve copies of these virtual machines in the long term.

Using virtual machines simplifies preservation of software and research papers because only the emulator needs to be maintained. However, there are currently several types of virtualization schemes, implemented as open source projects (Xen, KVM) or proprietary software (VMware, VirtualBox, . . .). This raises questions as to the long term maintenance of emulators and the compatibility of virtual machine file formats.

Programs preserved in a virtual machine are difficult to re-use because they reside in a separate environment. To communicate (send data, launch computations and retrieve results), one must go through the same procedures as working on a remote computer via a network connection. Consequently, such programs are difficult to include in other workflows or modify to run in a different computing environment.

To ensure the long term preservation of research, we believe that preserving the means to reimplement[2] is more important than preserving executability. These are a thorough documentation and a readable file format. Lepton uses literate programming to address the documentation issue. As to readability, Lepton favors LATEX which is a concise plain-text format and requires no specific software to read and edit contrary to PDF. We encourage authors to include input data as separate files so that the Lepton file remains small and easy to browse and edit in any text editor.

6. Comparison with existing software

6.1. Literate programming

The concept of "literate programming" was invented by D. E. Knuth in 1984 with the WEB program [2, 5], which he used to implement the TEX system. Knuth defined a vision of source code that is documented with the powerful capabilities of TEX and provided two programs to support this concept: `tangle` extracts source code and `weave` produces the documentation. Most current literate programming tools use the same tangle/weave approach with a different syntax, support for any programming language, and documentation format in LATEX or HTML.

In Lepton we have included the features that we find most appropriate for reproducible research. Instead of the tangle/weave approach in Noweb [3] and FunnelWeb [8], we prefer a single step approach similar to Nuweb [9] and Sweave [7] in which the Lepton file can be considered as a script itself. We use the simple syntax found in Noweb [3] with chunk options inspired from Sweave [7] for flexible control of the output format. In addition to the source code of literate programs, research reports must contain documented instructions for generating the program output, i.e. the analysis results. Sweave enables this for the R statistical software whereas Lepton can use any programming language and can combine several languages in the same document.

6.2. Executable papers

To reproduce the results in a research paper, the reader needs to know how the results where generated and in which conditions. Provenance-based approaches to reproducible research [10, 11] aim to systematically store how data, experiments and results were generated. These approaches guarantee only one successful execution of a research paper. In practice, results may be difficult to reproduce if the provenance information is incomplete — lacking a full description of the computing environment — or when figures are generated with different versions of the same software. Provenance information is stored in databases, and may be impossible to decipher without adequate software.

[2]Long-term preservation of research methods corresponds to portability, which is well-known to software engineers who are confronted with many different software architectures.

In contrast, executable papers make it possible to regenerate the results on demand, modify parameters and input data. The provided executable instructions indicate how to generate the results, and it suffices to describe the computing environment. Web servers (Collage [12], SHARE [13], R^2 [14], Paper Mâché [15], IPOL [16]) provide a convenient user interface to pre-configured computing ressources but usually restrict the computing environment to a limited set of software. Authors can provide complete, ready-to-use computing environments with virtual machines [13, 15]. In Lepton and [14, 17, 18], the research paper can be executed on a wider range of environments, including the local machine, but all the dependencies must be installed by the reader. Note that all executable papers require a computation engine and this can be preserved inside a virtual machine.

6.3. Access to ressources

It is well-accepted that reproducible papers should include input data, source code, and a description of the computing environment and how the results were generated; the considered frameworks differ in the level of access to these ressources. Papers that can be installed locally (Lepton and [14, 17, 18]) provide the best level of access. Although the computing environment needs to be set up, local papers with bundled input data are unaffected by broken URLs or network connectivity. Moreover, computing ressources provided by publishers are limited in cpu time and dataset size. As discussed in Section 5.4, papers encapsulated in virtual machines are difficult to integrate into new approaches.

