

SynT_EX

Drawing linguistic diagrams with L^AT_EX

Malhaar Shah, Department of Linguistics

malshah.com / mpshah@umd.edu

Winter Storm at the University of Maryland Language Science Center

January 2024

Welcome to the section of our L^AT_EX course where we will talk about using numbered examples with morpheme-by-morpheme glosses, drawing syntactic trees, and indicating syntactic transformations on both of those. There are two important parts to the philosophy here: the code should be pretty, and so should its output. Elegant code is more than just aesthetically satisfying to the programmer: it becomes quick to write with practice, and it saves you time by being easier to spot bugs in. After all, we're here to do linguistic analysis, not spend all day fighting L^AT_EX. I will address some common issues with achieving these aims, and show you the packages that I think are most reliable and efficient: `gb4e`, `movement-arrows`, and `forest`.

I will start by showing you numbered examples and glosses. We'll also talk about drawing arrows on an example sentence rather than a tree. Then, I will show you some basic trees and how to draw arrows marking transformations and how to indicate domains with arcs. Lastly, we'll talk about the nuts and bolts of alignment (skippable), and end with some more interesting things you can do, like multidominance trees.

1 Numbered examples

I will use `gb4e` to show you do numbered examples like (1).

(1) This is a numbered example.

`gb4e` is very easy to use and fairly powerful. Here is what I wrote above.

```
\begin{exe}
  \ex This is a numbered example.
\end{exe}
```

Warning! `gb4e` redefines some important keys like `_` and `^`. This can mean it does not play nicely with other packages, including bibliographies. This is easily rectified by adding `\noautomath` immediately after `\usepackage{gb4e}`. Phew!

1.1 Referencing examples and multiple examples

If you want to reference an example, standard L^AT_EX commands will work. These are `\label{}` and `\ref{}`.

```
\begin{exe}
  \ex \label{example} Using \ref{example} will
  insert the number of this example.
\end{exe}
```

It can be important to embed multiple examples within one another. gb4e allows this with `xlist`.

- (2) a. This is example (2a).
- b. This is example (2b).
- (3) This is example (3).
- (4) a. Another example.

To generate the above examples, I wrote the following code.

```
\begin{exe}
  \ex \begin{xlist}
    \ex \label{subexample1}
      This is example (\ref{subexample1}).
    \ex \label{subexample2}
      This is example (\ref{subexample2}).
    \end{xlist}
  \ex \label{nextexample}
    This is example (\ref{nextexample}).
  \ex
    \begin{xlist}
      \ex Another example.
    \end{xlist}
\end{exe}
```

To make the example numbers clickable links, you need `\usepackage{hyperref}`. To make the clickable links have a color, you can add this to your preamble for the deep blue I use:

```
\hypersetup{
  colorlinks=true,
  linkcolor=[RGB]{0,69,149},
}
```

`\ex` is not the only way of introducing an example sentence.

`\exi{id}` will mean `id` is used as the example's number.

(Hello) This is another example sentence.

```
\begin{exe}
  \exi{(Hello)} This is another example sentence.
\end{exe}
```

This can be very neat when repeating examples. Have a look. Here is (1) repeated:

(1) This is a numbered example.

```
\begin{exe}
  \exi{(\ref{numbered1})} This is a numbered example.
\end{exe}
```

In fact, gb4e abbreviates what I have just done as `\exr{}`. It also allows the repetition of a number with a prime mark with `\exp{}`.

(1') Here is a prime-numbered example.

When referring to examples inline, I never use 2b but always (2b). I find it annoying to have to type `(\ref{ })` every time. Instead, I define a macro in my preamble.

```
\newcommand{\eg}[1]{(\ref{#1})}
```

Don't ask what I do with all the time this saves me.

1.2 Glossing

Thankfully, not all linguistics is done about English by English speakers.

gb4e allows glossing easily, but you have to be a little careful as errors can easily creep in.

(5) Čia yra anotuot-as pavyzd-ys.
 here be.3 annotated-NOM.SG example-M.NOM.SG
 'Here is an annotated example.'

There are two important ingredients in the code for (5). After `\ex`, you need `\gll` for the first two lines of the gloss, broken up by `\\`. and then `\glt` for the idiomatic translation. **Make sure you use `\\` and not just a line break!**

```
\begin{exe}
  \ex \gll
  \v{C}ia yra anotuot-as pavyzd-ys. \\
  here be.3 annotated-{\sc nom.sg} example-{\sc m.nom.sg} \\
  \glt 'Here is an annotated example.'
\end{exe}
```

I like to use SMALL CAPS in my glosses. `\textsc{ }` is fine, but if you have places to be, I recommend `{\sc }` and placing the text to be capitalized within the braces.

Notice how gb4e automatically lines up the data and the gloss at word boundaries. If you do anything that monkeys around with how L^AT_EX sees units, it will not work.

```

\begin{exe}
  \ex \gll
  Atsarg-iai su {\it pa-svir-uo-ju \v{s}rift-u} \\
  careful-adv with prf-tilted-m.instr-def script-m.instr \\
  \glt ‘Careful with italicized text!’
\end{exe}

```

- (6) Atsarg-iai su *pa-sviruoj-u šrift-u*
 careful-ADV with PRF-tilted-DEF-M.INSTR script-M.INSTR
 ‘Careful with italicized text!’

Notice how *šrift-u* and its gloss (script-M.INSTR) are not aligned. Disgusting! L^AT_EX treats everything in the scope of `{\it }` as one thing. So, italicize each word separately.

```

\begin{exe}
  \ex \gll
  Atsarg-iai su {\it pa-svir-uo-ju} {\it \v{s}rift-u} \\
  careful-adv with prf-tilted-m.instr-def script-m.instr \\
  \glt ‘Careful with italicized text!’
\end{exe}

```

- (7) Atsarg-iai su *pa-sviruoj-u* *šrift-u*
 careful-ADV with PRF-tilted-DEF-M.INSTR script-M.INSTR
 ‘Careful with italicized text!’

This applies also to **boldface** and ~~strikeout~~ text. (This fact is particularly annoying with ~~strikeout~~, as it means you get ugly gaps in the struck-through text.)

1.3 Empty elements and annotation

Most of what is interesting goes unheard. Silent elements can sometimes be important in a gloss. Adding a trace to a sentence can mess up alignment with a gloss, of course.

- (8) [kaun-sii kitaab]₁ ek prasiddh bhaashaavid-ne *t*₁ likh-ii
 which-F book one famous linguist-ERG write.PST-FEM
 ‘Which book did a famous linguist write?’

Just add a `{}` in the gloss-line where the trace is.

```

\begin{exe}
  \ex \gll $[kaun-sii kitaab$]1 ek prasiddh
  bhaashaavid-ne $t_1$ likh-ii \\
  which-f book one famous
  linguist-erg {} write.pst-fem \\
  \glt ‘Which book did a famous linguist write?’
\end{exe}

```

- (9) [kaun-sii kitaab]₁ ek prasiddh bhaashaavid-ne t₁ likh-ii
 which-F book one famous linguist-ERG write.PST-FEM
 ‘Which book did a famous linguist write?’

Annotate an example with `\hfill [annotation]` in the `\glt` section.

- (10) [kaun-sii kitaab]₁ ek prasiddh bhaashaavid-ne t₁ likh-ii
 which-F book one famous linguist-ERG write.PST-FEM
 ‘Which book did a famous linguist write?’ (Hindi)

```
... \\
\glt ‘Which book did a famous linguist write?’
\hfill (Hindi)
\end{exe}
```

This is easy but (in my opinion) ugly. I like to have comments in the top-right, in the gloss-line, not the bottom-right. This requires another package called `cgloss` (loaded after `gb4e`). For this to work, you’ll need to add Alexis Dimitriadis’s `cgloss.sty` to your project, available from <http://staticweb.hum.uu.nl/medewerkers/alexis.dimitriadis/latex/cgloss.sty>.

