

xyling – L^AT_EX macros for linguistic graphics
using the **xypic** module
version 1.1, July 31, 2006

Ralf Vogel
Institut für Linguistik, Universität Potsdam
rvogel@ling.uni-potsdam.de
<http://www.ling.uni-potsdam.de/~rvogel/xyling>

December 31, 2004

This file explains the usage of the tree drawing macros in **xyling.sty**. Please also inspect the style file which contains detailed comments of the macros. These macros build on the **xypic** package by Kristoffer H. Rose, version 3.7.

xypic is loaded by **xyling.sty** with the **dvips** option in order to be able to produce coloured branches. This has a bad side effect: The use of the **dvips** option might cause branches to disappear when you produce a pdf file with **pdflatex**. Likewise, other packages that use postscript, like **pstricks**, might have such side effects. If you experience this, there are two ways of overcoming it:

1. erase the **dvips** option in the **xypic** call in line 57 of **xyling.sty**, i.e., use: `\RequirePackage[color,all]{xy}`. You will no longer be able to use coloured branches now, however. Similarly, you might have to avoid using other postscript packages, too.
2. create the pdf file via **dvips** and **ps2pdf** conversion. This is how this documentation pdf file is created.

Load **xyling** with `\usepackage{xyling}`.

The macros in this package model the construction of syntactic tree struc-

tures as a genuinely graphical problem: they contain three types of objects:

- the *grid*
- the *node labels*
- the *branches*

The branches and nodes are positioned relative to the grid. It is essential that each of these three elements is constructed independent of the other two. They can be modified without unwanted side effects. This is, to my mind, the most important difference to other tree drawing macro packages, where, in particular, the label width often changes the angles of the branches.

A second major advantage to my mind lies in using the excellent **xypic** package. **xypic** generates METAFONT code, so the drawing of trees is not dependent on postscript specials, as, e.g., in Wolfgang Sternefeld’s **ps-trees**. Hence, no postprocessing is required. The trees generated by postscript specials can neither be seen in a dvi previewer (at least not in my current version of **xdvi** for linux), nor can they be taken over into a pdf file by **pdflatex** (but it works in dvi to pdf conversion with **dvipdf**, or ps to pdf conversion with **ps2pdf**).

1 The Grid

A tree is a simple macro of the form `\Tree{ ... }`. It is an **xymatrix**, with `&` as column separator and `\\` as row separator. You should be familiar with this structure from tabulars and similar structures.

`\Tree` has an optional argument that can be used for changing the space between columns. The default is 0, which does not actually mean 0 space, but rather the default space. A negative value makes the tree smaller, a positive value makes it wider:

$$\begin{array}{lll} A & B & = \backslash\mathrm{Tree}\{A \& B\} \\ A & & B & = \backslash\mathrm{Tree}[1]\{A \& B\} \\ A & B & = \backslash\mathrm{Tree}[-1]\{A \& B\} \end{array}$$

The distance between rows can also be controlled using the “k”ontrolled version of `\Tree`, `\Treek`, which has one more obligatory argument, the distance between rows, which is defaulted to 2pc in `\Tree`. Check the examples in table 1.

VP	$=$	$\backslash\text{Tree}\{ \& VP \backslash\backslash Vzero \&\& NP \}$
V	NP	
VP	$=$	$\backslash\text{Treek}\{1\}\{ \& VP \backslash\backslash V \&\& NP \}$
V	NP	
VP	$=$	$\backslash\text{Treek}\{3\}\{ \& VP \backslash\backslash V \&\& NP \}$
V	NP	
VP	$=$	$\backslash\text{Treek}\{4\}\{ \& VP \backslash\backslash V \&\& NP \}$
V	NP	

Table 1: Effects of the first argument of $\backslash\text{Treek}$

2 The Nodes

A node is introduced with the command $\backslash\text{K}\{\dots\}$. Text is centered, both horizontally and vertically! This means that text boxes might not be aligned on the same baseline in `xypic`, using its standard command for text insertion, $\backslash\text{txt}$. You can see this here:

muss musg must Muss Musg

The code:

```
\Tree{\txt{muss~~} \txt{musg~~} \txt{must~~}
\txt{Muss~~} \txt{Musg~~}}
```

The command $\backslash\text{K}$ contains a strut of 18pt to prevent this. This is perfect for up to 12pt fonts:

muss musg must Muss Musg

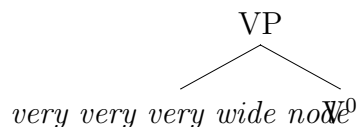
The code:

```
\Tree{\K{muss~~}& \K{musg~~}& \K{must~~}& \K{Muss~~}& \K{Musg~~}}
```

Node labels have the width 2pc, which is fine, because most node labels

consist of two upright letters, like NP, VP etc. Extremely wide labels do not affect the matrix, because the node width is determined independently of the content of the node – by the value `@W=2pc` in the definition of `\Tree` and `\Treek`.

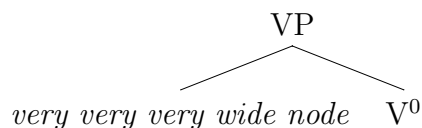
`\K` has an optional argument to move text freely horizontally – measured in pc. Positive values move the label to the right, negative values to the left. 1pc is about 0.5 em in a 12pt font. Horizontal adjustments are necessary for wide nodes:



The code:

```
\Tree{
      & \K{VP}      \B{dl}\B{dr}    \\
      \K{\emph{very very very wide node}} & & \K{V$^0$}  }
```

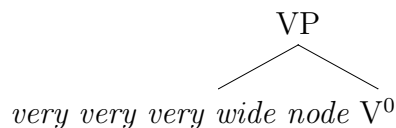
We have two solutions to this problem. The global one is widen the inter-column space with the optional argument of `\Tree`:



The code:

```
\Tree[1]{
      & \K{VP}      \B{dr}\B{dl}    \\
      \K{\emph{very very very wide node}} &      & \K{V$^0$}  }
```

The second solution is moving the wide node a bit to the left, using the optional argument of `\K`:



The code:

```
\Tree{
      & \K{VP}      \B{dr}\B{dl}    \\
      \K[-5]{\emph{very very very wide node}} & & \K{V$^0$}  }
```

`\Kk` offers full control over the vertical and horizontal dimension. The optional argument of `\Kk` is the same as that of `\K`, but there is an addi-

tional obligatory argument denoting the vertical adjustment. So the syntax is: `\Kk[horizontal adjustment]{vertical adjustment}{node label}`. The command `\Kk[1]{1}{VP}` places the node VP 1pc right of and 1pc higher than the standard position. This in fact allows you to put anything anywhere in the tree. You can add comments to the side of the tree etc.

