Axessibility 2.0: creating tagged PDF documents with accessible formulae

Axessibility 2.0: creating tagged PDF documents with accessible formulae

D. Ahmetovic, T. Armano, C. Bernareggi, A. Capietto, S. Coriasco, B. Doubrov, A. Kozlovskiy, N.

Sommario

I documenti PDF contenenti formule generati da LaTeX non sono solitamente accessibili mediante tecnologie assistive per persone con disabilità visive (i.e., screen reader e display Braille). Il pacchetto LaTeX `axessibility.sty` da noi sviluppato risolve questo problema, permettendo di creare documenti PDF in cui le formule vengono lette da tali tecnologie assistive, tramite l'inserimento di commenti nascosti. In questo articolo descriviamo l'evoluzione del pacchetto, che ora genera automaticamente anche il tagging delle formule. Il pacchetto non genera PDF/UA.

Abstract

PDF documents containing formulae generated by LaTeX are usually not accessible by assistive technologies for visually impaired people (i.e., by screen readers and Braille displays). The LaTeX package `axessibility.sty` that we developed manages this issue, allowing to create PDF documents where the formulae are read by such assistive technologies, through the insertion of hidden comments. In this paper we describe the evolution of the package, that now automatically generates also the tagging of the formulae. The package does not generate PDF/UA.

1 Introduction

PDF documents are widely used to digitally publish scientific content, such as papers or textbooks. Mathematical formulae, frequently contained within such documents, are not accessible by screen reader users because they are commonly rendered as bi-dimensional images. The burden of making digital documents accessible is often left to the document author, who needs to provide descriptions for each visual content in the form of alternate text. This procedure is time consuming, error-prone and it needs to be done by a sighted person. Additionally, in the case of mathematical formulae, a verbal description does not provide the same information as the original mathematical notation. In many cases no alternate text is even provided because authors are not aware of the accessibility needs of screen reader users.

In this paper, we show the features of the package `axessibility.sty` (whose a first version is also described in **?**) that provides the first method for an automatized production of accessible PDF documents with mathematical contents through LaTeX. We would like to highlight that this package does not produce fully tagged PDF, such as the standard PDF/UA, but it allows to obtain a PDF where formulae are marked and described using the `/Alt` and `/ActualText` attributes.

2 Related Work

Assistive technologies for people with visual impairments (e.g., screen readers, Braille displays, magnifiers) are used effectively and proficiently to read and edit digital documents containing structured text. Instead, still many accessibility issues remain for what concerns documents including mathematical formulae and images (e.g., diagrams, graphs, technical drawings) **?**, **?**. A number of studies have been conducted to improve non-visual access to scientific content, mainly along two research lines: to facilitate editing of scientific documents through non-visual tools, and to enable people with sight impairments to read scientific documents in digital formats.

The former research work has led to different multimodal systems that are now available to author scientific documents through non-visual tools. For instance, the LAMBDA editor **?** is used mostly by blind people to write and process text and mathematical formulae through Braille display and speech output. This sytem adopts a sequential code to represent mathematical notation, specifically designed for blind people and usable only in this editor. Hence, it has got widespread only among some communities of blind people and it cannot become a mainstream tool to produce accessible scientific content by sighted people, too. A different approach consists in editing LaTeX documents through speech and Braille support **?**, **?**, **?**, **?**, **?**, **?**. This approach has the advantage to rely on LaTeX, which is a de facto standard for authoring scientific documents. Unfortunately, since these tools are produced for a small community, due to the rapid evolution of technology, they often incur in maintainance and compliance issues.

For what concerns reading digital scientific documents, many studies have been undertaken to create non-visual reading tools for the most widespread digital formats. In particular, research has