- (11) [kaun-sii kitaab]₁ ek prasiddh bhaashaavid-ne t₁ likh-ii (Hindi)
 which-F book one famous linguist-ERG write.PST-FEM
 ‘Which book did a famous linguist write?’

To do this, add `\hfill (Hindi)` before `\glt` but after `\\`.

A quick note about brackets! In (10), I used a bracketed expression. For some reason, `gb4e` hates [being the first thing after `\gll`. Just enclosing the bracket in `$` fixes this.

You can add a grammaticality judgment after `\ex` by enclosing it in square brackets and putting braces around the rest of the sentence.

```
\begin{exe}
\ex[*]{This a bad sentence.}
\end{exe}
```

- (12) * This a bad sentence.

With glossed examples, put braces around the whole thing.

- (13) * yeh kharaab vaaky. (Hindi)
 this bad sentence
 Intended: ‘This is a bad sentence.’

Doing it another way will mean the judgment marker `*` will screw up the alignment and will need a `{}` in the gloss.

```

\begin{exe}
  \ex[*]{
    \gll yeh kharaab vaaky. \\  

    this bad sentence \\  

    \hfill (Hindi)  


    \glt Intended: ‘This is a bad sentence.’}
\end{exe}

```

1.4 Arrows in examples

Sometimes, you don’t want to draw a whole tree. (Or you don’t know how because we haven’t got there yet.) All the same, you might want to show a syntactic transformation. `gb4e` allows this natively but it’s a pain in the neck.

We’ll look at a package by Alan Munn (UMD [*93](https://github.com/alanmunn); [amunn.github.io](https://github.com/alanmunn)) called `movement-arrows` (don’t forget the -). This defines some helpful macros that allow annotating examples with arrows.

(14) What did a famous linguist write t ?


```

\begin{exe}
  \ex \mkword{What} did a famous linguist write \mkword{t}?
  \mvarrow{t}{What}
\end{exe}


```

Rather than repeating the word itself, you can label the nodes with square brackets. This is particularly helpful in representing ‘unpronounced copies’.

```

\begin{exe}
  \ex \mkword[target]{What} did a famous linguist
  write \mkword[source]{\sout{what}}?
  \mvarrow[source]{target}
\end{exe}


```

(15) What did a famous linguist write what ?


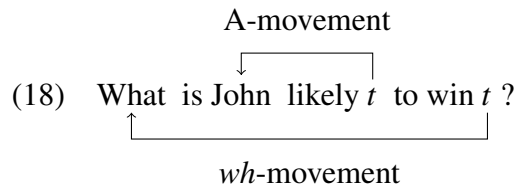
If you want the arrow to appear above the example, use `\mvarrow*`.

(16) What did a famous linguist write what ?


I am not sure why but you may need to adjust spacing of the text above or below using `\arrowstrut`. Add labels with square brackets like `[above = label]` or `[below = label]`.

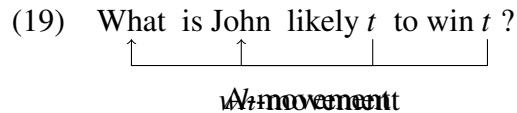
(17) What did a famous linguist write what ?

wh-movement

You can have multiple arrows in a single example.



```
\begin{exe}
  \ex \arrowstrut
      \mkword[wh]{What} is \mkword[subj]{John} likely
      \mkword[npt]{$t$} to win \mkword[wht]{$t$}?
      \mvarrow*[above = A-movement]{npt}{subj}
      \mvarrow[below = $wh$-movement]{wht}{wh}
\end{exe}
```

Suppose you wanted both arrows to be below. Disaster looms.

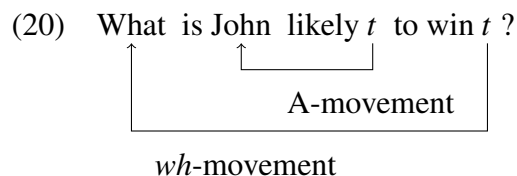


Oh dear.

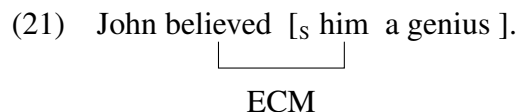
Adjust the height of an arrow with `\setlength{\arrowheight}{}`, and the position of the label with `xshift=` in the square brackets.

```
\begin{exe}
  \ex \mkword[wh]{What} is \mkword[subj]{John} likely
      \mkword[npt]{$t$} to win \mkword[wht]{$t$}?

      \mvarrow[below = A-movement, xshift=2em]{npt}{subj}
      \setlength{\arrowheight}{3em}
      \mvarrow[below = $wh$-movement, xshift=-2em]{wht}{wh}
\end{exe}
```



The package also includes an alternative to arrows in `\mvlink`. This is useful for representing, e.g., agreement dependencies.



In a glossed example, just include `\arrowgloss` after `\ex`, and put braces around the whole thing.

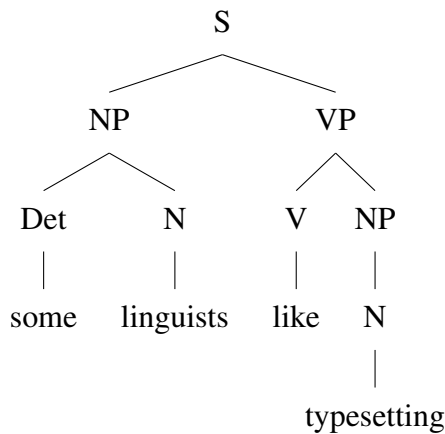
```
\begin{exe}
  \ex
    {
      \arrowgloss
      \gll ... \\
      ... \\
      \glt ...
    }
\end{exe}
```

2 Syntactic trees

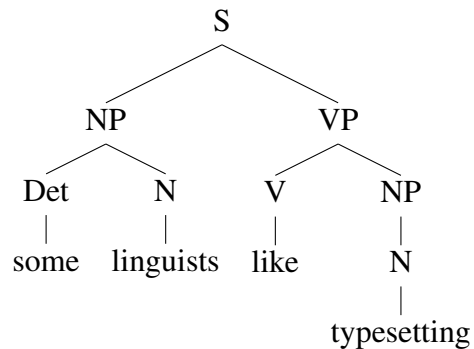
Everyone loves a good syntactic tree. No one likes an ugly one.

My favorite package for drawing trees is forest. Some old hands prefer packages like qtree. Here is a side-by-side comparison of the default trees in each.

(22) a. forest



b. qtree



Here is also a comparison of the code required to generate them.

```
(22') a. \begin{forest}
        [S [NP [Det [some]] [N [linguists]]]
          [VP [V[like]] [NP [N [typesetting]]]]]
\end{forest}

b. \Tree [.S [.NP [.Det [.some ] ] [.N [.linguists ] ] ]
        [.VP [.V [.like ] ] [.NP [.N [.typesetting ] ] ] ] ]
```

The spacing on a forest tree is extremely customizable. Many find the qtree trees outmoded. In fact, the width of the qtree diagrams was what led to the birth of forest. It is worth noting that

qtree is very finicky about the code - **including whitespaces**. This means the code and the trees are uglier.

So, I vastly prefer forest. It comes with a default [linguistics] option, so be sure to type `\usepackage[linguistics]{forest}`. Your trees will look *really* ugly if you don't.

