

Miscellaneous graphics

L^AT_EX for Linguists

Sylvia Blaho Patrik Bye Pavel Iosad

University of Tromsø

December 8, 2008

PDFL^AT_EX

- So far we have not paid attention to the engine that we're using and just used PDFL^AT_EX;
- PDF is a widespread filetype, but it is not T_EX's native output format;
- Natively, T_EX outputs a **d**evice **i**ndependent **f**ile (DVI), which is not good for distribution
- DVI files can be turned into Postscript (PS) files using `dvips`
- or directly into PDF files using `dvipdfm`

PDFL^AT_EX

- PS files can also be turned into PDF files through `dvips`
- This is good because PS is a very powerful graphic language and there are things that PDFL^AT_EX cannot accomplish
- In particular, there is a extremely powerful package called PSTricks
- However, if you typeset something with PS-specific stuff using PDFL^AT_EX, you will not get the result you want

Using plain L^AT_EX

- PDFL^AT_EX is set by default in T_EXShop, so we need to tell it if we want plain L^AT_EX
- One way is to choose “T_EX and Ghostscript” in the “Typeset” menu. But this needs to be done for every session, and is tedious
- You can also set “T_EX and Ghostscript” to be default by going to the “Preferences” → “Typesetting” → “Default script”. But then you’ll have to switch to PDFL^AT_EX manually...
- Probably the best way is to put
`%%!TEX TS-program = latex before \documentclass` for files where you know you’ll be using PS.

Why is this interesting?

- If you want to include external graphics, you need to know how you are going to typeset your document
- Many linguistics packages are built on PStricks, so you need to know how to work with plain L^AT_EX
- It may make sense to use T_EX+Ghostscript as your main typesetting engine, but...
- Especially with large documents, PDFL^AT_EX is faster
- You get fewer PDF-specific options (hyperlinks, microtype etc.)

Inserting graphics

- If you want to include graphics (e. g. plots or maps), the first thing you need is `\usepackage{graphicx}`
- To insert the picture, just use `\includegraphics{filename.extension}`
- In this case \LaTeX looks for the file in the same folder as the `.tex` file
- You can also give an absolute path (`/Users/user/Pictures/pic.eps`) (not recommended for sharing!), or a relative path (`Graphics/pic.eps`) (good practice to keep pictures and text separately for larger projects)

Graphics insertion

- The `\includegraphics` command takes some optional arguments
- `\includegraphics[scale=.5]{pic.jpg}` resizes the picture
- `\includegraphics[width=4cm,height=1in]{pic.jpg}` gives the absolute dimensions. One good idea is `width=\columnwidth`
- `\includegraphics[rotate=45]{pic.jpg}` rotates the picture

Graphics formats

- The format of the graphic is important. Plain L^AT_EX accepts only Encapsulated PostScript (EPS) files. Most graphics editors know how to export to that. **If you use plain L^AT_EX (e. g. for PStricks), you can only use .eps!**
- PDFL^AT_EX accepts files in .jpg, .png and .pdf formats. **It does not accept .eps!**
- If you have an .eps file, the best way to convert it to PDF is with the `epstopdf` command line utility included in most T_EX distros

Graphics insertion

- A useful trick is to have both an .eps and a .pdf (.jpg, .png) version of your graphics in the same folder and use `\includegraphics{}` without the file extension
- \LaTeX will then find the one that suits it depending on the engine you're running

Arrays

- Previously we looked at the `tabular` environment. The `array` environment functions in a similar way, but is used in math mode, i.e. $\$ \dots \$$. The values of *pos* ($= \{t, b\}$) and *columns* ($= \{l, r, c, p\{\text{width}\}\}$) are the same as for `tabular`.

```
\begin{array}[pos]{columns} rows \end{array}
```

- In linguistics, arrays are most immediately useful for representing feature matrices. The array must be typeset in math mode. Within math mode, normal text may be typeset using the command `\text{ }`.

