The Ti*k*Z-`dependency` package

Daniele Pighin

daniele.pighin@gmail.com

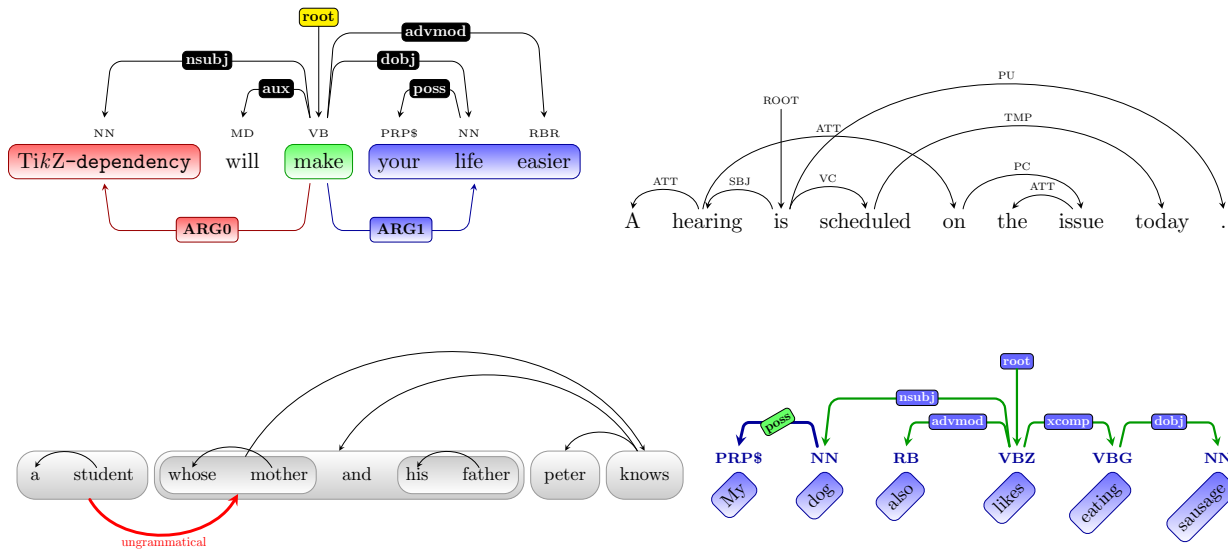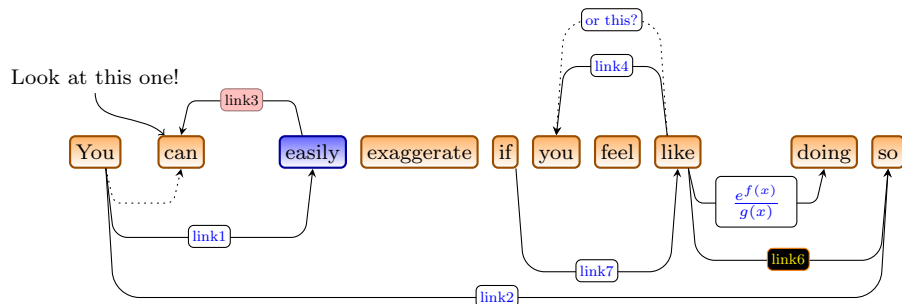http://danielepighin.net/cms

Version 1.1, January 2012

If your work is in any way connected with linguistics or natural language processing, chances are high that the moment will come when you will have to draw a dependency graph:

Ti*k*Z already has all the needed facilities to draw and style a dependency graph, but diving into the documentation just to understand how to draw one or two trees may turn out to be rather overwhelming.

Ti*k*Z-`dependency` is pretty flexible, and it can easily be used to draw quite complex graphs around sequences of tokens, not necessarily related to dependency parsing or to language, as shown in this useless toy example:

Hopefully, this package will make your life a little bit easier, and allow you to draw a dependency graph without too much Ti*k*Z prior knowledge. It will also allow you to draw cute graphs, which won't increase your impact factor but may help making your papers look more pleasant and elegant.

Contents

1 Installation

The simplest and fastest way to get things working is to copy the style file `tikz-dependency.sty` in the root directory of your latex project.

Alternatively, you can install it as you would install any other LaTeX package, by copying it in some directory searched by LaTeX and running `texhash`. The `texmf` directory under your home directory is most likely one of such directories (even though there are chances that it does not exist). If there is no `texmf` directory under your home, than create it:

```
$ cd ~
$ mkdir -p texmf
```

Extract the TikZ-dependency under this directory:

```
$ cd texmf
$ tar xvzf /path/to/tikz-dependency.tar.gz
```

Run `texhash` to let LaTeX know about the newly installed package:

```
$ texhash texmf
```

That should do it. Refer to LaTeX documentation for more details or in case of non-standard LaTeX installations.

Please, note that PGF/TikZ version 2.10 is required for TikZ-dependency to work as expected.

