Qtree, a LaTeX tree-drawing package[1]

Jeffrey Mark Siskind (tree drawing and core package)
Alexis Dimitriadis (parser and extensions)

Version 3.1b, 12 December 2008

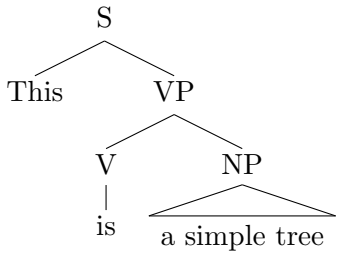
1 Overview

The *Qtree* package consists of QobiTree, a package of tree-drawing macros written by Jeff Siskind, and a front end that allows trees to be specified in bracket notation, using whitespace to separate tokens. Tree nodes can have labels of any size or complexity, and are automatically arranged on the page, usually with quite good results. Provisions exist for fine-tuning the default layout. The front end also centers trees (by default) and provides some other nice features.

A simple tree may look like this,

```
\Tree [.S This [.VP [.V is ] \qroof{a simple tree}.NP ] ]
```

which produces:



The node labels in trees may be quite complicated; they may contain font changes and math-mode text, line breaks introduced with `\\` (which produce centered lines), etc. The trees produced are constrained to a maximum depth of 20 levels, with a maximum of five branches at any one node. *Qtree* automatically adjusts for the width and height of tree labels, and is pretty good at arranging nodes on the page.

Trees are defined using a version of the bracket notation familiar to linguists. Tree elements are delimited by white space; braces can be used to enclose multi-word labels. *Qtree* does not rely on `\catcode` changes for its operation, allowing trees to be included in footnotes and other moving environments without problems. (But see the discussion of `\automath` below).

Thanks to recent improvements in LaTeX's support for PDF, *qtree* can now produce good-looking graphics for both DVI/PostScript and PDF output. By default, *qtree* will load the package pict2e.sty, which improves the native picture-drawing facilities of LaTeX and transparently supports both

[1] Thanks to Jeff Siskind for permission to distribute the QobiTree code. Please direct comments to Alexis Dimitriadis (*alexis@ling.upenn.edu*).

PostScript and PDF output.[2] This version of Qtree also provides hooks for customization, and contains many small enhancements and corrections of minor glitches.

Many of the new features were inspired by questions or requests from *qtree* users. Thank you all for your contributions to *qtree*, and keep them coming!

1.1 New features

1. Full PDF and PostScript support using pict2e.sty. This package, which extends the limited capabilities of the standard `picture` environment, was planned long ago by the LaTeX developers but has only recently been released. It provides enhanced drawing quality for both DVI and PDF output.

2. Customization hooks allow easy adjustments to the default style of leaf and branch labels, and to a number of other aspects of tree appearance.

3. Extraneous whitespace around trees has been removed.

4. Reorganized and rewritten code is easier to customize (with a moderate amount of LaTeX hacking skills).

5. The documentation has been extended and reorganized. An FAQ section has been added with explanations of various frequently requested tasks.

6. Several functions have been added, including a "balancing" command that will produce trees with more evenly sized branches.

7. *Qtree* is now (finally) available on CTAN.

1.2 Home page

The *qtree* home page is at:

http://www.ling.upenn.edu/advice/latex/qtree/.

[2] Earlier versions of *qtree* loaded eepic.sty, which does not work with PDF. You can still use *qtree* with eepic if you wish; see section 2.1.

Contents

2 Invocation

Qtree.sty is a LaTeX package. It should be installed in a directory of style files, and included with the LaTeX 2_ε command `\usepackage{qtree}`.

2.1 Package options

`[center]` Horizontally center trees on the line. This is the default behavior. (See also the commands `\qtreecentertrue` and `\qtreecenterfalse`, below).

`[nocenter]` Do not center trees. Trees are positioned on the page like ordinary text, and can be manually aligned to the left or right (or centered).

`[noload]` Suppress automatic loading of the graphics extensions library pict2e.sty. Pict2e supports both PostScript and PDF graphics, but automatic loading can be inhibited if it causes problems, or if you want to pass your own options to pict2e.

Earlier versions of *qtree* used eepic.sty as the graphics enhancement library; if you want to use it instead of pict2e, invoke *qtree* with the package option `[noload]` and load eepic.sty manually (using `\usepackage{eepic}`).

