On using HEVEA, a fast LaTeX to HTML translator

Luc Maranget

1. Introduction

HEVEA is a LaTeX to HTML translator. This article mostly intends to describe how to use HEVEA in practice. As everyone who actually tried to translate a reasonably complex LaTeX document to HTML knows, such a task is seldom automatic. A successful translation often requires to configure the translator or to instruct it about particular LaTeX constructs.

Section 2 explains my views on the difficulty to translate LaTeX into HTML and the basic principles of HEVEA design. The rest of the paper is devoted to describing the translation of various documents, including the quite involved task of altering a package.

HEVEA is available at `http://pauillac.inria.fr/~maranget/hevea/`.

2. Dream usage

Most users who discover a LaTeX to HTML translator such as HEVEA have the following expectation : getting a HTML-version of some document `doc.tex`, should be as simple as typing “`hevea doc.tex`”, provided “`latex doc.tex`” already works. And indeed, much of the work invested in the development of HEVEA aim at fulfilling this expectation. For instance, the design of HEVEA is much inspired by the design of LaTeX and, more anecdotally, the `hevea` command acts much like the `latex` as regards file searching or the usage of auxiliary files.

However, HEVEA is not a clone of LaTeX targeted to output HTML and we now explain why.

2.1. What is LaTeX ?

There is no such thing as a definition of LaTeX, there is no even such a thing as a reference implementation of LaTeX. Nevertheless there exists a `latex` program, but this program does not define LaTeX, it processes TeX, with additional constructs. Hence, full compatibility at the reference implementation level with LaTeX means producing a clone of `tex`, or adapting `tex` so that it produces HTML (the latter approach is the one of [1]).

Furthermore, on the one hand, LaTeX can be viewed as a mostly specifying language, telling more about the structure of a document, than about how it should be printed on sheets of paper. On the other hand, HTML is also a mostly specification language, also telling more about the structure of a document, than about how it should be displayed on a screen. As a matter of fact, the viewer's browser actually takes part to the formatting of the document. The browser performs the low-level part of the job.

Note that viewing both LaTeX and HTML as defining the structure of a document is wrong in my opinion. Authors can legitimately claim *some* control on the final aspect of their work, either printed or displayed, how much control is debatable. I would rather view both languages as high-level specification of how the final document should be rendered, which of course meets the document "structure". High-level formatting means being aware of the final medium, while ignoring the peculiarities of various medium implementations. A construct to format a body of text using two columns is an example of such a high-level specification. Numerous mandatory page break (in LaTeX) or lengths expressed in centimeters (in HTML) are counter-examples.

Provided that we accept LaTeX and HTML as being mostly specification languages implementing a LaTeX to HTML translator is by far less involved than reimplementing `tex`. For instance, while coping with (ordinary) tables and arrays, any LaTeX to HTML translator does little else than translating one specification into another, and this is quite easy. The same applies to most LaTeX environments that naturally translate into HTML block level elements, to LaTeX sectioning commands that naturally translate into `H` elements, etc. Additionally, we trade TeX exotic lexing conventions for more simple ones and this is not a small benefit.

To conclude, we intentionally give up the idea of handling the full TeX language and, instead, confine our attention to the LaTeX subset. We shall soon see how we define this subset.

2.2. Different media

Paper and web pages are different. None of the media is more powerful than the other, they are just different things.

The weaknesses of web-page formatting are met while generating HTML: HTML is not powerful enough to render some LaTeX constructs. Those limitations sometimes are arbitrary. For instance, we can render ordinary superscripting such as `$x^2$` ($x^2$) by `<I>x</I><SUP>2</SUP>`. But we cannot properly render limit-like superscripts or "stacked" symbols in the middle of a line — consider `$\stackrel{\star}{\rightarrow}$` ($\stackrel{\star}{\rightarrow}$). By contrast HEVEA handles such "stacked" symbols in displays, see Section 4.2.

This limitation of inline superscripts in HTML is particularly annoying in the case of vectors (`$\vec{u}$`, $\vec{u}$). There is no universally acceptable way to translate the sentence "Let $\vec{u}$ be a vector" into HTML. Instead of choosing a poor rendering of inline vectors (and other math accents such as `\hat`, `\overline` etc.), HEVEA does not even attempt to translate them. Instead, it issues a warning. Then, users can choose then own construct to replace vectors[1].

However, web pages have their own strengths, and hypertext links certainly are amongst them, But there are others, for instance, color changes are more acceptable on screen than on paper. Hence, for a particular document it may be acceptable to translate vectors by color changes.

The example of vectors shows that there cannot be a complete LaTeX to HTML translator. Or, if there is one, it chooses some rendering of a problematic construct, where users would have preferred another.

2.3. Guidelines in developing HEVEA

By the previous discussion, a "provably" compatible LaTeX to HTML translator is a reimplementation of `tex`. Moreover, such a compatible translator may not exist and not be what we want... Of course, compatibility is not be be neglected but focusing on it is not the most productive attitude. While developing HEVEA we focused on the following points

1. From the beginning, we have done our best effort to implement acceptable practice in LaTeX. We define such a practice by a liberal reading of Leslie Lamport's' Blue book [6], enlightened by the LaTeX companion [2]. Those books certainly have great impact on users practice. Moreover, the latter intend to promote a standard practice in writing LaTeX, pointing out packages whose commands have a clean LaTeX interface and are valid substitute for TeX-isms, as the calc package for instance. However implementing TeX-isms is rewarding, and we indulge in it from time to time, without granting full TeX compatibility.

