lvgloss.sty 0.11: flexible linguistic gloss macros

Waral xchi-qa-tz'ib'a-j *wi*
here PROSP-A1p-write-SS FOC

xchi-qa-tiki-b'a' *wi ojer tzij,*
PROSP-A1p-stand-CAUS FOC old word

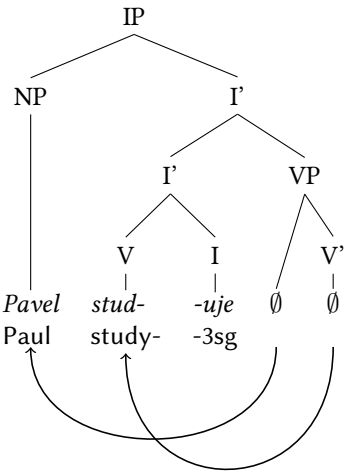
u-tik-ar-ib'al
A3s-plant-VERS-INSTR

u-xe'-n-ab'al *puch, r-onojel x-b'an* *pa tinamit K'iche',*
A3s-root-AP-INSTR PART A3s-all CPL-make:PASS P town K'iche'

r-amaq' *K'iche' winaq.*
A3s-nation K'iche' people

"Here we will write, / will erect the ancient words: / the planting, / the rooting, all that was done in the K'ichee' town, / the nation of K'ichee' people."

(*Popol Wuj*)



Subject	Object	Verb
Wičása ki man the	*mathó wą* bear a	*ktę* killed

Subject	Object	Verb
Mathó wą bear a	*wičása ki* man the	*ktę* killed

Subject	Object	Verb
Wičása ki man the	*ix?é wą* rock a	*wąyále* saw

Object	**Subject**	Verb
Ix?é wą rock a	*wičása ki* man the	*wąyále* saw

(*Van Valin 1985*)

Contents

1 Introduction **2**
1.1 Why? 2
1.2 Quick examples 3

2 How **6**
2.1 Declaring shortcuts 11

1 Introduction

This is a macro package for setting interlinear glossed text, which aims to be simple, flexible and easily extensible. Features include:

- Easy formatting of glossed examples (for instance, bolding words or phrases for emphasis).
- Hooks for creating your own macros that interact with glossed text in interesting ways.
- Special macros for tab-aligned glossed text, and for glossed text in syntax trees.

This package does *not* handle example numbering, but it plays well with other packages that do, including `expex`, `linguex` and `gb4e`.

1.1 Why?

Why create another set of glossing macros? There are already at least two widely used sets: the old reliable midnight macros which are used in linguex and gb4e, and the newer ones found in expex. So why write yet another?

The short answer is, flexibility and extensibility. The midnight macros do one specific thing very well. The macros in expex do a lot of specific things very well. But neither set can be extended beyond what the original author had in mind — or integrated with other formatting macros — without a lot of difficult digging into their internals.

The lvgloss package is an attempt at getting around those limitations. The front page of this manual shows some of the things you can do with it. In the top example, glossed text has been embedded in the LaTeX tabbing environment, in order to highlight the parallelisms in a piece of poetry. At the bottom left, glossed text has been embedded in a syntax tree. In the bottom right, a custom macro has been defined which draws a box around its argument — and this macro is able to operate on multi-word phrases in glossed text. For various reasons, all of these would have been hard to implement in other glossing packages; they are reasonably easy here.

1.2 Quick examples

The macro `\gl` aligns a sequence of words with a sequence of glosses.

dolōrem ipsum quia dolor sit amet
pain.acc itself.acc because pain be.subj.3sg love.subj.3sg

```
\gl{dol\=orem ipsum quia dolor sit amet}
   {pain.acc itself.acc because pain be.subj.3sg love.subj.3sg}
```

In loger examples, linebreaks can be added freely between words.