Often, especially in the terminal nodes, you might need more than one line of text. Use the command `\Below` for putting text below text inserted by `\K`. Check the following example:

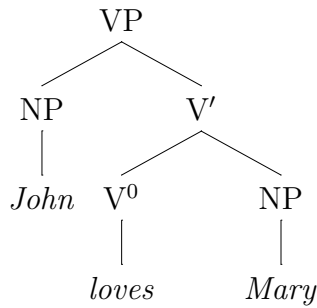
```
NP      \Tree{\K{NP}\B{d} \\  
|      \K{the}\Below{little}\Below{dog}}  
the  
little  
dog
```

`\Below` has an optional argument for the vertical adjustment, negative value lowers the text, positive value raises it. There also exists a controlled version, `Belowk`, which has the horizontal adjustment as its first obligatory argument in addition.

3 The Branches

Branch commands are inserted in the same field as their dominating node. The simplest command is `\B`. It has one obligatory argument, the description of the target field. It is given in the steps it takes to reach it. For instance, `\B{dl}` draws a branch to the field one step down and one step left from the current field. Use `u` and `r` for upwards and rightwards steps. Upwards branches will strike through the label, however, so this is not very useful. Branches should always go downwards.

We are now ready to draw our first tree:



This is the code:

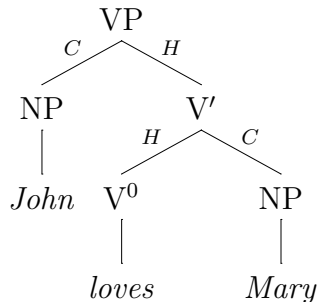
```

\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}}
}

```

`\B` has an optional argument, the vertical adjustment of the starting position of the branch, negative value, e.g. `\B[-1]{d}`, lowers the starting position, positive value raises it.

Some grammar formalisms, require labels on branches – like HPSG, where head and complement are explicitly denoted with the labels *H* and *C*. This is quite easy to integrate. The command `\B` is based on the `xypic` command `\ar` which has the option of putting a label on a line. It uses the syntax of subscripting and superscripting, i.e., `^{}` and `_{}` . If you put these commands right behind a branch command, you will get labels on the branch, `_{}` puts it on the branches left side, and `^{}` on its right. The label is written in math mode, i.e., slanted, unless you specify it to be in normal text font with, e.g. `\textnormal{}`. Here comes an example:¹



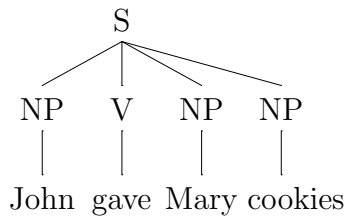
¹Thanks to Doug Arnold for his suggestion to include this feature.

```

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

```

There is no limitation on the number of branches either:

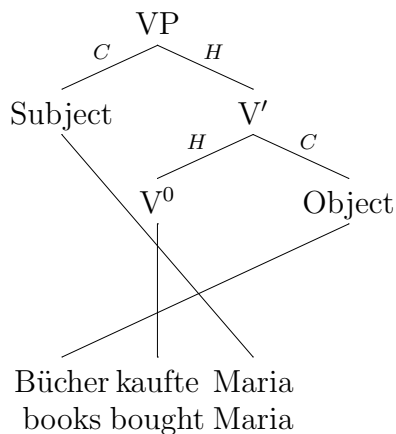


```

\Tree{
    & \K{S} \B{dl}\B{d}\B{dr}\B{drr}
    & \K{NP}\B{d} & \K{V}\B{d} & \K{NP}\B{d} & \K{NP}\B{d}
    & \K{John} & \K{gave} & \K{Mary} & \K[2]{cookies}
}

```

HPSG is also famous for crossing branches in cases where the linear order and the hierarchical structure do not match. You can, of course, also let branches cross. There are no system specific limits to what kinds of trees you might want to draw. An admittedly ugly looking example is the following German object initial sentence:

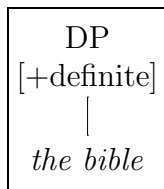


```

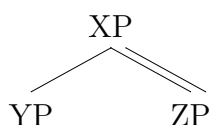
\Tree[.5]{
    & \K{VP}\B{dl}_C\B{dr}^H
    & \K{Subject}\B{ddrr} & \K{V$'}\B{dl}_H\B{dr}^C
    & \K{V$^0}\B{dd} & \K{Object}\B{ddl11}
    & \K{B\"ucher}\Below{books} & \K{kaufte} \Below{bought}
    & \K{Maria}\Below{Maria}
}

```

We also have two controlled versions of the branching command, `\Bkk` and `\Bk`. Both adjust the starting and target positions of branches. They have three arguments, 1: adjustment of starting position, 2: adjustment of target position, 3: destination. `\Bk` only allows for vertical adjustments and has numbers as its first two arguments. `\Bkk` has comma-separated number pairs as its first two arguments, for horizontal and vertical adjustment of starting and target position. This gives you full control over the branches. Some examples:



```
\fbox{\Tree{3}{\K{DP}\Below{[+definite]}\B[-5]{d}\ \ \ \K{\emph{the~bible}} } }
```



```
\Tree{ & \K{XP}\B{dl}\B{dr}\Bkk{2,0}{2,0}{dr} \ \ \K{YP}& & \K{ZP} }
```