[1] In the case of vectors a suggestion is made: loading the mathaccent package, which provides a default, quite unsatisfactory, rendering of vectors ($\vec{u}$) by superscripted arrows ($u^{\rightarrow}$).

2. More recently, implementing packages became the privileged direction for enlarging the scope of HEVEA, as a satisfactory effort against benefit trade-off .

3. HEVEA provides some constructs to control browser-like display. The acceptability of those by `latex` is a minor issue. Here, we may be wrong, given the emergence of pdf which exhibits hypertext capabilities.

4. Overall, we have considered compatibility at a higher abstraction level. The commands of LaTeX make it a highly configurable tool, we have insisted on giving users ways to configure `hevea` by writing code that tastes like LaTeX. Hence, implementing LaTeX commands and environments, and most of TeX macros, in a faithful and predictable manner is one of our major design choices.

3. HEVEA in practice

The rest of this paper describes the production of HTML versions of documents written independently by 13 authors (including me).

The documents describe programming projects, they are written in French and they size up from 2 to 9 pages. Authors provided one LaTeX source file and companion files, such as Postscript images or packages believed to be "non-standard". In practice only one author shipped an obsolete version of the algorithm package. They were required to do so, since the documents were to be gathered into a booklet.

Authors are computer scientists, they routinely use LaTeX. However they work in different areas and obviously have different habits. For instance, two authors gave document in old LaTeX (or 2.09) style, the other authors use LaTeX 2_ε.

Out of these 13 documents, three documents translated without any problem. However, problems with the remaining 10 documents were minor, and were solved in a few minutes and (except in one case) without altering the authors source. As a matter of fact, all documents load a specific projetX package, mostly intended to define page size. Definitions written to alter or complement the behaviour of HEVEA can be regrouped in a `projetX.hva` file, which will be loaded by `hevea`, where `latex` loads the `projetX.sty` file. Those definition are written in the language understood by HEVEA, which is, well, a dialect of LaTeX. At the end of the translation process, the `projetX.hva` file was made of 41 line or 25 definitions.

4. Warnings

It is worth noticing that all problems were unveiled thanks to HEVEA warnings. Warnings include a source line number which enables to spot quickly the problematic construct. This method is by far more productive than spotting problems from visual examination of browser display. The most common warning is the "`Command not found`" warning, HEVEA cannot possess its own version of every existing LaTeX command. They are various reasons :

1. Some constructs may not posses any clear equivalent in HTML. Then the warning draw users attention onto the problem and users are thus invited to choose a rendering.
2. Some command just are TeX. Such warnings sometimes reveal a difficult problem. Fortunately, most LaTeX users use true TeX code in very limited areas.
3. Some commands are defined in packages which `hevea` does not implement.
4. Some commands may have been forgotten.

There are others, more specific, warnings. For instance, HEVEA cannot implement negative lengths and thus warns users about them.

4.1. Simple "`Command not found`" warning

As an example of minor problematic rendering, one author used the symbol `\odot` ($\odot$). But HEVEA does not provides command `\odot`. In fact, HEVEA provides far less symbols than LaTeX, since it knows about two character sets only : iso-latin1 and a set of about 256 mathematical symbols[2] known as "the symbol font". An easy solution is to replace $\odot$ by a visually similar symbol, which `hevea` can produce. Here, I chose $\otimes$. Then it suffices to insert the following definition in the `projetX.hva` file :

```
\newcommand{\odot}{\otimes}
```

Other examples of similar harmless unknown commands, are `\varepsilon` (ε), defined as `\epsilon` (ϵ) and `\ell` (ℓ) defined as the letter "l".

[2] According to HTML 4.0 definition [7] more symbols are possible, and they can be specified without the dubious `<FACE=symbol>...</FONT>` element. However not all browsers are able to display the whole variety of symbols defined as "HTML entities"

As an example of chunks of TEX code inside a document, one document defines fonts the TEX way in its preamble. This can be solved using the image facility (see Section 5 below).

Case (3) (non-existent package) occurred in two occasions. We examine the first (and easiest) case. One document loads the fguill package. This small package defines two commands `\guillemotleft` and `\guillemotright` (« and »). A clean solution is to implement the fguill package in HEVEA. That is, we need to create a `fguill.hva` file that defines both commands and to put this file somewhere in `hevea` search path, since `hevea` reacts to `\usepackage{`*pgk*`}` by searching the file *pgk*`.hva`. Here, given our french computer, the `fguill.hva` file simply contains:

```
\newcommand{\guillemotleft}{«}
\newcommand{\guillemotright}{»}
```

Of course, this solution relies on iso-latin1 being privileged both as input encoding and output encoding.