2.1 Basic trees and triangles

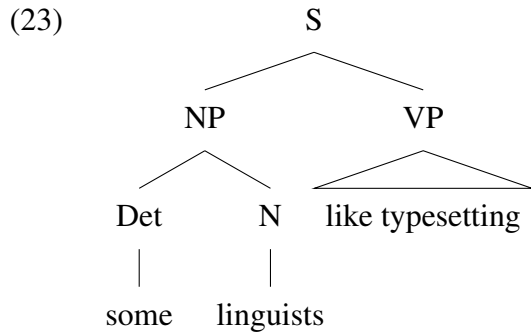
In forest, `\begin{forest}` will create an environment that allows syntactic bracket notation to be read as instructions to draw a tree. If you find the notation confusing, it can help to use line-breaks and indentation to make things a little more readable.

```
\begin{forest}
  [S
    [NP
      [Det
        [some]
      ]
      [N
        [linguists]
      ]
    ]
    [VP
      [V
        [like]
      ]
      [NP
        [N
          [typesetting]
        ]
      ]
    ]
  ]
\end{forest}
```

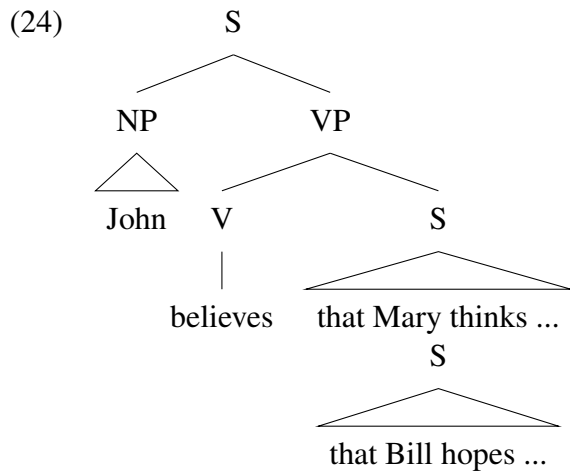
Basically, sharing a level of indentation means sharing a level of embedding. I personally don't care much for this way of formatting things.

Sometimes syntacticians are in a foul mood and simply do not want to draw a tree. Or, on a very very rare occasion, we feel a little lazy. Use '`, roof`'.

```
\begin{forest}
  [S [NP [Det [some]] [N [linguists]]]
    [VP [like typesetting, roof]]
\end{forest}
```



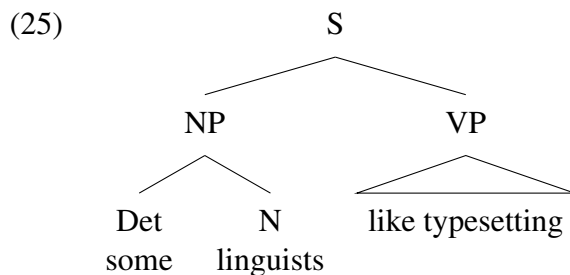
You can still embed after roof.



```
\begin{forest}
  ... [S [that Mary thinks ... \\ S, roof
    [that Bill hopes ..., roof]]] ...
\end{forest}
```

Notice how the S dominating the most embedded sentence is actually not part of that bracket. If you want to get rid of the lines between the non-terminal and terminal nodes (and this can be theoretically important), use \\ instead of [].

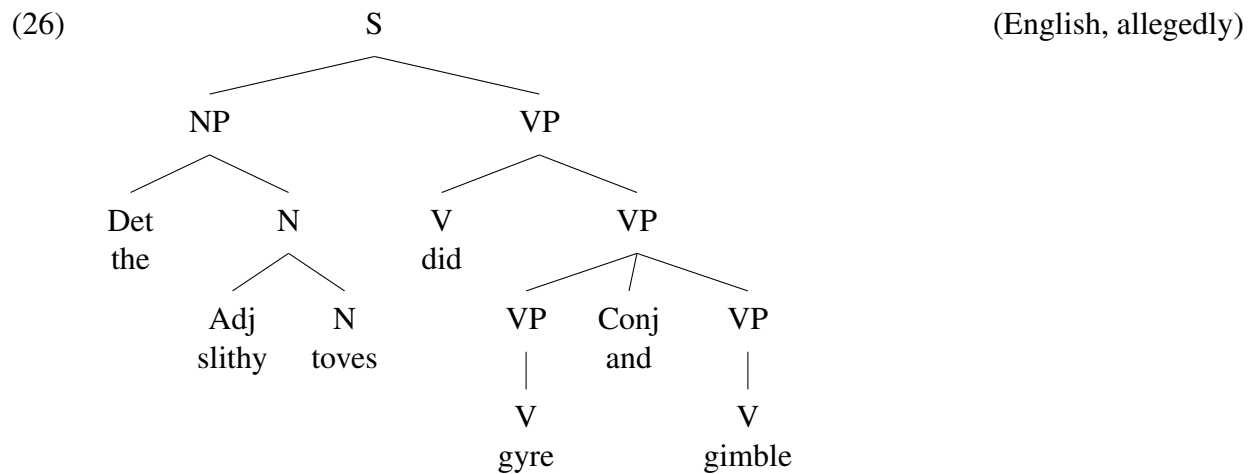
```
\begin{forest}
  [S [NP [Det \\ some] [N \\ linguists]]
  [VP [like typesetting, roof]]]
\end{forest}
```



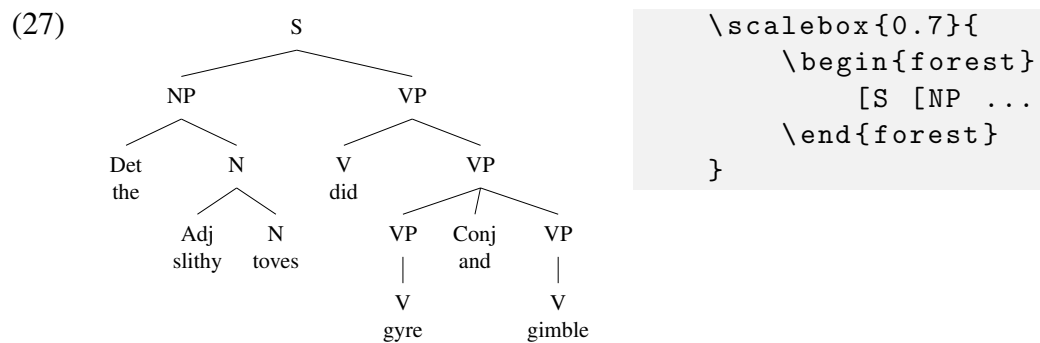
Labelling a tree works fine with `\hfill (Label)` after the tree.

```
\begin{exe}
  \ex \begin{forest}
    [S [NP [Det \ the][N [Adj \ slithy][N \ toves]]]
      [VP [V \ did] [VP [VP [V \ gyre]] [Conj \ and]
        [VP [V \ gimble]]]]]
  \end{forest}
  \hfill (English, allegedly)
\end{exe}
```

Note that this works *without* the `cgloss.sty` package.



Adjust the size of a tree with `\scalebox{n}{}` where `n` is the scaling factor.



Syntacticians will often like to leave nodes unlabelled to reduce clutter. Sadly, by default, the [linguistics] style for forest does not produce the beautiful tree in (28a), but the yucky tree in (28b).



However, it does come with an option called, aptly, `nice empty nodes`. Just typing it in at the start of the forest environment will put it into use.

```
begin{forest}
nice empty nodes
  [FP [YP] [[F] [XP]]]
\end{forest}
```

You can define a style using `\forestset{}`.

