

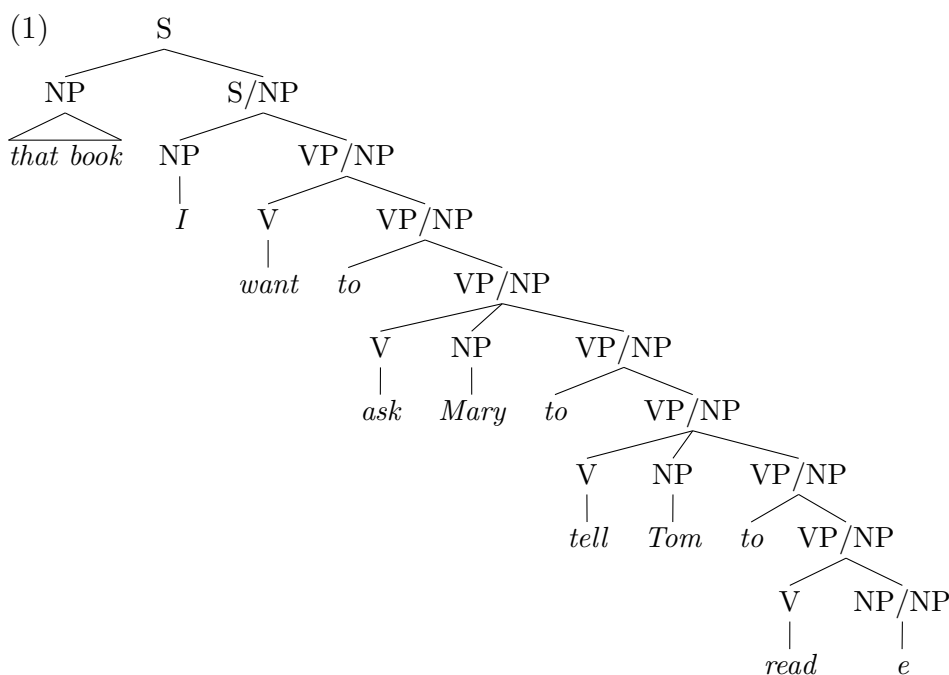
Linguistics Trees Preprocessor

Avery Andrews

v. 3.0 (`pst-nodes` based)

March, 2006

Consider the tree shown in (1) drawn from Newmeyer (1986) following GKPS:



This would require some drama to do by hand, but the `LingTrees` package provided here makes it reasonably easy.

The package consists of a preprocessor that converts an indented list tree format into \LaTeX input (plain \TeX doesn't seem to work), and some macro files which lay out the trees and draw the lines. There are currently two possibilities for the line-drawing, one a partial emulation and extension of Emma Pease's `tree-dvips` based on the powerful graphics package `PSTricks` (the default), and the other the original `tree-dvips`, in case there is a problem with using `PSTricks`. Some of the more advanced features won't work with `tree-dvips`; these are noted as they come up below.

This package is probably most suited for irregularly-branching trees with wide node-labels, as found in LFG or HPSG. Some other packages worth looking at are `pst-jtree` (John Frampton), `qtree` (Jeffrey Siskind and Alexis Dimitriadis), `xytree` (Koaunghi Un), and for unusual situations, the `pst-tree` module of PSTricks (which is oriented towards tree-like structures in mathematics, rather than linguistics, and can do wierd things such as change the direction of the branching in the middle of the tree).

1 Setup

The current version of the preprocessor is a Python script `trees.py`.¹ For Windows (98 thru XP), there is a free-standing executable with installer that doesn't require Python to be installed that can be downloaded from <http://arts.anu.edu.au/linguistics/People/AveryAndrews/Software/latex>. Most Linux distros and later versions of MacOS will have an adequate version of Python included (the preprocessor works for at least Python 1.5.2 thru Python 2.4.2). Otherwise, Python will have to be installed. Instructions for installing and running `trees.py` under various OS's can be found in the file `00_readme.txt`.