As an example some command that HEVEA should have known, another document uses the `\to` command (→), which I did not know about. Apparently, this command is defined in the TEX book [5] as an equivalent of the `\rightarrow` command. HEVEA should probably be aware of this. Here writing `\let\to\righarrow` in the `projetX.hva` file solves the problem. More, this definition is now included into HEVEA main configuration file.

4.2. Mathematics

The documents include some mathematics. In sharp contrast with TEX, HTML was not designed for typesetting mathematics and one cannot expect a satisfactory rendering of every formula. However, HEVEA usually does a decent job.

One author is a researcher in complexity theory and its For instance one of the documents is written by a researcher in complexity theory and he is not frightened by mathematics. His document includes various formulas, amongst which we choose the following two :

```
\[
V=\int_0^{\infty}\frac{\sqrt x\,\ln^5x}{(1-x)^5}\,{\rm d}x
\]
```

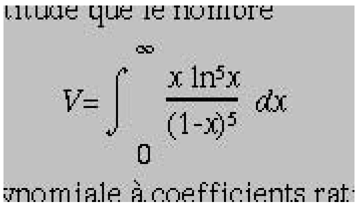
and

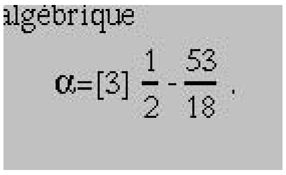
```
\[
\alpha=\sqrt[3]{\frac12-\frac{5\sqrt3}{18}}.
\]
```

LATEX typesets these formulas as follows:

$$V = \int_0^\infty \frac{\sqrt{x}\,\ln^5 x}{(1-x)^5}\,\mathrm{d}x \qquad \alpha = \sqrt[3]{\frac{1}{2} - \frac{5\sqrt{3}}{18}}.$$

A first run of HEVEA produces a warning: "`Command not found: \sqrt`". And indeed, HEVEA has no `\sqrt` command, since "*big square roots*" are difficult to render in HTML[3]. The warning cannot be ignored since there is currently no rendering of roots:





The first formula is simply wrong. Additionally, the second formula is rather strange, since the optional argument `[3]` appears in output.

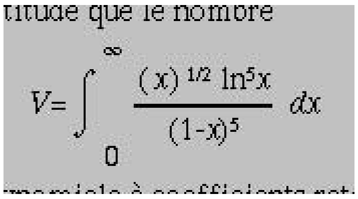
We now attempt a definition of roots as fractional exponents, by adding a definition into the `projetX.hva` file.

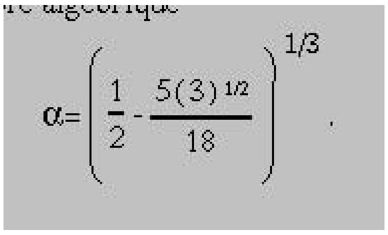
```
\newcommand{\sqrt}[2][2]{\left(#2\right)^{1/#1}}
```

HEVEA now has a definition for `\sqrt`, as a command that takes two arguments, the first of which being optional with default value "2". That is, we adopt the same interface as the `\sqrt` command of LATEX. The command body includes constructs that are meaningfully to HEVEA: big delimiters and exponents. Any person with some knowledge of LATEX can design such a replacement definition. Rendering is now understandable, if not satisfactory[4].

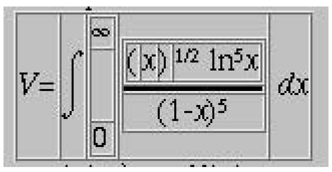
[3] However the LATEX to HTML translator TTH [3] handles root signs, which HEVEA does not.

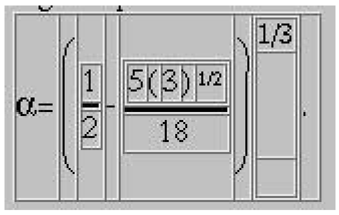
[4] Ironically, one of the motivations of TEX design was D.E. Knuth frustration with roots being replaced by fractional exponents by pre-TEX typesetting systems.





Explaining HeVeA internals in detail would be out of scope, telling a little about them does not harm. To typeset mathematics, HeVeA mostly relies on two techniques: symbols are build from the limited set of glyphs offered by the symbol font, and nested HTML tables are extensively used to express formula structure. Here, the vertical parenthesis are made of two vertical stacks of five glyphs, and the following picture illustrate how tables are organized.





Although HeVeA successfully handles the previous formulas, it cannot translate any TeX mathematics. Apart from missing symbols, HeVeA suffers from two, more severe, limitations:

— Inline (as opposed to display) mathematics are a real problem, since inserting a `TABLE` element necessarily produces a line break in displayed HTML. Fortunately, complicated inline mathematics are quite rare, and authors generally have complicated formulas displayed (*i.e.* they use `$$` or `\[`... `\]`).

— When it comes to typesetting mathematics TeX is really very powerful. HeVeA is by nature less sophisticated, at some point it just gives up. For instance, TeX box dimensions are characterized by three quantities (height, width and depth or baseline vertical position). In the context of formulas, the HeVeA analogs to TeX boxes are HTML tables and HeVeA only handles one approximative quantity: the number of rows in a table. This means that HeVeA is not the right tool to process mathematical texts — but HTML neither is the right tool to display them. In case a document includes complicated formulas, they should be translated into images, as explained in the next section.