However, software and data sets that are protected by licenses cannot be readily distributed with research papers. Collage, [12] and SHARE [13] implement some level of digital rights and user management; these features can be complemented by the publisher. As indicated in Section 5.3, Lepton is designed without licensing features so that readers can produce results in the same conditions as the authors of the research paper.

6.4. Review and reuse

In comparison with other tools, Lepton is focused on documentation and re-usability of the published material. Lepton is designed for work-in-progress research that should be reviewed and improved rather than polished approaches ready for publication. Consequently, fiddling with the contents of a research paper by authors and readers is unobstructed by the publication system. In contrast, web-based approaches are restricted by the user interface. Temporary results generated during method development can clutter provenance-based system.

In web-based frameworks, the format of the research paper is restricted. For example, results must appear inside frames in Collage. In Lepton and Sweave, results can appear anywhere in the documentation, and parts of the documentation can be programmatically generated. Other types of documents can be generated such as Beamer presentations.

Depending on the documentation format some elements of interactivity can be included in reproducible papers. The HTML format can provide interactive visualizations and execution of code snippets in [12, 11]. In Lepton and Sweave, the research paper is a static PDF file which ensures that all the results coincide with the presented source code. Lepton depends on LaTeX only for formatting code chunks and can be easily extended for HTML output and interactive visualization via browser plugins.

Literate research papers document the methods to access the ressources provided with the manuscript as opposed to "code-data dumps". To ensure that source code is properly documented and reviewed, we suggest that it should be included in the main publication medium, in the appendix or as a full publication. Additionally, executable instructions provide the instructions needed to re-use the provided input data sets, including those in proprietary formats.

As indicated in Section 5.4, long-term reproducibility is dependent on the quality and readability of the documentation so that the research method can be reimplemented. In web-based systems and specific file formats such as HDF [17], research papers may become unreadable when the corresponding software ceases to be maintained. When even a PDF reader or an HTML browser is no longer available, two formats are ultimately readable: Lepton uses plain text, and [18] uses natural language.

7. Conclusion

Lepton is a powerful tool for producing technical reports, research manuscripts as well as other types of documents. All the elements in a project can be embedded in the same Lepton file: input data, source code, executable instructions and the documentation for all of these. By enabling command execution in external interpreters, parts of the document can be programmatically generated and Lepton files can be turned into self-contained executable programs.

Lepton takes its inspiration from literate programming and suggests to use the tools and approaches developped in software engineering for code review and re-use. As such, Lepton can be used with any programming language and any documentation format. With its unobtrusive syntax, Lepton can be quickly adopted and deployed in a wide variety of environments. Nevertheless, the task of writing thorough documentation remains the author's responsibility.

To meet the demands of reproducible research in computational science, we propose to write research papers as literate executable programs and provide Lepton as the tool to implement this approach. This piece of software is stable, extensible and will be able to accomodate the interactive visualization of research results as well as future publication systems and formats.

References

[1] S. Li-Thiao-Té, Lepton User Manual.
URL http://www.math.univ-paris13.fr/~lithiao/ResearchLepton/Lepton.html

[2] D. E. Knuth, Literate programming, THE COMPUTER JOURNAL 27 (1984) 97–111.

[3] N. Ramsey, Literate programming simplified, Software, IEEE 11 (5) (1994) 97–105.

[4] G. Brandl, T. Hatch, A. Ronacher, Pygments.
URL http://pygments.org/

[5] D. Knuth, S. U. C. S. Dept, Literate programming, Center for the Study of Language and Information, 1992.

[6] M. Larkin, G. Blackshields, N. Brown, R. Chenna, P. McGettigan, H. McWilliam, F. Valentin, I. Wallace, A. Wilm, R. Lopez, et al., Clustal w and clustal x version 2.0, Bioinformatics 23 (21) (2007) 2947–2948.

