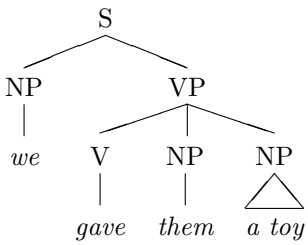
The `parsetree` Package for Drawing Trees in LaTeX

Doug Arnold
(LaTeX for Linguists)

July 2003

This is my favoured and recommended method for drawing standard linguistic trees where the nodes have no more than three daughters. It is very simple, and interacts well with other packages. Here is a simple example.




```
\begin{parsetree}
    ( .S.
        (.NP.  'we')
        ( .VP.
            (.V.    'gave' )
            (.NP.   'them')
            (.NP. ~ 'a toy')
         )
    )
\end{parsetree}
```

Apart from information on basic usagebasic usage (section 1) you will find information about the use of the package for more complex examples, and use in conjunction with `avm.sty`, and `tree-dvips`) for drawing really rather complicated thingscomplicated things (section 2).

The package is available from the usual archive sites, but I have not been able to find documentation other than what is in the source code. So I have made this up myself. My apologies to the author (Eirik Hektoen) for misrepresentations.

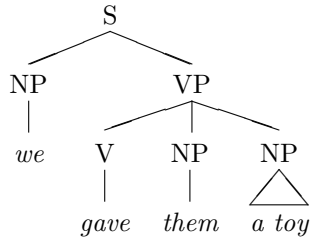
This documentation is available in printed form: PostScript and DVI. The style/package file parsetree.sty is also available here.

1 Basic Use

In the preamble, put:

```
\usepackage{parsetree}
```

In the text put:



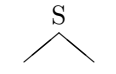

```
\begin{parsetree}
    ( .S.
        (.NP.  'we')
        ( .VP.
            (.V.    'gave' )
            (.NP.   'them')
            (.NP. ~ 'a toy')
         )
    )
\end{parsetree}
```

Basic usage is as follows:

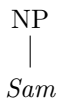
- `\begin{parsetree}`...`\end{parsetree}`
- A (sub-) tree consists of a "(", a node label, some daughters, and a ")".
- A terminal node consists of just the node label. Terminal node labels are written in quotes: `'baby'`, `'gave'`
- Non-terminal node labels are surrounded by dots, e.g. `.S.`,
- Putting a `~` before a node label puts it under a triangle, rather than a vertical line.
- Node labels can be arbitrarily complex (see examples below).
- *Warning*: no more than **three** daughters are allowed per node.
- *Warning*: don't use "(", or ")", or "`.`" or "`~`" inside node labels (see below on how to avoid this restriction).
- The following commands can be used to vary the appearance of trees, see below for examples of their use.

`\pthorgap`	horizontal gap between sisters (default 12pt)
`\ptvergap`	vertical gap between mother/daughter (default 12pt)
`\ptnodefont`	font and strut height/depth of non-terminal nodes
`\ptleaffont`	fond and strut height/depth of leaves

Here are some more simple examples:




```
\begin{parsetree}
(.S. .NP. .VP.)
\end{parsetree}
```



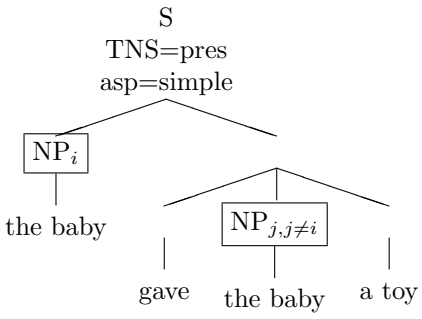

```
\begin{parsetree}
(.NP. 'Sam')
\end{parsetree}
```

2 Interaction with other packages

Node labels can be arbitrarily complicated. Just to give you an idea, the following examples show nodes containing arrays, co-indexing subscripts, and frame boxes, and nodes without node lables; nodes labelled with feature structures (made with the `avm.sty` package); node labels containing round brackets and other characters that `parsetree` treats specially; and nodes with lines linking remote parts of the tree (made with the `tree-dvips` package).