5. The image file feature

Sometimes, HEVEA just cannot process its input, but it remains acceptable to have part of the input processed by LaTeX and then to replace such input in the HTML output by included GIF (or PNG) images.

HEVEA provides some support for doing this. Any text enclosed in the special `toimage` environment is echoed into the *image* file (which is a LaTeX source file). Additionally he special command `\imageflush` outputs a strict page break into the image file, while it outputs the appropriate `IMG` element into the generated HTML file. Some constructs of the source document are echoed to the image file without user intervention. This includes the `\documentclass` command and the `\usepackage` commands with all their arguments. Then, a later run of the companion `imagen` script produces one image per page in the image file. The `imagen` script first calls `latex` on the image file, and then a variety of image processing tools.

5.1. Fancy symbols as images

HEVEA image feature can be used to replace problematic symbols (which we would like not to change) by small images.

For instance, one author loads the `amssymb` package and then defines command `\Z` as :

```
\newcommand{\Z}{\mathbb{Z}}
```

The symbol $\mathbb{Z}$ (`\Z`) stands for the set of relative integers, following french conventions. As students are familiar with this symbol, I decided not to change it. Notice that the image file includes `\usepackage{amssymb}` and thus, the command `\mathbb` can be used inside in it. Hence, a solution is to define `\Z` in the `projetX.hva` file.

```
\newcommand{\Z}{\begin{toimage}$\mathbb{Z}$\end{toimage}\imageflush
```

After a run of `hevea` on this documemt `transpos.tex`, the image file `transpos.image.tex` is as follows :

```
\documentclass{article}
\usepackage{projetX}
\usepackage{amssymb}
\pagestyle{empty}
```

```
\thispagestyle{empty}
\begin{document}
$\mathbb{Z}$
\clearpage% page: 0
$\mathbb{Z}$
\clearpage% page: 1
\end{document}
```

Notice that there are two invocations of `\Z` in this document. In HTML output, the two occurrences of `\Z` are replaced by `<IMG SRC="transpos001.gif">` and `<IMG SRC="transpos002.gif">`. The images themselves are produced by issuing the command `imagen transpos`. However, when given the command-line option `-fix`, the `hevea` command will call `imagen` automatically.

The author's source can be left as it is. This results from HEVEA semantics for `\newcommand` : if the defined command already exists, then `hevea` does not fail, as `latex` would. Instead, `hevea` issues a warning and does nothing. Here, the `hevea` command loads the `projetX.hva` file before processing the author's definition. Hence, `hevea` definition for `\Z` is the one from the `projetX.hva` file.

The `\Z` command story does not stop here, another author defined it for the same purpose, but in a different way, by short-circuiting LaTeX font selection scheme.

```
\font\twelvesym=msbm10 at 12pt\font\tensym=msbm10\font\sevensym=msbm7
\font\fivesym=msbm5
\newfam\symfam
\textfont\symfam=\tensym\scriptfont\symfam=\sevensym\scriptscriptfont
\symfam=\fivesym
\def\sym{\fam\symfam\tensym}
\def\Z{{\sym Z}}
```

The solution is exactly the same as in the previous case: send all that source into the image file. But here we cannot avoid altering author's document. Furthermore, the document should remain processable by LaTeX. One solution is bracket previous definitions into special comments:

```
%BEGIN IMAGE
\font\twelvesym=msbm10 at 12pt\font\tensym=msbm10\font\sevensym=msbm7
 ...
\def\Z{{\sym Z}}
%END IMAGE
```

HEVEA handles the special comments as `\begin{toimage}`... `\end{toimage}`, while LaTeX ignore them (since they are comments!). As a consequence the image file now includes the above definition of `\Z`. Then, it suffices to adopt the following definition for `\Z` in the `projetX.hva` file:

```
\newcommand{\Z}{\begin{toimage}\Z\end{toimage}\imageflush}
```

This definition only appears absurd, it is not. It is intended for the consumption of HEVEA and occurrences of `\Z` result in outputting the following two lines into the image file.

```
\Z
\clearpage
```

The first line results from the interpretation of `\begin{toimage}\Z\end{toimage}` while the second line results from the interpretation of `\imageflush`.

As they stand, the two solutions for the `\Z` command problem are not compatible, since we now have two conflicting definitions for `\Z` in the `projetX.hva` file. In practice we adopted a different solution, which we describe at the end of the next section.

5.2. Included images

Many authors shipped images with their document. Authors use various commands to include their images. For instance the document `solide.tex` uses the `\epsfbox` command from the `epsf` package. To translate these Postscript images into GIF images automatically, it suffices to define `\epsfbox` in the `projetX.hva` file.

```
\newcommand{\epsfbox}[1]
{\begin{toimage}\epsfbox{#1}\end{toimage}\imageflush}
```

Observe that this example is more involved than the previous one, since the parameter `#1` needs to be substituted. As a consequence of this substitution, the `solide.image.tex` file includes the following lines:

```
\epsfbox{solide.eps}
\clearpage% page: 0
\epsfbox{deplacement.eps}
\clearpage% page: 1
```

Some authors are attentive readers of the LaTeX "*reference*" books [6, 2], those authors use the `\includegraphics` command from the graphics or graphicx packages. HEVEA implements these packages, with definitions similar to the one we just saw.