Nē =que porrō quisquam est quī dolōrem ipsum
not =and further anyone.nom be.3sg who.masc.nom pain.acc itself.acc
quia dolor sit amet consectetur adipisci
because pain be.subj.3sg love.subj.3sg seek.subj.3sg reach.inf
velit
wish.subj.3sg

```
\gl{N\=e =que porr\=o quisquam est qu\={\i} dol\=orem ipsum
     quia dolor sit amet consectetur adipisci velit}
   {not =and further anyone.nom be.3sg who.masc.nom pain.acc itself.acc
     because pain be.subj.3sg love.subj.3sg seek.subj.3sg reach.inf
     wish.subj.3sg}
```

A free translation can be introduced by `\ft`, and a right-aligned label after either the gloss or its free translation can be introduced by `\lb`.

Nē =que porrō quisquam est quī dolōrem ipsum
not =and further anyone.nom be.3sg who.masc.nom pain.acc itself.acc

quia dolor sit amet consectetur adipisci
because pain be.subj.3sg love.subj.3sg seek.subj.3sg reach.inf

velit
wish.subj.3sg

Nor, furthermore, is there anyone who loves, seeks or wishes to obtain pain itself, just because it is pain.

Cicero, “On the Ends of Goods and Evils”

```
\gl{N\=e =que porr\=o quisquam est qu\=i dol\=orem ipsum
     quia dolor sit amet consectetur adipisci velit}
  {not =and further anyone.nom be.3sg who.masc.nom pain.acc itself.acc
     because pain be.subj.3sg love.subj.3sg seek.subj.3sg reach.inf
     wish.subj.3sg}
\ft{Nor, furthermore, is there anyone who loves, seeks or wishes to obtain
    pain itself, just because it is pain.}
\lb{Cicero, ‘‘On the Ends of Goods and Evils’’}
```

Words that are grouped together by braces are aligned as a single unit.

Ēdormī crāpulam!
sleep off hangover

Sleep off that hangover!

```
\gl{\=Edorm\={\i} cr\=apulam!}
   {{sleep off} hangover}
\ft{Sleep off that hangover!}
```

Font commands such as `\bfseries` will only affect a single word that follows them.

dolōrem ***ipsum*** *quia dolor sit amet*
pain.acc itself.acc because pain be.subj.3sg love.subj.3sg

```
\gl{dol\=orem \bfseries ipsum quia dolor sit amet}
   {pain.acc itself.acc because pain be.subj.3sg love.subj.3sg}
```

To make such a command affect *all* subsequent words, precede it with an exclamation mark, as in `!{\bfseries}`.

dolōrem *ipsum* *quia* *dolor* *sit* *amet*
pain.acc itself.acc because pain be.subj.3sg love.subj.3sg

```
\gl{dol\=orem !{\sc} ipsum quia dolor !{\mdseries} sit amet}
   {pain.acc itself.acc because pain be.subj.3sg love.subj.3sg}
```

Similarly, formatting commands like \\ or \par need to be preceded with an exclamation mark: `!{\\}`, `!{\par}`.

dolōrem *ipsum*
pain.acc itself.acc
quia *dolor* *sit* *amet*
because pain be.sbj.3sg love.sbj.3sg

```
\gl{dol\=orem ipsum !{\\}
     quia dolor sit amet}
   {pain.acc itself.acc because
     pain be.sbj.3sg love.sbj.3sg}
```

A single forward slash can be used as a shortcut to produce a line break: the example above could also have been written.

```
\gl{dol\=orem ipsum / quia dolor sit amet}
   {pain.acc itself.acc because pain be.sbj.3sg love.sbj.3sg}
```

What glossing macros do is take horizontally organized input — like this

```
\gl{lorem ipsum dolor sit amet}
   {1     2     3     4   5}
```