4 Abbreviations

The usage of branching commands can be made much easier by the definition of macros for most frequently used combinations. These are the ones defined in `xyling.sty`. Define new ones if you need them and/or change those given here:

```
% \D goes to the position in the same column, one row below.
% The optional argument is the vertical adjustment of the
% starting position.
```

```
\newcommand{\D}[1][0]{\B[#1]{d}}
\newcommand{\Dk}[2]{\Bk{#1}{#2}{d}}
```

```
% \Dk is the \Bk version, i.e., it has two obligatory arguments
% defining the vertical adjustments of starting position and
% target position.
```

```
\newcommand{\DL}[1][0]{\B[#1]{dl}}
\newcommand{\DLL}[1][0]{\B[#1]{dll}}
\newcommand{\DLLL}[1][0]{\B[#1]{dlll}}
```



```

\newcommand{\DR}[1][0]{\B[#1]{dr}}
\newcommand{\DRR}[1][0]{\B[#1]{drr}}
\newcommand{\DRRR}[1][0]{\B[#1]{drrr}}

\newcommand{\DLk}[2][0]{\Bk[#1]{#2}{d1}}
\newcommand{\DLLk}[2][0]{\Bk[#1]{#2}{d11}}
\newcommand{\DLLLk}[2][0]{\Bk[#1]{#2}{d111}}
\newcommand{\DRk}[2][0]{\Bk[#1]{#2}{dr}}
\newcommand{\DRRk}[2][0]{\Bk[#1]{#2}{drr}}
\newcommand{\DRRRk}[2][0]{\Bk[#1]{#2}{drrr}}

% Abbreviations for binary branches:

\newcommand{\V}[1][0]{\DL[#1]\DR[#1]}
\newcommand{\VV}[1][0]{\DLL[#1]\DRR[#1]}
\newcommand{\VR}[1][0]{\DL[#1]\DRR[#1]}
\newcommand{\VL}[1][0]{\DLL[#1]\DR[#1]}
\newcommand{\VVV}[1][0]{\DLLL[#1]\DRRR[#1]}
\newcommand{\VVR}[1][0]{\DLL[#1]\DRRR[#1]}
\newcommand{\VVL}[1][0]{\DLLL[#1]\DRR[#1]}
\newcommand{\VLL}[1][0]{\DLLL[#1]\DR[#1]}
\newcommand{\VRR}[1][0]{\DL[#1]\DRRR[#1]}

% Define more if you need them.

% Let's make further abbreviations. A default VP has a
% \V branch. So let's define \VP accordingly, and likewise
% Vbar, IP, CP etc.:

\newcommand{\VP}[1][0]{\K{VP$_{#1}$}\V}
\newcommand{\Vbar}{\K[.5]{V$'$}\V}

\newcommand{\IP}[1][0]{\K{IP$_{#1}$}\V}
\newcommand{\Ibar}{\K[.5]{I$'$}\V}

\newcommand{\CP}[1][0]{\K{CP$_{#1}$}\V}
\newcommand{\Cbar}{\K[.5]{C$'$}\V}
\newcommand{\Nbar}{\K[.5]{N$'$}\V}

\newcommand{\PP}[1][0]{\K{PP$_{#1}$}\V}
\newcommand{\DP}[1][0]{\K{DP$_{#1}$}\V}

```

```

\newcommand{\AuxP}[1] [] {\K{IP$_{#1}$}\V}
\newcommand{\Auxbar}{\K[.5]{I$'}\V}

% Nodes reasonably without branch:

\newcommand{\NP}[1] [] {\K{NP$_{#1}$}}
\newcommand{\Nzero}[1] [] {\K[1]{N$^{0}_{#1}$}}
\newcommand{\Vzero}[1] [] {\K[1]{V$^{0}_{#1}$}}
\newcommand{\Izero}[1] [] {\K{I$^{0}_{#1}$}}
\newcommand{\Auxzero}[1] [] {\K{I$^{0}_{#1}$}}
\newcommand{\Czero}[1] [] {\K{C$^{0}_{#1}$}}
\newcommand{\Pzero}[1] [] {\K{P$^{0}_{#1}$}}
\newcommand{\Dzero}[1] [] {\K{D$^{0}_{#1}$}}

% Note that all of these commands, except for the
% Xbar nodes, have an optional argument for an index.
% If you use indices longer than one letter, you might
% need to do horizontal adjustment and have to use
% \K instead.

% This default system only needs new nodes for the
% terminals. Let's define \T parallel to \K.
% \T has a vertical branch to the upper node, text is
% italicised. The optional argument, as before,
% allows for horizontal adjustment of the text:

\newcommand{\T}[2] [0] {\POS[]\POS+(0,2)
\ar@{-}[u]-(0,4) \POS[]\K[#1]{\textit{#2}}}

```

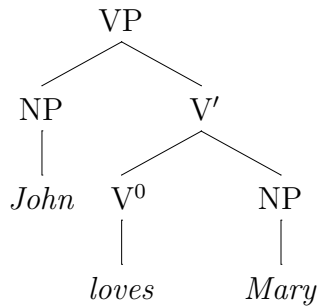
The code for the tree on page 5 can now be reduced dramatically:

```

\Tree {
      & \VP
      & \NP
      & \Vbar
      & \Vzero
      & \NP
      & \T{loves}
      & \T{Mary}
}

```

Here is the tree, just to assure you:



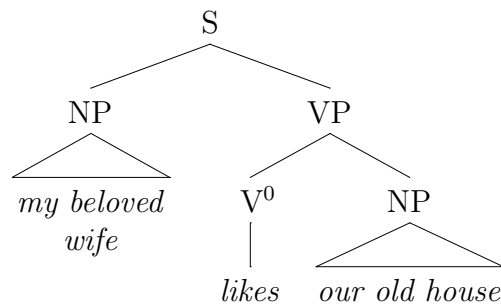
Chomskyan syntacticians like movement traces, here is one:

```
\newcommand{\Trace}[1][i]{\K[0.5]{t$_{#1}$}}
```

The optional argument, as with the `\XP`, `\Xzero` commands above, is the index, here defaulted to ‘i’.

5 Triangles

Phrase structure trees often use triangles as abbreviations for larger structures which shall not be fledged out in full detail. The command `\TRI` can be used instead of branching commands, introducing a triangle:²



²Notice that the top node in this tree, S, is inserted exactly in the middle between two columns – due to the horizontal adjustment of 5.2pc. The horizontal distance between the centers of two neighbour fields is 10.4pc. The command `\Bkk` is then used to get symmetric branches from this position, by adjusting the starting position horizontally by 5.2pc. You see that the label and branch placement are not limited to the possibilities specified by the grid. It only provides a usually satisfactory default.

```

\Tree{ && \K[5.2]{S}\Bkk{5.2,0}{0,0}{dl}
      \Bkk{5.2,0}{0,0}{drr}          \\\
      &\NP\TRi &&      &      \VP          \\\
& \K{\emph{my beloved}}\Below{\emph{wife}} &&
      \Vzero &&          \NP\TRi[2]      \\\
      &&& \T{likes} && \K{\emph{our old house}}
      }

```

Triangles need the nodes they refer to in the next lower row, in particular, don't forget the node [d] from the insertion point of the triangle. You will get an error message otherwise. The text below the triangle is inserted in the field [d] from the insertion point

Notice also that the first column in the tree is empty. It's only purpose is providing a field for the left bottom edge of the left triangle.

\TRi is a symmetric triangle. It has an optional argument, the horizontal adjustment, negative value makes it narrower, positive value makes it wider, as exemplified in the triangle for '*our old house*' in the tree above.