- $x^2 + \text{je ne sais quoi} = z^2$

```
 $\$x^2 + \text{\text{je ne sais quoi}} = z^2\$$ 
```

Arrays

son
cont
voice

```
$$\begin{array}{l}  
son\\  
cont\\  
voice  
\end{array}$$
```

Arrays

son
cont
voice

```
 $\begin{array}{l}  
 \text{son} \\ \text{cont} \\ \text{voice} \\ \end{array} $
```

Arrays

–son

–cont

–voice

```

 $\begin{array}{l}
\text{\text{\$-\$son}}\\
\text{\text{\$-\$cont}}\\
\text{\text{\$-\$voice}}
\end{array}$ 

```

Delimiters

$$\left[\begin{array}{l} -\text{son} \\ -\text{cont} \\ -\text{voice} \end{array} \right]$$

```
 $\left[ \begin{array}{l} \\ \text{\text{\$-\$son}} \\ \text{\text{\$-\$cont}} \\ \text{\text{\$-\$voice}} \end{array} \right]$ 
```

Rewrite rules

$$\begin{bmatrix} -\text{son} \\ -\text{cont} \\ -\text{voice} \end{bmatrix} \rightarrow \begin{bmatrix} -\text{son} \\ +\text{cont} \\ -\text{voice} \end{bmatrix} / \text{---}[Wd\dots]$$

```

 $\left[ \right.$ 
 $\begin{array}{l} \text{\text{\$-\$son}} \\ \text{\text{\$-\$cont}} \\ \text{\text{\$-\$voice}} \end{array}$ 
 $\right]$ 
 $\rightarrow$ 
 $\left[ \right.$ 
 $\begin{array}{l} \text{\text{\$-\$son}} \\ \text{\text{\$+\$cont}} \\ \text{\text{\$-\$voice}} \end{array}$ 
 $\right]$  /
 $\underline{\quad}$ 
 $[_{Wd} \ \text{\text{\ldots}} \ \$]$ 

```

Delimiters

Sometimes it might be desirable to open an array with a `{` delimiter while leaving the righthand side empty. In this case, the right delimiter has to be explicitly specified as null using a full stop: `\right.`

Delimiters

$$\begin{bmatrix} -\text{son} \\ -\text{cont} \\ -\text{voice} \end{bmatrix} \rightarrow \begin{bmatrix} -\text{son} \\ +\text{cont} \\ -\text{voice} \end{bmatrix} / \text{---} \left\{ \begin{array}{l} [Wd\dots \\ \acute{V}\dots \end{array} \right.$$

```

 $\left[ \right.$ 
 $\begin{array}{l} \text{---son} \\ \text{---cont} \\ \text{---voice} \end{array}$ 
 $\right]$ 
 $\rightarrow$ 
 $\begin{array}{l} \text{---son} \\ \text{+cont} \\ \text{---voice} \end{array}$ 
 $/$ 
 $\text{---}$ 
 $\left\{ \begin{array}{l} [Wd\dots \\ \acute{V}\dots \end{array} \right.$ 

```

The xyling package

- The `xyling` package is a useful tool for tree building and tree-like diagrams
- It lets one build quite advanced trees with arrows, curves and suchlike
- It works best in `plain LATEX`. In PDF`LATEX` it works, but the diagonal lines are scraggy. Also no colours in PDF`LATEX`

Building a tree

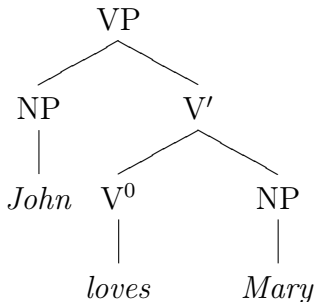
- A “tree” in xyling is not unlike an array or a matrix. Nodes are positioned in a matrix-like structure with `&` being a column delimiter and `\` starting new rows.
- The command `\K{}` is used for text in the node
- The command `\B{}` is used for drawing branches. It takes the arguments `d,u` (for down and up) and `l,r` for left-right (as many times as needed)