— and pair the words off into a bunch of vertical boxes, like this:

lorem	ipsum	dolor	sit	amet
1	2	3	4	5

But sometimes you want to issue a command that will "escape" from this process: that will not get paired off or boxed up, but will end up on its own and outside of any box. For instance, if you want to force a line break using the \par command, or make an adjustment to the spacing between words, it really needs to take place *between* boxes, like so:

lorem	ipsum	\par	dolor	\kern3pt	sit	amet
1	2		3		4	5

The midnight macros don't have any escape mechanism that will make this possible. In expex, there are some built-in shortcuts that work a little like escaped commands (for instance, if you type a forward slash character in the middle of a gloss, it will tell expex "start a new line here," and so this is a little bit like being able to escape a `\par` command.) — but these shortcuts are hardcoded into the package, and it's difficult to create new ones.

This package has an escape mechanism that can get arbitrary bits of LaTeX code out into the space in between words. It also lets you define your own single-character shortcuts that are equivalent to bits of escaped code. (It comes with a few shortcuts predefined, but you can override these.) Basically nothing is hard-coded, and everything can be redefined. This lets you play all sorts of fancy formatting tricks like the ones on page 1 — but also, more prosaically, it means you can have your line breaks wherever you want them:

```
\gl{lorem ipsum !{\par} dolor !{\kern2pt} sit amet}
   {1     2             3                 4   5}
```

2 How

glhangindent
betweenglspace
withinglspace
aboveglspace
everygla
everyglb
glescape

Set up dimensions to control spacing, and macros to set the default font for object language and metalanguage words. Specify the escape character. All of these are easily overrideable by the user.

```
\newtoks\ta\newtoks\tb
\newdimen\glhangindent\glhangindent=2em
\newdimen\betweenglspace\betweenglspace=\jot
\newdimen\withinglspace\withinglspace=0pt
\newdimen\aboveglspace\aboveglspace=0pt
\def\glspace{\penalty0\hspace{0pt}}
\def\everygla{\slshape}
\def\everyglb{\sffamily}
\def\glescape{!}
%\end{macrocode}
%\end{macro}
%\end{macro}
%\end{macro}
%\end{macro}
%\end{macro}
%\end{macro}
%\end{macro}
%\begin{macro}{\addtokens...\to}
%\begin{macro}{\pop...\to}
%\begin{macro}{\popoff...\to...\remainderin}
%\begin{macro}{\split}
%\begin{macro}{\ssplit}
%Some utility macros. These give us push and pop operations on token lists,
%and a string split operation on the contents of macro arguments.
%\begin{macrocode}
\long\def\addtokens#1\to#2{\ta={#1}\tb=\expandafter{#2}\edef#2{\the\tb\the\ta}}
```

```
\def\pop#1\to#2{\expandafter\popoff#1\to#2\remainderin#1}
\long\def\popoff#1 #2\to#3\remainderin#4{#3={#1}\def#4{#2}}
\long\def\split#1{\expandafter\ssplit#1\xyzzy}
\long\def\ssplit#1#2\xyzzy{\gdef\istchar{#1}\gdef\restchars{#2}}
%\end{macrocode}
%\end{macro}
%\end{macro}
%\end{macro}
%\end{macro}
%\end{macro}
%\begin{macro}{\ifnotin}
%A rather devious macro to test whether one argument is found
%within another. We will use it to check whether a character
%has been designated as a ``special'' or ``passthru'' character.
%(This lets us get away without any monkeying around with catcodes:
%declaring, say, \verb+[+ as a special character does not change
%the catcode associated with \verb+[+ --- which is important if we
%want our metalanguage to contain brackets.)
%\begin{macrocode}
\def\ifnotin#1#2{%
  \def\@ifnotin##1#1##2##3\zyzzy{\ifx\ifnotfound##2}%
  \expandafter\@ifnotin#2#1\notfound\zyzzy}
%\end{macrocode}
%\end{macro}
%\begin{macro}{\checkspecial}
%\begin{macro}{\checkpassthru}
%\begin{macro}{\makespecial}
%\begin{macro}{\makepassthru}
%The first two macros here check whether a character has been designated
%as a special or passthru character, using \verb+\ifnotin+.  The second
%two macros are used to designate new special or passthru characters.
%\begin{macrocode}
\newif\ifspecial
\def\dvglspecials{}
\def\checkspecial#1{%
  \ifnotin#1{\dvglspecials}\specialfalse\else\specialtrue\fi}
\def\dvglpassthru{}
\def\checkpassthru#1{%
  \ifnotin#1{\dvglpassthru}\specialfalse\else\specialtrue\fi}
\def\makespecial#1{%
  \edef\dvglspecials{\dvglspecials#1}}
\def\makepassthru#1{%
  \edef\dvglpassthru{\dvglpassthru#1}}
%\end{macrocode}
%\end{macro}
%\end{macro}
%\end{macro}
%\end{macro}
%\begin{macro}{\stacksymbol}
%This will stack two words atop one another, with appropriate struts and spacing.
```

```
%\begin{macrocode}
\def\stacksymbol#1#2{%
  \mbox{\vtop{\hbox{#1\/\strut}\nointerlineskip\hbox{#2\/\strut}}}}%
%\end{macrocode}
%\end{macro}
%\begin{macro}{\glossword}
%This is the default macro for formatting a pair of object-langauge and metalanguage
%words. In addition to stacking them, it invokes the proper font commands
\def\glossword#1#2{%
  \stacksymbol{\everygla#1}{\everyglb#2}%
  \glspace}
\def\glosswordmacro#1#2{\glossword{#1}{#2}}
```

