A Quickstart Guide for Drawing Tree Diagrams in LaTeX

Lisa Shao

October 21, 2025

Abstract

This guide aims to provide a beginner-friendly introduction to using the `forest` and `qtree` packages in LaTeX to draw clean tree diagrams for your linguistic assignments. ;-)

Contents

1 The LaTeX Basics: Preamble and Packages **1**

2 Drawing Trees with the `forest` Package **2**
2.1 Environment Setting 2
2.2 Simple Tree Structure 2
2.3 Adjusting node spacing 3
2.4 Complex Tree Structure 4
2.5 Features 6
2.6 Movements 7
2.7 Final Notes 8

3 Drawing Trees with the `qtree` Package **9**
3.1 Examples 9
3.2 Limitations 9

4 Appendix: Set theory **10**

1 The LaTeX Basics: Preamble and Packages

LaTeX (pronounced /l'tek/) is powerful because of its **Preamble**, which defines the document type and loads special packages that give you extra features. Using LaTeX ensures your work is **automatic, professional, and consistent**.

The preamble is where you tell the compiler what tools you'll be using. Since we're drawing trees, we need these packages:

```
\usepackage[linguistics]{forest}
\usepackage{amsmath, amssymb}
\usepackage{tikz}
```

Always put this line in your preamble to tell LaTeX you'll be using this tool.

A Note The `forest` package is actually built on top of the incredibly powerful **TikZ** package. We won't study TikZ directly because its manual is over 1,000 pages, but `forest` gives us an easy, user-friendly way specifically for drawing trees.

2 Drawing Trees with the `forest` Package

The `forest` package uses a simple bracket notation to represent the hierarchy of a tree.

2.1 Environment Setting

Before any tree coding starts, you must create the **environment** where the code will run. **NEVER** forget these lines:

```
\begin{forest}
type your codes here...
\end{forest}
```

Note: If you use an editor like VS Code, type `\begin{forest}`, and the closing command `\end{forest}` will often appear automatically!

2.2 Simple Tree Structure

Each set of brackets `[]` represents a single **node** in the tree. The first item inside the brackets is the node's label (the **mother**), and anything else inside represents its **daughters**.

Example 1: A Single Branch

The code `[N [Kitty]]` produces:

(1)


Example 2: Two Branches (*Kitty meows*)

To draw a branching structure, you simply place the daughter nodes next to each other, separated by a space, inside the mother node's brackets.

The sentence *Kitty meows* has a structure like: S → DP+VP.

We can firstly start with the POS (part of speech) labeling since it is unary in this case.

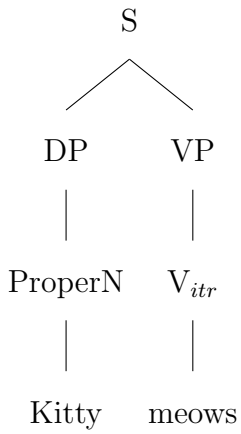
```
\begin{forest}
[ProperN[Kitty]]
[V_{itr} [Meows]]
\end{forest}
```

Then we can do the syntactic analysis, adding **S, DP and VP**.

```
\begin{forest}
      [S
[DP [ProperN [Kitty]]]
[VP [V_{itr} [meows]]]
      ]
\end{forest}
```

This produces:

(2)



2.3 Adjusting node spacing

Although `forest` will arrange the nodes in the tree for you, you can still adjust both the horizontal and the vertical spacing, and the empty space around the nodes.

The default setting is

```
\forestset{default preamble={for tree={s sep=10mm, inner sep=0, l=0}}}
```

Something to note:

- The horizontal spacing is controlled by the `s sep` command, the vertical (or level) spacing by the `l` command.

- The `inner sep` command controls the empty space around the nodes.

You can specify absolute values for these parameters, as in the example below, or increase or decrease their default values as calculated by forest.

This is done either by multiplication (e.g. l*=3 multiplies the default level distance by 3), or by addition or subtraction (e.g. l+=3mm adds 3mm to the default level distance, l-=3mm subtracts 3mm).

2.4 Complex Tree Structure

Let's analyze the transitive sentence *Andy likes Billy* with a complete phrase structure.