[7] F. Leisch, Sweave: Dynamic generation of statistical reports using literate data analysis, in: W. Härdle, B. Rönz (Eds.), Compstat 2002 — Proceedings in Computational Statistics, Physica Verlag, Heidelberg, 2002, pp. 575–580, iSBN 3-7908-1517-9.
URL http://www.stat.uni-muenchen.de/~leisch/Sweave

[8] R. Williams, et al., FunnelWeb user's manual (1992).
URL http://www.ross.net/funnelweb/reference/index.html

[9] P. Briggs, J. D. Ramsdell, M. W. Mengel, S. Wright, K. Harwood, Nuweb Version 1.57 A Simple Literate Programming Tool.

[10] M. Gavish, D. Donoho, A universal identifier for computational results, Procedia Computer Science 4 (0) (2011) 637 – 647, proceedings of the International Conference on Computational Science, ICCS 2011. doi:10.1016/j.procs.2011.04.067.
URL http://www.sciencedirect.com/science/article/pii/S1877050911001256

[11] D. Koop, E. Santos, P. Mates, H. T. Vo, P. Bonnet, B. Bauer, B. Surer, M. Troyer, D. N. Williams, J. E. Tohline, J. Freire, C. T. Silva, A provenance-based infrastructure to support the life cycle of executable papers, Procedia Computer Science 4 (0) (2011) 648 – 657, proceedings of the International Conference on Computational Science, ICCS 2011. doi:10.1016/j.procs.2011.04.068.
URL http://www.sciencedirect.com/science/article/pii/S1877050911001268

[12] P. Nowakowski, E. Ciepiela, D. Harezlak, J. Kocot, M. Kasztelnik, T. Bartyński, J. Meizner, G. Dyk, M. Malawski, The collage authoring environment, Procedia Computer Science 4 (0) (2011) 608 – 617, proceedings of the International Conference on Computational Science, ICCS 2011. doi:10.1016/j.procs.2011.04.064.
URL http://www.sciencedirect.com/science/article/pii/S1877050911001220

[13] P. V. Gorp, S. Mazanek, Share: a web portal for creating and sharing executable research papers, Procedia Computer Science 4 (0) (2011) 589 – 597, proceedings of the International Conference on Computational Science, ICCS 2011. doi:10.1016/j.procs.2011.04.062.
URL http://www.sciencedirect.com/science/article/pii/S1877050911001207

[14] F. Leisch, M. Eugster, T. Hothorn, Executable papers for the r community: The r2 platform for reproducible research, Procedia Computer Science 4 (0) (2011) 618 – 626, proceedings of the International Conference on Computational Science, ICCS 2011. doi:10.1016/j.procs.2011.04.065.
URL http://www.sciencedirect.com/science/article/pii/S1877050911001232

[15] G. R. Brammer, R. W. Crosby, S. J. Matthews, T. L. Williams, Paper mâché: Creating dynamic reproducible science, Procedia Computer Science 4 (0) (2011) 658 – 667, proceedings of the International Conference on Computational Science, ICCS 2011. doi:10.1016/j.procs.2011.04.069.
URL http://www.sciencedirect.com/science/article/pii/S187705091100127X

[16] N. Limare, J.-M. Morel, The ipol initiative: Publishing and testing algorithms on line for reproducible research in image processing, Procedia Computer Science 4 (0) (2011) 716 – 725, proceedings of the International Conference on Computational Science, ICCS 2011. doi:10.1016/j.procs.2011.04.075.
URL http://www.sciencedirect.com/science/article/pii/S1877050911001335

[17] K. Hinsen, A data and code model for reproducible research and executable papers, Procedia Computer Science 4 (0) (2011) 579 – 588, proceedings of the International Conference on Computational Science, ICCS 2011. doi:10.1016/j.procs.2011.04.061.
URL http://www.sciencedirect.com/science/article/pii/S1877050911001190

[18] S. M. Veres, J. P. Adolfsson, A natural language programming solution for executable papers, Procedia Computer Science 4 (0) (2011) 678 – 687, proceedings of the International Conference on Computational Science, ICCS 2011. doi:10.1016/j.procs.2011.04.071.
URL http://www.sciencedirect.com/science/article/pii/S1877050911001293