Extracting Interlinear Glossed Text from LaTeX Documents

Mathias Schenner, Sebastian Nordhoff
Language Science Press, Freie Universität Berlin, Habelschwerdter Allee 45, 14195 Berlin
mathias.schenner@langsci-press.org, sebastian.nordhoff@langsci-press.org

Abstract

We present `texigt`, a command-line tool for the extraction of structured linguistic data from LaTeX source documents, and a language resource that has been generated using this tool: a corpus of interlinear glossed text (IGT) extracted from open access books published by Language Science Press. Extracted examples are represented in a simple XML format that is easy to process and can be used to validate certain aspects of interlinear glossed text. The main challenge involved is the parsing of TeX and LaTeX documents. We review why this task is impossible in general and how the `texhs` Haskell library uses a layered architecture and selective early evaluation (expansion) during lexing and parsing in order to provide access to structured representations of LaTeX documents at several levels. In particular, its parsing modules generate an abstract syntax tree for LaTeX documents after expansion of all user-defined macros and lexer-level commands that serves as an ideal interface for the extraction of interlinear glossed text by `texigt`. This architecture can easily be adapted to extract other types of linguistic data structures from LaTeX source documents.

Keywords: Interlinear Glossed Text, LaTeX, Parsing

1. Introduction

We present a tool for the extraction of structured linguistic data from LaTeX source documents and a language resource that has been generated using this tool: a corpus of interlinear glossed text (IGT) extracted from open access books published by Language Science Press.

Many linguistic research papers and books are written in LaTeX, using more or less semantic markup. Our goal is to take advantage of this existing structural information and systematically extract linguistic data structures from properly formatted LaTeX documents. This route can complement existing tools for harvesting IGT from PDF documents, in particular ODIN (Lewis and Xia, 2010), which have to cope with much more noise in the source channel resulting from lossy pdf-to-text conversion. However, parsing LaTeX comes with its own challenges, as detailed in section 5.

In addition to *extracting* data, the tool we are presenting can also be used for *validating* LaTeX encodings of IGT. For example, it can check whether the number of words and morphemes in the source line match those in the gloss lines or whether the formatting used in gloss lines conforms to certain conventions codified by the Leipzig Glossing Rules (Bickel et al., 2008).

2. Interlinear Glossed Text

Interlinear glossed text (IGT) is a commonly used format for describing linguistic data, typically comprised of several word-aligned lines and a single free translation line. A simple example, taken from Holton and Robinson (2014, p.163), is shown in (1).

(1) Ke'e pi-ga-ussar.
fish 1PL-3SG-catch
'We're catching fish.'

The first line contains a phrase in the original language, with words split into morphemes separated by a dash. The second line provides a word-aligned morpheme-by-morpheme analysis, and the final line gives a free translation of the whole phrase in English.

3. Representations of IGT in LaTeX

While the LaTeX format itself (Lamport, 1994) does not directly support typesetting interlinear glossed text (without resorting to table markup or other means of manual formatting), there are several third-party packages that have been designed to add a convenient syntax for encoding IGT. Popular choices in linguistic communities are `gb4e`,[1] `linguex`[2] and `ExPex`.[3] We will focus on describing the widely adopted `gb4e`, although our conversion tool supports other packages as well.

The `gb4e` package provides a list-like environment for numbered examples that may contain IGT blocks. The basic syntax for glossed examples is illustrated in Figure 1. Examples are contained in an `exe` environment and prefixed by an `\ex` command. A block of word-aligned lines is introduced by `\gll` (two lines) or `\glll` (three lines), and each line is terminated by a hard line break (`\\`). An example item may contain an arbitrary number of subexamples. This is encoded by wrapping the subexamples in an `xlist` environment, as illustrated in Figure 2.

There are several other ways of encoding IGT using `gb4e`. For example, the package exposes some unofficial (undocumented) abbreviations for opening and closing example containers that are sometimes used in the wild. The use of these `\ea` and `\z` commands is illustrated in the real-world example in Figure 3.

Even if the package did not provide these shortcuts directly, LaTeX users are free to define their own macros with similar functionality. Moreover, a document written in LaTeX

[1]Available at http://ctan.org/pkg/gb4e, accessed on 2016-03-01.