At the above URL there are also links to the 'TreeRunner' GUI for running the preprocessor and other L^AT_EX components, which I find to be a useful alternative to setting up a `make` file for small projects.

The macro files are `lingtrees.sty`, `pst-tree-dvips-emu.sty` and `trees-without-pstricks.sty`; they will have to be put wherever you put custom T_EX/L^AT_EX files (`c:\localtexmf\tex\latex` for MikTeX). You'll also need the PSTricks package (including the `pst-nodes` module), and `tree-dvips` if you're planning to use that, both available on CTAN and included in many T_EX/L^AT_EX distributions. If you have to install either of these yourself, note that they include files with the extension `.pro` defining special commands for PostScript; these must be put somewhere where the `dvips` program can see them (`localtexmf/dvips` in MiKTeX).

¹The original was written in C; a slightly modified version of the source can be found on Chris Manning's (La)TeX web-page <http://nlp.stanford.edu/~manning/tex/>.

2 Basic Usage

The document needs to load the `lingtrees.sty` style file, with a line like this before the `\begin{document}`:

(2) `\usepackage{lingtrees}`

This will load up the PSTricks-based implementation, to use the old `tree-dvips`-based one instead, use `lingtrees-without-pstricks` instead.

In the document, a tree begins with `. [`, ends with `.]`, and has each node represented by a line of text, where nesting is indicated by increasing depth of embedding, so that sisters must be indented equally, i.e.:

```
(3) . [
      S
        NP
          John
        VP
          V
            bit
          NP
            Fido
      . ]
```

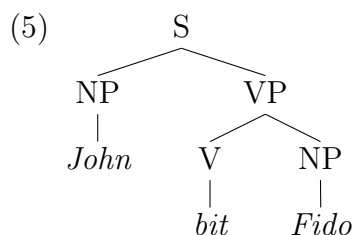
Blank lines are not allowed, and error messages will be triggered by various kinds of discrepancies, although there are plenty of ways of getting the wrong output without producing an error message, so the results need to be checked carefully.

After running the preprocessor, you run \LaTeX . Many current `.dvi` viewers can then show the lines, at least in simple cases, but:

- (4) a. some drawn lines get wrongly obliterated by other material
b. some things aren't positioned correctly

So if you really want to be sure that you're seeing how it's going to be, you'll need to run `dvips` and look at it with a postscript viewer. The TreeRunner GUI can launch these automatically.

The result of the above tree is:



A complete file of this example can be found in `examples/simple.txp`.

3 Work Methods

The method used above is to have your source document be say `paper.txp`, and run the preprocessor every time you change the text. I personally don't tend to do things this way, since I find it alarming to have anything overwriting big files with the extension `.tex`. So an alternative method is to put all of the trees in a file of their own, with a name ending in `_trees.tex`, and then use the preprocessor's `.>` directive to send the output of the next tree to the filename coming after the `.>` (on the same line). So that if for example the preprocessor encounters the line `.> trees/firsttree.tex`, then the following tree gets written to the file `trees/firsttree.tex` (and the material in between the `.>`-line and the start of the tree, but nothing after the tree). Then you use a `\input trees/firsttree.tex` statement in your document to include the tree (I find it tidy to put them in a subfolder). This works nicely if you're the sort of person who rarely gets around to setting up makefiles for writing projects (since you only have to run the preprocessor when you change a tree, which is really not that often, in practice). The technique is also useful for keeping trees synchronized between a paper and a handout or presentation. To send more than the next tree to a specific file, use `.>> [filename]`; output will then go to file name until either the next `.>>` directive, or a `.<<` directive, which reverts the output to the default output file.