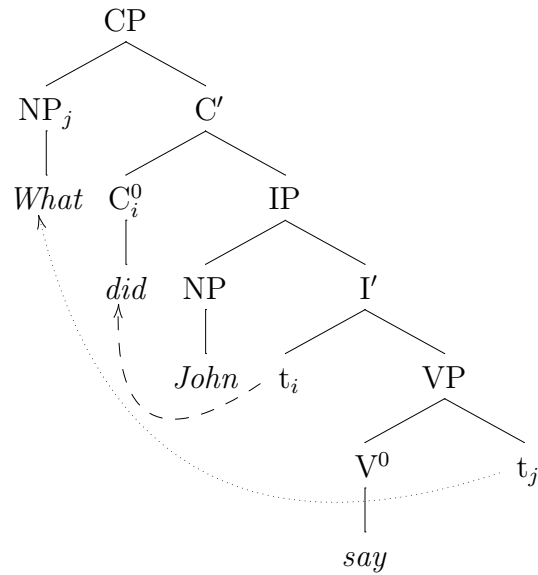
\DTRi and \TRiD are rectangles, \DTRi has a [d] branch on the left, \TRiD on the right, the optional argument allows for horizontal adjustment.

The controlled version of the triangle, \Trik, has two obligatory arguments for the left and right horizontal adjustment, which can be specified separately here. This potentially leads to asymmetric triangles. The syntax is:

\Trik{left}{right}, where \Trik{0}{0} = \TRi[0] = \TRi.

6 Links

It occurs very frequently that relations between elements of a tree shall be indicated with linking arrows, for instance to illustrate syntactic movement. Links are inserted with the commands \Link and Linkk. Both have as optional argument the line style, which could be one of -, ->, <-, ., .>, <., --, -->, <--, a.o. \Linkk has as its first argument the curving specification. It is defaulted to 1.33pc in \Link. This might not be sufficient in many cases. The second obligatory argument of \Linkk and the first obligatory argument of \Link is the destination field specification.

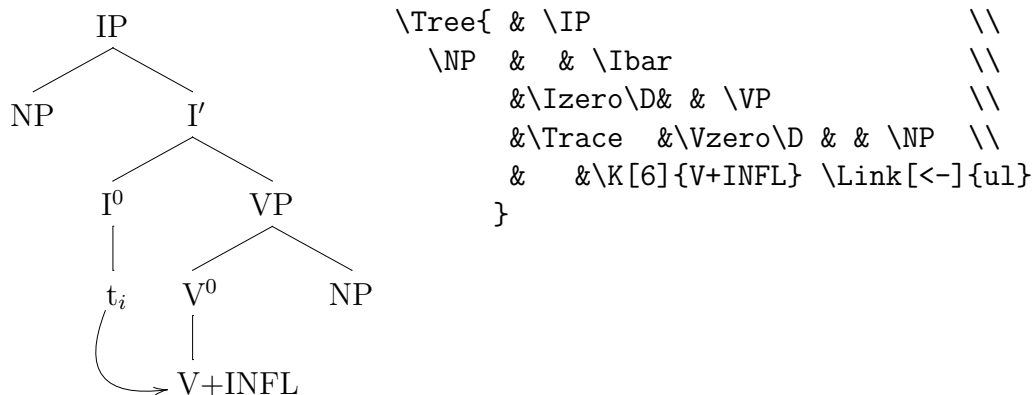


The code is as follows:

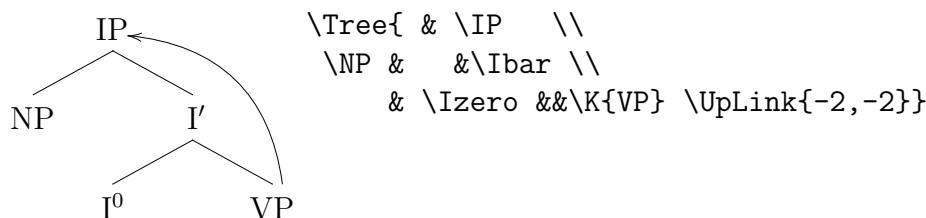
```
\Tree{      & \CP \\  
          \NP[j] && \Cbar \\  
          \T{What}& \Czero[i] && \IP \\  
              &\T{did} & \NP && \Ibar \\  
              &      &\T{John}&\Trace  
                  \Linkk[-->]{2.5}{11u}&&\VP \\  
              &      &&\Vzero && \Trace[j]  
                  \Linkk[.>]{4.5}{-3,-6}\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

Note that it is possible to specify the target location in terms of the number of ‘hops’, as done for the long link here: `\Linkk[.>]{4.5}{-3,-6}` states that the target of the link is 3 fields up and 6 fields left of the current field. This is shorter than specifying it with “uuu111111”.

English affix hopping according to Chomsky (1981):



Finally, the commands `\Link` and `\Linkk` link labels to the left of the tree. Links which are to the right of the tree, on its top, are specified with `\UpLink` and `\UpLinkk`:



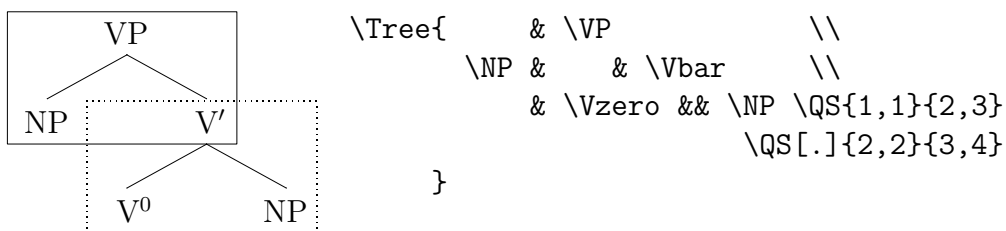
7 Highlighting

Node labels are often highlighted with surrounding circles or squares, and, in particular in presentations, with colours. Several ways of doing this are integrated in `xyling`.

7.1 Highlighting sections of a tree with a rectangle

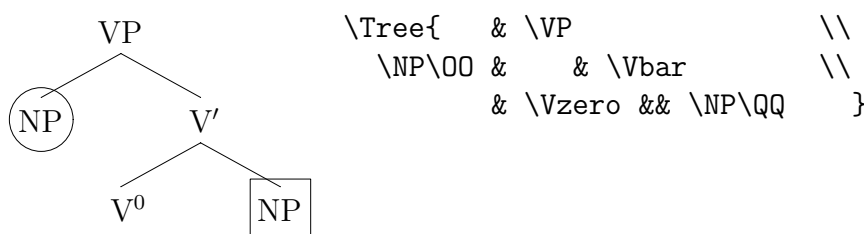
The command `\QS` draws a rectangle around two fields of the tree specified by their absolute address. The syntax of the command is:

`\QS[line style]{top left corner}{bottom right corner}`. The absolute address of a field is a number pair. The field “1,1” is the top left field of the grid. The fields in the first row are 1,1 1,2 1,3 etc. Those in the second row are 2,1 2,2 2,3 etc. Because the field addresses are given in absolute terms, it makes no difference where within the `\Tree` specification a `\QS` command appears:



7.2 Highlighting node labels with circles and squares

The command `\OO` draws a circle around the node label in the current field. The command `\QQ` draws a square around it:



As you can see, the branches go inside the circles and squares. We need to define branches which start and/or end at circles or squares.

The following are branches that always work. Starting point for all branches is the same point on the circle. This makes branches rather short, and changes the angles of diagonal branches:

```

\newcommand{\Bo0}[1]{\Bk{-1.2}{1.2}{#1}}
% branch with circled start and target

```

```

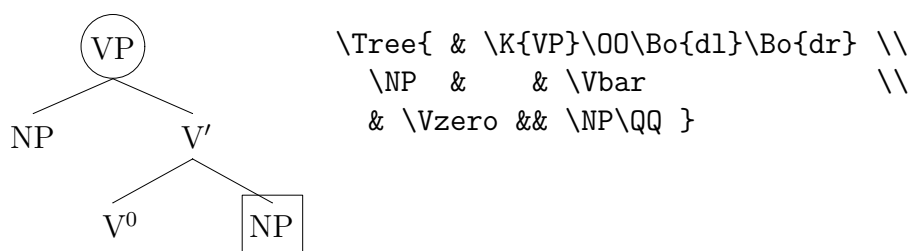
\newcommand{\Bo}[1]{\Bk{-1.2}{0}{#1}}
% branch with circled start

```

```

\newcommand{\B0}[1]{\Bk{0}{1.2}{#1}}
% branch with circled target

```



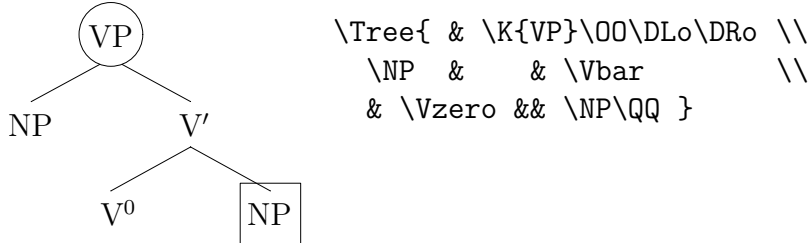
The two [dl] and [dr] branches are no longer parallel. To get this, the branches should start at different sides of the circle, as if they initiated at the old position within the circle:

```
\newcommand{\DRo}[1][dr]{\Bkk{1.6,-1}{0,0}{#1}}
\newcommand{\DLo}[1][dl]{\Bkk{-1.6,-1.0}{0,0}{#1}}

\newcommand{\DRo0}[1][dr]{\Bkk{1.6,-1}{-1.8,1}{#1}}
\newcommand{\DLo0}[1][dl]{\Bkk{-1.6,-1}{1.8,1}{#1}}

\newcommand{\DR0}[1][dr]{\Bkk{0,0}{-1.8,1}{#1}}
\newcommand{\DL0}[1][dl]{\Bkk{0,0}{1.8,1}{#1}}
```

This helps:



The same has to be done with branches to and from squares:

```
\newcommand{\DRq}[1][dr]{\Bkk{1.5,-1}{0,0}{#1}}
\newcommand{\DLq}[1][dl]{\Bkk{-1.5,-1}{0,0}{#1}}

\newcommand{\DRQ}[1][dr]{\Bkk{0,0}{-2,1}{#1}}
\newcommand{\DLQ}[1][dl]{\Bkk{0,0}{2,1}{#1}}

\newcommand{\DRqQ}[1][dr]{\Bkk{1.5,-1}{-2,1}{#1}}
\newcommand{\DLqQ}[1][dl]{\Bkk{1.5,-1}{2,1}{#1}}

\newcommand{\DRoQ}[1][dr]{\Bkk{1.5,-1}{-2,1}{#1}}
\newcommand{\DLoQ}[1][dl]{\Bkk{-1.5,-1}{2,1}{#1}}

\newcommand{\DRq0}[1][dr]{\Bkk{1.6,-1}{-1.8,1}{#1}}
\newcommand{\DLq0}[1][dl]{\Bkk{1.6,-1}{1.8,1}{#1}}

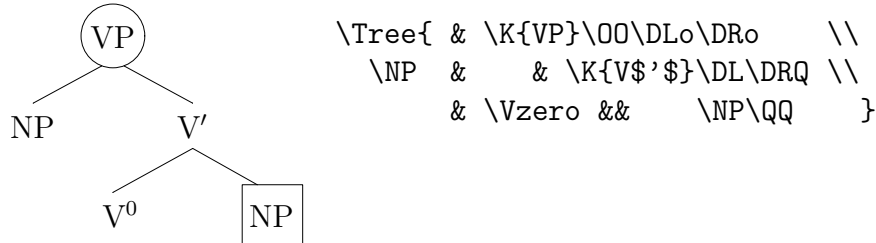
\newcommand{\Dq}{\Dk{-1.1}{0}}
% only squared starting node
```



```

\newcommand{\DQ}{\Dk{0}{1}}
% only squared target node
\newcommand{\Dq0}{\Dk{-1.1}{1.2}}
% squared start and circled target
\newcommand{\DoQ}{\Dk{-1.3}{1}}
% circled start and squared target
\newcommand{\DqQ}{\Dk{-1.1}{1}}
% square start and squared target