[2]Available at http://www.ctan.org/pkg/linguex, accessed on 2016-03-01.

[3]Available at http://www.ctan.org/pkg/expex, accessed on 2016-03-01.

has full access to the low-level commands of the underlying TeX language (Knuth, 1984). While LaTeX provides a well-organized semantic markup layer for documents, its abstractions are not enforced in any way and can easily be broken by making use of low-level primitive commands. The listing in Figure 4 may serve as an arbitrary example. Although the last four lines might not look like idiomatic LaTeX, they are perfectly valid and in the context set up by the preceding lines they are in fact equivalent to the listing in Figure 2 and produce identical output.

In general, the concrete syntax a document author wishes to use can be customized almost without restrictions. TeX is dynamically programmable down to the lexer level. For example, authors are free to choose their own escape character (instead of the conventional backslash), they can define their own grouping characters (instead of the conventional braces), they can define their own comment character (instead of the conventional percentage sign), they can define their own macro names and environment names, and so on. All of these settings can have local scope, so authors are free to use a wild mix of mutually inconsistent settings across their document. While these options for controlling TeX may seem exotic or marginal to some casual LaTeX users, they are the bread and butter of LaTeX package writers. Many commonly used commands crucially depend on these functionalities. For example, the `\verb` command and the `verbatim` environment allow the author to enter special characters without escaping by temporarily changing the category codes of these characters behind the scenes.

4. Extracting IGT from LaTeX

While there are many tools for rendering IGT and other linguistic data structures to a LaTeX representation for typesetting, there are almost no tools that can go the other way. We are trying to fill this gap with a tool called `texigt`[4] that extracts IGT instances from LaTeX documents and converts them to an XML representation described in section 6.

One main challenge is that IGT encodings in LaTeX documents may contain arbitrary TeX and LaTeX macros from various sources, including (a) standard macros for font styles, citations, footnotes, cross-references or even low-level formatting instructions, (b) macros from imported packages, like `tipa`[5] for phonetic symbols or `leipzig`[6] for standard glossing abbreviations, and (c) additional user-defined macros of arbitrary complexity, as illustrated in Figure 4. In other words, robust extraction of IGT data from LaTeX documents is only possible on the shoulders of a full-blown TeX parser.

```
\begin{exe}
\ex\glll
  sw1m1-sw1m2 sw2m1 sw3m1-sw3m2\\
  g1w1m1-g1w1m2 g1w2m1 g1w3m1-g1w3m2\\
  g2w1m1-g2w1m2 g2w2m1 g2w3m1-g2w3m2\\
\glt 'This is the translation line'
\end{exe}
```

Figure 1: Encoding of IGT using the `gb4e` LaTeX package, showing an example consisting of one source line, two gloss lines and a translation line. Each of the aligned lines contains three words which in turn consist of one or two morphemes.

```
\begin{exe}
  \ex
  \begin{xlist}
    \ex First subexample
    \ex Second subexample
  \end{xlist}
\end{exe}
```

Figure 2: Nested unglossed example sentences using the `gb4e` LaTeX package.

```
\ea
\label{ex:4:8}
\langinfo{Western Pantar}{AP}%
  {\citealt{Holton2010}} \\
\gll Ke'e pi-ga-ussar. \\
   fish \textsc{1pl-3sg}-catch \\
\glt 'We're catching fish.'
\z
```

Figure 3: A glossed example sentence from the LaTeX sources of Holton and Robinson (2014, p.163) using the `gb4e` package with unofficial environment abbreviations and a custom `langinfo` command for supplying metainformation.

[4]See https://github.com/langsci/xmlbooks for further details.

[5]Available at http://www.ctan.org/pkg/tipa, accessed on 2016-03-01.

[6]Available at http://www.ctan.org/pkg/leipzig, accessed on 2016-03-01.

```
\catcode`|=0 \catcode`E=13
\catcode`)=1 \catcode`(=2

|let|+|begin
|let|-|end
|let|*|ex
|defE#1{|+)exe(|*|+)xlist(#1%
        |-)xlist(|-)exe(}

E)
|* First subexample
|* Second subexample
(
```

Figure 4: An alternative representation of a simple `gb4e` example in somewhat obscure but perfectly valid LaTeX. Notice that ‘`)(`’ is a well-balanced empty group in this context, as far as TeX is concerned. If this listing is confined to a local group, it is completely equivalent to Figure 2 and produces identical output.