These possibilities are illustrated in `examples/outputfiles_trees.txp`, which makes the trees for `examples/outputfiles.tex`. Using `.>` after `.>>` will produce an error message. Summing up the preprocessor directives, they are:

- (6) .[: begin a tree
- .] : end a tree
- .- : continuation line (see below)
- .> *file* : write next tree to *file*
- .>> *file* : write following material until .<< to *file*
- .<< : stop writing to file

4 Line-drawing Options

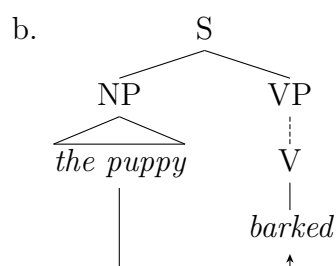
Line-drawing options are separated from the node label and each other by a vertical bar (‘|’). There are currently five basic ones which will work with either the PSTricks or `tree-dvips` as the line-drawer, the last three mutually exclusive:²

- (7) a. `tag TAG`, where TAG is any alphabetic string, sets TAG as the tag of the node, for later use by custom line-drawing commands.
- b. `tri` connects the node to its mother with a ‘triangle of laziness’.
- c. `none` causes the line to the mother to be omitted
- d. `dash` causes the line to the mother to be dashes
- e. `dot` causes the line to the mother to be dotted its mother

Here’s a sample illustrating some of the options:

²And due to Chris Manning.

(8) a. .[
 S
 NP
 the puppy|tri|tag a
 VP
 V|dot
 barked|tag b
 .]
 \abarnodeconnect ab



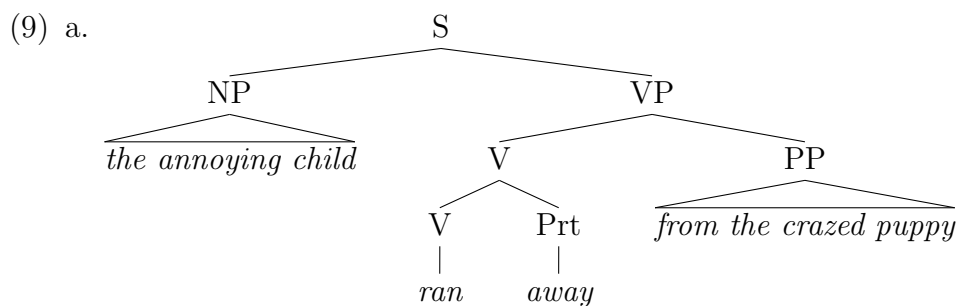
Some more advanced possibilities will be illustrated later.

5 Node-Building Options

There are various options for controlling the spacing and style of nodes.

5.1 Spacing

Sometimes the default spacing produces not-so-good results. For example the vanilla input (b) below produces the too-wide-and-spread-out result (a):

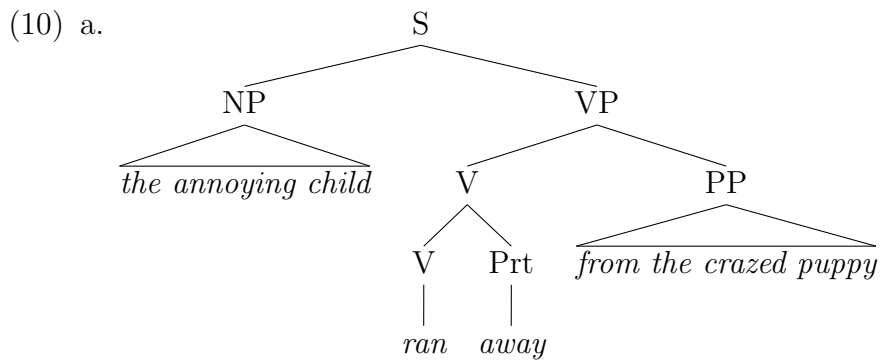


```

b. .[
    S
    NP
    the annoying child|tri
    VP
    V
    V
    ran
    Prt
    away
    PP
    from the crazed puppy|tri
.]