focused on web publishing Microsoft Word, LaTeX and PDF documents. In recent years, mathematical content has been published on the web through images of formulae, by embedding MathML in the web page or through MathJax, a JavaScript display engine for mathematical formulae. Images of formulae are inaccessible to screen readers, hence they can be adapted to be read by screen readers only through a proper alternative text (e.g., the LaTeX equivalent). On the contrary, MathML and MathJax can be used to create accessible web pages. MathML, especially the content markup, can be interpreted by most common screen readers to generate a verbal description of the formula **?**, **?**. Moreover, MathPlayer, a web browser plugin for rendering MathML on the screen, through speech output and on Braille devices, enables hierarchical navigation of mathematical formulae, including bi-dimensional notations such as matrices **?**. MathJax can be embedded in web pages making available adaptable accessibility features for representing and navigating formulae (e.g., LaTeX, ASCIIMath or CSS representation) **?**, **?**. Taking Microsoft Word into account, mathematical formulae can be read by the speech synthesizer or on a Braille display through MathPlayer. Nonetheless, due to the visual features of Microsoft Word, interaction with screen readers is often not easy. LaTeX documents can be read by people with sight impairments either reading the source file on the Braille display or through editors that support speech reading of LaTeX (e.g., ChattyInfty by Science Access Net) **?**, **?**, **?**, **?**. Furthermore, also converters from LaTeX to some national Braille codes for mathematics are available **?**. Since national Braille codes can represent only a limited amount of mathematical notations, these converters can transform only a subset of the source LaTeX document.

For PDF files, frequently used as a medium for publishing digital scientific documents, the accessibility of mathematical content has been developed in the scope of the so-called Tagged PDF, which embeds the document semantics directly into the visual representation of the page. Both ISO 32000-1:2008 (specifying PDF 1.7) and the recent ISO 32000-2:2017 (for PDF 2.0) suggest the use of MathML syntax for describing the semantics of mathematical formulae. In addition, PDF 2.0 standard opens the door for any alternative syntax (for example, the original LaTeX representation of the formula), which can be associated with any structure element in Tagged PDF. However, due to the novelty of this approach, it is not yet supported by the screen readers and, thus, may be considered only in the long-term scope.

Another approach widely supported by the majority of the screen readers is to add accessibility features to mathematical content as alternate text. It can be specified manually using, for example, a proprietary editor such as Adobe Acrobat. Guidelines have been produced to create accessible PDF according to this procedure **?** with a focus on mathematical content **?**, **?** and **?**.

However, this approach requires the availability of a suitable editor, and it entails additional labor from the document author. Furthermore, alternate text most often does not carry the same semantic value as the original mathematical content. Yet another approach consists in transforming PDF files into LaTeX or HTML+MathML documents by performing OCR **?**, **?**. However, the resulting document has to be proofread because of possible recognition errors. Proofreading process is usually time consuming and it has to be done by a sighted person who can compare the PDF document with the OCR result.

3 The `axessibility` LaTeX package

We provided a solution to the problem described above through our package `axessibility`, see, e.g., **???**. In its most recent version, release 2.0, which will soon be available in CTAN, we employed the `tagpdf` package, created by Ulrike Fisher, replacing the `accsupp` package, on which the 1.x versions of `axessibility` package relied. The package implements insertion of the original LaTeX formulae as properties of the Span elements containing visual representation of the mathematical content in the resulting PDF document, by means of the commands provided by the `tagpdf` package.

In more detail, each inline or display formula in the source LaTeX document is wrapped into a marked content sequence (see the documentation of the `tagpdf` package for more details on the difference between structure elements and marked content sequences in Tagged PDF). In addition, the original formula is added to this marked content sequence as `/ActualText` and `/AltText`. These properties are read by screen readers and braille displays instead of the `ASCII` representation of the formula, which is often incorrect. Additionally, the package adds a minimal Tagged PDF structure to the output PDF. This includes at the moment the top level Document structure element to mark the beginning and the end of the document and the P (paragraph) tag for each formula. Further extension of this set of tags (like automatic tagging of all paragraphs, section headers, etc) is still a work in progress.

As the `tagpdf` package, the `axessibility` 2.0 package is currently experimental and it is aimed for individual tests and experiments.

3.1 Usage

To create an accessible PDF document for visually impaired people, the authors just need to include the `axessibility` package into the preamble of their LaTeX project. The supported mathematical

environments will then automatically produce the `/ActualText` and `/AltText` contents and include them in the produced PDF file. Formulae will also be automatically tagged, as well as the `document` environment. The tagging of other text tokens (paragraphs, sections, etc.), at the moment, has to be inserted manually, under the guidelines of the `tagpdf` package.

The environments for writing formulae which are presently supported are `\(`, `\[`, `equation*`, `equation`, `align*`, and `align`. Hence, any formula inserted using one of these environments is accessible and tagged in the corresponding PDF document. The click-copy of the formula LaTeX code from the PDF reader, to be pasted elsewhere, is presently not working with this new release.

Inline and displayed mathematical modes activated by the old sintaxes $...$ and $$...$$ are not supported by the `axessibility` package (as in the previous versions). However, external scripts provided as companion software can address, at some extent, the problem of source files where the old TeX sintax is used (see Section **??** below).