For this reason, `texigt` is built on top of a separate open source TEX parsing library, `texhs`[7], also developed at Language Science Press, mainly for the conversion of linguistic books from LATEX to XML, HTML and EPUB. It is written in Haskell (Marlow, 2010) and provides an interface for accessing the abstract syntax tree of LATEX documents (as will be detailed in section 5). This tree is then walked by `texigt`, looking for and analyzing IGT containers. After converting special characters to Unicode and discarding irrelevant formatting instructions, `texigt` detects word and morpheme boundaries in aligned lines and constructs an internal representation for all encountered IGT instances that is later serialized to the XML format described in section 6.

5. Parsing LATEX

The major task involved in the extraction of IGT is the parsing of LATEX documents. Ideally, we want a representational layer that abstracts away from irrelevant differences in concrete syntax, like between Figures 2 and 4. In this section we review the challenges involved and explore how `texhs` tries to provide clean interfaces to the structure of arbitrary LATEX documents. Since LATEX is basically a superset of TEX, we will focus on parsing TEX in the following subsections.

5.1. Parsing TEX is impossible

How hard is it to write a robust TEX parser? For a start, here are three facts that suggest that the task is at least not trivial.

First, TEX is a powerful and flexible programming language. For example, it has been used to control a Mars rover (Hicks, 2009) or to embed functional programming idioms (Jeffrey, 1990).

Second, the TEX language lacks a formal grammar or specification. This is not to deny that its reference implementation has an excellent and extensive documentation (Knuth, 1986), but this lengthy treatment does not replace a concise formal specification.

Third, TEX is an unusual and highly dynamic programming language. An interesting symptom of this is that the TEX engine is conventionally described as a living organism (Knuth, 1984, chapter 7) with eyes, mouth and stomach, rather than in terms of compiler theory.

However, at the core of the third issue is the fact that lexing, parsing and evaluation are deeply intertwined in TEX. In fact, as Erdweg and Ostermann (2011) point out, TEX is a dynamic language that *cannot be parsed* in general. There are TEX documents that do not have a syntax tree at all. Here is a simple example from Erdweg and Ostermann (2011, 398):

```
\def\app#1#2{#1#2}
```

Is the body of this macro definition a macro application or a text sequence? Well, it can be both, depending on the arguments:

```
\def\id#1{#1}
\app a b
\app \id c
```

[7]Available at `https://github.com/synsem/texhs`, accessed on 2016-03-01.

In the first application of `\app` it is a text sequence. In the second application of `\app` it is a macro application. And its gets worse:

> Since TEX is a Turing-complete language, the property whether a program has a parse tree is even undecidable. (Erdweg and Ostermann, 2011, 398)

Several language features prevent a static analysis of TEX documents. Erdweg and Ostermann (2011) mention dynamic scoping, higher-order arguments, the lexical macro system (macro arguments and macro bodies need not have complete syntax trees) and the custom macro call syntax using delimited parameters. In addition, TEX features commands like `\expandafter` that modify the evaluation order, conditionals that induce their own grouping structure independent of the regular block structure and a category code system for dynamically classifying characters.

5.2. Faking a TEX parser

Since lexing, parsing and evaluation (expansion) are deeply intertwined in TEX, it is *impossible* to write a (correct) static TEX parser. If we do not want to restrict ourselves to an artificial subset of TEX that can be statically parsed, we need to mimic TEX's behavior and intermingle parsing and evaluation to some degree. All of the more powerful LATEX-to-Markup converters follow this strategy, including `LaTeXML`[8] and `plasTeX`[9].