\merge
bracketcount
maxbracketcount

The meat and bones of the thing: this is what I called "zipwith-with-exceptions" in the explanation up above, but "merge" is shorter and easier to type. All three arguments should be token list macros (so the usage is `\merge\x\and\y\to\z`). The first two arguments should contain its input; the last will contain its output at the end. The input token lists will be destroyed in the course of the macro, so I hope you weren't using them for anything.

```
\newcount\bracketcount\newcount\maxbracketcount%
\def\merge#1\and#2\to#3{%
```

If there's still stuff in our first token list, pop an item off the front. Then, split the first character off of that first item.

```
  \ifx\empty#1\else%
    \pop#1\to\ta%
    \def\istchar{}\def\restchar{}%
    \edef\temp{\the\ta}\split\temp%
```

If that first character is an escape, add the rest to our output token list.

```
    \ifx\istchar\glescape%
      \expandafter\addtokens\restchars\to#3%
      \expandafter\addtokens\space\to#3%
    \else%
```

If that first character is a passthrough character, add it *and* all the rest to the output token list.

```
      \expandafter\checkpassthru\istchar\ifspecial%
        \expandafter\addtokens\istchar\to#3%
        \expandafter\addtokens\restchars\to#3%
        \expandafter\addtokens\space\to#3%
```

(This is a cheat. For `\gltree` down below, we'll want to know the deepest bracket nesting that occurs in the course of a gloss. Here we have some code to calculate that. It is hard-wired to work only with square brackets, and only when square brackets have been declared as passthru characters.)

```
        \if\istchar[\advance\bracketcount by 1\fi%
        \ifnum\bracketcount>\maxbracketcount%
          \maxbracketcount=\bracketcount\fi
        \if\istchar]\advance\bracketcount by -1\fi%
```

Okay, back to the important stuff. If that first character is a special character, generate the appropriate control sequence name from it, make the remaining characters in the item its argument, and add that to the output token list. (So if * is a special character, and we see *argument as an item in the first argument, we'll end up sending \gl*<argument> to output.

```
      \else%
        \expandafter\checkspecial\istchar\ifspecial%
          \expandafter\addtokens\csname gl\istchar\endcsname\to#3%
          \expandafter\addtokens\expandafter<\restchars>\to#3%
          \expandafter\addtokens\space\to#3%
        \else%
```