Below, an example of LaTeX code, illustrating the usage of `axessibility`, jointly with `tagpdf`.

```
\documentclass{article}
\usepackage{etoolbox,axessibility}

\begin{document}

\tagstructbegin{tag=P}
  \tagmcbegin{tag=P}
        A simple displayed formula:
  \tagmcend
\tagstructend

\begin{equation*}
x=\frac{3a^2}{n+m}
\end{equation*}

\tagstructbegin{tag=P}
  \tagmcbegin{tag=P}
        A multiline formula, aligned,
            with label:
  \tagmcend
\tagstructend
\begin{align}
70xy^2+105x^2y-35xy7
& = 35\left(2xy^2+3x^2y-xy7\right) =
    \\
& = 35x\left(2y^2+3xy-y7\right) =
    \\
& = 35xy\left(2y+3x-7\right)
\end{align}

\end{document}
```

We observe that, in these cases, the author can write the formulae without adding anything else. Moreover, inside the source code of the PDF file, we find `/ActualText` and `/AltText` contents, with the (Hex) LaTeX code inside, automatically generated by the `axessibility.sty` package, as well as the equation tags, namely:

```
/P
<</MCID 1
/Alt <FEFF002000200078003D005C
      006600720061006300200078
      00330061005E0032007D007B
      006E002B006D007D0020>
/ActualText <FEFF002000200078003D005C
             006600720061006300200078
             00330061005E0032007D007B
             006E002B006D007D0020>
>>
```

and

```
/P
<</MCID 3
/Alt <FEFF00370030007800790 05E
       0032002B0031003000350078
       005E00320079002D00330035
       0078007900370020002600 20
       003D002000330035005C006C
       0065006600740020002800 32
       00780079005E0032002B0033
       0078005E00320079002D0078
       00790037005C007200690067
       0068007400200029002000 3D
       0020005C005C002000260020
       003D002000330035007800 5C
       006C006500660074002000 28
       00320079005E0032002B0033
       00780079002D00790037005C
       0072006900670068007400 20
       00290020003D0020005C005C
       002000260020003D00200033
       003500780079005C006C0065
       0066007400200028003200 79
       002B00330078002D0037005C
       0072006900670068007400 20
       0029>
/ActualText <FEFF00370030007800790 05E
       0032002B0031003000350078
       005E00320079002D00330035
       0078007900370020002600 20
       003D002000330035005C006C
       0065006600740020002800 32
       00780079005E0032002B0033
       0078005E00320079002D0078
       00790037005C007200690067
       0068007400200029002000 3D
       0020005C005C002000260020
       003D002000330035007800 5C
       006C006500660074002000 28
       00320079005E0032002B0033
       00780079002D00790037005C
       0072006900670068007400 20
       00290020003D0020005C005C
       002000260020003D00200033
       003500780079005C006C0065
       0066007400200028003200 79
       002B00330078002D0037005C
       0072006900670068007400 20
       0029>
>>
```

respectively. Here the `/Alt` and `/ActualText` keys are followed by the UTF-16 encoded values in the Hexadecimal format. So, this makes our solution fully Unicode compliant.

We note that such use of `/Alt` and `/ActualText` keys is not fully aligned with the best practices of PDF accessibility techniques. But it does open the door for real world tests and further experiments. In particular, the screen reader will read correctly the LaTeX commands. Moreover, the JAWS and NVDA dictionaries that we created provide the reading in the natural language, in the case that the user does not know the LaTeX commands. It is strongly recommended to use the most recent version of `tagpdf` (available through the *GitHub* website), as well as the most updated versions of the TexLive distribution.

3.2 Technical Overview

In `axessibility` we first load the requested packages, configure `tagpdf`, and define a pair of internal variables.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{axessibility}

\RequirePackage{tagpdf}
\tagpdfsetup{tabsorder=structure,
    uncompress,activate-all,
    interwordspace=true}
\tagpdfifpdftexT
 {
\pdfcompresslevel=0
  %set language / can also be done
      with hyperref
  \pdfcatalog{/Lang (en-US)}
  \usepackage[T1]{fontenc}
  \input glyphtounicode
  \pdfgentounicode=1
 }
\tagpdfifluatexT
 {
  %set language / can also be done
      with hyperref
  \pdfextension catalog{/Lang (en-US)}
  \RequirePackage{fontspec}
  \RequirePackage{luacode}
  \newfontface\zerowidthfont{freeserif
      }
\directlua{
pdf.setcompresslevel(0)
pdf.setmajorversion(2)
pdf.setminorversion(0)
}
}

\RequirePackage{amsmath}
\RequirePackage{amssymb}
\RequirePackage{xstring}

\newtoks\@mltext
\newtoks\@mltexttmp
```

Then, we redefine the `document` environment, so that the PDF file is automatically tagged at the Document level.