If TikZ is not installed on your system, you can easily install it in the same way. Visit http://sourceforge.net/projects/pgf/ with your favorite browser, download the latest version of the package and uncompress in some of the directories searched by LaTeX. The `texmf` directory under your home directory should be a good place to start:

```
$ cd ~
$ unzip /path/to/pgf_version.zip -d texmf
```

Run `texhash`:

```
$ texhash texmf
```

Everything should be fine now. For more detailed instructions on how to install TikZ, please refer to its manual at http://ftp.gui.uva.es/sites/ctan.org/graphics/pgf/base/doc/generic/pgf/pgfmanual.pdf.

To use TikZ-dependency, the first thing that you will need to do is to require it in your preamble:

```
\usepackage{tikz-dependency}
```

TikZ-dependency will automatically load TikZ and its other requirements.

2 A step-by-step introduction

In this section you will see how simple it is to draw a dependency graph. We will go throw all the required steps one by one, from the creation of a suitable environment, to the actual drawing of the graph, to the optional steps that can make your graph look nicer.

We will start by trying to reproduce the following example:

But first let's agree on some terminology. I will call **words** the tokens of the sentence, and **dependencies** the relations between words. A dependency actually consists of two entities: the **edge** between the two words and its **label**, which is the type of the relation.

2.1 An environment to draw dependency graphs

The first step will be to create a suitable environment to draw your dependency graphs.

\begin{dependency}[⟨*key-value-pairs*⟩]
  ⟨*environment contents*⟩
\end{dependency}

All the commands and configuration keys that you will see in this document are available within a dependency environment. Since it is an overloaded version of `tikzpicture`, all the things that you would do within a `tikzpicture` environment (such as drawing paths, edges, nodes, whatever) you can also do within a dependency. At the moment, the only reason why you'd rather use a dependency environment is because you can pass it some optional configuration keys (i.e., [⟨*key-value-pairs*⟩]) that can affect the style of the whole dependency graph. The documentation of these configuration keys is scattered around the documentation, though many of them are described in Section 6.

So, to draw a new dependency tree you will generally start by writing something like this:

```
\begin{dependency}
    %  The code to actually draw the
    %  dependency graph will go here
\end{dependency}
```

At this stage it looks pretty empty, so let's get started and fill it up with some contents.

2.2 Writing the sentence

\begin{deptext}[⟨*key-value-pairs*⟩]
  ⟨*environment contents*⟩
\end{deptext}

The deptext environment is used to lay out the words of the parsed sentence.

Within deptext, you will write a sentence separating each word with \&, similarly to what you would do in a `tabular` environment. Please, not that in this case the separator is actually a LaTeX command (i.e., a backslash followed by an ampersand), and not just a simple ampersand. To terminate the sentence, add the command \\ (i.e., two consecutive backslashes) after the last word, just as you would do to end a row in a table. The [⟨*key-value-pairs*⟩] can bee used to change the way the sentence looks, and more details about it are coming soon.

After adding the deptext environment with the text of our example sentence, the incomplete dependency graph would look like this:

My dog also likes eating sausage

```
\begin{dependency}
   \begin{deptext}
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
\end{dependency}
```

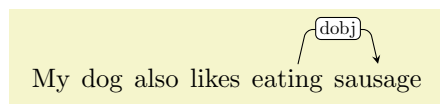For the TikZ-savvy user, deptext is just a wrapper around TikZ's own matrix environment. The matrix is a matrix of nodes, with nodes in empty cells and some default styling. All the [⟨*key-value-pairs*⟩] passed to the deptext are first parsed by TikZ-dependency, then handed to the underlying `matrix`. If you know what you are doing and you want more control on the final result, you can have it.

2.3   Adding dependencies

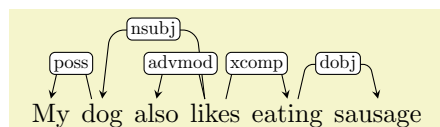\depedge[⟨*key-value-pairs*⟩]{⟨*from-word*⟩}{⟨*to-word*⟩}{⟨*label*⟩}

This command is used to add dependencies between words. The [⟨*key-value-pairs*⟩] argument is optional, and it can be used for styling the edge and its label (again, we will come to that part shortly). The three mandatory arguments define the origin and destination of the dependency edge, and the label of the dependency. The arguments {⟨*from-word*⟩} and {⟨*to-word*⟩} are the offsets of the words to be linked, counting from 1. {⟨*label*⟩} is just any string, to be used as a label for the edge. \depedge commands should be issued *after* closing the deptext environment.