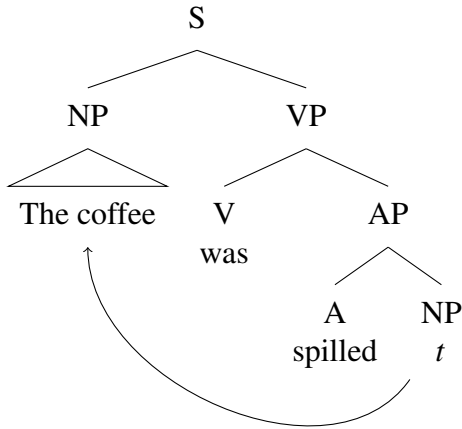
```
\forestset{
  style_name/.style={
    for tree{
      ...
    }
  }
}
```

To change the default style for all your trees, use `\forestset{default preamble = { ... }},` and insert your preferred style. There is more at the end about editing styles.

2.2 Drawing on trees: movement and phase-boundaries

Syntacticians can't say 'movement' without making a corresponding hand-gesture on an imaginary tree in front of them. Let's get that on the .pdf too.

(29)



```
\begin{forest}
  [S [NP [The coffee, roof, name=subj]]
  [VP [V \\\ was] [AP [A \\\ spilled]
  [NP \\\ $t$, name=trace]]]]
  \draw[->] (trace) to[in=-90, out=235] (subj);
\end{forest}
```

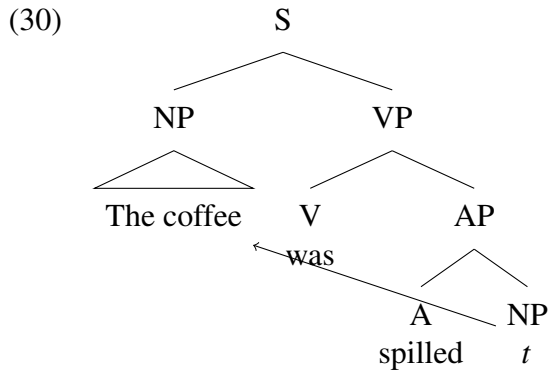
Here are the relevant ingredients: I named the nodes involved in the movement with `, name=`. After the tree was done, I used `\draw[->]`. This is from TikZ. Learning to use that is a course in itself.

It is worth dwelling on the anatomy of `\draw`.

- It ends with a semi-colon!
- It needs a source and a target - much like a syntactic transformation. These are indicated by the `(trace)` and `(subj)` which flank `to`.
- It needs to know what kind of arrow to draw: `[->]`, `[<->]`, `[<-]` are all valid, but for raising transformations, you'll want `[->]`.
- It *can* (but need not) be told the angles of its journey. These are the `in` and `out` options for `to`. That is, this is perfectly valid code.

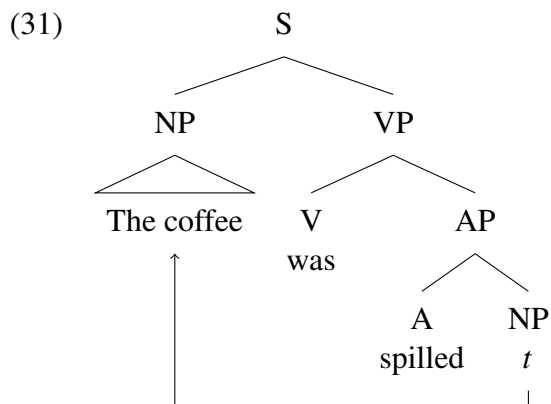
```
\draw[->] (trace) to (subj);
```

But this will draw an arrow 'as the crow flies', or as the ant dead-reckons.



Angles are fine as the options for to - that is what I have used in my code above - as are compass directions, as in to[in=south, out=south west].

Some people prefer blockier lines for movement.



Instead of to, I used the following code.

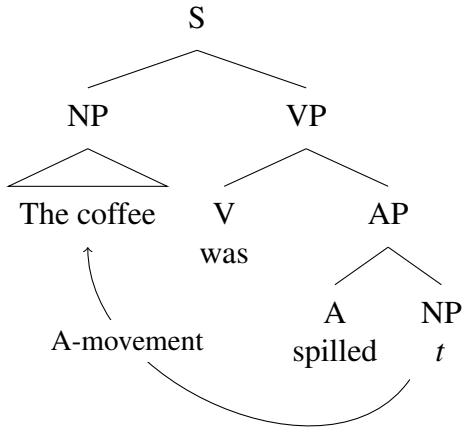
```
\draw[->] (trace) |- +(0, -2em) -| (subj);
```

The +(0, -2em) means the arrow will be offset downwards by 2em, leading to the slight vertical line below *t*. The |- and -| mean there is a vertical line (possibly of length=0) at the start and end points of the arrow.

You can label the arrow with node.

```
\begin{forest}
  [S [NP [The coffee, roof, name=subj]]
  [VP [V \\\ was] [AP [A \\\ spilled]
  [NP \\\ $t$, name=trace]]]]
  \draw[->] (trace) to[in=-90, out=235] (subj);
\end{forest}
```

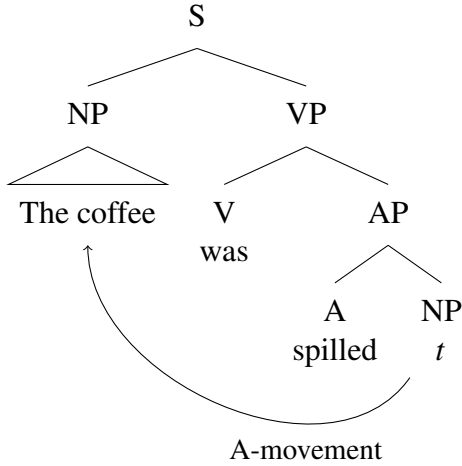
(32)



```
\draw[->] (trace) to[in=-90, out=-125]
node[pos=0.75, fill=white, font=\small]{A-movement}
(subj);
```

pos ranges from 0 (start of path) to 1 (end of path). fill means there is a background to the text. If you want the label to float near the arrow rather than sit on it, you can use xshift and yshift.

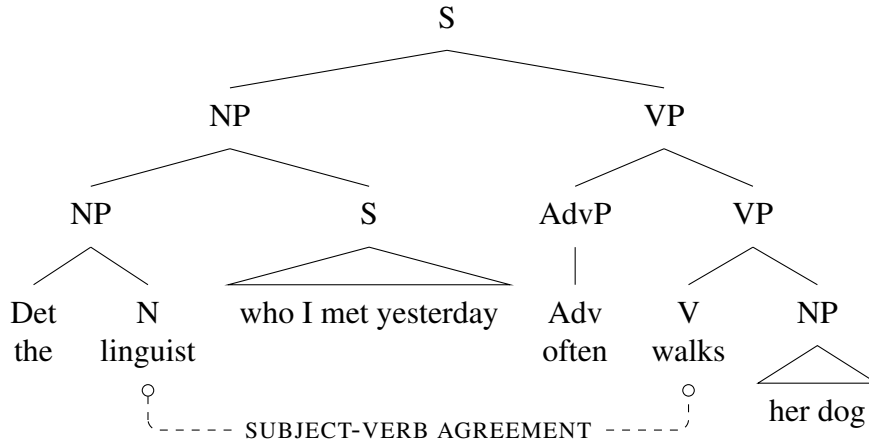
(33)



```
\draw[->] (trace) to[in=-90, out=-125]
node[pos=0.3, font=\small, yshift=-0.8em]{A-movement}
(subj);
```

If you want more arrow styles, you may need \usetikzlibrary{arrows}. This allows you to use, e.g., [o-o], which I quite like for Agreement dependencies.