```

We can make it look better by increasing the vertical mother-daughter spacing by setting the `\daughterskip` dimension, and decreasing the horizontal spacing between sisters with the `\sisterskip` dimension:



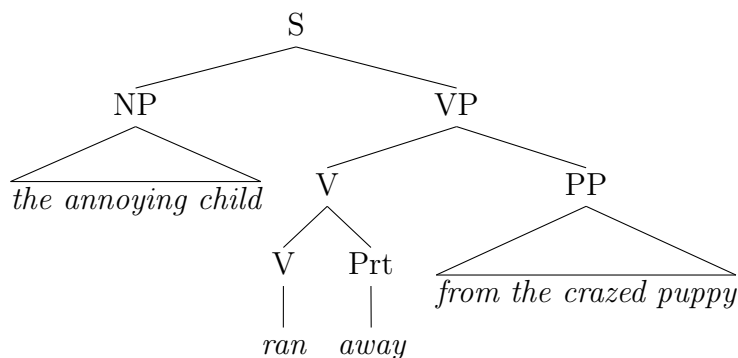
```

b. \daughterskip=3ex
   \sisterskip=1em
   .[
   S
   NP
   the annoying child|tri
   VP
   .
   .
   .
   .]

```

This is a bit better, but the triangles are still too flat, and there's a bit too much space between the subject NP node and the top V. These problems can be addressed with the `\daughtergap` command, placed somewhere in a node to set the vertical spacing to its daughters, and the `\sistergap` command, placed somewhere in a node to set the spacing between it and its *preceding* daughter:

(11) a.

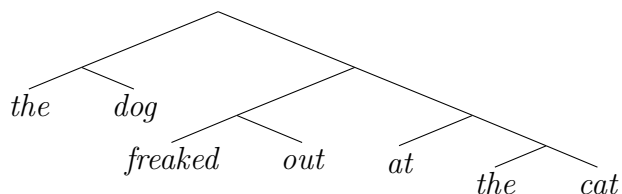


b. `\daughterskip=3ex`
`\sisterskip=1em`
`. [`
`S`
`NP\daughtergap{4ex}`
`the annoying child|tri`
`VP\sistergap{0em}`
`V`
`V`
`ran`
`Prt`
`away`
`PP\daughtergap{5ex}`
`from the crazed puppy|tri`
`.]`

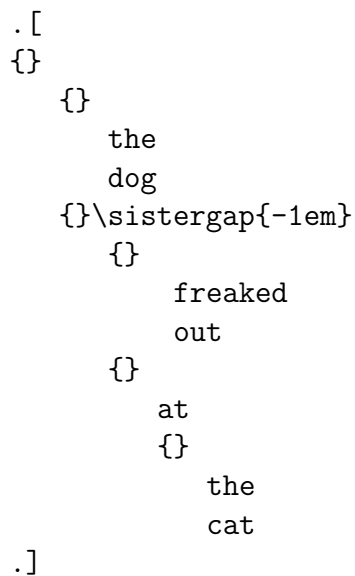
5.2 Nodes without Labels

Sometimes we want at least some nonterminal nodes to lack labels. Such trees look rather bad unless the lines all make constant angles, which can be effected with the `\treeslanratio{rise/run}` command:

(12) a.



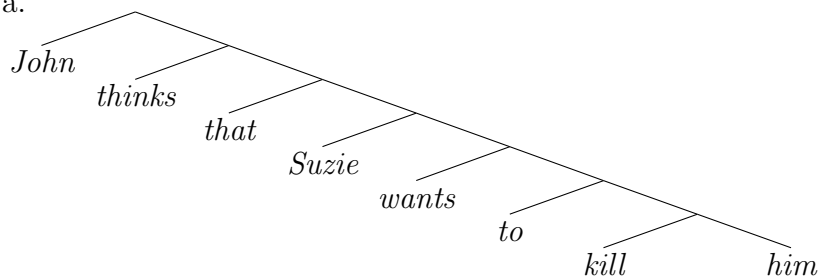
b. `\treeslantratio{4/10}`



This works by seizing control of the vertical spacing, so that `\daughterskip` ceases to have any effect.

Some trees look also look better with the length of the lines kept mostly uniform, which can be achieved by setting the `\nodewidth` dimension, which causes all nodes to be treated as having the value of that dimension as their width:

(13) a.