To draw an edge between the words *eating* and *sausage* in the example, you would set {⟨*from-word*⟩} to 5, {⟨*to-word*⟩} to 6 and {⟨*label*⟩} to *dobj*. After adding this first edge, the example looks as follows:


```
\begin{dependency}
   \begin{deptext}
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \depedge{5}{6}{dobj}
\end{dependency}
```

By repeating this operation for all the dependencies in the sentence, we end up with a first complete version of the dependency tree:


```
\begin{dependency}
   \begin{deptext}
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
\end{dependency}
```

As you can see, Ti*k*Z-dependency automatically does some basic routing of the edges of the graph, by drawing dependency edges higher or lower depending on how far apart the two words are. The graph looks decent at this stage, but it looks a little bit cramped and it could definitely use some more space between the words.

2.4   Getting word spacing right

You can either add some space between all words, or you can add some space between two specific words. Let's see how it is done.

2.4.1   Setting the space between every two words

When I first introduced the deptext environment, I said in passing that it can take some optional arguments. Since deptext is a wrapper around Ti*k*Z's own matrix, if you know some Ti*k*Z you can pass to deptext any key-value pairs that matrix would understand. If you have absolutely no idea of how optional parameters (key-value configuration options) work in Ti*k*Z, you are warmly invited to take a look at Appendix A. If you do not understand what is going on, you will be able to copy the example but your ability to exploit the flexibility of the package will be largely impaired.

As far as we are concerned here, what we want to discuss is the `/tikz/column sep` configuration key, which can be used to add more space between all the words in the sentence.

In this documentation, I will always use the fully qualified name of a key when documenting it (see, for example, the description of `/tikz/column sep` immediately below. Anyway, you are not required to use the fully qualified name of keys when using them, and using their base name (e.g., `column sep`) is sufficient. In fact, in the examples I will generally use the shorter form.
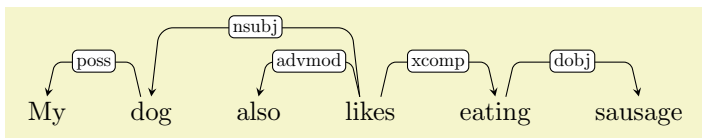

`/tikz/column sep`=⟨*length*⟩   (default `0cm`)

If you want to add, say, 0.2cm between every two words, you can do it so by passing `column sep`=`0.2cm` as the optional argument of the `deptext` environment.


```
\begin{dependency}
   \begin{deptext}[column sep=0.2cm]
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
\end{dependency}
```

It looks better already, but the fact that some dependency labels are longer than the horizontal segment of the edge doesn't look to nice. So, we could increase the space between all the words, for example to 0.7cm:
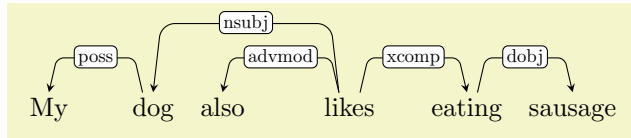


```
\begin{dependency}
   \begin{deptext}[column sep=0.7cm]
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
\end{dependency}
```

Now the labels are fine, but the words are a little bit too far apart, at least for my taste. So I would rather go back to `/tikz/column sep`=`0.2`, and find a more precise way of spacing only the problematic word pairs. In this case, they are adjacent words connected by a dependency with a long label, i.e., *My/dog*, *also/likes* and *likes/eating*.

2.4.2   Spacing word individually

Additional space between two words can be added by using the optional argument of the `\&` command that we used to separate words. For example, writing `\&`[`0.5cm`] will add 0.5cm to the default word spacing set with via the `column sep` option. If we take care of the three problematic pairs individually, we can obtain something like this:

```
\begin{dependency}
   \begin{deptext}[column sep=0.2cm]
      My \&[.5cm] dog \& also \&[.7cm] likes \&[.4cm] eating \& sausage \\
   \end{deptext}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
\end{dependency}
```
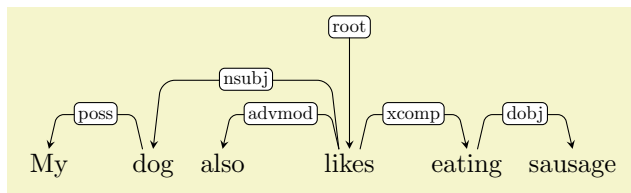
2.5   Adding a root node

If you want to add a to the dependency graph, you can do so by means of the `\deproot` command.

`\deproot`[⟨*key-value-pairs*⟩]{⟨*root-offset*⟩}{⟨*label*⟩}

It is basically a variant of `\depedge`. Instead of connecting two words, it will create a node labeled ⟨*label*⟩ just above the word having offset ⟨*root-offset*⟩, and an arrow between them. In our example, the root word is the verb *likes*, which occupies the fourth position:
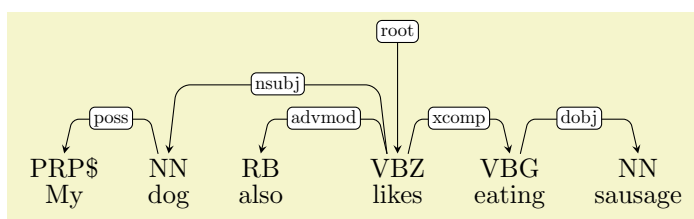


```
\begin{dependency}
   \begin{deptext}[column sep=0.2cm]
      My \&[.5cm] dog \& also \&[.7cm] likes \&[.4cm] eating \& sausage \\
   \end{deptext}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
   \deproot{4}{root}
\end{dependency}
```

This completes the crash course on the basics of dependency graph drawing. The next sections will tell you how to control edge positioning, how to build more complicated graphs with several layers of text, how to style the graphs to your liking and how to include your graphs in larger pictures.