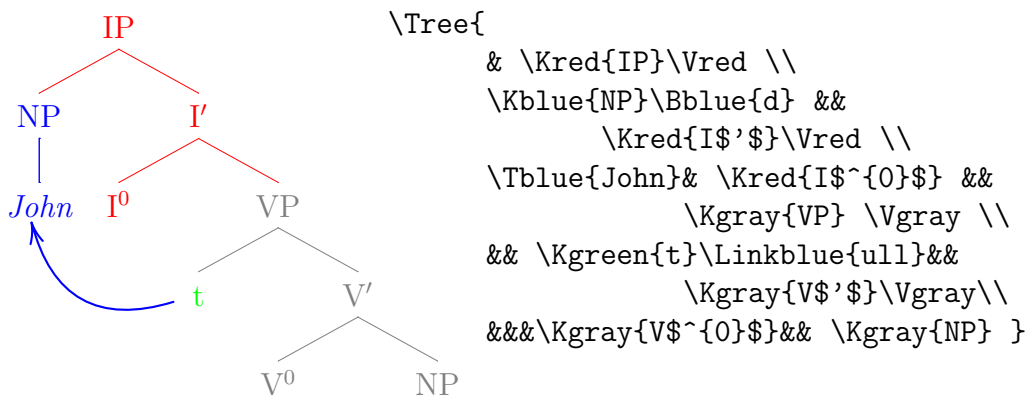
```



The style file contains many abbreviations for the branch commands including circles and squares around node labels.

7.3 Highlighting with Colours

Especially in presentations, colours are an option for highlighting parts of a tree. `xypic` uses the postscript extensions to make this possible. Check the `xypic` documentation for more on colours, especially for defining new colours. I defined blue, red, green and gray versions of the standard commands `\Bkk` and `\Kk` and the commands derived from them. So it is quite painless to integrate colour:



There also are macros for coloured circles and squares around nodes. They are called `\OOred`, `\QQred` etc. But they do not fully function yet. They

have to be inserted as first element within a field, right after the $\&$, and must be followed by an empty group, $\{\}$.

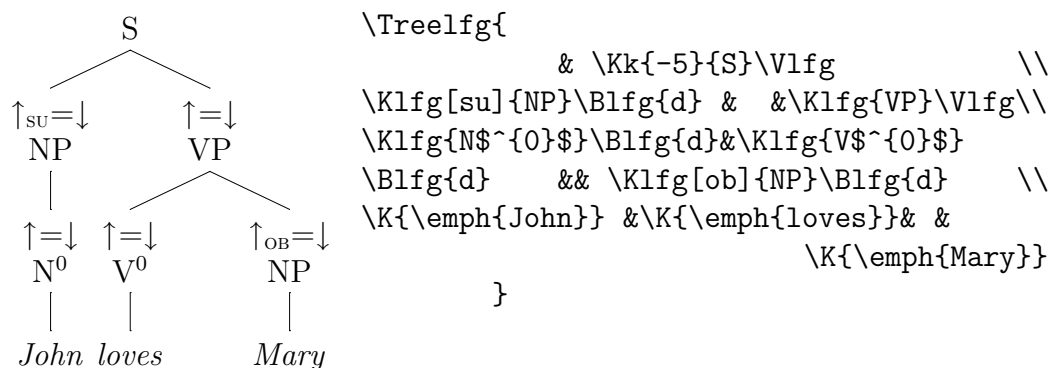
8 Other Structures

8.1 Lexical Functional Grammar

On the basis of the general commands for grids, labels and branches, it is easy to define commands for other phrase structure formats, or any kind of graphical object that mixes text and lines. The following macros help drawing LFG trees:

```
\newcommand{\Treelfg}[2][0]{\Treek[#1]{3}{#2}}
\newcommand{\Klfg}[2][]{%
  \K{$\uparrow$ _{\textsc{\scriptsize #1}}=\downarrow$}\Below{#2}}
\newcommand{\Blfg}[1]{\Bk{-5}{0}{#1}}
\newcommand{\Vlfg}{\Blfg{dl}\Blfg{dr}}
```

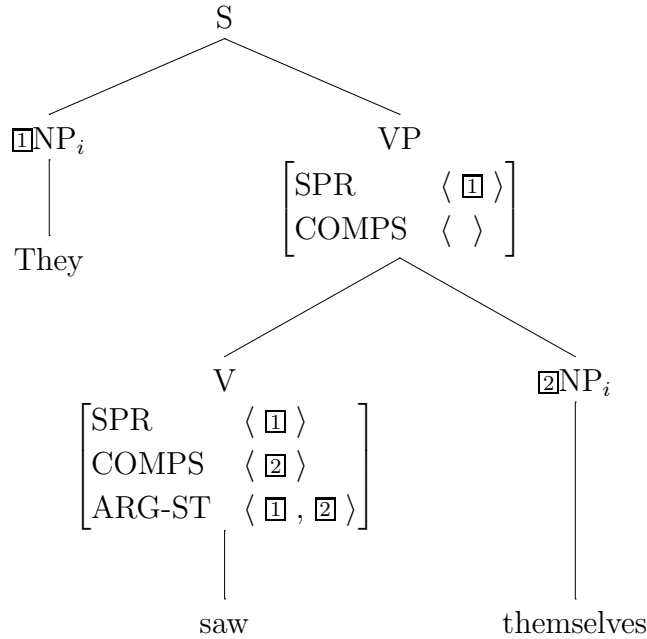
Note that $\backslash\text{Klfg}$ produces two lines: The inheritance specification and the category label. The optional argument is the grammatical function, which, if available, is indexed to the \uparrow in the first line. Here comes an example:



Much of this can be abbreviated in the manner exemplified above. Feel free to do so. Command names like $\backslash\text{Klfg}$ etc. could, of course, also be shortened.

8.2 Head Driven Phrase Structure Grammar

The following structure is from the HPSG introduction book “Syntactic Theory. A Formal Introduction” (second edition, 2003) by Sag, Wasow & Bender, page 207:



How did we draw the tree? First, we use Christopher Manning’s avm.sty to define the AVM structures of each node separately:

```

\newcommand{\AVMone}{\begin{avm}\@1NP$_{i}$\end{avm}}
\newcommand{\AVMtwo}{\begin{avm}
    \[    SPR & \q< \@1 \q> \\\
          COMPS & \q<~ \q> \] \end{avm}}
\newcommand{\AVMthree}{\begin{avm}
    \[    SPR & \q< \@1 \q> \\\
          COMPS & \q< \@2 \q> \\\
          ARG-ST & \q< \@1 , \@2 \q> \] \end{avm}}
\newcommand{\AVMfour}{\begin{avm}\@2NP$_{i}$\end{avm}}

```

Next, we make up the tree, and call the four AVM nodes within \K or \Below. The most tricky issue is the determination for the tree widths and how to deal with the unequal sizes of the nodes. As you see in the code below, I exploited three features of xyling: setting row and column distance to a new

value, adjusting branches, and the huge nodes VP and V were extended over two rows!