(34)

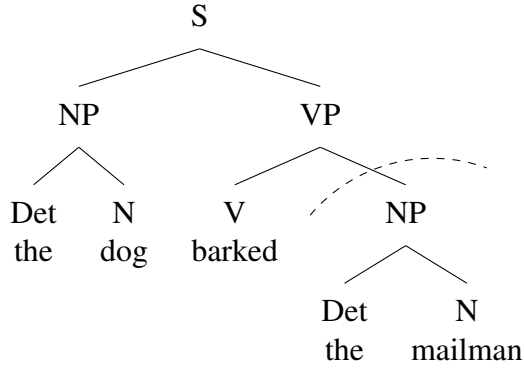


```
\draw[o-o, dashed, rounded corners] (subj) |- +(0, -3em) -|
node[pos=0.25, font=\small, fill=white
{\textsc{subject-verb agreement}}
(verb);
```

You can see some additional options like dashed and rounded corners but you should now be in a position to understand this.

We can also use `\draw` to help with boundaries, or marking domains of the syntax as important. For example, if I want to mark the edge of an NP as a ‘cyclic domain’, I can.

(35)

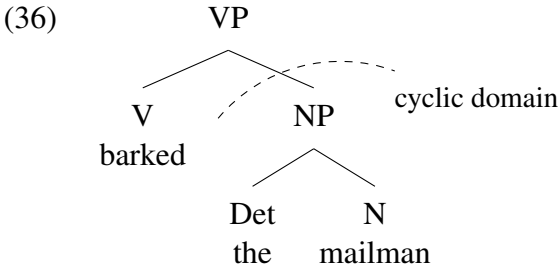


```
\draw[dashed] ([xshift=-3em]NP_edge)
arc[start angle=140, end angle=70, radius=5em];
```

`NP_edge` was the name for the object NP node. This draws an arc that leaves the start point ($x=-3em$ from the object NP node) at 140° and ends up reaching the end, $5em$ away, at 70° .

We can add a label to this arc in the usual way.

```
\draw[dashed] ([xshift=-3em]NP_edge)
arc[start angle=140,end angle=70,radius=5em]
node[pos=1.3, xshift=1em, font=\small]{cyclic domain};
```

One last cool thing about drawing on trees.

forest allows you to just use the current node without labelling it. Rather than naming the NP node, I can add the `\draw` command, enclosed in some braces, just after the square bracket that closes the relevant node. The source is an empty bracket (here with `[xshift]`).

```

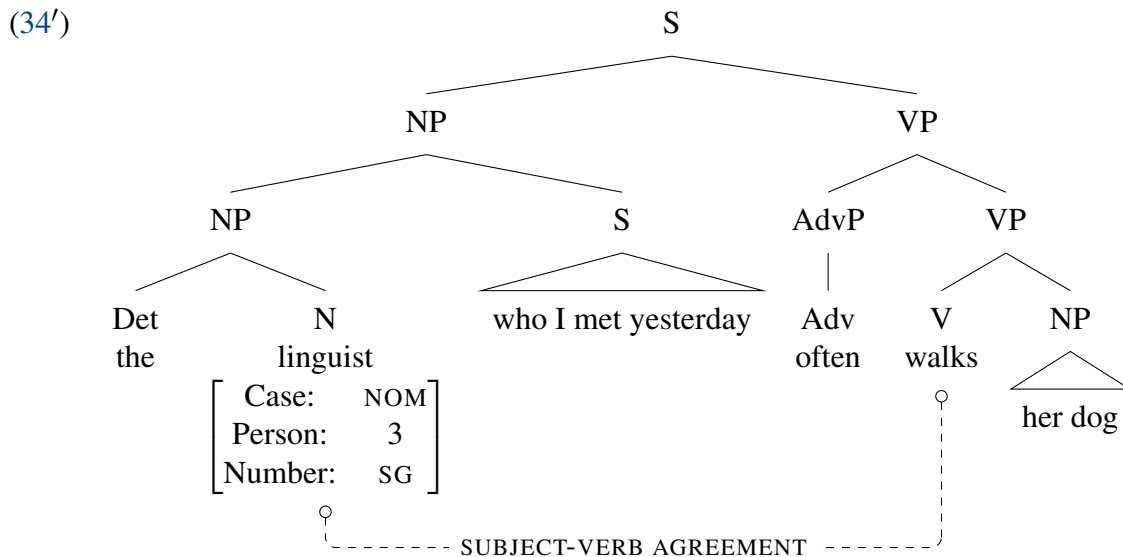
\begin{forest}
  [VP [V \ \ barked]
    [NP [Det \ \ the] [N \ \ mailman]]
    {\draw[dashed] ([xshift=-3em])
      arc[start angle=140,end angle=70,radius=5em]
      node[pos=1.3, xshift=1em, font=\small]{cyclic domain};}
  ]
\end{forest}

```

This produces exactly the same diagram as (36). So, you should pretty much never need to label a trace of movement. Just input the `\draw[->] ()` to `[...` immediately after its closing bracket.

2.3 Mathmode in trees: features and semantics

Non-terminal nodes often have features, and it is often important to talk about their features. Here's (34) repeated with some features for the subject NP.



Here's the code.

```
\begin{forest}
  [S [NP [NP [Det \ the]
  [N \ linguist \
  {$\begin{bmatrix}
    \text{Case:} & \textsc{nom} \\
    \text{Person:} & \text{3} \\
    \text{Number:} & \textsc{sg}
  \end{bmatrix}$}
  , name=subj]]
  [S [who I met yesterday, roof]]]
  [VP [AdvP [Adv \ often]]]
  [VP
    [V \ walks \]

    {\draw[o-o, dashed, rounded corners]
      ([yshift=-1.5em]) |- +(0, -5em) -|
      node[pos=0.25, font=\small, fill=white]
      {\textsc{subject-verb agreement}}
      (subj);}

    [NP [her dog, roof]]]]]
\end{forest}
```

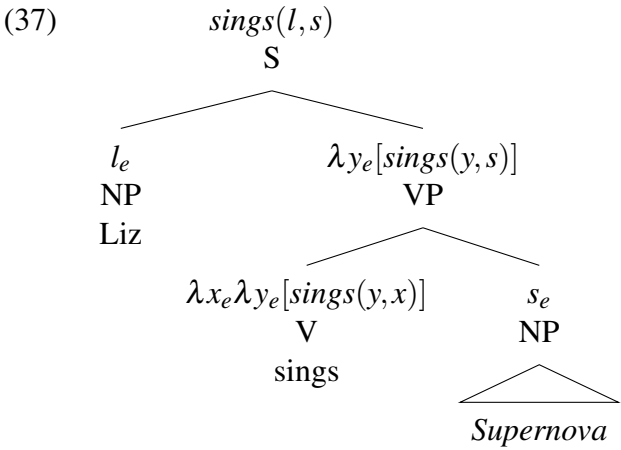
Do you see that I did the trick where I didn't name the V node but just added `\draw` after its closing bracket? Neat, right? I also needed to add `yshift` to get it to work right, but that was easy.

Here's the specific code for the feature matrix, which required the `amsmath` package. (It's a good idea to load this *before* `gb4e`.)

```
[N \ linguist \
  {$\begin{bmatrix}
    \text{Case:} & \textsc{nom} \\
    \text{Person:} & \text{3} \\
    \text{Number:} & \textsc{sg}
  \end{bmatrix}$}
  , name=subj]]
```

As you can see, just by enclosing it in braces, `forest` let me just use math-mode as normal.