3   Multiple layers of text

When drawing a dependency graph it is common to have a layer of part-of-speech (POS) tags just above the words. Achieving this with Ti*k*Z-`dependency` is pretty straightforward. Since the `deptext` environment is already a matrix, all you have to do is to add another row of text to the matrix:
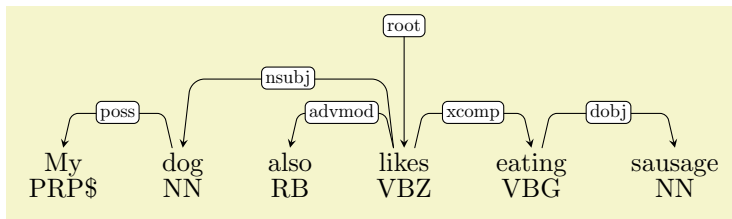
```
\begin{dependency}
    \begin{deptext}[column sep=.5cm]
        PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
        My \& dog \& also \& likes \& eating \& sausage \\
    \end{deptext}
    \deproot{4}{root}
    \depedge{2}{1}{poss}
    \depedge{4}{2}{nsubj}
    \depedge{4}{3}{advmod}
    \depedge{4}{5}{xcomp}
    \depedge{5}{6}{dobj}
\end{dependency}
```

In this case, we have added the POS tags on top of the words. Of course, we could have also added them below:


```
\begin{dependency}
    \begin{deptext}[column sep=.7cm]
        My \& dog \& also \& likes \& eating \& sausage \\
        PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
    \end{deptext}
    \deproot{4}{root}
    \depedge{2}{1}{poss}
    \depedge{4}{2}{nsubj}
    \depedge{4}{3}{advmod}
    \depedge{4}{5}{xcomp}
    \depedge{5}{6}{dobj}
\end{dependency}
```

/tikz/row sep=⟨*length*⟩   (default `0cm`)

Similarly to how you can increase the space between columns in a matrix, you can also increase inter-row spacing. For example, to add 0.5ex between every two rows you can add `row sep`=`0.5ex` to the optional argument of the `deptext` environment:

```
\begin{dependency}
    \begin{deptext}[column sep=.7cm, row sep=.5ex]
        My \& dog \& also \& likes \& eating\\
        PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG\\
    \end{deptext}
    \deproot{4}{root}
    \depedge{2}{1}{poss}
    \depedge{4}{2}{nsubj}
    \depedge{4}{3}{advmod}
    \depedge{4}{5}{xcomp}
    \qquad
\end{dependency}
\quad
\begin{dependency}
    \begin{deptext}[column sep=.7cm, row sep=2ex]
        My \& dog \& also \& likes \& eating\\
        PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG\\
    \end{deptext}
    \deproot{4}{root}
    \depedge{2}{1}{poss}
    \depedge{4}{2}{nsubj}
    \depedge{4}{3}{advmod}
    \depedge{4}{5}{xcomp}
\end{dependency}
```
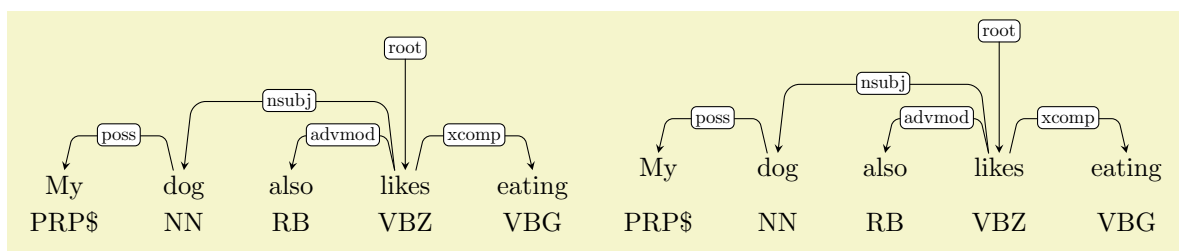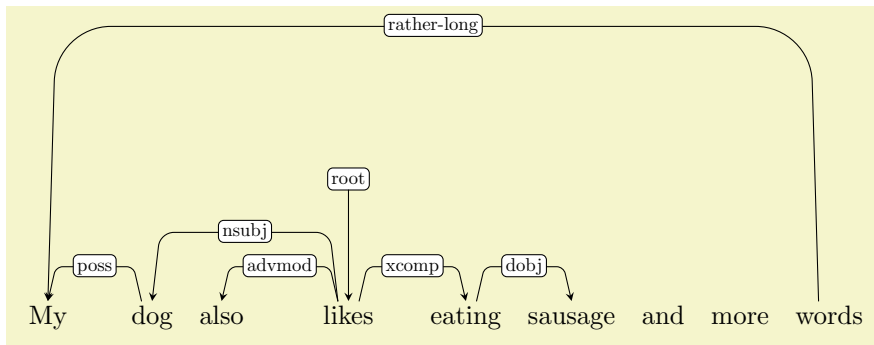
4   More control over edges