```

\Tree[3]{3}{   & \K{S}\V \\                                % adjust grid!
\Kk{-1}{\AVMone}\D && \K{VP}
                  \Below[-4]{\AVMtwo}\B[-13]{ddl}\B[-13]{ddr}\\ % Branches go ddr/ddl!
\K{They}          \\ % skipped row for VP!
                  & \K{V}
\Below[-6]{\AVMthree} && \Kk{-1}{\AVMfour}\B{dd}\\ % Branch goes dd!
&\K{}\D[-1]          \\ % Branch starts here!
&\K{saw}              && \K{themselves}}

```

Note an incompatibility of avm.sty: When I first experimented with xyling.sty and avm.sty, the linebraks in my avm’s disappeared. It took me a while to figure out that this was not due to xyling, but caused by the tabularx package which I had loaded for another purpose. So, if you use avm.sty, don’t use tabularx!

8.3 Arrows in Text

Even when we do not use trees, but only strings or bracketed notations, we sometimes want to express relations between elements within these notations by arrows. It is possible to use xyling macros for this, too. We first define a new grid, \TExt. It has an obligatory argument – the text, which can be split into several fields by the field separator &. The distance between fields is set to 0, such that even parts of words can be put in subsequent fields without visible effect:

```


noodles noodles \TExt{\textnormal{nood} & \textnormal{les} &
                      \textnormal{~~} & \textnormal{noodles}}

```

The fields within the \TExt environment are inserted with \Txt or \TXT. They mainly provide a strut to yield correct horizontal alignment. \Txt has a blank before the text, \TXT has none. I.e., if a word is split over two fields, the second part of the word should be introduced with \TXT. Likewise, the first word of the whole string.

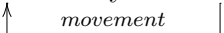

The arrows differ in their depths, such that several arrows can be used within one \TExt without overlapping. The narrowest connector is called SCOn, we then have \COn, \COnn and \COnnn. They have an optional argument for the line style, – for solid, -- for dashed, . for dotted. The commands are

inserted at the starting point of the arrow. The obligatory argument is the target field destination, e.g. [11], [r] etc. A simple example:

What did you see t ?	<code>\TExt{\TXT{What} & \Txt{did you see} & \Txt{t}\COn{11} & \Txt{?}}</code>
	

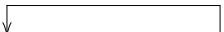
The command `\COnk` allows for control of the link depth with its first obligatory argument. The other arguments are as before.

We also have connectors with labels, which can be above or below the arrow. This depends on whether the arrow goes from left to right or vice versa. `\COnl` with right to left yields a label above the arrow:

What did you see t ?	<code>\TExt{\TXT{What} & \Txt{did you see} & \Txt{t}\COnl{11}{movement} & \Txt{?}}</code>
	
What did you see t ?	<code>\TExt{\TXT{What} & \Txt{did you see} & \Txt{t}\COnr{11}{movement} & \Txt{?}}</code>
	

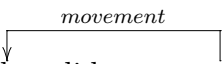
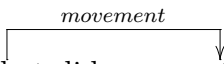
8.3.1 Arrows over the text: Added by gina nov 06 2006

To make the arrows appear above the text change "COn" to "OVr" in the above examples, copied below.

What did you see t ?	<code>\TExt{\TXT{What} & \Txt{did you see} & \Txt{t}\OVr{11} & \Txt{?}}</code>
	

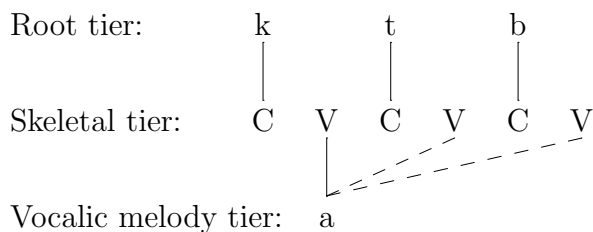
The command `\OVrk` allows for control of the link depth with its first obligatory argument. The other arguments are as before.

We also have connectors with labels, which can be above or below the arrow. This depends on whether the arrow goes from left to right or vice versa. `\OVrl` with right to left yields a label above the arrow:

What did you see t ?	<code>\TExt{\TXT{What} & \Txt{did you see} & \Txt{t}\OVrl{11}{movement} & \Txt{?}}</code>
	
What did you see t ?	<code>\TExt{\TXT{What}\OVrr{rr}{movement} & \Txt{did you see} & \Txt{t} & \Txt{?}}</code>
	

8.4 Autosegmental Phonology

Not being an phonologist, I can only offer some beginner's applications, but I will gladly add macros that you might supply. Just send them to me. Here is one application that is based on the previously introduced `\TExt` matrix. It is not limited to having just one row, and one application of this feature are the structures used in autosegmental phonology. Have a look at this:



The code uses the controlled matrix for text, `\TExtk`, with inter-column space in pc as mandatory argument, and inter-row space in pc as optional argument. The command `\S1` provides the slots of the tier. At the beginning of each row, the tier's name is given with the macro `\Tname` which has as optional argument the name width in cm. Links between slots are variants of our branch commands, `\Ln` draws a solid line, and `\Lnd` a dashed line. `\Lnu` is an abbreviation for `\Ln{u}`. Define more macros, if you need them. By the way, the example above is Arabic for 'he/she wrote'. This is the code:

```

\TExtk[1]{1}{
\Tname[5]{Root tier:}&
\Sl{k}      &      & \Sl{t}      &      & \Sl{b}      &      \\
\Tname[5]{Skeletal tier:}&
\Sl{C}\Lnu&\Sl{V}&\Sl{C}\Lnu&\Sl{V}&\Sl{C}\Lnu&\Sl{V}\\
\Tname[5]{Vocalic melody tier:}&
& \Sl{a}\Lnu\Lnd{urr}\Lnd{urrrr}
}

```

8.5 Graphical Schemata – Arrows instead of lines

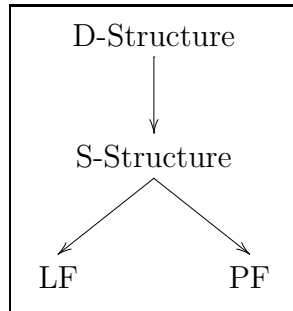
The branching command is based on the `xypic` arrow command `\ar`. It can do a lot more than just drawing solid lines. It can, for instance, draw arrows. As this is sometimes useful, we integrate an arrow command, `\AR`:

```

\newcommand{\ARkk}[3]{\POS[]-(0,4)+(#1)\ar@{->}[#3]+(0,2)+(#2)}
\newcommand{\ARK}[3]{\ARkk{0,#1}{0,#2}{#3}}
\newcommand{\AR}[2][0]{\ARK{#1}{0}{#2}}

```

This helps us draw the GB schema:



The code is rather small:

```

\fbbox{
\Treek[.5]{3}{
& \K{D-Structure}\AR{d}      \\
& \K{S-Structure} \AR{dl}\AR{dr}\\
\K{LF} &      & \K{PF}      }
}

```

We can also use alternative line styles, like dashes, dots, and waves:

```

\newcommand{\Bdot}[2][0]{\POS[]-(0,4)\POS+(0,#1)%
& \ar@{.}[#2]+(0,2)}
\newcommand{\ARdot}[2][0]{\POS[]\POS-(0,4)\POS+(0,#1)%
& \ar@{.>}[#2]+(0,2)}

\newcommand{\Bdash}[2][0]{\POS[]-(0,4)\POS+(0,#1)%
& \ar@{--}[#2]+(0,2)}
\newcommand{\ARDash}[2][0]{\POS[]\POS-(0,4)\POS+(0,#1)%
& \ar@{-->}[#2]+(0,2)}

```

```

\ar@{-->}[#2] + (0,2)}

\newcommand{\Bwav}[2][0]{\POS[] - (0,4)\POS+ (0,#1)%
\ar@{~}[#2] + (0,2)}
\newcommand{\ARwav}[2][0]{\POS[] \POS- (0,4)\POS+ (0,#1)%
\ar@{~>}[#2] + (0,2)}

```

Joost Kremers, University of Frankfurt/Main, <j.kremers@em.uni-frankfurt.de>, has kindly defined a “generalised branch” command that I happily integrate here:

```

\newcommand{\GBkk}[4]{\POS[] - (0,4) + (#1)\ar@{#4}[#3] + (0,2) + (#2)}
\newcommand{\GBk}[4]{\GBkk{0,#1}{0,#2}{#3}{#4}}
\newcommand{\GB}[3][0]{\GBk{#1}{0}{#2}{#3}}

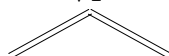
```

The branch commands are the same as the `\Bkk`, `\Bk` and `\B` commands, except that they have one more mandatory argument, and the final mandatory argument is the line style, which can be as follows:

- solid line
- dashed line
- = double line
- == dashed double line
- . dotted line
- : dotted double line
- ~ wavy line
- ~~ dashed wavy line

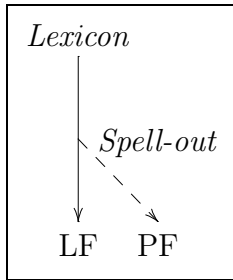
To yield arrows, you only need to extend the line style with leading or final `<` or `>`, as e.g. in `\GB{=>}{dl}`.

```

VP      \Tree{ & \K{VP}\GB{dl}{=}\GB{dr}{=}\ & }


```

The minimalist syntax model is often schematised with a dashed arrow to PF:



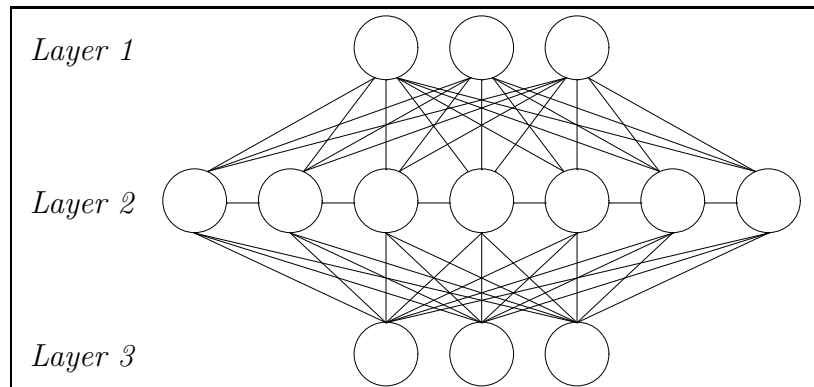
```
\fbox{
\Treek{2.5}{
\K{\emph{Lexicon}}\AR{dd}  \\\
\ARdash[3]{dr} & \K{\emph{Spell-out}}\\\
\K{LF}      & \K{PF} } }
```

It might sometimes be useful to have branches and arrows to the field on the right in the same row. The command `\R` makes this possible. The syntax is: `\R[line style]{destination}`. If you want to, you can write phrase structure rules that way ... ☺ ... :

$S \rightarrow NP \ VP$

```
\Tree[-1]{\K{S}\R[->]{rr} && \K[5]{NP~~VP}}
```

Versions of `\R` that link circles and squares are also defined in `xyling.sty`. Another funny application are neural nets:



```
\Treek[.5]{4}{
\K[-2]{\emph{Layer 1}}&&&
\OO \DLLoO\DLloO \DoO \DRoO\DRRoO\DRRRoO\DRoO[drrrr]&
\OO \DLLLoO\DLLoO\DLloO \DoO \DRoO\DRRoO\DRRRoO &
\OO \DLloO[dllll]\DLLLoO\DLLoO\DLloO \DoO \DRoO\DRRoO \\\
\K[-2]{\emph{Layer 2}}&
\OO \BoO{drr}\BoO{drrr}\BoO{drrrr} \RoO{r} &
\OO \BoO{dr}\BoO{drr}\BoO{drrr} \RoO{r} &
\OO \BoO{d}\BoO{dr}\BoO{drr} \RoO{r} &
\OO \BoO{dl}\BoO{d}\BoO{dr} \RoO{r} &
\OO \BoO{dll}\BoO{dl}\BoO{d} \RoO{r} &}
```

```

\00 \Bo0{d111}\Bo0{d11}\Bo0{d1} \Ro0{r} &
\00 \Bo0{d1111}\Bo0{d111}\Bo0{d11} \\\
\K[-2]{\emph{Layer 3}} &&\00 & \00 & \00 }

```

Compare the two different connection styles between layers 1,2 and 2,3, respectively.

9 Some Extras

These are two handy macros for constructing metrical grids and glosses. They have nothing to do with `xypic`. I include them because I find them useful.

Metrical Grids The command `\PX` has one obligatory argument, the piece of text which is to receive grid marks, and, as optional argument, the number of grid marks, defaulted to 1:

```

      ×
×      ×
×      ×
Farah Fawcett Majors \PX[2]{Fa}rah \PX{Faw}cett \PX[3]{Ma}jors

```

`\PX` has center alignment of text and mark, for left and right alignment use `\PXl` and `PXr`. These macros require the `ifthen` package.

Glosses For linguistic glosses, I normally use `cgloss4e.sty`. But when I want to gloss only one or two words within an example, `cgloss` is not that handy, as you must make as many glosses as there are words within an example, which leads to a sequence of empty groups. `\PG` has two arguments for the word and its gloss:

```

John loves Mary \PG{John}{\textsc{nom}} loves
NOM ACC \PG{Mary}{\textsc{acc}}

```

`\PG` aligns word and gloss at the left edge, for center and right alignment use `\PGc` and `\PGr`.

For bug reports, comments and suggestions, contact:
rvogel@ling.uni-potsdam.de