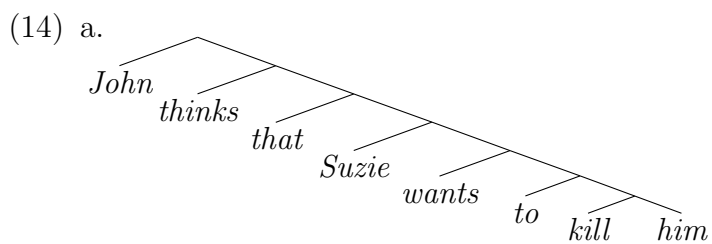


```

b. \nodewidth=6em
   \treeslantratio{7/20}
   \sisterskip=1em
   .[
   {}
   John
   {}
   thinks
   {}
   that
   {}
   Suzie
   {}
   wants
   {}
   to
   {}
   kill
   him
   .]

```

To my eye, the final branch looks a bit too wide; we can fix this by using the `dtwidths` preprocessor command to set a different width for the daughters of some of the nodes near the bottom. It takes a bit of fiddling to make things look nice:



```

b. \nodewidth=5em
   \treeslantratio{7/20}
   .[
   {}
   John
   {}

```

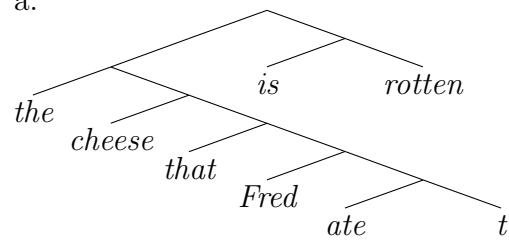
```

    thinks
    {}
      that
      {}
        Suzie
        {}|dtwidths 4.5em
          wants
          {}|dtwidths 3.5em
            to
            {}|dtwidths 3em
              kill
              him
      .]

```

Fixed width nodes can easily result in labels overlapping each other; one way to deal with this is to emulate the `pst-jtree \longer` command by putting in a right-branch with a `nolabel` node and no line:

(15) a.



```

b. \hspace{4em}
   \nodewidth=5em
   \treeslantratio{7/20}
   .[
   {}
     {}
       {}
         the
         {}
           cheese
           {}
             that
             {}
               Fred
               {}
                 ate
                 t
       {}|none
     {}
       is
       rotten
   .]

```

5.3 Stacking

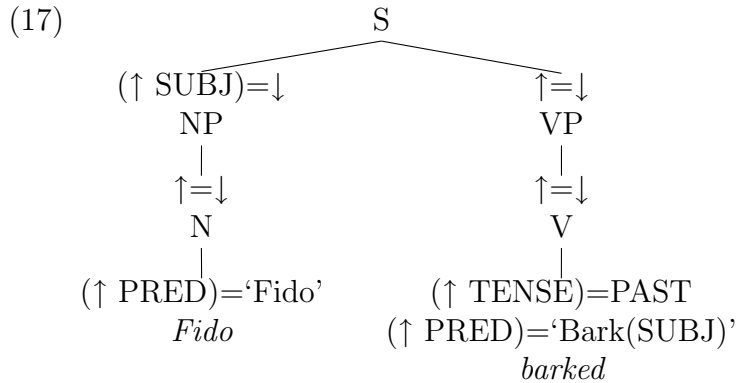
For writing LFG papers, there is a facility for ‘stacking’ labels and annotations, by separating the things to be stacked with the ‘/’ symbol. So for an annotated tree we can write:

```

(16) .[
      S
      ($\uparrow$ SUBJ)=$\downarrow$/NP
      $\uparrow$=\downarrow$/N
      ($\uparrow$ PRED)='Fido'/Fido
      $\uparrow$=\downarrow$/VP
      $\uparrow$=\downarrow$/V
      ($\uparrow$ TENSE)=PAST/($\uparrow$ PRED)='Bark(SUBJ)'/barked
    .]

```

which produces:



To include / in the text, enclose it in braces.

5.4 Node Styles

The default is that the bottom line of a terminal nodes is set in italics, other things in roman. If you want the terminals to be, say slanted rather than italics, you can do this by redefining `\tstyle` as follows: `\def\tstyle{\sl}`. Similarly using `\rm` if you want them to be set in roman. The nonterminal style can be changed with the same kind of redefinition of `\ntstyle`.