The `texhs` library we are using for `texigt` approaches this problem in the following way that turned out to be successful in practice. Broadly speaking, TEX and LATEX commands are partitioned into two sets: First, *lexer-level commands* are macros that need to be evaluated during lexing because they may change the behavior of the lexer. Examples include category code changes (`\catcode`), modifications of the expansion order (`\expandafter`), user-level macro definitions (`\let`, `\def`, `\newcommand`), and conditionals (`\if`, `\ifx`). Second, *document-level commands* are macros that need not be evaluated during lexing or parsing. Examples include formatting instructions (`\emph`, `\textit`), sectioning commands (`\section`, `\chapter`), anchors and cross-references (`\label`, `\ref`), and bibliographic citations (`\cite`).

Using this strategy, `texhs` provides access to an abstract syntax tree of a LATEX document after expansion of all user-defined macros and lexer-level commands. Depite of their radical differences in concrete syntax the practically equivalent LATEX listings in Figures 2 and 4 receive an identical representation at this level. Their common abstract syntax tree is shown in Figure 5.

`texigt` uses the representation at this level in order to extract `gb4e` example containers and glossed example sentences.

[8]Available at `http://dlmf.nist.gov/LaTeXML/`, accessed on 2016-03-01.

[9]Available at `http://tiarno.github.io/plastex/`, accessed on 2016-03-01.

6. Representations of IGT in XML

In order to capture as much information as possible from the LaTeX source document, we are using a data model that closely mirrors the one used by LaTeX IGT packages, in particular `gb4e`. The XML serialization format for this data model is the primary output of the `texigt` conversion tool. It is illustrated in Figure 6 and described by a minimalistic RelaxNG schema shown in Figure 7 and available at `https://github.com/langsci/xmlbooks`, where a tutorial-style introduction to this format is also provided.

This data model is simpler than most of the IGT representations that have been proposed in the past and it should be straightforward to convert the `texigt` XML output to other formats like Xigt (Goodman et al., 2015), TypeCraft (Beermann and Mihaylov, 2014) or GrAF (Ide and Suderman, 2007).

7. Application: Extracting IGT from LangSci Books

We have used `texigt` to extract IGT from books published by Language Science Press, an open access publisher of linguistic monographs that uses LaTeX for typesetting. The corpus is available at `https://github.com/langsci/xmlbooks`, together with documentation and a simple query tool. The number of glossed and unglossed examples extracted from each book is shown in Table 1.

8. Conclusion and Future Directions

`texigt` is a tool for extracting and validating IGT embedded in LaTeX documents. There are at least two directions that seem promising for further development. First, the extraction of IGT can be improved by supporting extensions like grammaticality judgments or explicit indications of constituent structure in the form of labeled bracketing, empty categories or co-indexing. Second, non-IGT data structures that are commonly used in linguistics could be extracted as well by taking advantage of LaTeX encoding conventions for syntax trees (`qtree`[10], `forest`[11]), semantic formulas, feature structures (AVMs) or discourse representation structures.

9. Acknowledgements

This work was supported by a grant from the DFG to Stefan Müller and Martin Haspelmath (FKZ MU 2822/8-1). See `http://hpsg.fu-berlin.de/~stefan/Pub/lsp-dfg.html` and `http://langsci-press.org/` for more information about Language Science Press.

[10]Available at `http://www.ctan.org/tex-archive/macros/latex/contrib/qtree/`, accessed on 2016-03-01.

[11]Available at `http://www.ctan.org/pkg/forest`, accessed on 2016-03-01.

```
Group "exe" [] [
  Command "ex" [],
  Group "xlist" [] [
    Command "ex" [],
    Str "First",
    Space,
    Str "subexample",
    Space,
    Command "ex" [],
    Str "Second",
    Space,
    Str "subexample",
    Space
    ]
  ]
```

Figure 5: The internal representation produced by the `texhs` parsing modules for the `gb4e` examples in Figure 2 and Figure 4 (slightly simplified). Notice that despite of the substantial differences in concrete syntax between these two listings, they are both represented by the same value shown here after the `texhs` parsing stage.

```
<example>
  <language>Western Pantar</language>
  <reference>Holton2010</reference>
  <label>ex:4:8</label>
  <alignedwords>
    <word>
      <morpheme>
        <block type="src">Ke'e</block>
        <block type="imt">fish</block>
      </morpheme>
    </word>
    <word>
      <morpheme>
        <block type="src">pi</block>
        <block type="imt">1pl</block>
      </morpheme>
      <morpheme>
        <block type="src">ga</block>
        <block type="imt">3sg</block>
      </morpheme>
      <morpheme>
        <block type="src">ussar</block>
        <block type="imt">catch</block>
      </morpheme>
    </word>
  </alignedwords>
  <translation>
    We're catching fish.
  </translation>
</example>
```

Figure 6: XML representation of the glossed example sentence shown in Figure 3.