This can also be useful for adding semantic formulae to trees.



```

\begin{forest}
  [${sings(l, s)}$ \ \ S
    [${l_e}$ \ \ NP \ \ Liz ]
    [${\lambda y_e [sings(y, s)]}$ \ \ VP
      [${\lambda x_e \lambda y_e [sings(y, x)]}$ \ \ V \ \ sings]
      [${s_e}$ \ \ NP] [\textit{Supernova}, roof]]
    ]
\end{forest}

```

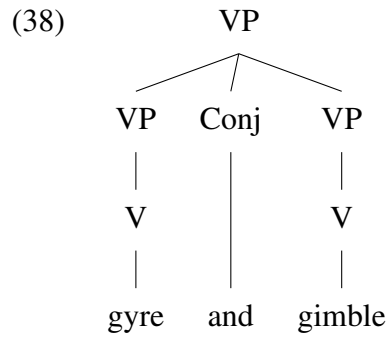
The one thing to be careful of here is to make sure you use braces after the \$s, otherwise the square brackets you use *within* mathmode (in the semantic formulae) are in danger of being interpreted as instructions to draw a forest branch.

I think this is pretty much the ‘core’ L^AT_EX you need to do syntax. The rest of the handout explores some more customization options and shows some advanced L^AT_EX you need for some advanced syntax.

3 More advanced forestry

3.1 Editing alignment, and styles

There's a lot you can do in a style. You can adjust the spacing between nodes and the length of the branches. For instance, if you want the terminal nodes to be aligned, you can use `tier=word`.

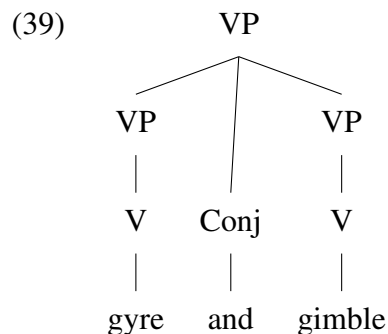


It doesn't matter what you call the tier.

```
\begin{forest}
  [VP [VP [V [gyre, tier=word]]]
  [Conj [and, tier=word]]
  [VP [V [gimble, tier=word]]]]
\end{forest}
```

You can have multiple tiers in one tree and they will all be aligned.

```
\begin{forest}
  [VP [VP [V,tier=head [gyre, tier=word]]]
  [Conj, tier=head [and, tier=word]]
  [VP [V, tier=head [gimble, tier=word]]]]
\end{forest}
```



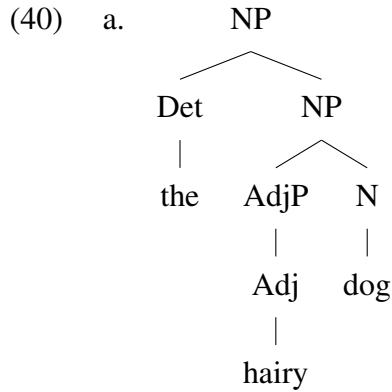
If you try and make a node the same `tier` as a node it dominates, `forest` will crash. So don't.

There's a few important pieces of alignment in `forest`. These include `calign`, `s sep`, and `l`.

- `calign` is the alignment of a parent node with respect to its children.

- s sep is the horizontal spacing between nodes.
- l is a rather complicated beast which is the vertical separation between nodes.

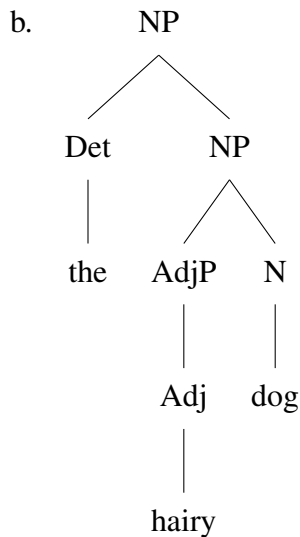
We can mess around with these at the start of a tree with for tree.



```

\begin{forest}
  for tree={l sep=0}
  [NP [Det [the]]
  [NP [AdjP [Adj [hairy]]]
  [N [dog]]]]
\end{forest}

```

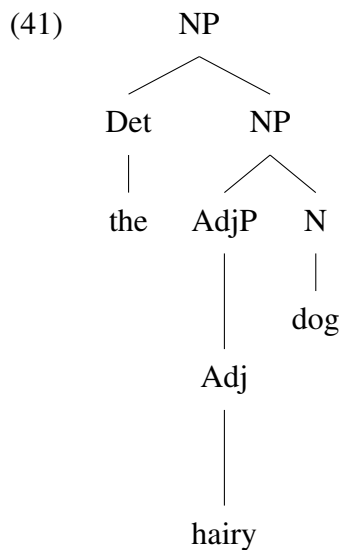


```

\begin{forest}
  for tree={l sep=2em}
  [NP [Det [the]]
  [NP [AdjP [Adj [hairy]]]
  [N [dog]]]]
\end{forest}

```

We can even change them for a subtree.

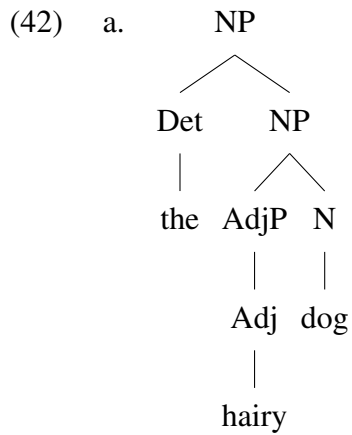


```

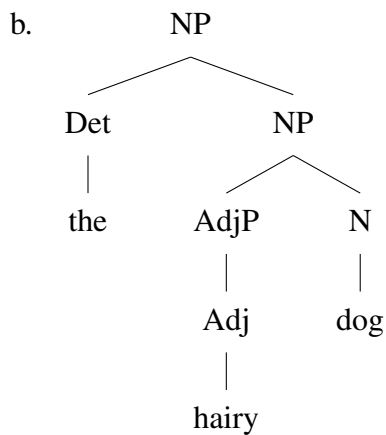
\begin{forest}
  [NP [Det [the]]
  [NP
  [AdjP,for tree={l sep=3em}
  [Adj [hairy]]]
  [N [dog]]]]
\end{forest}

```

Now let's look at `s sep`.



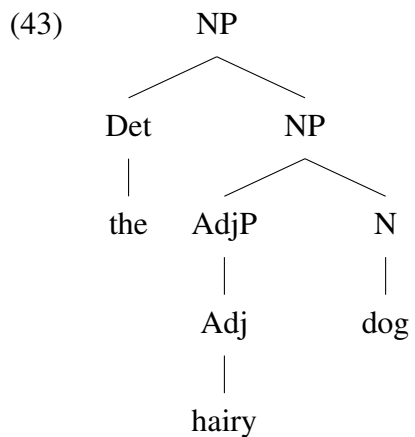
```
\begin{forest}
  for tree={s sep=0}
  [NP [Det [the]]
  [NP [AdjP [Adj [hairy]]]
  [N [dog]]]]
\end{forest}
```



```
\begin{forest}
  for tree={s sep=2em}
  [NP [Det [the]]
  [NP [AdjP [Adj [hairy]]]
  [N [dog]]]]
\end{forest}
```

Strictly speaking, `s sep` does *not* change the spacing between two nodes. If there are two sibling nodes [X Y], `s sep` defines the distance between the right edge of X's subtree and the left edge of Y's subtree. This means you never have to worry about your trees accidentally overlapping when changing values for `s sep`.