1. Start by encoding each word with its POS label: `[ProperN [Andy]]`, `[V [likes]]`, `[ProperN [Billy]]`.

2. Group them into phrases using brackets: → `[NP [ProperN [Andy]]` and `[NP [ProperN [Billy]]`.

3. Finally, link everything up to the main S node.

Example 3: A whole sentence (*Andy likes Billy*)

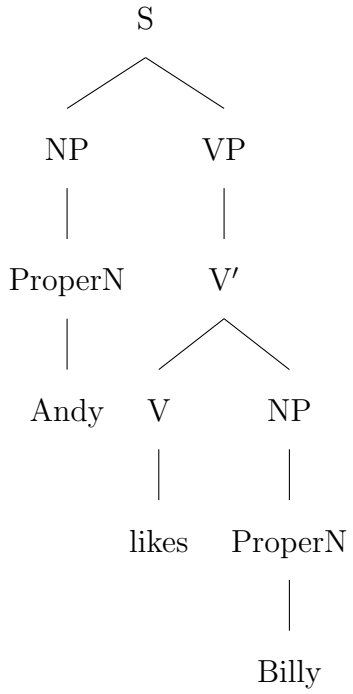
The complete code looks like this (Notice the use of V $'$ for V-bar, and that we must always ensure every bracket is closed!)

```
\begin{forest}
[S
    [NP
        [ProperN [Andy]]
    ]
    [VP
        [V$'$
            [V [likes]]
            [NP
                [ProperN [Billy]]
            ]
        ]
    ]
```

```
]
\end{forest}
```

And here is the output:

(3)



Note: in syntax, there is triangle to represent a whole phrase. We use the `root` command, just type `[...,roof]` after the node.

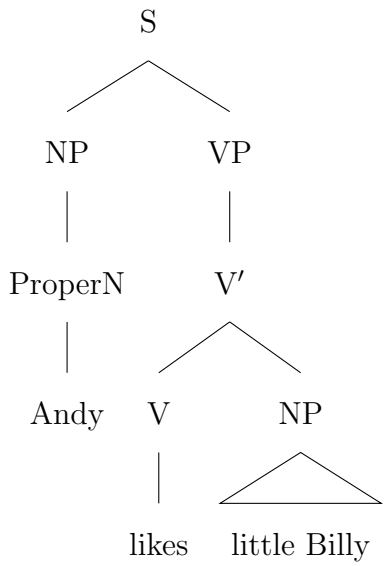
Example 4: Roof (*Andy likes little Billy*)

```
\begin{forest}
[S
    [NP
        [ProperN [Andy]]
    ]
    [VP
        [V$'$
            [V [likes]]
            [NP [little Billy, roof]
            ]
        ]
    ]
]
```

```
\end{forest}
```

Then we can get:

(4)



In the next section, we'll look at how to add types like $\langle e \rangle$ and $\langle t \rangle$ for semantics or ϕ-features for formal syntax

2.5 Features

In formal Semantics (this course), a tree should be drawn like this:

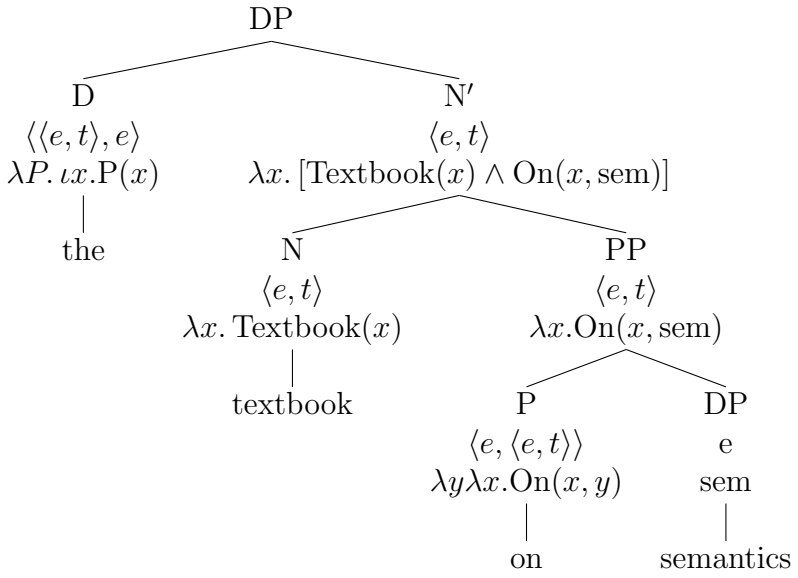


Figure 1: Adapted from Heim and Kratzer (1998, p. 5)

For semantic types, we use $\{\langle ... \rangle\}$ after part of speech and before the word. Here is an example:

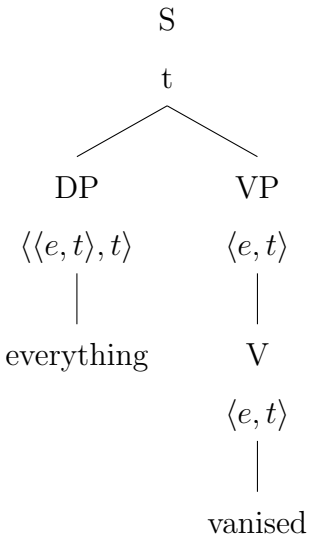
Example 5: semantic types(*Everything vanished*)

We can type like this:

```
\begin{forest}
for tree = { s sep = 10mm , inner sep = 0pt, l = 0pt}
[S \\ t
    [DP\\{$\langle \langle e,t \rangle, t\rangle$} [everything]]
    [VP\\{$\langle e,t \rangle$}[V\\{$\langle e,t \rangle$}[vanised]]]
]
\end{forest}
```

Then the first semantic tree you have drawn in your life is:

(5)



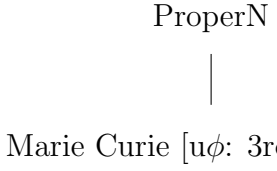
Adding a feature needs to be done like [feature], which is to put curly brackets outside square brackets.