In Section 2.3 I hinted to the fact that the height of the horizontal segment of a dependency edge is proportional to the distance between the linked words. As you may have already guessed, this can be a problem for . Consider the following example, in which I added some more words and a long dependency:


```
\begin{dependency}
    \begin{deptext}[column sep=0.2cm]
        My \&[.5cm] dog \& also \&[.7cm] likes \&[.4cm] eating \& sausage \& and \& more \& words \\
    \end{deptext}
    \depedge{2}{1}{poss}
    \depedge{4}{2}{nsubj}
    \depedge{4}{3}{advmod}
    \depedge{4}{5}{xcomp}
    \depedge{5}{6}{dobj}
    \deproot{4}{root}
    \depedge{9}{1}{rather-long}
\end{dependency}
```

To decide how high a dependency should be drawn, Ti*k*Z-`dependency` uses the formula $s|x - y|$, where x and y are the offsets of the two words, and s is the value of the optional parameter /depgraph/edge unit distance.

/depgraph/edge unit distance=⟨*length*⟩   (default 3ex)

   This key represents the distance of the horizontal segment of the edge between two adjacent words. That is, if two words are 5 positions apart, the distance of the edge will be 5 times the value of /depgraph/edge unit distance. Whenever you draw a edge with \depedge you can override this value as part of the

optional argument of the command. For example, by changing its value in the previous and setting it to `1.5ex` it is possible to halve the total height of the problematic edge:


```
\begin{dependency}
   \begin{deptext}[column sep=0.2cm]
      My \&[.5cm] dog \& also \&[.7cm] likes \&[.4cm] eating \& sausage \& and \& more \& words \\
   \end{deptext}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
   \deproot{4}{root}
   \depedge[edge unit distance=1.5ex]{9}{1}{rather-long}
\end{dependency}
```
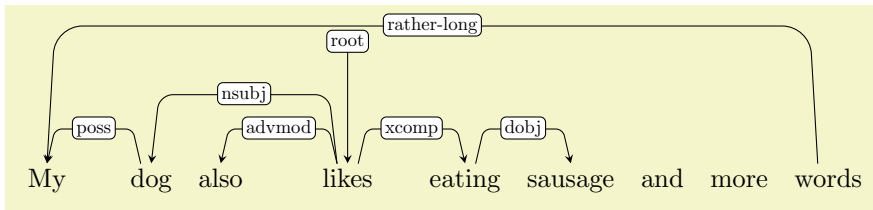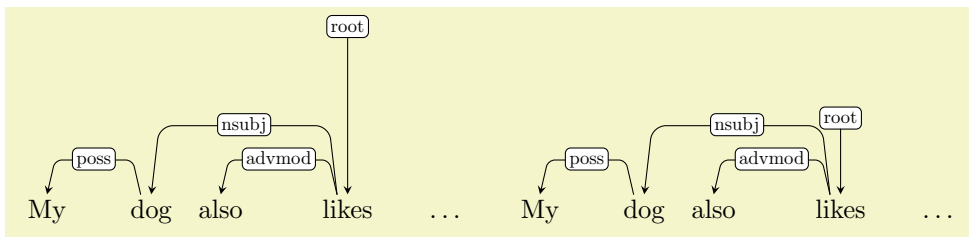
`/depgraph/edge unit distance` can also be used to tune the distance of the root node from the sentence. By default, the root node is set at a distance equal to 4 times the value of the parameter. By changing its value you can bring it closer of further:
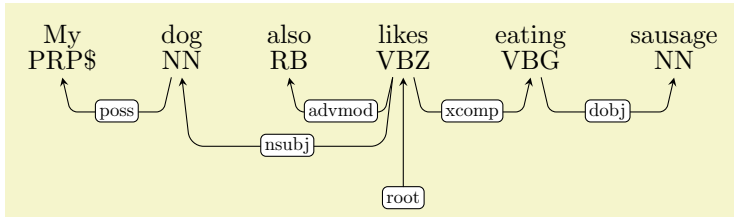


```
\begin{dependency}
   \begin{deptext}[column sep=0.2cm]
      My \&[.5cm] dog \& also \&[.7cm] likes \&[.4cm] \dots \\
   \end{deptext}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \deproot[edge unit distance=4ex]{4}{root}
\end{dependency}
\quad
\begin{dependency}
   \begin{deptext}[column sep=0.2cm]
      My \&[.5cm] dog \& also \&[.7cm] likes \&[.4cm] \dots \\
   \end{deptext}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \deproot[edge unit distance=2ex]{4}{root}
\end{dependency}
```

At this point, you may be wondering: why are we always drawing dependencies above the text? Indeed, turning links around is not only possible, but also quite easy, and the next key-value pairs allow you to control this behaviour.