A more profound effect on nodes (which didn't used to work right) can be achieved by redefining the `\nodeannotation#1{}` command, which is wrapped around each line in a stack (the same for terminal and nonterminal). The default definition is:

(18) `\gdef\nodeannotation#1{#1}`

But if you want to set each line of each node in math mode, without having to write in lots of `$`'s, you can do it by redefining `\nodeannotation` as follows:

(19) `\def\nodeannotation#1{\$#1\$}`

For a more likely use, which can involve a new preprocessor feature as well, suppose you want to set trees whose nodes are avms, as produced by Christopher Manning's `avm.sty` file. If all of the nodes are going to be avms, we can spare ourselves the annoyance of typing out lots of `\begin{avm}` and `\end{avm}` commands by redefining `\nodeannotation`:

(20) `\def\nodeannotation#1{\begin{avm}#1\end{avm}}`

6 PSTricks-based techniques

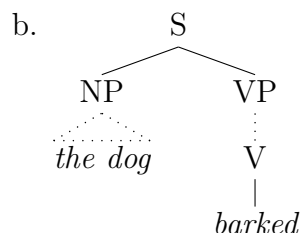
The use of PSTricks enables a substantial assortment of further possibilities. `lingtrees.sty` uses the `pst-nodes` module of PSTricks via a partial emulation of `tree-dvips`, provided by `pst-tree-dvips-emu.sty`, which has its own included documentation `pst-tree-dvips-emu.doc.pdf`. In combination with `lingtrees`, you can combine techniques from the emulator (somewhat more convenient for certain linguistics tasks) and PSTricks itself (much more powerful).

6.1 More Line-Drawing Options

The easiest is `dotted`, which gives PSTricks-style round dots instead of very short dashes:

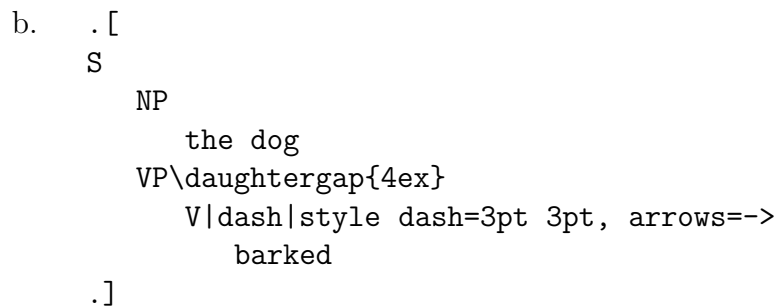
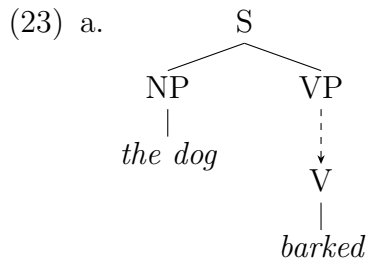
(22) a.

```
. [
  S
    NP
      the dog|tri|dotted
    VP
      V|dotted
        barked
  . ]
```

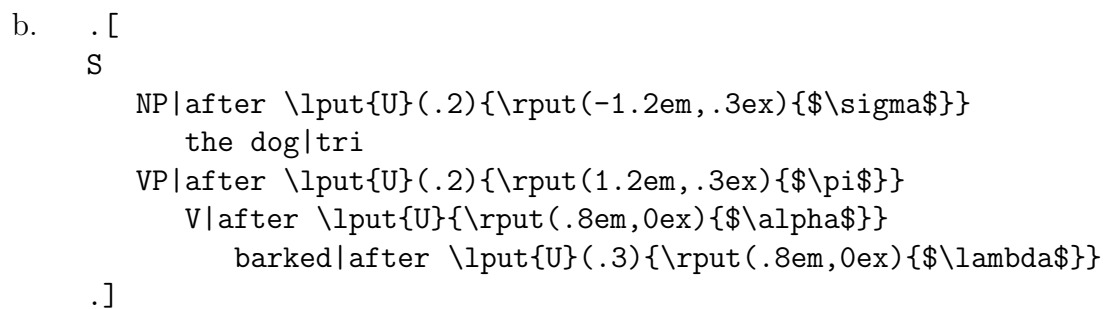
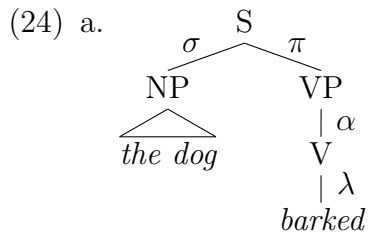
b. 