```
\makeatletter
\let\begin@document=\document
\let\end@document=\enddocument
\renewcommand{\document}{\
    begin@document\tagstructbegin{tag=
    Document}}
\renewcommand{\enddocument}{\
    tagstructend\end@document}
\makeatother
```

Subsequently, we redefine the inline formula environment, to make it accesible, inserting its (hidden) LaTeX code. We also define an internal command to produce a space (which is useful in passing parameters to some of our redefined environments).

```
\makeatletter
\newenvironment{temp@env}{%
  \relax\ifmmode\@badmath\else$\fi%
 \collect@body\wrap}{%
  \relax\ifmmode\ifinner$\else\
      @badmath\fi\else \@badmath\fi}
\protected\def\(#1\){\begin{temp@env
    }#1\end{temp@env}}
\makeatother

\newcommand{\auxiliaryspace}{ }
```

The core of the package is represented by the wrapping procedures. The first one, `\wrap`, is used for both the inline, as well as the displayed single line, formulae environments (numbered and unnumbered), which we redefine, to obtain their automatic tagging and insertion of the corresponding LaTeX code in the `/ActualText` and `/AltText` contents. The wrapper receives as parameter the code within the environment, obtained by means of the `\collect@body` command (from the `amsmath` package), and passes it to the tagging commands defined in `tagpdf`.

```
\makeatletter
\long\def\wrap#1{
\tagstructbegin{tag=P,alttext-o=\
    detokenize\expandafter{#1},
    actualtext-o=\detokenize\
    expandafter{#1}}
 \tagmcbegin{tag=P,alttext-o=\
     detokenize\expandafter{#1},
     actualtext-o=\detokenize\
     expandafter{#1}}
 #1
 \tagmcend
\tagstructend
}
\makeatother

\makeatletter
\renewenvironment{equation}{%
 \incr@eqnum
  \mathdisplay@push
  \st@rredfalse \global\@eqnswtrue
```

```
 \mathdisplay{equation}%
  \collect@body\wrap\auxiliaryspace}{%
  \endmathdisplay{equation}%
  \mathdisplay@pop
  \ignorespacesafterend
}
\makeatother

\makeatletter
\renewenvironment{equation*}{%
  \mathdisplay@push
  \st@rredtrue \global\@eqnswfalse
  \mathdisplay{equation*}%
  \collect@body\wrap\auxiliaryspace}{%
  \endmathdisplay{equation*}%
  \mathdisplay@pop
  \ignorespacesafterend
}
\makeatother

\makeatletter
\protected\def\[#1\]{\begin{equation
    *}#1\end{equation*}}
\makeatother
```

The next two, `\wrapml` and `\wrapmlstar`, perform the same task for the multiline environments. We need a different routine here, due to the more involved typesetting procedure of multiline environments like `align` and `align*`, which are likewise redefined.

```
\makeatletter
\long\def\wrapml#1{
\def\@mltext{\detokenize\expandafter
    {#1}}
\def\@mltexttmp{}
\StrBehind[6]{\@mltext}{ }[\@mltexttmp
    ]
\StrGobbleRight{\@mltexttmp}{1}[\
    @mltext]
\tagstructbegin{tag=P,alttext-o=\
    detokenize\expandafter{\@mltext},
    actualtext-o=\detokenize\
    expandafter{\@mltext}}
 \tagmcbegin{tag=P,alttext-o=\
     detokenize\expandafter{\@mltext},
     actualtext-o=\detokenize\
     expandafter{\@mltext}}
 #1
}
\makeatother

\makeatletter
\long\def\wrapmlstar#1{
\def\@mltext{\detokenize\expandafter
    {#1}}
\def\@mltexttmp{}
\StrBehind[5]{\@mltext}{ }[\@mltexttmp
    ]
\StrGobbleRight{\@mltexttmp}{1}[\
    @mltext]
\tagstructbegin{tag=P,alttext-o=\
    detokenize\expandafter{\@mltext},
    actualtext-o=\detokenize\
    expandafter{\@mltext}}
 \tagmcbegin{tag=P,alttext-o=\
     detokenize\expandafter{\@mltext},
     actualtext-o=\detokenize\
     expandafter{\@mltext}}
 #1
}
\makeatother