/depgraph/edge above=⟨*boolean*⟩   (default `true`)
/depgraph/edge below=⟨*boolean*⟩   (default `false`)

These two keys control the position of the links, which can be drawn either above or below the layer(s) of text. You can pass any of them as part of the optional arguments of the `dependency` environment, and change the position of all the links in the graph at once. Writing the name of the key is sufficient, so writing `edge below`=true or just `edge below` produces the same result:
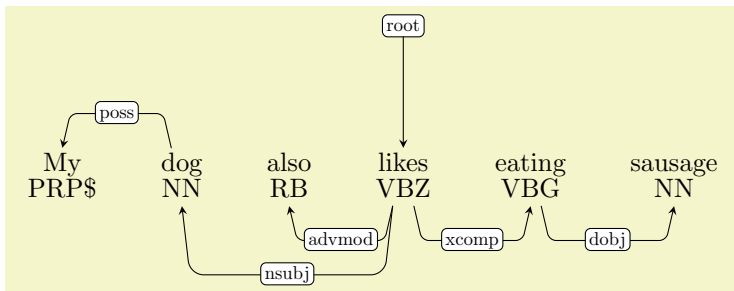
```
\begin{dependency}[edge below]
   \begin{deptext}[column sep=.7cm]
      My \& dog \& also \& likes \& eating \& sausage \\
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
   \end{deptext}
   \deproot{4}{root}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
\end{dependency}
```

/depgraph/edge below and /depgraph/edge above can also be used to control the positioning of individual edges. In this case, you won't pass the keys to the dependency environment, but instead to a specific \depedge or \deproot whose behaviour you want to modify. Say that you wanted all the edges to be drawn below, but the *root* and *poss* edges should be on top. You can pass /depgraph/edge below to dependency, and /depgraph/edge above to the two \depedges to be placed above:


```
\begin{dependency}[edge below]
   \begin{deptext}[column sep=.7cm]
      My \& dog \& also \& likes \& eating \& sausage \\
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
   \end{deptext}
   \deproot[edge above]{4}{root}
   \depedge[edge above]{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
\end{dependency}
```

Other than /depgraph/edge unit distance, three more parameters can affect the way edges are drawn. As shown in Figure 1, they control several aspects related to the spacing among edges and between edges and text, plus the inclination of the non-horizontal segments of the edge. All these properties can be set both at the dependency level (which will affect all the edges in the graph) or at the \depedge level, which will only affect individual links.

/depgraph/edge slant=⟨*length*⟩   (default 3pt)