Dots seem to be almost invisible with the usual `lingtrees` line thickness, so they're set a bit thicker with the dimension `treedotwidth`, which is set to 0.6pt in `lingtrees.sty`.

Any `\psset`-able property of the connecting line can be set with the `style` option: what comes after 'style' must be an OK argument of `\psset`. For example:



The `after` option is followed by any well-formed string of \LaTeX text, which appears in the output after the node connection command. This is especially useful for using the `PSTricks` label-positioning commands, which appear after the command drawing the lines they are to attach labels to. Here's an example where we use these to attach Greek letter labels to some of the lines:



It is a bit tricky to get the positions right, but I think that it's easier to position them relative to the lines than to the nodes.

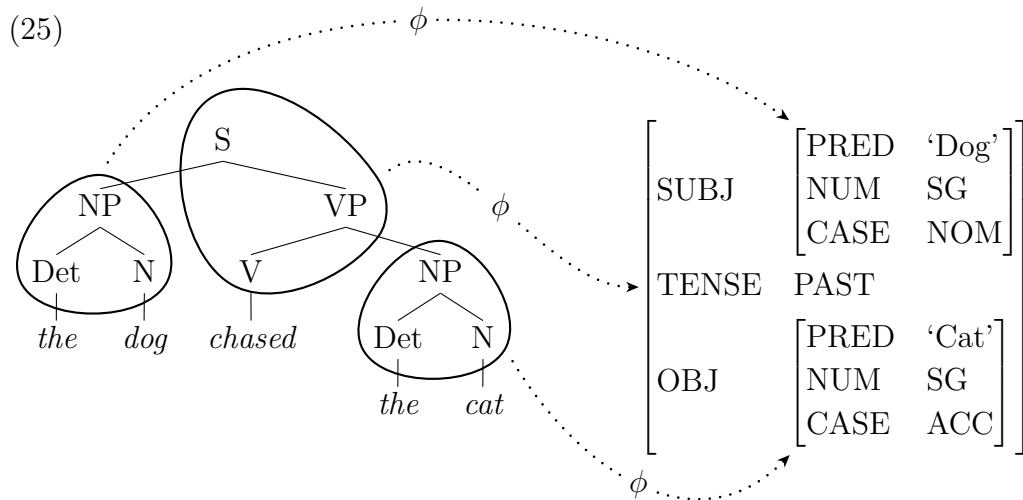
My `.dvi` viewer (Yap) doesn't get the positions of things placed by `lput` right, so check the results in postscript if things look wrong.

The `none` and `after` options together can be used to entirely replace the line-drawing command that the preprocessor normally produces, but a slightly more convenient option is `connect`, whose following material is put in place of the usual line-drawing command (`\nodeconnect`), in front of the mother and daughter arguments. So with a `connect` option such as `\nodeconnect<\pccoil>`, (see `pst-tree-dvips-emu.doc.pdf`), you could replace a normal line with a coil.

A final preprocessor option is `&|width&`, followed by a dimension, which sets the width of a particular node, rather than its daughters.