`calign` is the most involved. It is the way a parent node is horizontally aligned with its children. Here is one nice option it can take.



```
\begin{forest}
  for tree={
    calign=fixed edge angles,
    calign angle = 65}
  [NP [Det [the]]
  [NP [AdjP [Adj [hairy]]]
  [N [dog]]]]
\end{forest}
```

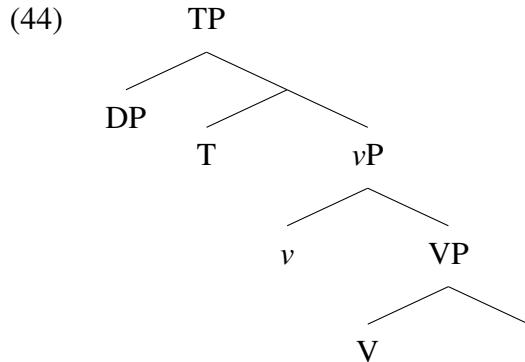
The angle between a node and its children will always be 65°. (This is quite qtree like.) I often use the following settings. I don't like the default `s sep` or `l settings` for nice empty nodes.

```

delay={where content={
  {shape=coordinate, for siblings={anchor=north}}{}}},
for tree={calign=fixed edge angles, calign angle=65, l=1em}

```

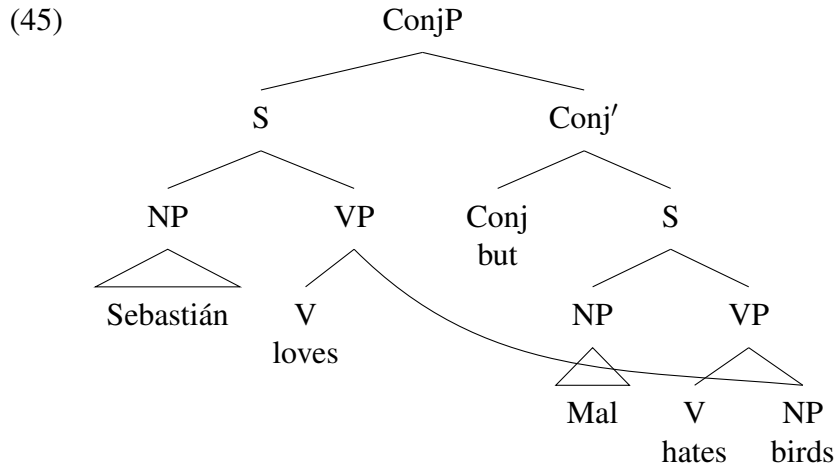
This allows me to draw trees like this. But it doesn't play terrifically nicely with ternary branching.



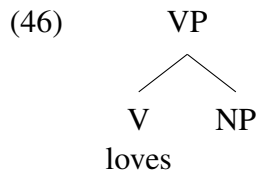
These are fairly qtree like, but they still end up being much less wide. There should be enough here for you to tweak things to your own preference.

3.2 Advanced syntax: multidominance and spanning

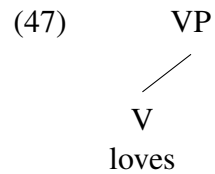
Sometimes, syntacticians want to treat one constituent as simultaneously the daughter of two nodes.



We already have most of the resources required to do this. The one major addition is `phantom`. This spooky command will make a branch invisible. The first VP is actually as in (46) with the left branch erased. (For 'mirror theory' aficionados, `phantom` is a godsend.)

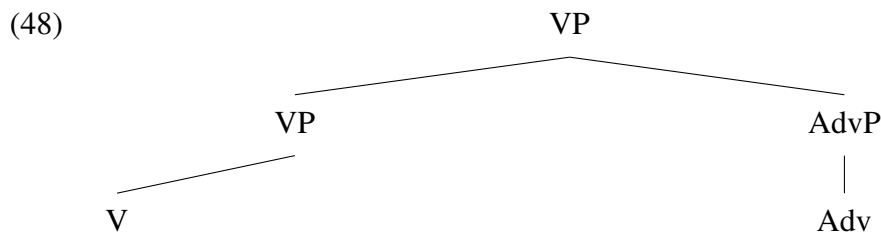


```
\begin{forest}
  [VP [V \\ loves]
    [NP, phantom]]
\end{forest}
```



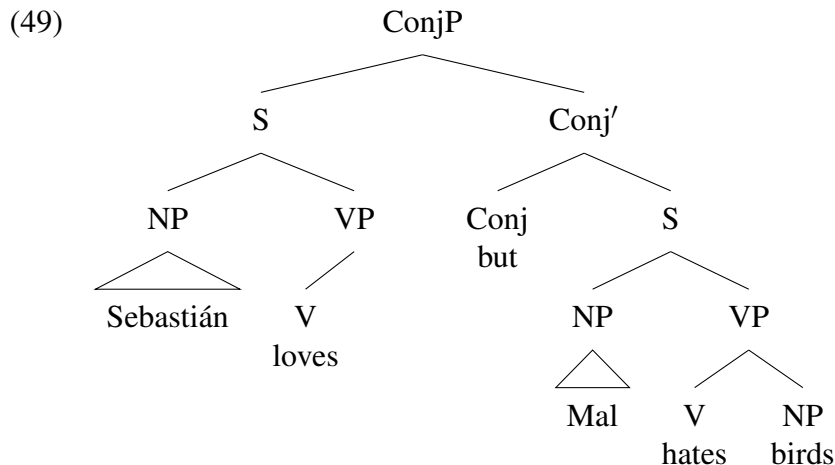
forest still pretends there is NP written in the left branch for spacing reasons. That is, if I had written more, the VP would be wider.

```
\begin{forest}
  [VP [VP [V
    [a really really really long sentence to show you, phantom]]
    [AdvP [Adv]]]
\end{forest}
```



Note that phantom also eradicates the label. If you want to keep the label but just delete the edge, use `no edge` instead of `phantom`.

Now that we have our invisible object NP, we need to link the left VP to the right VP with `\draw{}`.



I used `{\draw[-](.south) to[out=-45, in=175] (obj.north);}` after the first VP bracket, and named the object NP `obj`. `.south` means it will start from the south of that node, i.e., the bottom of the VP label, and then it will go to the north of the NP label.

```
\begin{forest}
```

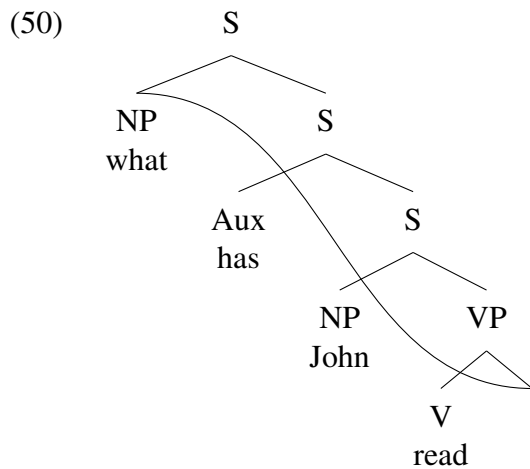


```

[ConjP [S [NP [Sebasti\prime{a}n, roof]]
[VP
[V \ \ loves] [NP ,phantom]
]
{\draw[-] (.south) to[out=-45, in=175] (obj.north);}
]
[Conj\prime$ [Conj \ \ but] [S [NP [Mal, roof]]
[VP [V \ \ hates]
[NP \ \ birds, name=obj]]]
]]
\end{forest}

```

Some syntacticians will use such trees instead of traditional movement dependencies.