2.2 Customization parameters

`\qtreecentertrue` `\qtreecenterfalse` Trees are horizontally centered on the line by default, but you can turn centering on and off at any point with these commands. They obey the usual scoping rules; if used inside an `enumerate` environment, for example, their effect will only last until the end of that environment. (See also the `[(no)center]` package options).

Length parameters The following macros can be redefined to customize various aspects of tree construction. (Note that they are all *macros,* not counters).

`\qtreepadding` The amount of whitespace inserted around labels and leaf nodes as the tree is built. The default is the value of the LaTeX parameter `\tabcolsep` (usually set to `6pt`). It is safe to change the value of this macro for different trees, but modifying it in the middle of a tree could lead to somewhat strange behaviour.

`\qroofpadding` The amount of whitespace inserted around text placed under a roof; that is, the width of the "eaves" of the roof that extend beyond the text. The default is `0.4em`.

`\qtreeunaryht` The height of the line connecting the labels of non-branching nodes to their leaves. The default is `2ex`.

Customization hooks The following macros are empty by default, but they can be defined to customize the appearance of labels and leaves. They can safely change font styles etc., since they are invoked inside another environment.

`\qtreeinithook` Called at the beginning of each tree, before processing begins. It can be used for definitions or modifications of commands that should only be in effect during tree construction. (For example, to use the `\small` font size for all trees).

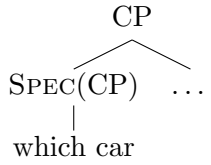
`\qtreefinalhook` Called after the completed tree has been printed out; it might be used, for example, to end an environment opened with `\qtreeinithook`.

`\qleafhook` `\qlabelhook` `\qleafhook` is called for each leaf node, and `\qlabelhook` for each non-terminal label. They can be used to define default fonts, etc., for text in these places. Their use is explained in section 3.6.

3 Basic usage

3.1 Tree syntax

`\Tree` The front end of *qtree* reads a tree description written in the familiar (to linguists) bracket notation. Tree labels are delimited by whitespace. To make a multi-word node label, enclose it in braces. Note also that TEX discards the spaces immediately after *control sequences* (commands whose name consists of a backslash followed by letters); hence if a node label ends with a control sequence, like `\ldots` in the following example, you need to enclose it in braces too.




```
\Tree [.CP [.{\sc Spec}(CP) {which car} ] {\ldots} ]
```

3.2 Label matching

For convenience, a label for a non-terminal node can be written either after the left bracket or after the right bracket corresponding to that node. Thus the following are equivalent:

```
\Tree [.S when [.NP the cat ]    sleeps ]
\Tree [.S when [    the cat ].NP sleeps ]
```

To help keep braces matched when editing large trees, the front end allows the option of writing a label after both the left and the right bracket of the same node, as shown for the node NP below. In this case the two labels provided must be identical, token for token.

```
\Tree [.S when [.NP the cat ].NP sleeps ]
```

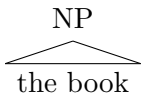
3.3 Roofs

\qroof
\qroofx
\qroofy

The command `\qroof` draws a triangular "roof" above a phrase that is treated as a unit. It can appear anywhere a *leaf* can appear. The slope of the roof is equal to the ratio `\qroofy / \qroofx` (these counters may be reset to any pair of integers between zero and six; the default is 1/3).

To create a roof labeled *NP* over the phrase *the book,* write

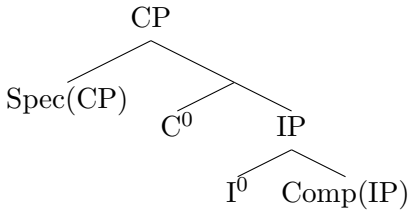
```
\qroof{the book}.NP
```



If the phrase contains line breaks introduced with `\\`, the resulting lines are flush left, not centered. Again, it is possible for the "phrase" to be a construction of arbitrary complexity; but the syntax of `\qroof` does not allow further branches of the tree to appear under the roof, since a roof is meant to cover material that is not analyzed.[3] See also the discussion of roofs in the following section.