   /depgraph/edge slant controls the slant of the diagonal segments in a dependency edge. As shown in Figure 1, what it actually controls is the horizontal offset between the initial (final) of the edge and its first (second) corner. By default it is set to 3pt, which makes the segments just slightly sloped. By setting it to 0, you can make the lateral segments completely vertical; by increasing its value you can make the edges look less slender.
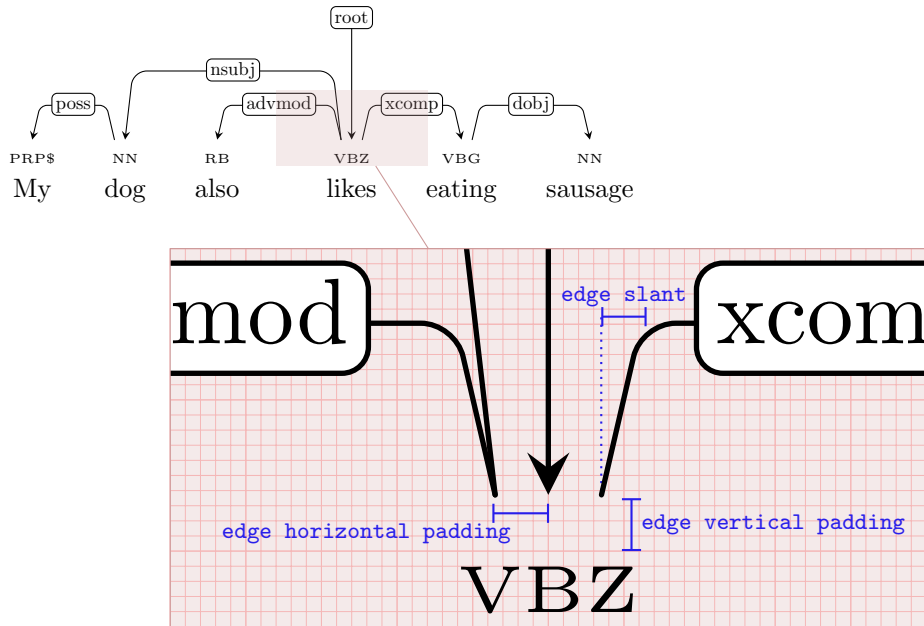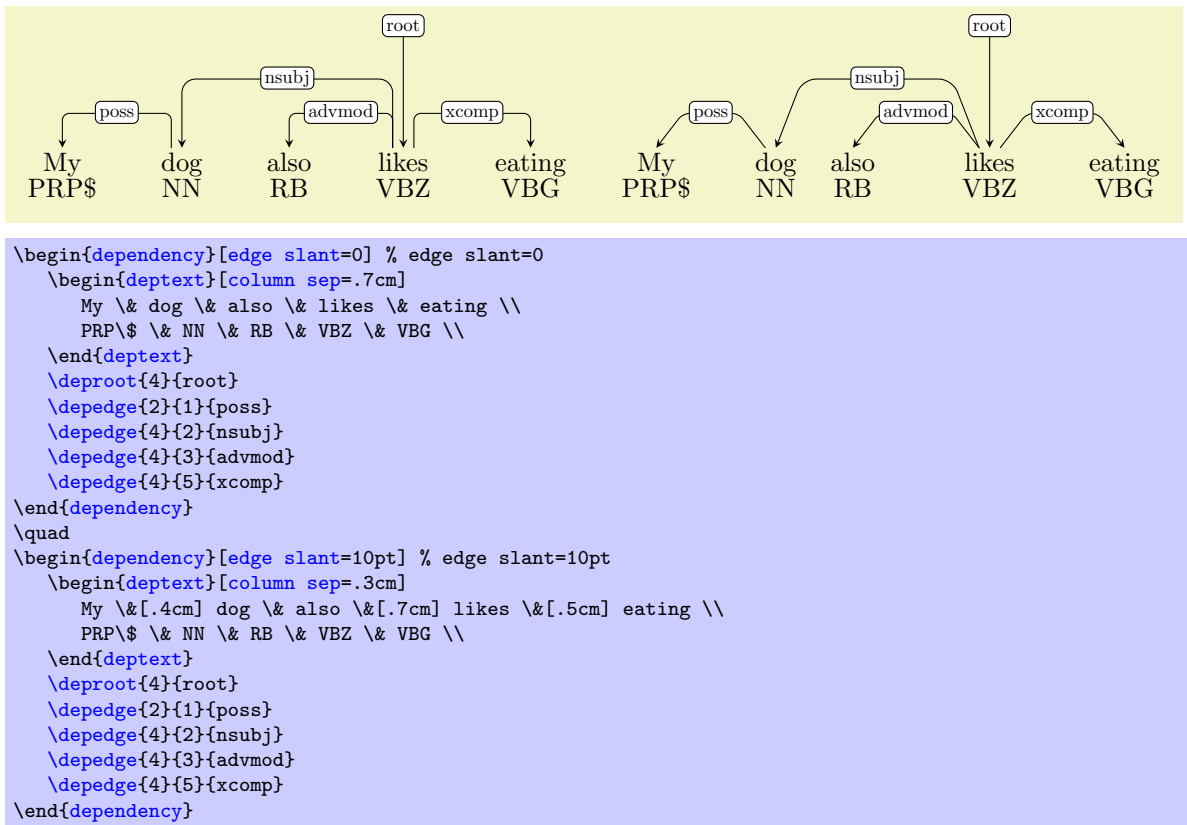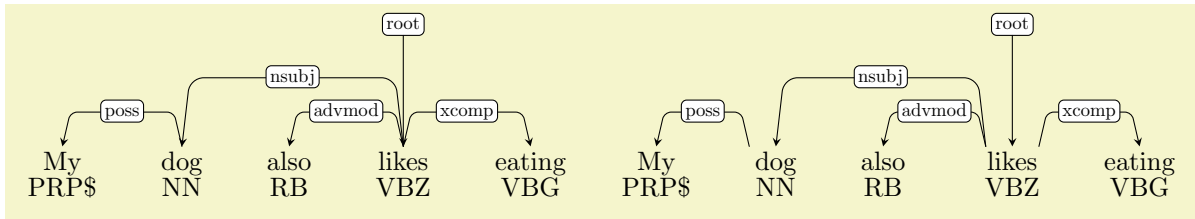
Figure 1: Edge-drawing parameters.


```
\begin{dependency}[edge slant=0] % edge slant=0
    \begin{deptext}[column sep=.7cm]
        My \& dog \& also \& likes \& eating \\
        PRP\$ \& NN \& RB \& VBZ \& VBG \\
    \end{deptext}
    \deproot{4}{root}
    \depedge{2}{1}{poss}
    \depedge{4}{2}{nsubj}
    \depedge{4}{3}{advmod}
    \depedge{4}{5}{xcomp}
\end{dependency}
\quad
\begin{dependency}[edge slant=10pt] % edge slant=10pt
    \begin{deptext}[column sep=.3cm]
        My \&[.4cm] dog \& also \&[.7cm] likes \&[.5cm] eating \\
        PRP\$ \& NN \& RB \& VBZ \& VBG \\
    \end{deptext}
    \deproot{4}{root}
    \depedge{2}{1}{poss}
    \depedge{4}{2}{nsubj}
    \depedge{4}{3}{advmod}
    \depedge{4}{5}{xcomp}
\end{dependency}
```