It is worth noticing that we finally solved the the problem of command `\Z` in a general way by defining command `\mathbb` as follows:

```
\newcommand{\mathbb}[1]
{\begin{toimage}$\mathbb{#1}$\end{toimage}\imageflush}
```

6. A real difficulty

The one problem that frightened me the most was the absence of an HEVEA implementation of the algorithm package.

One author shipped us a quite obsolete version of some algorithm package, as an `algorithm.sty` file and used it to typeset a rather lengthy algorithm:

```
\begin{algorithm}{LLL$(b_1,b_2,\ldots,b_n)$}
\\ $b_1^*$\=$b_1$, $B_1$\=$<b_1^*,b_1^*>$
\\ \For $i$\=$2$ \To $n$ \Do
   \> \\ $b_i^*$\=$b_i$
      \\ \For $j$\=$1$ \To $i-1$ \Do
      \> \\ $\mu_{i,j}$\=$<b_i,b_j^*>/B_j$,
            $b_i^*$\=$b_i^*-\mu_{i,j}b_j^*$
      \<
      \\ $B_i$\=$<b_i^*,b_i^*>$
 ...
\end{algorithm}
```

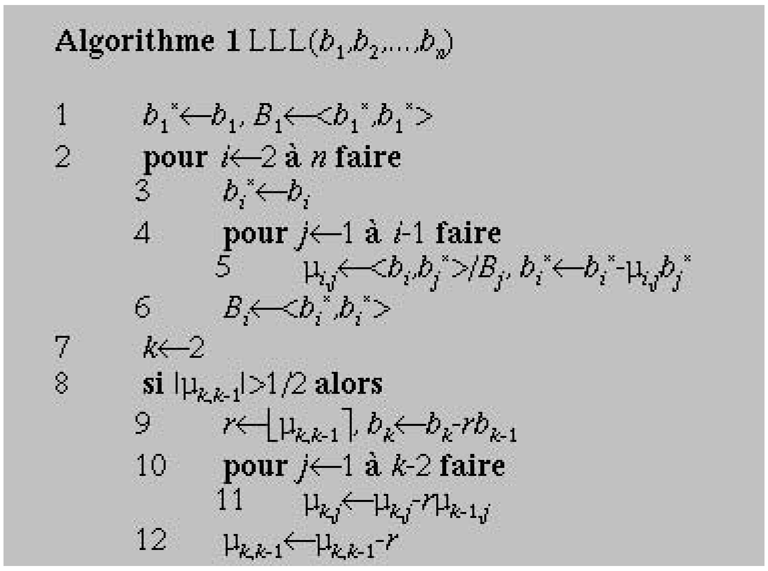
The key commands are `\\`, which starts a new line, `\>`, which increases indentation, and `\<` which decreases indentation. This can be confirmed by looking at `latex` output, then one also discovers line numbers.

An easy solution would of course have been to insert `\begin{toimage}`, `\end{toimage}` and `\imageflush` somewhere. But, in some sense, this is giving up and I was ready for a slightly more involved solution. Thus, I started writing an `algorithm.hva` file. My first attempt was rather minimal.

```
\input{algorithm.sty}
```

Doing so, I hoped that the algorithm package was written using more LaTeX than TeX. Then I ran `hevea`, without even looking at `algorithm.sty`. To my surprise, `hevea` did not crash and there was not even a single warning. Unfortunately, the output (see figure 1) was *almost* right. Everything looks

Figure 1: A first attempt of adapting algorithm for HEVEA.



```
Algorithme 1 LLL($b_1$,$b_2$,...,$b_n$)

1       $b_1^* \leftarrow b_1$, $B_1 \leftarrow <b_1^*,b_1^*>$
2       pour $i \leftarrow 2$ à $n$ faire
        3       $b_i^* \leftarrow b_i$
        4       pour $j \leftarrow 1$ à $i$-1 faire
                5       $\mu_{i,j} \leftarrow <b_i,b_j^*>/B_j$, $b_i^* \leftarrow b_i^* - \mu_{i,j} b_j^*$
        6       $B_i \leftarrow <b_i^*,b_i^*>$
7       $k \leftarrow 2$
8       si $|\mu_{k,k-1}|>1/2$ alors
        9       $r \leftarrow \lfloor \mu_{k,k-1} \rceil$, $b_k \leftarrow b_k - r b_{k-1}$
        10      pour $j \leftarrow 1$ à $k$-2 faire
                11      $\mu_{k,j} \leftarrow \mu_{k,j} - r\mu_{k-1,j}$
        12      $\mu_{k,k-1} \leftarrow \mu_{k,k-1} - r$
```

fine except line numbers, which should remain on the left instead of sticking to algorithm lines.