3.4 Branching without labels

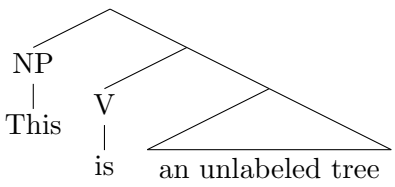
Sometimes we want to draw an abbreviated tree without a label on some or all intermediate nodes. *Qtree* will handle such trees correctly:




```
\Tree [.CP Spec(CP) [ C^0 [.IP I^0 Comp(IP) ] ] ]
```

Starting with qtree version 3.1b, a roof without a label will be attached to the tree branches above it. Note that the period between the roof text and the label does need to be given. If you use this option, you will probably want to adjust the roof angles so that they match the slope of the branch above it. For binary branching, the correct values are `\qroofx=2`, `\qroofy=1`.

```
\qroofx=2
\qroofy=1
\Tree [.S This [ [.V is ] \qroof{a simple tree}. ] ]
```



[3] *Qtree* internally implements a roof as a large leaf.

3.5 Subscripts, superscripts and primes

Qtree defines the following formatting commands in such a way that they are only in effect during tree construction. They have their regular meaning (or none) in the rest of the document.

`\automath` `\noautomath`

Trees are constructed in a special environment in which things like `NP_i`, `N^0`, automatically format their subscripts or superscripts in math mode, giving NP_i and N^0, respectively. The command that arranges this is called `\automath`, and can also be enabled outside the tree environment, if desired. (It is turned off with `\noautomath`).

Because this feature relies on `\catcode` changes for its operation, `\automath` has no effect when called inside footnotes or floats; in such trees, all sub- or superscripts should be explicitly placed in math mode, as you would ordinarily do. Alternately, you can turn on `\automath` before entering the footnote or float.

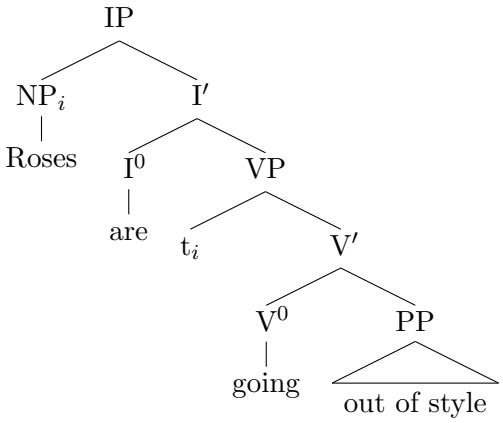
`\0,\1,\2`

As a further convenience, constructions like `X$'$`, producing X', can be abbreviated `X\1`. (If you simply type `X'` you get X', with an apostrophe rather than a prime). There is also `X\2`, producing X'', and `X\0`, producing X^0. These commands also arrange for subtle improvements in the centering of labels that use them, by using `\rlap` to set the superscript.

`\qtreeprimes`

If you want to use commands like `\1` in your text (outside of trees), you can cause all three to be defined by putting the command `\qtreeprimes` in your preamble. The version defined in this way does not use `\rlap` around the superscripts, since this is inappropriate for running text. Thus you get X', X'', and X^0. (Notice the position of the punctuation compared to the previous paragraph). The versions that use `\rlap` will still be used inside trees.

Here is an example using some of these features:




```
\Tree
[.IP [ Roses ].NP_i [.I\1 [ are ].I\0
   [.VP t_i [ [ going ].V\0 \qroof{out of style}.PP ].V\1 ].VP
].I\1 ]
```

Granted, by the time the examples get this big, the bracketed format isn't

all that readable; but it's certainly no worse than any other tree format, and you can add white space to make it a little better.

3.6 Customizing text appearance

`\qleafhook` `\qlabelhook`

The macros `\qleafhook` and `\qlabelhook` don't do anything by default, but they can be defined to control the appearance of the text in leaves and in branch labels, respectively. The macros can be defined either with or without an argument; the following examples illustrate both options.

```
\newcommand{\qlabelhook}{\bf}
\newcommand{\qleafhook}[1]{\emph{#1}}
\newcommand{\qlabelhook}{\framebox}
```

Details: For some effects it is important to know that the argument of the hook is not just the text of the leaf or label, but a small `tabular` environment (which inserts whitespace controlled by `\qtreepadding`). In simplified form, the code that uses the hooks looks like this (where `#1` is the text of the branch label).

```
... \qlabelhook {\begin{tabular}[t]{c} #1 \end{tabular}}
```