10. Bibliographical References

Beermann, D. and Mihaylov, P. (2014). TypeCraft collaborative databasing and resource sharing for linguists. *Language Resources and Evaluation*, 48(2):203–225.

Berghäll, L. (2015). *A Grammar of Mauwake*. Number 4 in Studies in Diversity Linguistics. Language Science Press, Berlin.

Bickel, B., Comrie, B., and Haspelmath, M. (2008). The Leipzig glossing rules: Conventions for interlinear morpheme-by-morpheme glosses. http://www.eva.mpg.de/lingua/resources/glossing-rules.php, accessed on 2015-10-16.

Erdweg, S. T. and Ostermann, K. (2011). Featherweight TEX and parser correctness. In Brian Malloy, et al., editors, *Software Language Engineering: Third International Conference, SLE 2010*, number 6563 in LNCS, pages 397–416, Berlin/Heidelberg. Springer.

Goodman, M. W., Crowgey, J., Xia, F., and Bender, E. M. (2015). Xigt: Extensible interlinear glossed text for natural language processing. *Language Resources and Evaluation*, 49:455–485.

Hicks, S. (2009). Rapid prototyping in TEX. *The Monad.Reader*, 13:5–16.

Holton, G. and Robinson, L. C. (2014). The linguistic position of the Timor-Alor-Pantar languages. In Marian Klamer, editor, *The Alor-Pantar Languages: History and Typology*, number 3 in Studies in Diversity Linguistics, pages 155–198. Language Science Press, Berlin.

Ide, N. and Suderman, K. (2007). GrAF: A graph-based format for linguistic annotations. In *Proceedings of the Linguistic Annotation Workshop*, pages 1–8, Prague. Association for Computational Linguistics.

Jeffrey, A. (1990). Lists in TeX's mouth. *TUGboat*, 11(2):237–245.

Knuth, D. E. (1984). *The TEXbook*, volume A of *Computers & Typesetting*. Addison-Wesley, Reading, Massachusetts.

Knuth, D. E. (1986). *TEX: The Program*, volume B of *Computers & Typesetting*. Addison-Wesley, Reading, Massachusetts.

Lamport, L. (1994). *LATEX: A Document Preparation System*. Addison-Wesley, Reading, Massachusetts.

Lewis, W. D. and Xia, F. (2010). Developing ODIN: A multilingual repository of annotated language data for hundreds of the world's languages. *Literary and Linguistic Computing*, 25(3):303–319.

Simon Marlow, editor. (2010). *Haskell 2010 Language Report*. Haskell.org, https://www.haskell.org/definition/haskell2010.pdf, accessed on 2012-10-10.

```
start = examples
examples = element examples
  { exampleitem* }
exampleitem = element exampleitem
  { example, examples }

example = element example {
  element language { text }?,
  element reference { text }?,
  element label { text }?,
  alignedwords,
  freewords }

alignedwords = element alignedwords
  { word* }
word = element word
  { morpheme* }
morpheme = element morpheme
  { block* }
block = element block
  { blocktype, text }
blocktype = attribute type
  { "phon" | "ortho" | "src" |
    "imt" | ... }

freewords =
   element source { text } |
   element translation { text }
```

Figure 7: Minimalistic XML schema for representing linguistic example sentences in RELAX NG Compact Syntax.

source	glossed	unglossed
berghall.xml	1467	5
cangemi.xml	10	4
dahl.xml	409	36
handschuh.xml	369	24
klamer.xml	352	102
schackow.xml	1561	2
wilbur.xml	327	0
total	4495	173

Table 1: Count of glossed and unglossed example sentences extracted from books published by Language Science Press. The XML files are named after the last name of the first author or editor of the book; for instance, berghall.xml contains examples extracted from Berghäll (2015).