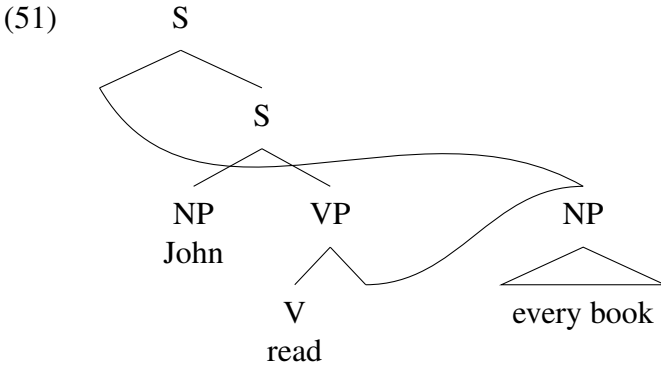


```

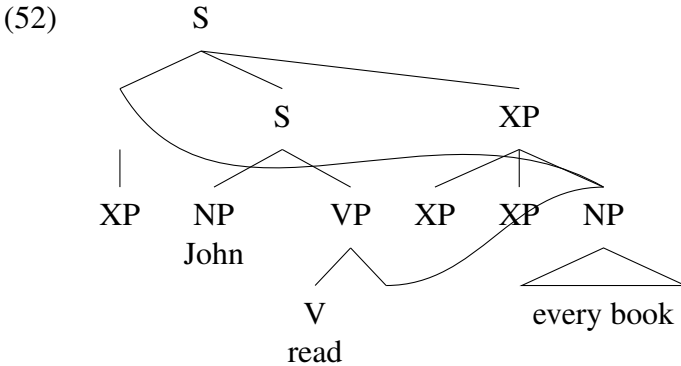
\begin{forest}
[S [NP \ \ what, name=wh]
[S [Aux \ \ has]
[S [NP \ \ John]
[VP [V \ \ read] [\ \ , name=object [NP, phantom]]]
]]]
\draw[-] (wh.north) to[out=0, in=180] (object.north);
\end{forest}

```

Suppose we want a phrase to be displayed off to the side with a connection to the relevant nodes.



This is a little more complicated. I used a bunch of phantom sisters to embedded S to set up the tree. Here is what it looks like without any phantoms.



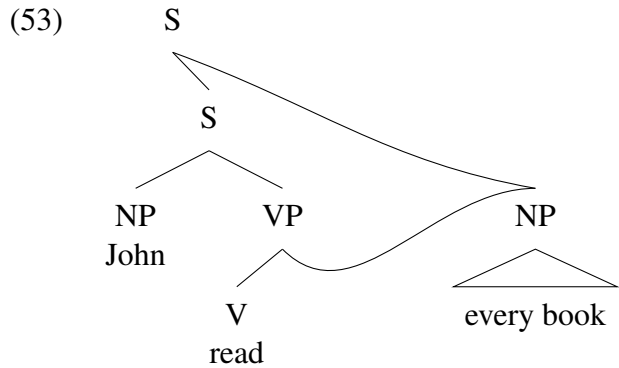
```

\begin{forest}
  [S, calign=midpoint, calign primary child = 1,
  calign secondary child = 2
  [\, name=scope [XP,phantom]]
  [S [NP \, John] [VP [V \, read] [\, name=object]]]
  [XP,phantom [XP,phantom] [XP,phantom]]
  [NP, name=qfier]]
  \draw[-] (scope.north) to[out=-60, in=150] (qfier.north);
  \draw[-] (object.north) to[out=0, in=180] (qfier.north);
\end{forest}

```

This requires forcing the topmost S node to align horizontally between its invisible left daughter and its second daughter. Without that, it will try and align to the center of the whole tree. I did this using `calign=midpoint`, which aligns the node between its primary and secondary child. The default takes those to be its first and last child, but I stipulated with `calign primary child = 1` and `calign secondary child = 2` that it will be the first and second child. So, S will align to the midpoint of the first and second child.

If we wanted the line to come directly from the VP and S, this is also possible.

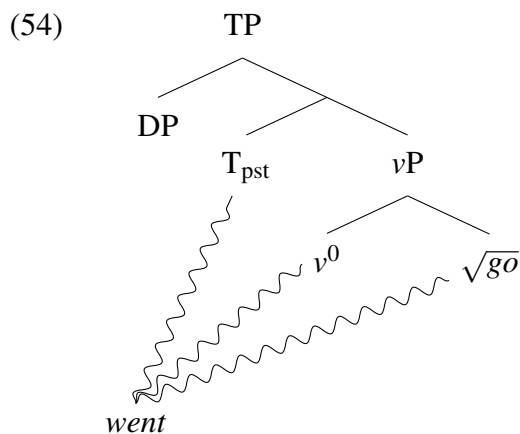


```

\begin{forest}
  [S, calign=midpoint, calign primary child = 1
  calign secondary child = 2, name=scope [XP, phantom]
  [S [NP \ \ John]
  [VP, name=object [V \ \ read] [XP, phantom]]]
  [XP, phantom [XP, phantom] [XP, phantom]
  [NP, name=qfier [every book, roof]]]]
  \draw[-] (scope.south) to[out=-20, in=170] (qfier.north);
  \draw[-] (object.south) to[out=-45, in=180] (qfier.north);
\end{forest}

```

Some syntacticians also believe in multiple heads being ‘exponed’ by one string, in a ‘span’. For example, an abstract Tense, verbalizer, and root element might all get together as *went*.



This, believe it or not, did not take much work. I added a node, offset from the categorizer head v , and named it exponent.

```

\node[align=center, font=\it] at
  ([xshift=-6em, yshift=-5em]categorizer)(exponent){went};

```

To draw the squiggly lines, I had to add `\usetikzlibrary{decorations.pathmorphing}`. That gave me the snake option for `\draw{}`.

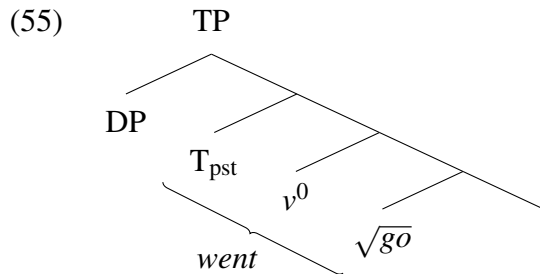
```

\draw[-, decorate, decoration=snake](exponent.north) to (tense);

```

Adding `decorate` and `decoration=snake` to the options for `\draw{}` meant that the line that would otherwise be straight was drawn with squiggles. I made sure to use `(exponent.north)`, because otherwise, the lines would all leave from different points.

Some people like to indicate spans with brackets. This is a little more involved because trying to get the `xshift` and `yshift` values right requires a lot of trial-and-error.



```
\draw [decorate, decoration={brace}]
      ([xshift=-2.5em, yshift=-0.1em]root.south east) --
      ([xshift=-0.5em, yshift=-0.6em]tense.west)

      node [pos=0.5, anchor=east, xshift=0.5em,
            yshift=-1em]{\textit{went}};
```

This required `\usetikzlibrary{decorations.pathreplacing}` for the brace decoration.

It's worth noting that unless you're grouping together a bunch of stuff on the same edge (like above), the brace is no fun to align.

4 Packages used

Here is an easy to reference list for the packages used for all the trees above.

```
\usepackage[linguistics]{forest}
\usepackage{hyperref}
\usepackage{amsmath}
\usepackage{gb4e}           %Load after amsmath!
\noautomath                %Required for gb4e to work nicely
\usepackage{cgloss}        %Load after gb4e!
                           %Also add cgloss.sty to your project.

\usepackage{movement-arrows}
\usetikzlibrary{decorations.pathmorphing}
\usetikzlibrary{decorations.pathreplacing}
\usetikzlibrary{arrows}
```