This arrangement makes it possible for leaves and labels to contain line breaks (\\). But one consequence is that the hooks cannot put their argument in math mode, since a table cannot appear inside math mode. Fortunately, there is a simple work-around: We get the same effect by making *qtree* use the `array` environment in place of `tabular`.[4] To put only branch labels (not leaves) in math mode, define the following hook:

```
\def\qlabelhook{\let\tabular=\array \let\endtabular=\endarray}
```

3.7 Escaping the parser

`!-escapes`

For fine control of the tree-building process, we must sneak certain directives past the front end. If a word begins with an exclamation mark, the entire word (i.e., up to the next space) will be passed through unchanged, except for stripping off the "!". Braces can be used to pass through larger groups. This is most often used for the manual width-adjustment commands `\faketreewidth` and `\qsetw` (see below), but can also be used to override various *qtree* settings for part of the tree. Note that `\qroof` is part of the tree syntax, and should *not* be preceded by an exclamation mark.

3.8 Adjusting inter-node spacing

`\qsetw`

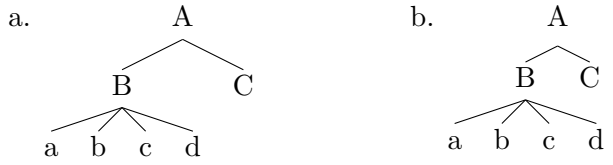
The command `\qsetw{`⟨*length*⟩`}` (where ⟨*length*⟩ might be `0.5in`, `36pt`,

[4] This works because `tabular` and `array` are internally implemented identically, except for switching on math mode in `array`.

etc.) tells *qtree* to override its default calculation of the width of the *just-finished* node (that's the leaf or branch ending just to the *left* of where the directive was issued), and instead consider that width to be ⟨*length*⟩.

`\faketreewidth` Similarly, `\faketreewidth{`⟨*text*⟩`}` sets the width of the last node to be equal to the width of ⟨*text*⟩ (which again can contain '`\\`' commands etc.) ⟨*text*⟩ is not actually typeset but is used just to compute the fake width of the most recently completed leaf or subtree.

For example, the default placement rules would produce tree (a) below. By setting the width of the subtree headed by B to `1cm`, we get tree (b).

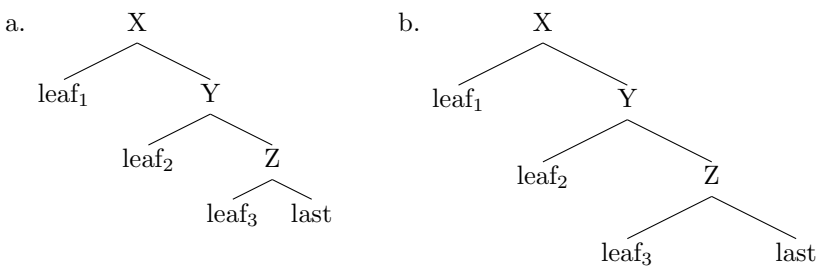



```
\Tree [.A [ a b c d ].B !\qsetw{1cm} C ]
```

When you use `\qsetw` or `\faketreewidth` you are on your own. They can either shrink or enlarge the space taken by the node and may result in trees with overlapping labels.

3.9 "Balanced" trees

`\qbalance` Regular binary-branching trees often end up with smaller and smaller branches as you descend into the tree; you can see an example below. If you don't like this look, you can get "balanced" trees by adjusting the width of the last leaf node. For a balanced tree, this should be exactly *three* times as wide as the leaves on its left;[5] For binary-branching trees, the command `\qbalance` can be used. It sets the width of the node just before it to be three times the width of the *previous* node. If it does not do what you need, you'll need to set the correct width manually. In the example below, the width of the leaf *"last"* is set to be three times the width of $leaf_3$.




```
\Tree [.X leaf_1 [.Y leaf_2 [.Z leaf_3 last !{\qbalance} ] ] ]
```

[5] The geometrical justification for this is left as an exercise to the reader.

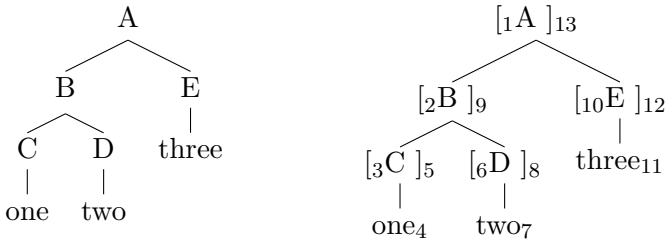
4 Using qtree

4.1 How to convert a tree to brackets

Reading or writing a complex tree in bracket notation is not terribly easy for humans; it helps to have an editor that can show matching braces as they are typed in. The procedure described here should allow you to easily convert a tree to bracket notation. If you don't have any difficulty with this, just skip this section and do it any way you want!