2.1 Boxes, Subscripts, Unlabelled Nodes

An example with some nodes boxed, and some nodes unlabelled (just put nothing between the '.' and '.'):




```
\begin{parsetree}
    ( .\begin{tabular}{c}S\\TNS=pres\\asp=simple\end{tabular}.
        (.\fbox{NP$_i$}. .the baby.)
        ( . .
            (. . .gave. )
```

```
            (.\fbox{NP$_{j , {j \neq i}}$}.
                .the baby. )
            (. . .a toy.)
        )
    )
\end{parsetree}
```

2.2 AVMs on Nodes

For making trees labelled with AVMs (Feature Structures), a useful hint is to first use `\newcommand` to define commands to draw the AVMs (this means you can check they are correct before you try to combine them into a tree), and makes the tree much easier to read.

We define three commands to draw the AVMs: `\fsA`, `\fsB`, and `\fsC`, e.g.

```
\newcommand{\fsB}{\begin{avm}
           \sort{f}{\[ cat & det \\
                       agr & \@1   \\
                       def & \@2   \] }
                 \end{avm} }
```

One can, of course test these just by putting:

```
        \fsA
        \fsB
        \fsC
```

To draw the tree, we put these commands in the node labels, preceded by some commands to adjust the space between nodes (see below):

$$f\begin{bmatrix} \text{cat} & \text{np} \\ \text{agr} & \boxed{1} \begin{bmatrix} \text{num} & \text{sing} \\ \text{per} & \text{1st} \\ \text{gender} & \text{masc} \end{bmatrix}_{\text{agr}} \\ \text{def} & \boxed{2} + \end{bmatrix}$$

$$f\begin{bmatrix} \text{cat} & \text{det} \\ \text{agr} & \boxed{1} \\ \text{def} & \boxed{2} \end{bmatrix} \qquad f\begin{bmatrix} \text{cat} & \text{n} \\ \text{agr} & \boxed{1} \end{bmatrix}$$

```
\begin{parsetree}
        \pthorgap{75pt}
        \ptvergap{40pt}
```

```
    ( .\fsA .
          .\fsB .
          .\fsC .
    )
\end{parsetree}
```

This example uses the `parsetree` declarations: `\pthorgap` and `\ptvergap`:

- The gap between sisters is controlled by the `\pthorgap` declaration (default is 12pt)
- The vertical distance between mother and daughters is controlled by the `\ptvergap` declaration (default is 12pt).

These can be set globally, or anywhere within individual `parsetree` environments.

You can also alter the font and height/depth allowed for non-terminal node labels and leaves with `\ptnodefont` and `\ptleaffont`. The following are the defaults:

```
\ptnodefont{\normalsize\rm}{11pt}{3pt}  % font and strut height/depth: nodes
\ptleaffont{\normalsize\it}{11pt}{3pt}  % font and strut height/depth: leaves
```

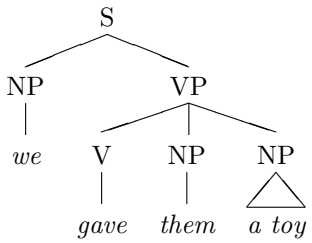
2.3 Special Characters in Node Labels

`Parsetree` treats some characters specially (e.g. round brackets, dot, tilde), using them as commands to draw trees. This can be a problem if you want these characters in your trees. However, there is a simple solution.

The `parsetree` environment is simply a wrapper that: (a) makes these special characters 'active' (i.e. special), and (b) calls the commands `\ptbegtree` `\ptendtree` at the beginning and end of the environment respectively. If we use these latter commands directly, then these special characters retain their normal meanings, and can appear in node lables. However, we must now use the underlying `parsetree` commands to draw the tree:

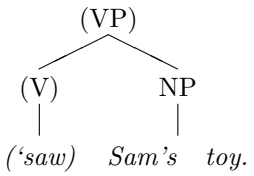
- replace "(" with `\ptbeg`, and ")" with `\ptend`
- replace `.N.` by `\ptnode{N}`.
- replace `'N'` by `\ptleaf{N}`.
- replace `~` with `\pttritrue`

Here is our original example:




```
\ptbegtree
    \ptbeg \ptnode{S}
        \ptbeg \ptnode{NP}  \ptleaf{we} \ptend
        \ptbeg \ptnode{VP}
            \ptbeg \ptnode{V}    \ptleaf{gave}  \ptend
            \ptbeg \ptnode{NP}   \ptleaf{them}  \ptend
            \ptbeg \ptnode{NP} \pttritrue  \ptleaf{a toy} \ptend
         \ptend
    \ptend
\ptendtree
```