That's all the special cases. If we've gotten this far, we know we're building a normal glossword. Make sure the second argument isn't empty; pop off an item from the second argument; package those two items together into a glossword; and away we go. One tricky thing: \glosswordmacro is expanded exactly once here. So if we do \def\glosswordmacro\foo, and the current items are first and second, we'll end up outputting \foo{first}{second}.

```
          \ifx\empty#2\else%
            \pop#2\to\tb%
            \edef\temp{\expandafter\noexpand\glosswordmacro{\the\ta}{\the\tb} }%
            \expandafter\addtokens\temp\to#3%
          \fi%
        \fi%
      \fi%
    \fi%
```

And recurse!

```
    \merge#1\and#2\to#3%
  \fi%
}
```

\putgl This is just a wrapper around \merge, to compensate for the fact that \merge wants an extra space at the end of its first two arguments. Ideally, in a future version, we'll add some more code here to make sure there's one *and only one* trailing space, since multiple spaces also make \merge barf.

```
\long\def\putgl#1#2\into#3{%
  \def\x{#1 }\def\y{#2 }\def#3{}%
  \merge\x\and\y\to#3%
}
```

\gl Here's the basic user-facing gloss macro. Take two arguments and merge them together. Do a little formatting. excecute the output we got from from \merge. Do a little more formatting. The stuff having to do with indentation and spacing is at the end, so we can make reasonably sure it will still be in effect when the paragraph is broken. But we don't actually break the paragraph here, in case the user wants to add a label in the right margin (see \lb below).

```
\long\def\gl#1#2{%
  \putgl{#1}{#2}\into\z%
```

```
  \ifvmode\vskip\aboveglspace\fi%
  \z%
  \hangafter1\hangindent=\glhangindent%
  \lineskiplimit=\betweenglspace\lineskip=\betweenglspace%
  \rightskip=0pt plus 1fil%
}
```

`\gltree` `\tikzglossword` `nodecount`

This one not only creates a sequence of glossboxes, but feeds it to `tikz-qtree`'s `\Tree` macro as an argument. To make this go smoothly, we need to define a replacement for `\glossbox` which will create a `tikz` node instead of just a TEXbox. We also keep track of how many nodes we've created so that we can give each one a name — the first one's named "1," the second "2," and so on.

(I do not understand why the `\noexpand`s in this macro are necessary, or why they work. but they do seem to be necessary, and they do seem to work. So be it.)

```
\newcounter{nodecount}%
\def\nc{\addtocounter{nodecount}{1}}
\def\tikzglossword#1#2{%
  \node[align=left](\noexpand\arabic{nodecount}){%
    \noexpand\stepcounter{nodecount}\noexpand\everygla#1\noexpand\\\noexpand\everyglb#2};}%
```

Right. Now we can define `\gltree` itself. This is pretty straightforward. We declare left and right brackets to be passthru characters; we initialize the bracket counter; we do a little formatting; and away we go. The tree is set up to be `\maxbracketcount`$*30$ points tall; that leaves room for each level in the tree to have a height of 30 points.

```
\newcommand{\gltree}[3]{%
  \begingroup%
  \bracketcount=0\maxbracketcount=0%
  \makepassthru[\makepassthru]%
  \begin{tikzpicture}[every leaf node/.append style={inner sep=1}, baseline=base, sibling distance=
  \setcounter{nodecount}{1}%
  \let\glosswordmacro=\tikzglossword%
  \putgl{#1}{#2}\into\z%
  \multiply\maxbracketcount by 30%
  \expandafter\Tree\z%
  #3%
  \end{tikzpicture}%
  \endgroup%
}
```