Tree example

```

\Tree{ & \K{VP}\B{dl}\B{dr}\
\K{NP}\B{d} & & \K{V$'$}\B{dl}\B{dr}\
\K{\emph{John}}& \K{V$^{0}}\B{d} & & \K{NP}\B{d}\
& \K{\emph{loves}} & & \K{\emph{Mary}}}}

```



More xyling

- You can also have roofs with the command `\TRi`
- You can circle nodes with the command `\O0`. If you do, use `\Bo` for branches starting at circled nodes, `\BO` for branches with circled targets and `\BoO` where both start and target are circled
- For arrows, use `\Link{}` (with the same arguments as `\B{}`) and an optional argument setting the style of the arrow: e. g. `\Link[-->]{}` is a dashed arrow
- `xyling` also defines macros handy for syntax, for example `\VP` is equivalent to `\K{NP}` and two branches on either side. See the documentation for explanations

More xyling

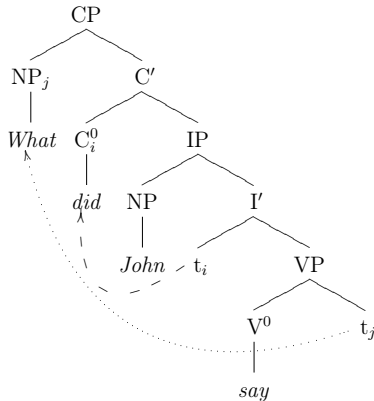
- Most commands in `xyling` have versions with a `k` at the end, which allow control over finer details of placement. See the documentation
- More generally, `xyling` can be used to typeset all sorts of structures with nodes and connecting lines
- It also includes a few handy macros not directly related for trees, for example for metrical grids.

More xyling

```

\Tree{ & \CP \\
\NP[j] && \Cbar \\
\T{What}& \Czero[i] && \IP \\
&\T{did} & \NP && \Ibar \\
& &\T{John}&\Trace
\Linkk[-->]{2.5}{11u}&&\VP \\
& &&\Vzero && \Trace[j]
\Linkk[.>]{4.5}{-3,-6}\\
& &&\T{say}&& }

```



The package `colortbl`

- Sometimes we need to shade the cells in a table (for example when we're doing OT tableaux using `tabular` or just for readability)
- A good way to do it, which is PDF \LaTeX -friendly and works with `ctable`, is provided by the package `colortbl`
- It provides an easier way to implement table shading than `colortab`
- Its main commands are `\columncolor{}`, `\rowcolor{}` and `\cellcolor`

The package `colortbl`

- The colour specifications are taken from the `color` package. There are many, but the intuitive approach won't get you into trouble (remember American spelling: `gray`, not `grey`)
- The command `columncolor` should be used after `>` in the column specification. It has the syntax
`\columncolor[colour name]{colour intensity}%`
`[left overhang][right overhang]`
- The overhangs regulate how far the coloured panel stretches with respect to the widest piece of text in the column. If nothing is given, the colours will be contiguous.

Example of colortbl

```
\begin{tabular}{>{\columncolor[gray]{.8}}1%  
>{\color{white}\columncolor[gray]{.2}}1}  
one & two \\  
three & four \\  
\end{tabular}
```

one	two
three	four

The package `colortbl`

- There is also the command `\rowcolor` with the same syntax. You just issue it at the start of a row
- Finally, there is a command `\cellcolor`, again with the same syntax. It's pretty obvious what it does
- You can also colour rules (but is that a good idea?). See the documentation for details.

TikZ

- PGF/TikZ is an extremely powerful graphics package. It can do only slightly less than PSTricks, but it is PDF \LaTeX -friendly and very human-readable
- As we do not have much time, we will only look at some very simple trees with TikZ
- We start with the first tree we did with `xyling`

TikZ basics

- To start using TikZ, we just say
`\usepackage{tikz}`
- The easiest way to starting drawing in TikZ is to use the `tikzpicture` environment (especially if we're drawing trees)
- A basic unit in the TikZ world is the **node**. The syntax for creating nodes is
`\node[drawing options] (name) {content} ;`
Mind the semicolon!
- As the syntax indicates, only the content is obligatory (but can be empty, i. e. `{}`)