\makeatletter
\renewenvironment{align}{%
  \collect@body\wrapml\auxiliaryspace
 \start@align\@ne\st@rredfalse\m@ne
}{%
  \math@cr \black@\totwidth@
  \egroup
 \ifingather@
    \restorealignstate@
    \egroup
    \nonumber
    \ifnum0=`{\fi\iffalse}\fi
  \else
    $$%
  \fi
  \ignorespacesafterend
  \tagmcend
  \tagstructend
}

\renewenvironment{align*}{%
  \collect@body\wrapmlstar\
      auxiliaryspace
  \start@align\@ne\st@rredtrue\m@ne
}{%
  \endalign
}

\makeatother

\endinput
```

We are presently working to make `\wrapml` and `\wrapmlstar` more flexible, so that they will work correctly with all the other multiline environments provided by the `amsmath` package. This will make all of them accessible and tagged, as those illustrated above. At the moment, the package works correctly when typesetting with both `pdfLaTeX` as well as `luaLaTeX`.

4 Supporting Software

In addition to the *Axessibility* package, we developed additional software to address two use cases: 1) Preprocessing Scripts for the application of *Axessibility* on existing documents, and 2) Screen Reader Dictionaries for natural language reading of formulae made accessible with *Axessibility*. We are currently working on these supporting software, to fix some of the issues we detected through user's reports and suggestions, and to expand their applicability range.

4.1 Preprocessing Scripts

Axessibility restricts the syntax that can be used to write mathematical formulae to specific environments and math mode syntax. Instead, existing documents may contain unsupported syntax, and therefore cannot be used with *Axessibility* without being first opportunely edited. We provide *Axesscleaner*, an external script written in Python and Perl, through which it is possible to substitute unsupported commands and environments with suitable replacements, thus enabling the use of *Axessibility* on existing LaTeX documents.

An additional issue lies in the usage of user-defined macros in the LaTeX code. While this is a common practice to avoid code repetitions and simplify document authoring, it can limit the accessibility of formulae with *Axessibility*. Indeed, *Axessibility* is transparent to commands used in math environments, which means that it will include standard LaTeX as well as custom macros within the PDF replacement text. However, custom commands used by an author may bear no meaning for other readers. Thus, *Axesscleaner* also replaces user defined macros with their content, in order to only contain standard LaTeX code within the PDF replacement text.

4.2 Screen reader dictionaries

Mathematical formulae included as PDF replacement text using *Axessibility* are easy to read by LaTeX proficient users, using either a screen reader or a braille display. However, for novice users, the LaTeX code read by a screen reader may be difficult to comprehend.

To address this problem, we also provide dictionaries for *NVDA* and *JAWS* screen readers, which convert LaTeX commands contained within the PDF replacement text created by *Axessibility* into their natural language counterparts (*e.g.*, '\frac{2}{3}' becomes "two thirds"). We are currently developing additional screen reader scripts to enable interactive navigation of formulae, and we are exploring more sophisticated natural language processing techniques to personalize formula reading considering their complexity and context, as well as user's proficiency with math.

5 Acknowledgements

The authors wish to thank the several volunteers with visual impairment who provided their fundamental contribution.

▷ D. Ahmetovic
Dipartimento di Informatica, Università degli Studi di Milano
dragan.ahmetovic@unito.it

▷ T. Armano
Dipartimento di Matematica "G. Peano", Università degli Studi di Torino
tiziana.armano@unito.it

▷ C. Bernareggi
Dipartimento di Informatica, Università di Milano
cristian.bernareggi@unimi.it

▷ A. Capietto
Dipartimento di Matematica "G. Peano", Università degli Studi di Torino
anna.capietto@unito.it

▷ S. Coriasco
Dipartimento di Matematica "G. Peano", Università degli Studi di Torino
sandro.coriasco@unito.it

▷ B. Doubrov
Dual Lab, Belgium
boris.doubrov@duallab.com

▷ A. Kozlovskiy
Dual Lab Bel, Belarus
k.sasha1994@gmail.com

▷ N. Murru
Dipartimento di Matematica "G. Peano", Università degli Studi di Torino
nadir.murru@unito.it