Now, we have to look at `algorithm.sty` in order to understand where line numbers are produced. One quickly finds an `algorithmline` LaTeX style counter (defined with `\newcounter`) and the following command:

```
\def\instr@{\refstepcounter{algorithmline}%
\item[{\algonumberstyle\thealgorithmline}\hfill]}
```

Obviously, the `\instr@` command outputs the line numbers and the `algorithm` environment must be some kind of `list` environment.

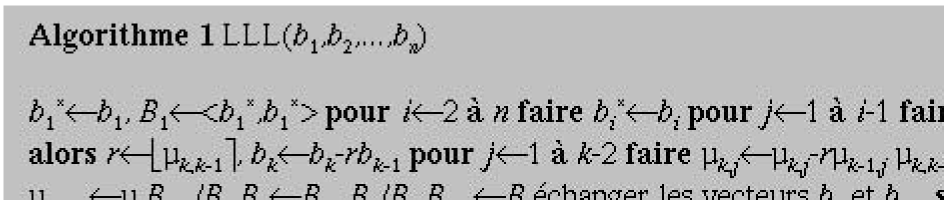
In fact, such `list` environments are nested in algorithm presentation. The `\>` command starts a new `list` environment with augmented `\labelspace` (space between label and item), while the `\<` closes it and restore `\labelstep` to its previous value (the indentation value is kept into a length register and such registers are global). Moreover `\instr@` is some internal name for `\\` (*i.e.* there is a definition `\let\\=\instr@` somewhere). As a consequence `\\` increases and then typesets the line number, then LaTeX inserts some `\labelspace` space, the value of `\labelspace` being controlled by the nesting of `list` environment. All this explains the above browser rendering, since HEVEA translates `list` environments into `DL` (description list) elements, ignoring `\labelspace`. Furthermore, my browser systematically indents nested `DL` elements.

The package is cleanly written, with internal names for all commands. which makes it easier to change the behavior of some commands by redefining them after the package is loaded. First I decide to get rid of `list` environments, because of the systematic indentation introduced by browsers. This can be done by redefining the `Blo@ck` environment, which is the internal version of `\>` and `\<`, the former being defined in `algorithm.sty` as `\begin{Blo@ck}` and the latter as `\end{Blo@ck}`. The `algorithm.hva` file now is:

```
\input{algorithm.sty}
\renewenvironment{Blo@ck}{}{}
```

A test run of `hevea` now gives a lot of "`\item outside a list-making environment`" warnings and line breaks and line numbers have disappeared (see figure 2).

Figure 2: Suppressing `list` environments.



Algorithme 1 LLL(b_1,b_2,...,b_n)

b_1^*←b_1, B_1←<b_1^*,b_1^*> **pour** i←2 **à** n **faire** b_i^*←b_i **pour** j←1 **à** i-1 **fai**

alors r←⌊$\mu_{k,k-1}$⌉, b_k←b_k-rb_{k-1} **pour** j←1 **à** k-2 **faire** $\mu_{k,j}$←$\mu_{k,j}$-$r\mu_{k-1,j}$ $\mu_{k,k-}$

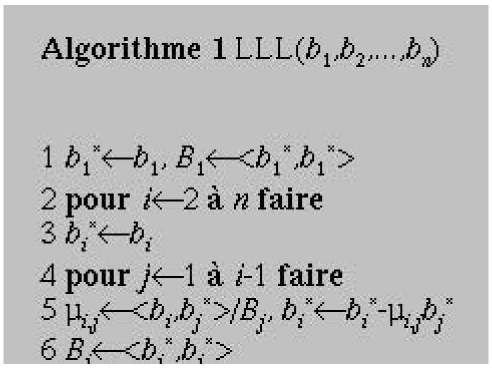
μ ←μ B /B B ←B B /B B ←B échanger les vecteurs b et b

One easily restores them with the following redefinition of `\instr@`.

```
\renewcommand{\instr@}{\@br%
\refstepcounter{algorithmline}{\algonumberstyle\thealgorithmline}}
```

Command `\@br` is one of HEVEA internal commands: it outputs a `<BR>` tag, which browsers interpret as a line break. Now, output (see figure 3) is *almost* perfect, except for indentation which is missing.

Figure 3: Restoring line breaks and line numbers.



Algorithme 1 LLL(b_1,b_2,...,b_n)

1 b_1^*←b_1, B_1←<b_1^*,b_1^*>
2 **pour** i←2 **à** n **faire**
3 b_i^*←b_i
4 **pour** j←1 **à** i-1 **faire**
5 $\mu_{i,j}$←<b_i,b_j^*>/B_j, b_i^*←b_i^*-$\mu_{i,j}b_j^*$
6 B_i←<b_i^*,b_i^*>

Restoring the indentation is more involved. HEVEA does not implement length registers, but it features counters. A `block@depth` counter is introduced, and the `Blo@ck` environment now keep track of its nesting level.

```
\newcounter{block@depth}
\renewenvironment{Blo@ck}
  {\stepcounter{block@depth}}
  {\addtocounter{block@depth}{-1}}
```