1. Draw the tree you want to enter on a piece of paper, so you can look at it.
2. Imagine that the tree is a large peninsula, and your pencil is a boat sailing around it. Starting just to the left of the root node, move downwards, following the outline of the tree until you come back to the root node (on the right side, having moved counterclockwise around the tree), without crossing any of the tree's lines.
3. (a) Every time you are at the left side of a non-terminal node, type a left bracket, and the label for that node.
   (b) Every time you are at a leaf node, type in the contents of that node.
   (c) Finally, every time you are on the right of a non-terminal node, type a right bracket (and the node name again, if you want to help keep them straight).

To see this in practice, consider the following tree; in the variant on the right, the numbered subscripts show the order in which the brackets and labels are written.



Accordingly, we would generate the tree by typing the following:

```
\Tree [.A [.B [.C one ] [.D two ] ].B [.E three ] ].A
```

4.2 Placing trees on the page

Numbered examples etc. A tree generated with *qtree* can be placed in a numbered example environment, in footnotes or `\parbox`es, inside math formulas, tables, pictures, etc. The tree nodes can themselves contain

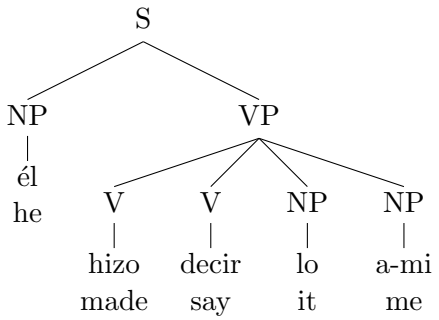
arbitrarily complex material—although, unfortunately, it is not possible to embed a recursive call to *qtree*. If you need to do this, save the intermediate tree in a `\box`. An example is shown in section 5.2.

Trees are centered horizontally on the line by default, but this feature can be turned off with `\qtreecenterfalse` (see section 2.2). If you are not satisfied with the horizontal placement of the tree, you can adjust it by judicial use of `\hskip`.[6]

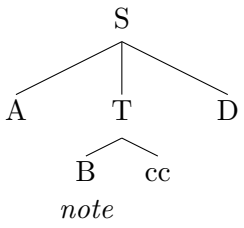
Qtree attempts to align the topmost label of the tree with the baseline of the text, similarly to the effect of the [t] option for `\parbox` alignment. To center trees vertically on the baseline, enclose the entire tree in a `tabular` environment.

Side by side trees Multiple trees, or text and trees, can be arranged side by side. This can generally be done by just arranging commands one after another; it usually helps to turn off tree centering. If necessary the positioning can be adjusted with `\hskip`.

(2) a.



b.



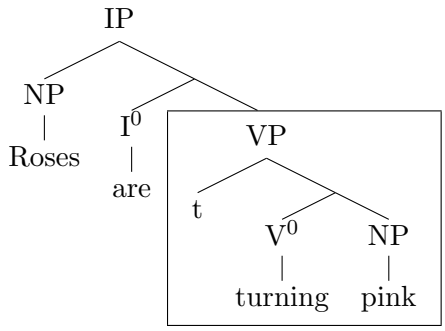

```
\begin{enumerate}
\qtreecenterfalse
\item[(2)] a. \Tree [.S [.NP \'el\\he ]
        [.VP [.V hizo\\made ] [.V decir\\say ]
             [.NP lo\\it ] [.NP a-mi\\me ]      ].VP ]
  \hskip 0.3in
  b.  \Tree[ A [.T {B\\ \em note} cc ].T D ].S
\end{enumerate}
```

5 Advanced features

5.1 Putting a frame around part of a tree

`\qframesubtree` The command `\qframesubtree` will put a rectangular box ("frame") around part of a tree. It should be placed just *after* the closing bracket of the subtree that should be boxed.