Basic Feature: Put the feature directly after the node's label, but before the children.

We use line breaks (\\) and custom spacing (`s sep`) to achieve this.

Example 6: ϕ-feature(*Marie Curie*)

(6)



2.6 Movements

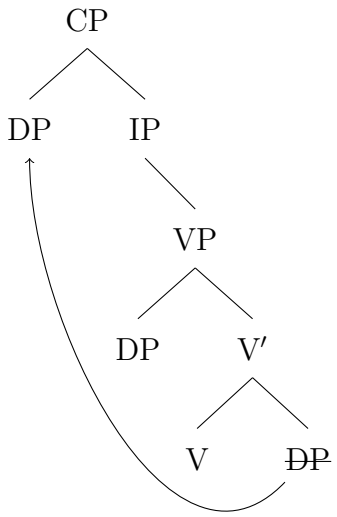
We need to draw arrows for movements, and this requires two steps:

1. **Naming Nodes:** Inside a node's square brackets, use the option `,name=LABEL`. For movement, you usually name the source (`src`) and the target (`tgt`).

2. **Drawing the Arrow:** The `\draw` command is placed *after* the entire `forest` environment. **Here we are using a `TikZ` command. So the package `TikZ` should be added**

Example 7: movement(*A simple Wh-Movement example*)

(7) **Wh-Movement example**



The `phantom` *node is useful for maintaining horizontal spacing without drawing a line, and* `\sout{...}` *strikes through the copy.*

2.7 Final Notes

Here are some basic possible errors you might encounter when the compiler breaks dowm. :(

Check through all these

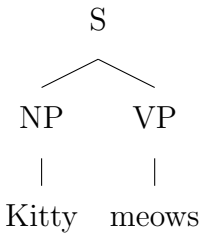
- Always **close every bracket**.
- Start simple, then delve into complexity.
- Some are not included in main text, so please use comments to annotate your code for future reference.

3 Drawing Trees with the qtree Package

The `qtree` package uses a bracketed string format inside the \Tree command. It's great for quick, small trees.

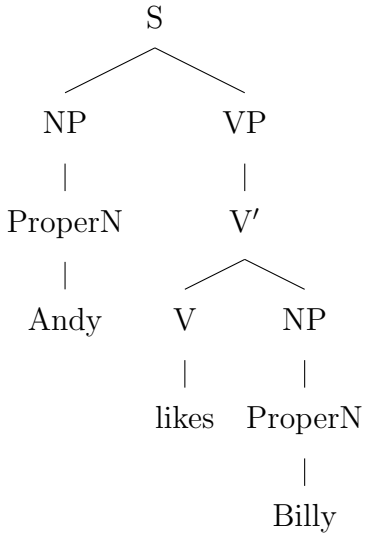
Basic syntax: `\Tree [.S [.NP Kitty ] [.VP meows ] ]`

It will produce:



3.1 Examples

For *Andy likes Billy*



3.2 Limitations

As it cannot do the followings:

- No semantic types or lambda annotations
- No arrows or movement lines
- No control over spacing or styling
- No node naming or referencing

This is not our main focus.

4 Appendix: Set theory

description	*command*	*output*
set brackets	`\{1,2,3\}`	$\{1,2,3\}$
element of	`\in`	$\in$
not an element of	`\not\in`	$\notin$
subset of	`\subset`	$\subset$
subset of	`\subseteq`	$\subseteq$
not a subset of	`\not\subset`	$\not\subset$
contains	`\supset`	$\supset$
contains	`\supseteq`	$\supseteq$
union	`\cup`	$\cup$
intersection	`\cap`	$\cap$
big union	`\bigcup_{n=1}^{10}A_n`	$\bigcup_{n=1}^{10} A_n$
big intersection	`\bigcap_{n=1}^{10}A_n`	$\bigcap_{n=1}^{10} A_n$
empty set	`\emptyset`	$\emptyset$
power set	`\mathcal{P}`	$\mathcal{P}$
minimum	`\min`	$\min$
maximum	`\max`	$\max$
supremum	`\sup`	$\sup$
infimum	`\inf`	$\inf$
limit superior	`\limsup`	$\limsup$
limit inferior	`\liminf`	$\liminf$
closure	`\overline{A}`	$\overline{A}$

Thanks! Have fun with coding!