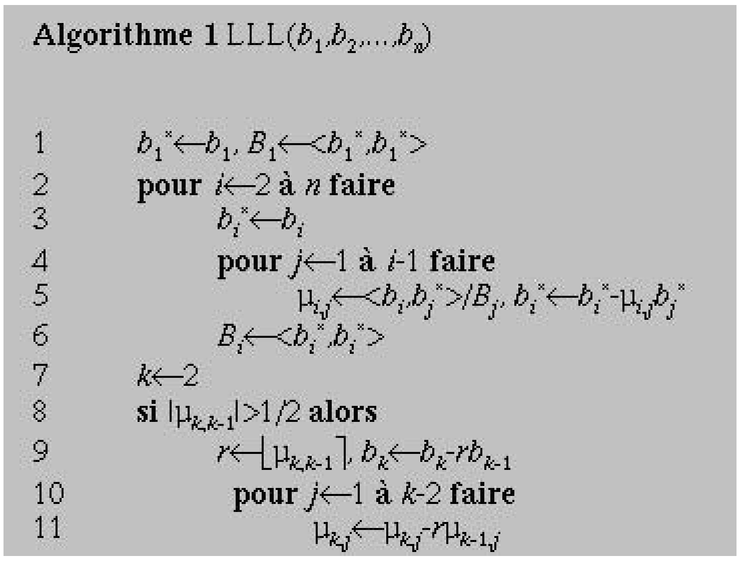
It remains, given an integer value d, to output some space quantity d times. Let us first assume that such a command `\do@indent` exists, then we have our final implementation of `\instr@`.

```
\renewcommand{\instr@}{\@br%
\refstepcounter{algorithmline}{\algonumberstyle\thealgorithmline}%
\@doindent{\value{block@depth}}}
```

To write `\@doindent`, I use another counter and the `\whiledo` command from the ifthen package.

```
\usepackage{ifthen}%optional, hevea loads ifthen by default
\newcounter{algo@}
\newcommand{\algo@indent}{\hspace{10ex}}
\newcommand{\@doindent}[1]
{\setcounter{algo@}{#1}%
\algo@indent%
\whiledo{\value{algo@} > 0}
  {\algo@indent\addtocounter{algo@}{-1}}}
```

Figure 4: Final rendering of the algorithm.



```
Algorithme 1 LLL($b_1$,$b_2$,...,$b_n$)

1     $b_1^* \leftarrow b_1$, $B_1 \leftarrow <b_1^*,b_1^*>$
2     pour $i \leftarrow 2$ à $n$ faire
3         $b_i^* \leftarrow b_i$
4         pour $j \leftarrow 1$ à $i$-1 faire
5             $\mu_{i,j} \leftarrow <b_i,b_j^*>/B_j$, $b_i^* \leftarrow b_i^* - \mu_{i,j} b_j^*$
6         $B_i \leftarrow <b_i^*,b_i^*>$
7     $k \leftarrow 2$
8     si $|\mu_{k,k-1}|>1/2$ alors
9         $r \leftarrow \lfloor \mu_{k,k-1} \rceil$, $b_k \leftarrow b_k - r b_{k-1}$
10         pour $j \leftarrow 1$ à $k$-2 faire
11             $\mu_{k,j} \leftarrow \mu_{k,j} - r \mu_{k-1,j}$
```

Rendering (see figure 4) is almost perfect, except for line numbers which should be right-justified, a minor problem. Overall, I was extremely lucky, implementing packages for HEVEA usually is more complicated. Here, the package performs a rather simple task, and above all it is written in LATEX that HEVEA understands. As a benefit of the approach of slightly altering the original package, observe that we did not need to worry about other commands from this package, such as the `algorithm` environment itself (it shows a number and a title) and the various keywords (*e.g.* `\For`, which gets translated to **pour**).

7. Writing a paper and a screen version at the same time

I authored one document and was aware that it was to be processed by HEVEA. This enabled me to perform a few actions to make this translation easier. The first action is to load the hevea package:

```
\usepackage{hevea}
```

The hevea package is a LaTeX package (HEVEA ignores `\usepackage{hevea}`), it provides definitions for constructs which HEVEA is aware of by default.

7.1. Commands for hypertext links

The hevea package provides a innocuous definitions for the `toimage` environment and for the `\imageflush` command. But it also features a LaTeXversion of high-level hypertext commands. Figure 5 describes the most significant such commands, with HEVEA and LaTeX behavior.

Figure 5: High-level hypertext commands

Macro	HEVEA	LaTeX
`\ahref{`*url*`}{`*text*`}`	make *text* an hyperlink to *url*	echo *text*
`\footahref{`*url*`}{`*text*`}`	make *text* an hyperlink to *url*	make *url* a footnote to *text*, *url* is shown in typewriter font
`\ahrefurl{`*url*`}`	make *url* an hyperlink to *url*.	typeset *url* in typewriter font
`\ahrefloc{`*label*`}{`*text*`}`	make *text* an hyperlink to *label* inside the document	echo *text*
`\aname{`*label*`}{`*text*`}`	make *text* an hyperlink target with label *label*	echo *text*
`\mailto{`*address*`}`	make *address* a "mailto" link to *address*	typeset *address* in typewriter font

As a first example, defined the author as follows:

```
\author{Luc Maranget\footnote{\mailto{Luc.Maranget@inria.fr}}}
```

As a consequence, both HTML and paper versions of my document include a footnote with my email address, furthermore the footnote is clickable.