[6] Earlier versions of *qtree* inserted stray spaces in front of trees, requiring frequent use of `\hskip`. This problem has now been fixed.




```
\Tree [.IP [.NP Roses ] [ [.I^0 are ]
    [.VP t [ [.V^0 turning ] [.NP pink ] ] ].VP !{\qframesubtree} ]]
```

Only complete subtrees can be framed with this method. To enclose a single label node in a box, use a frame command directly. To box all labels or leaves, you can use the format customization hooks. For example, the following will put a box around every label:

```
\renewcommand{\qlabelhook}[1]{\framebox{#1}}
```

5.2 Embedding one tree in another

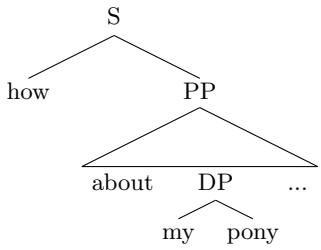
The internal implementation of *qtree* only supports one tree being built at one time; hence it does not allow a complete tree to be recursively embedded as part of another one. This is not actually needed very often (trees are already recursive, after all); for most constructions that seem to require it, there's a simpler way to get the same effect. But it might be necessary, for example, if you need to put a subtree under a roof.

Fortunately it's straightforward to embed one tree in another using TEX's facilities, without ever calling *qtree* recursively. We can do this by saving the first tree in a named box, and then using it as part of the larger tree. Here's how:

```
\newsavebox{\partbox} % Declare this only once, in your preamble!

% Build the subtree and save it in the box
\setbox\partbox=\hbox{\Tree [.DP my pony ] }

% Use the subtree in the containing tree
\Tree [ how
  \qroof{about \usebox{\partbox} ...}.PP
].S
```



5.3 Overriding branch drawing

Qtree is designed to produce nice trees automatically; it is not designed for fine control over their appearance. Nevertheless, technically inclined

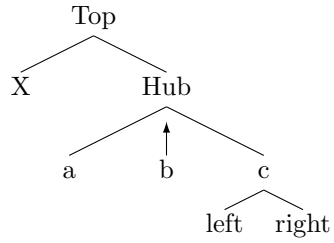
users could use the ! escape mechanism to temporarily redefine the branch-drawing routine and draw the branches any way they want, specifying wavy or dotted lines between selected nodes, arrows at line tips, etc.

`\qdraw@branches` The code that draws the lines from a tree node to its daughters is encapsulated in the macro `\qdraw@branches`. It takes one argument, the number of branches, and uses graphics commands to draw a picture containing the branched lines. The default drawing is very simple; the actual width of the picture is controlled through the value of the parameter `\unitlength` (see the documentation of the standard LaTeX environment `{picture}`), which will be set to *half* the required distance between the branch tips before `\qdraw@branches` is called.

In the following simple example we override the macro (and later restore it), to shorten the line that connects "b" to its parent and add an arrow at its end (with `\vector`). Since the custom macro will only be used once, it supports ternary branching only and ignores its argument.

```
\makeatletter
  \newcommand{\myLines}[1]{% Three-way only
  \begin{picture}(4,1)
    \put(0,0){\line(2,1){2}}
    \put(2,0){\vector(0,1){0.7}}
    \put(4,0){\line(-2,1){2}}
  \end{picture}}
\let\qdrawReal=\qdraw@branches
\newcommand\brOverride{\let\qdraw@branches=\myLines}
\newcommand\brRestore{\let\qdraw@branches=\qdrawReal}
\makeatother

\Tree [.Top X [.Hub a b [.c left  right ]
  !{\brOverride} ].Hub !{\brRestore} ].Top
```

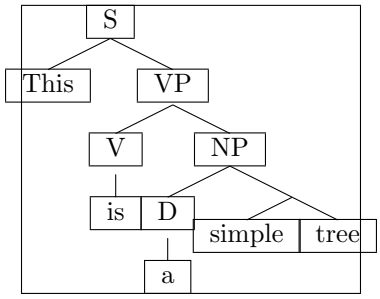


The branch-drawing code above is a simple modification of the commands found in the style file, but it could be replaced with commands using a drawing environment of your choice, and inserting arbitrary graphic elements. Note, however, that the horizontal placement of child nodes on the page is not affected by what the drawing routine does; drawing wider or narrower branch lines will not cause the text to move with them.