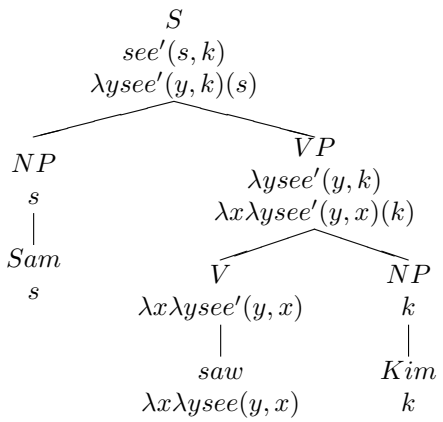
And here is an example, with 'special' characters thrown in:




```
\ptbegtree
  \ptbeg \ptnode{(VP)}
      \ptbeg \ptnode{(V)}    \ptleaf{(‘saw)}  \ptend
      \ptbeg \ptnode{NP}     \ptleaf{Sam’s~~~toy.} \ptend
   \ptend
\ptendtree
```

Here is a more complicated and realistic example (to simplify things, I have defined `\npile` to produce a node label consisting of an array — of course, with something this complicated, you would probably want to simplify things even further, but this is just for exemplification).

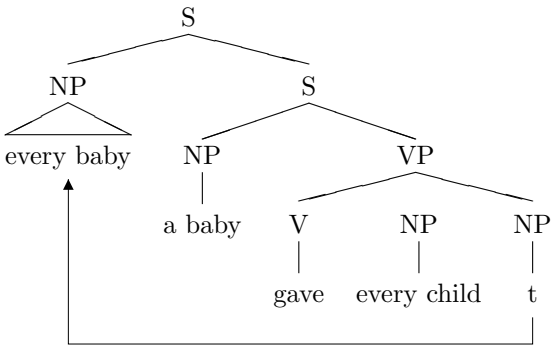
```
\newcommand{\npile}[1]{%
  \ptnode{               %
    \( \begin{array}{c}#1%
       \end{array}       %
    \)   }}
```




```
\ptbegtree
 \ptbeg
 \npile{S\\
        see' (s, k)\\
        \lambda y see' ( y,k ) (s)}
    \ptbeg
    \npile{NP\\s}
      \npile{Sam\\s}
    \ptend
    \ptbeg
    \npile{VP\\ \lambda y see' ( y,k )\\
                 \lambda x \lambda y see' ( y,x ) (k)}
            \ptbeg
            \npile{V\\ \lambda x \lambda y see' ( y,x ) }
                \npile{saw\\ \lambda x \lambda y see ( y,x )}
            \ptend
            \ptbeg
            \npile{NP\\k}
                  \npile{Kim\\k}
            \ptend
     \ptend
\ptend
\ptendtree
```

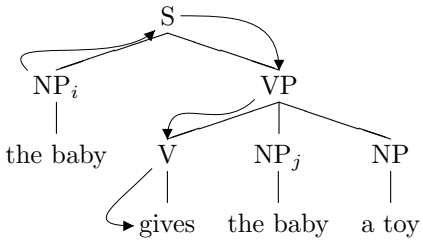
2.4 Non-Local Lines

The `tree-dvips` package can be used to draw 'non-local' lines, as in the following example, supposed to show something like "Quantifier Raising":




```
 \begin{parsetree}
( .S.
  (.NP. ~ .\node{1}{\strut every baby}. )
  (.S.
      (.NP. .a baby.)
      ( .{VP}.
            (.V. .{gave}. )
            (.NP. .every child.)
            (.NP. .\node{2}{\strut t}.)
         )
    )
)
\abarnodeconnect[-10pt]{2}{1}
\end{parsetree}
```

Here is a more complicated example:




```
\begin{parsetree}
    ( .\node{1}{S}.
        ( .\node{2}{NP$_i$}. .the baby.)
        ( .\node{3}{VP}.
            (.\node{4}{V}. .\node{5}{gives}. )
            (.NP$_j$. .the baby.)
            (.NP. .a toy.)
         )
    )
\anodecurve[tl]{2}[bl]{1}{20pt}
\anodecurve[r]{1}[t]{3}{20pt}
\anodecurve[bl]{3}[t]{4}{20pt}
\anodecurve[bl]{4}[l]{5}{20pt}
\end{parsetree}
```