Since I encouraged students to contact me by giving them my email address in a footnote, I also wished to collect my answers to their questions on another web page (in French, *La page de suivi*), and I wanted all students to be aware of this page. This is a perfect job for the `\footahref` command:

```
Important, il existe une
    \emph{page de \footahref{\base/suivi.html}{suivi}}.
```

The word *suivi* gets decorated with a footnote in the LaTeX version and with an hyperlink in the HTML version. The command `\base` is defined somewhere else as the URL where my text will finally be. It can be a good idea to define it as "`.`" for HEVEA and as an absolute URL for LaTeX.

I also intended to make both versions of the document to reference the other. Here I need to have different texts in both versions. To that end, I used the TeX style `\ifhevea` command, which HEVEA sees as true and LaTeX sees as false (provided of course the hevea package is loaded).

```
\begin{center}\large
\ifhevea
Cet inonci en \ahref{\base/anagramme.ps}{Postscript}.
\else
La page~web de cet inonci est disponible
      en \ahrefurl{\base/anagramme.html}.
\fi
\end{center}
```

Finally, the document has a small bibliography, including a reference to a paper which is down-loadable. Here, the explicit URL should appear in both versions and I used the `\ahrefurl` command.

```
\bibitem{tst} Jon Bentley and Bob Segdgewick,
     ``\emph{Fast Algorithms
for Sorting and Searching}'',
Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete
Algorithms, Juanary 1997.
\ahrefurl{\url{http://www.cs.princeton.edu/~rs/strings/}}.
```

Notice that the argument to `\ahrefurl` is first processed by the `\url` command from the url package. For our purposes we can see `\url` command as echoing its argument verbatim, as LATEX `\verb` does. Indeed the URL includes the active character `~`, which is not to be interpreted with its usual meaning of non-breakable space. That way, inserting the link and typesetting the URL remains two separate tasks. It requires a bit of typing from users, but they can understand more easily what happens in case of problem. Furthermore, writing the LATEX version of `\ahrefurl` is trivial, which is not the case of `\url` whose implementation has been performed by someone else, who arguably knows TEX much better than I do.

7.2. Anticipated usage of the image feature

My document includes a PIC image. Such an image is described in a specific language [4], and can then be translated into TEX by a specific `gpic` (Unix) command. For instance I have an image `dico.pic` and I translate it into TEX by issuing the (Unix) command:

```
# gpic -t < dico.pic > dico.tex
```

Just after `tex` has processed the source included in the `dico.tex` file, the image is present in the box `\graph` and hence can be put somewhere by `\box\graph`.

My code for the inserting the `dico` image reads as follows:

```
\begin{gpic}\input{dico.tex}\end{gpic}
```

That is, I follow the practice of hiding gory details by a clean LATEX interface, here an environment. The LATEX definition of the `gpic` environment resides in some local `gpic.sty` file:

```
\newenvironment{gpic}{\begin{center}}{~\box\graph~\end{center}}
```

The HEVEA definition of the `gpic` environment resides in some local `gpic.hva` file:

```
\newenvironment{gpic}
  {\begin{toimage}}
  {\box\graph\end{toimage}\begin{center}\imageflush\end{center}}
```

My document includes the line `\usepackage{gpic}`, so that both `latex` and `hevea` find the proper definition.

It is worth noticing that in HEVEA case, `\imageflush` appears centered, since this is where the final link to the GIF image is inserted. Whether the `\box\graph` is centered or not is irrelevant, since this is food for the `latex` run `imagen`, which later crops all margins.

8. Conclusion

I hope that the few examples described in this paper are enough to convince the readers that HEVEA is worth a try. Above all, I hope that it will help HEVEA users to appreciate HEVEA verbose reaction in front of source code it cannot translate. I would like them to consider such numerous warnings more as an assistance than as a nuisance.

Bibliography

[1] E. Gurari, *TeX4ht: LaTeX and TeX for Hypertext.* Software and documentation, `http://www.cis.ohio-state.edu/~gurari/TeX4ht/`.

[2] M. Gooseens, F. Mittelbach, A. Samarin. *The LaTeX Companion* Addison-Websley, 1994.

[3] I. Hutchinson. *TTH, the TeX to HTML translator.* Software and documentation, `http://hutchinson.belmont.ma.us/tth/`

[4] B.W. Kernighan. *PIC – A Graphics Language for Type•setting (User Manual).* AT&T Bell Laboratories, Computing Science Technical Report No. 116. `http://cm.bell-labs.com/cm/cs/cstr/116.ps.gz`, revised May, 1991.

[5] D.E. Knuth. *The TeXbook.* Addison-Websley, 1984.

[6] L. Lamport. *A Document Preparation System System, LaTeX, User's Guide and Reference Manual.* Addison-Websley, 1994.

[7] D. Ragget, A. Le Hors and I. Jacobs. *HTML 4.0 Reference Specification.* `http://www.w3.org/TR/REC-html40`, 1997.