5.4 Visualising tree construction

If you're curious about how *qtree* assembles leaves and labels into a tree, you can try the command `\qtreeshowframes`. It causes trees to be drawn with boxes around their leaf nodes and labels; in the example below we also add a frame around the entire tree, to show TeX's idea of its bounding box:

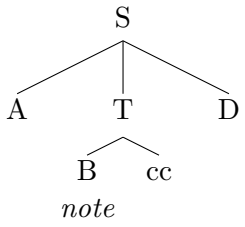
```
{ \qtreeshowframes
  \frame{\Tree [.S This
    [.VP [.V is ]
         [.NP [.D a ] [ simple tree ] ]
    ]] }}
```



5.5 The low-level interface

The guts of *qtree* are the tree macros written by Jeff Siskind, named *QobiTree.* Using the original interface (which is still accessible with this package) the example tree shown on page 11 would be written like this:

```
\begin{center}
\leaf{A}
    \leaf{B\\ \em note} \leaf{cc}
    \branch{2}{T}
\leaf{D}
\branch{3}{S}
\qobitree
\end{center}
```



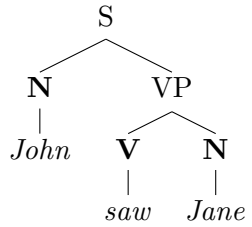
These macros operate like a stack machine. You push TEX boxes onto the stack of tree nodes, then you pop them off to make branching nodes which get pushed back on the stack. The *qtree* front end internally generates these commands to build trees.

The command`\branch` requires you to specify the number of stack elements that will be made into children of the new node. The front end uses an internal counter, `\nbranches`, to keep track of how many children a node has.

It is possible, and in some cases even useful, to write your own commands that add bits of tree structure to the qtree stack. The command `\mylex` in the following example formats a singleton branch label in bold above lexical items (in italics), while leaving other branch labels in the default roman font.

```
\newcommand\mylex[2]{\advance\nbranches by1
    \leaf{\emph{#2}}\branch{1}{\bf #1}}

\Tree [.S !\mylex{N}{John}
        [.VP !\mylex{V}{saw} !\mylex{N}{Jane} ]]
```



We use `!` to bypass the front end, in order to manipulate the tree stack directly. The macro pushes a leaf onto the stack and then includes it under a singleton branch with a bold label. It also increments the counter

`\nbranches` to let the front end know that the resulting node has been added to the stack. With some understanding of how qtree works, much more complicated commands are possible. (In extreme cases, you might consider bypassing the front end altogether and relying exclusively on the stack interface).

6 How do I . . . ? (The qtree FAQ)

Make my tree fit in the page? Try one or more of the following: reduce the font size with `\small` or another size command, *before* you begin the tree; reduce the amount of space between subtrees with `\qsetw` or `\faketreewidth`; abbreviate unimportant parts of your tree with `\qroof`; consider placing your tree sideways in the page, using one of the packages that provide rotation commands.

Draw movement arrows from one node of a tree to another? Use Emma Pease's tree-dvips.sty, or some other package for drawing lines between locations on the page. Despite its name, tree-dvips is not narrowly tailored to creating trees; its many line- and arrow-drawing commands can be used to decorate trees drawn with *qtree*.

Use qtree with *pdflatex*? Earlier versions of *qtree* did not have good pdflatex support. The current version supports it through the package pict2e.sty. If it is not already available on your system, it is highly recommended that you install it. See section 2 for details.

Place a tree in a footnote, figure, or macro argument? *Qtree* does work in floats, but the `\automath` feature is not available in such environments. If you're having problems, you're probably using sub- or superscripts without switching to math mode. See section 3.5.

Center my trees vertically on the baseline? Put them inside a trivial `tabular` environment. (This can be automated with the hooks `\qtreeinithook`, `\qtreefinalhook`). To avoid extra whitespace around the tree, use the following column specification (see the documentation of `{tabular}` for an explanation):

```
\begin{tabular}[c]{@{}l@{}} \Tree ...
```

Line up the text from all the leaf nodes on one horizontal line? As far as I can tell, *qtree*'s design is incompatible with this style of tree. I'd love it if there was an easy way to give *qtree* this capability, but if there is, I haven't figured it out.

Draw dashed or dotted branches between certain nodes? This requires a bit of LaTeX hacking skills, but it can be done. See section 5.3 for details.

Automatically select a particular font for branch or leaf labels? Redefine `\qlabelhook` and/or `\qleafhook`. See section 3.6.

Embed one call to qtree inside another? *Qtree* cannot be called recursively. If you do need to embed one tree in another (it's very rarely necessary), save the smaller tree in a `\savebox` and insert it into the bigger one. See section 5.2 for the details.