6.2 Together with PSTricks

A major reason for using PSTricks as the basis is to be able to combine the powerful facilities of the latter with the convenience of the tree-formatter and the `tree-dvips` commands it uses. For example, one can produce something like this, where PSTricks allows loops to be drawn around sets of nodes, and labels to be attached to arrows:



This was produced as follow. First, the tree, with labels placed on many of the nodes (also some preliminary stuff to help the diagram fit into the space on the page):

```
(26) \makebox[0ex]{}\[5ex]
      \hspace*{-1.8em}
      \sisterskip=1.3em
      .[
      S|tag s
        NP|tag su
          Det|tag sud
            the
          N|tag sun
            dog
        VP|tag vp
          V|tag v
            chased
          NP\sistergap{2.5em}|tag ob
            Det|tag obd
              the
            N|tag obn
              cat
      .]
```

And then some commands to place points at positions relative to the corners of the nodes, and run closed curves through those points. These begin with the PSTricks `SpecialCoor` command, which allows a wide range of ‘Special Coordinate’ specifications to be used, such as, here, the name of a node in parentheses as the specification of a point location:

```
(27)  {\SpecialCoor
      % loop around subject
      \putpoint{sutop}{1ex}(90)[t]{su}
      \putpoint{sunrt}{1ex}(-40)[r]{sun}
      \putpoint{sudlt}{1ex}(220)[l]{sud}
      \psccurve(sutop)(sunrt)(sudlt)
      % loop around S, VP V
      \putpoint{stop}{2ex}(120)[t]{s}
      \putpoint{vl}{1ex}(220)[l]{v}
      \putpoint{vpr}{1ex}(-40)[br]{vp}
      \putpoint{vprb}{1ex}(40)[tr]{vp}
      \psccurve(stop)(vl)(vpr)(vprb)
      % loop around object
      \putpoint{obtop}{1ex}(90)[t]{ob}
      \putpoint{obnrt}{1ex}(-40)[r]{obn}
      \putpoint{obdlt}{1ex}(220)[l]{obd}
      \psccurve(obtop)(obnrt)(obdlt)
```

The difference between `\putpoint` and the PSTricks command `\uput` is that `\putpoint` defines the ‘corners’ (tl, tr, br, bl) and ‘edges’ (t, r, b, l) of the point, which are all the same location, but allows the `pst-tree-dvips-emu.sty` commands `\(a)nodeconnect` and `\(a)nodecurve` to work, and sets the default direction of incoming arrows for the latter in accord with the standard conventions of `tree-dvips`.

So next we produce the f-structure, done with Chris Manning’s `avm.sty` package. In `avm.sty`, the `\!{lab}{stuff}` command is an abbreviation inside `avm`’s for `\node{lab}{stuff}`, and it works when the nodes are implemented by `pst-tree-dvips-emu.sty` as well as original `tree-dvips.sty`:

```
(28) \hskip 2em
      \begin{avm}
      \!{f}{\[SUBJ & \!{subj}{\[PREL & ‘Dog’\}
          NUM & SG\}
```

```

        CASE & NOM\]]\\
TENSE & PAST\\
OBJ & \!{obj}{\[PRED & 'Cat'\}
        NUM & SG\\
        CASE & ACC\]} \]}
\end{avm}

```

And so all that remains is to draw the arrows from the loops to the f-structures (representing the c-structure- f-structure correspondence). One could use the the PSTricks `\pccurve` command for this, but `\anodecurve` is more convenient, although sometimes we want to override the default angles for the corners and the edges. After each connection comes a `\mput*` command to label the arrow.

```

(29) % properties of phi arrows
      \treelinewidth=1pt
      \psset{linestyle=dotted}
      % arrow to whole f-structure
      \anodecurve[tr]{vprb}[1]f
      \mput*{ $\phi$ }           %label for this
      % arrow to the SUBJq
      \anodecurve[t]{sutop}[t1]{subj}[angleA=60,angleB=145]
      \mput*{ $\phi$ }
      \anodecurve[br]{obnrt}[b1]{obj}(.9)[angleA=-60]
      \mput*{ $\phi$ }
    }

```

I wish I could say it was easy to produce diagrams like this with this method, but that would be an untruth. However it is possible, and isn't ridiculously hard.

7 Availability

This package may be found at <http://arts.anu.edu.au/linguistics/people/AveryAndrews/Software/LaTeX>. I plan to submit it to CTAN fairly soon.