`\gltab`

This one wraps our sequence of glossboxes in LATEX's `tabbing` environment, and sets up some shortcuts for commonly used commands in that environment. The `minipage` is necessary to keep us from getting a linebreak and extra vertical space at the beginning and end of the `tabbing` environment. The width of the `minipage` is set arbitrarily; this is dumb and we should do better in a future version.

```
\long\def\gltab#1#2{%
  \begingroup%
  \makeglshortcut={\=}\makeglshortcut>{\>}\makeglshortcut/{\\[\betweenglspace]}%
  \putgl{#1}{#2}\into\z%
  \begin{minipage}[t]{0.9\textwidth}\begin{tabbing}\z\end{tabbing}\end{minipage}%
```

```
  \endgroup%
}
```

\ft \lb These are for putting a "free translation" after a gloss (\ft) and for putting a "label" to the right of a gloss (\lb). Nothing too complicated.

```
\def\ft#1{\par\nopagebreak[4]\lineskiplimit=0pt\lineskip=1pt\addvspace{\jot}#1\strut}
\def\lb#1{{\unskip\nobreak\hfil\penalty0\hskip2em\mbox{}\nobreak\hfill\mbox{\strut#1}}}
```

2.1 Declaring shortcuts

We have a macro \makespecial which will tell the parser "treat this character as special."

But that alone won't get us anywhere interesting. We also need to *define* the macro that will be executed when the special character in question is encountered. And it would be nice if we could wrap both steps — calling \makespecial and defining the macro — in a user-friendly package. That's what this last section does.

Suppose we want to declare / as a shortcut character which inserts a line break. Step one is simple:

```
\makespecial{/}
```

Now let's consider how the parser will respond after we've done this. Suppose we then execute this command:

```
\gl{a b * c}{1 2 3}
```

The output which is sent to be typeset will look like this

```
\glossword{a}{1}
\glossword{b}{2}
      \gl/<>
\glossword{c}{3}
```

where \gl/ is a control sequence — one which could not normally be input directly. And if we execute the command

```
\gl{a b /argument c}{1 2 3}
```

The output which is sent to be typeset will look like this:

```
\glossword{a}{1}
\glossword{b}{2}
  \gl/<argument>
\glossword{c}{3}
```

Step two, then, is to define a macro that will do something useful in those contexts. If the slash were a normal character, this would be easy:

```
\def\gl/<#1>{\\\ifx#1\empty\else\vskip{#1}\fi}
```

But since the slash isn't a normal character, we need to do it this way instead:

```
\expandafter\def\csname gl/\endcsname<#1>{\par\ifx#1\empty\else\vskip{#1}\fi}
```

\makeglshortcut Well, that's just what \makeglshortcut does: carries out Step One and Step Two for an arbitrary character and arbitrary macro code.

```
\long\def\makeglshortcut#1#2{%
  \makespecial{#1}%
  \expandafter\gdef\csname gl#1\endcsname<##1>{#2}}
```

\makeglsurround \makeglsurround does the same thing, except that in Step Two it creates a delimited macro with several arguments. This requires some unpleasant fucking around with token registers and \expandafter.

```
\long\def\makeglsurround#1#2#3{%
  \xdef\dvglspecials{\dvglspecials#1#2}%
  \ta=\expandafter{\csname gl#1\endcsname}%
  \tb=\expandafter{\csname gl#2\endcsname}%
  \expandafter\expandafter\expandafter\gdef%
     \expandafter\expandafter\the\ta%
     \expandafter<\expandafter##\expandafter1\expandafter>%
     \expandafter##\expandafter2\the\tb<##3>{#3}}
```

/ And here we have definitions for the built-in shortcuts.

```
\makeglshortcut/{\\\ifx#1\empty\else\vskip{#1}\fi}
\makeglshortcut[{\stacksymbol{[}{[}}
\makeglshortcut]{\stacksymbol{]$_{#1}$}{]$_{#1}$}}
\makeglsurround ** {%
  \rlap{\raisebox{1.5em}{\footnotesize\sf#3}}%
  \fbox{#2\unskip}\glspace}
\def\doarrow\glossword#1\glossword#2\glossword#3\glossword{\linkto\glossword#1\under\glossword#2\fr
\makeglsurround<<{\doarrow#2}
```