7 Troubleshooting

Disclaimer: I welcome any comments, feature requests, or reports of other problems. But as usual, no guarantees, promises, etc. can be made about the present or future state of this code.

The placement of trees on the page has changed Version 3 of *qtree* has fixed some bugs in horizontal spacing (most visibly, extra spaces used to be inserted before trees), and made countless small adjustments in spacing. Trees are now centered exactly on the line, and they generate *no* white space around them.[7] They are thus easy to position and integrate with other visual objects.

Unfortunately, this creates a compatibility problem, as it was not practical to provide a "backwardly compatible" spacing option. If you have used a lot of `\hspace` commands to adjust the placement of your trees, they will no longer work properly. On the positive side, it should now be very easy to arrange trees the way you want them. If you need to process files that have a lot of old trees with spacing adjustments, you can install the package qtree221.sty, which contains version 2.21 of qtree. The new commands and formatting options of *qtree* version 3 are not available in qtree221.sty.[8]

The change-over is a one-time incompatibility. Future revisions will retain the spacing of version 3.

LaTeX says that pict2e.sty "will not be written soon" Pict2e.sty *has* finally been written. You need to update your TeX distribution or just download the package. If you cannot do that for any reason, load *qtree* with `\usepackage[noload]{qtree}` to suppress loading of pict2e.sty.

[7] If you put a tight box around a tree (using `\frame`), it will just touch the sides of the tree.

[8] qtree221.sty is not available on CTAN. It can be downloaded from the Qtree home page, http://www.ling.upenn.edu/advice/latex/qtree/.

Some very short lines are not drawn This problem is caused by the limited inventory of line shapes in the original LaTeX picture environment. For example, the tree fragment `[.X a b ]` will produce invisible branch lines from `X` to `a` and `b`, but the lines will reappear if the labels are made wider. If you use pict2e, the problem should go away.

Some not-so-short lines are not drawn My dvi previewer (yap) sometimes fails to display some lines; but the lines are really there, and can be seen at higher magnifications, or when the file is printed or converted to PostScript. So it's only a problem with the previewer; if it gets in your way, suppress loading of pict2e with the package option `[noload]`.

TeX is running out of registers *Qtree* is a real pig for counter registers: it uses more than 60 of them. In combination with other packages that use a lot (e.g., the memoir class), this can go over the limits of the original TeX engine. Fortunately most distributions have now quietly switched to using eTeX internally, which makes many more registers available (you still invoke it simply as `latex`, so you probably don't even know you're using it). But you need to tell eTeX that the extra registers are there. If you run out of registers, try including the package etex.sty and see if it fixes the problem.[9]

There's too much space below my trees, but only when I use pdflatex Qtree should produce identical output for dvi and pdf. The problem is caused by a bug in older versions of color.sty, which is loaded internally by many pdf-aware packages.[10] Update the package color from CTAN (the entire directory) and the problem will go away.

Qtree will not work with journal style X Any number of things could be going wrong, of course, but start by checking if the journal's style redefines the `tabular` environment. *Qtree* makes internal calls to `tabular`, so this is a frequent source of problems. Usually the style's writer has saved the original definition of `\tabular` under a different name, so all you need to do is arrange for the original definition to be restored during the calls to `\Tree`.

You can define `\qtreeinithook` to carry out the necessary redefinitions. It is called at the beginning of each call to `\Tree`, with local scope (so that any redefinitions it makes are automatically canceled at the end of the call to `\Tree`). For example, the JNLE style (nle.sty) saves the standard commands to begin and end a table as `\oldtabular` and `\endoldtabular`, respectively, and the replacement macros result in *r e a l l y w i d e* trees.

[9] If you include etex.sty and strange things start happening, you probably aren't running the eTeX engine; in that case, *don't* load etex.sty. You'll have to install a newer TeX distribution first.

[10] The bug is in the redefinition of `\ usebox` in the file pdftex.def.

The following will restore the original definitions for calls to `\Tree` only.

```
\def\qtreeinithook{\let\tabular=\oldtabular
         \let\endtabular=\endoldtabular}
```

Kluwer's house style saves the original definitions as `\klu@tabular` and `\klu@endtabular`; they can be restored in the same way. (You'll need to use `\makeatletter` since these names contain an `@`-sign).