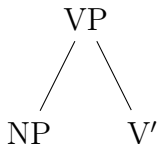
Trees with TikZ

- In TikZ, trees are built out of nodes, using the `child` command after the content and specifying the child as being a node
- So to have a VP node with NP and V' children, we say

```

\begin{tikzpicture}
\node {VP}
child {node {NP}}
child {node {V'$'$}} ;
\end{tikzpicture}

```



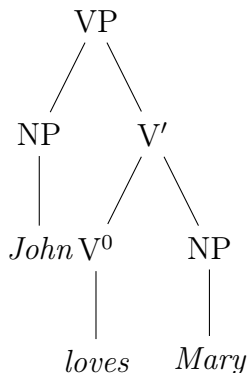
Trees with TikZ

- child statements can be nested:

```

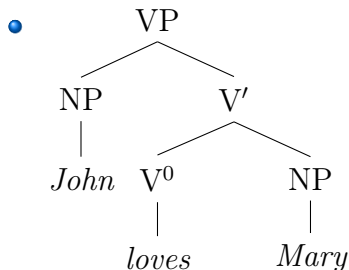
\begin{tikzpicture}
\node {VP}
  child {node {NP}
    child {node {\emph{John}} } }
  child {node {V$'$}
    child {node {V$^{0}$}
      child {node {\emph{loves}} }} }
  child {node {NP}
    child {node {\emph{Mary}} }
  } } } ;
\end{tikzpicture}

```



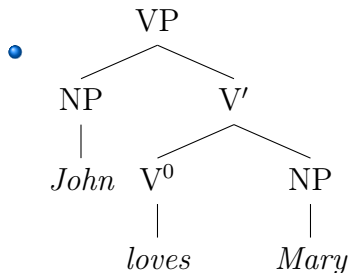
Trees with TikZ

- Now we need to make sure the branches are properly aligned and spaced. To do this, we issue options to the `tikzpicture` environment
- Let's try this:
`\begin{tikzpicture}[parent anchor=south,%
child anchor=north, sibling distance=7em]`



Trees with TikZ

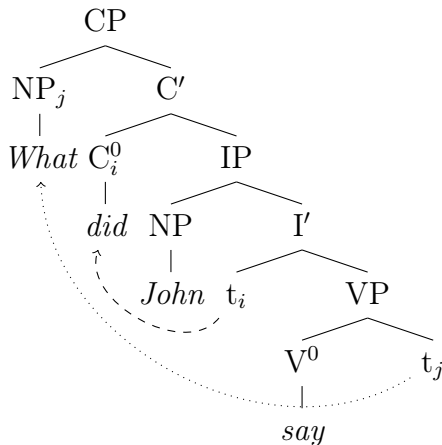
- This is better, but alignment is slightly off. To do this, we pass the options `text height=1.5ex, text depth=.25ex` to `\begin{tikzpicture}`



- Note how the lines aren't scraggy

Node naming with TikZ

- One useful feature in TikZ is the ability to give names to nodes and then use them in drawing. Take the following tree from the xyling example.



- To give a node a name, put a (name) between node and the content
- Now it is possible to draw a line from a node to a node using their names. For example, in that tree I have nodes called (tj) and (what)
- To draw a straight line:
`\draw[->,dotted] (tj) -- (what) ;`
- To draw a curved line:
`\draw[->,dotted] (tj) to [bend left=angle] (what);`

Node naming with TikZ

- TikZ is a very versatile tool with a huge manual, and well worth learning
- For example, it also has a `\matrix` which works like arrays or `xyling` trees
- You can also draw things like flowcharts and finite-state automata with TikZ, but there's too little space so see the manual

Other linguistic packages

- `OTtblx`: for OT tableaux (**PS only**)
- `pst-jtree`: a very powerful (syntactic) tree package (**PS only**)
- `pst-autoseg`: typesets autosegmental representations (**PS only** and quite difficult to use, as it's plain `TEX`)
- `avm`: for attribute-value matrices à la HPSG