/depgraph/edge horizontal padding=⟨*length*⟩   (default 4pt)

TikZ-dependency employs a very simple strategy to avoid edges outgoing from and incoming into a word to start/terminate on the same spot, thus making it easier to understand the direction of a dependency. The algorithm is very simple: all incoming links arrive in the middle of a word; all outgoing links heading left leave the node from a point which is a little bit to the left; all those heading right leave from a spot a little bit to the right. /depgraph/edge horizontal padding is the value of this "little bit". By setting it to 0, all the edges will come and go from the same spot in the middle of the word.

By increasing its value, the incoming and outgoing edges on a word will look more apart.
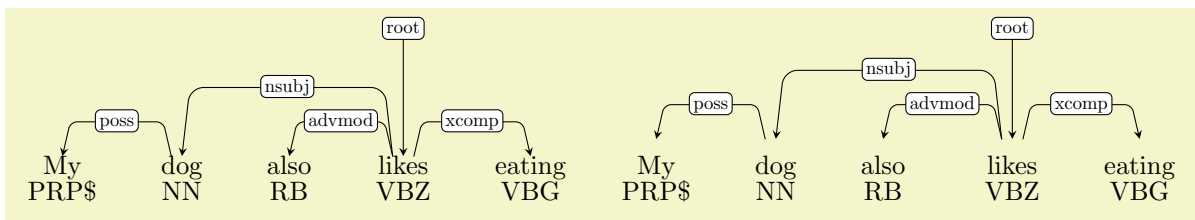


```
\begin{dependency}[edge horizontal padding=0] % edge horizontal padding=0
   \begin{deptext}[column sep=.7cm]
      My \& dog \& also \& likes \& eating \\
      PRP\$ \& NN \& RB \& VBZ \& VBG \\
   \end{deptext}
   \deproot{4}{root}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
\end{dependency}
\quad
\begin{dependency}[edge horizontal padding=10pt] % edge horizontal padding=10pt
   \begin{deptext}[column sep=.7cm]
      My \& dog \& also \&[.2cm] likes \& eating \\
      PRP\$ \& NN \& RB \& VBZ \& VBG \\
   \end{deptext}
   \deproot{4}{root}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
\end{dependency}
```

/depgraph/edge vertical padding=⟨*length*⟩   (default `0ex`)

/depgraph/edge vertical padding is the extra distance between a word and incoming/outgoing edges. By increasing (decreasing) its value, you can send the edges further (bring them closer) to the text.
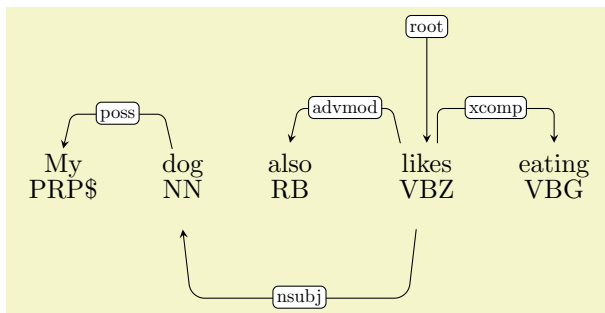
```
\begin{dependency}[edge vertical padding=-.5ex] % edge vertical padding=-.5ex
    \begin{deptext}[column sep=.7cm]
        My \& dog \& also \& likes \& eating \\
        PRP\$ \& NN \& RB \& VBZ \& VBG \\
    \end{deptext}
    \deproot{4}{root}
    \depedge{2}{1}{poss}
    \depedge{4}{2}{nsubj}
    \depedge{4}{3}{advmod}
    \depedge{4}{5}{xcomp}
\end{dependency}
\quad
\begin{dependency}[edge vertical padding=1ex] % edge horizontal padding=1ex
    \begin{deptext}[column sep=.7cm]
        My \& dog \& also \&[.2cm] likes \& eating \\
        PRP\$ \& NN \& RB \& VBZ \& VBG \\
    \end{deptext}
    \deproot{4}{root}
    \depedge{2}{1}{poss}
    \depedge{4}{2}{nsubj}
    \depedge{4}{3}{advmod}
    \depedge{4}{5}{xcomp}
\end{dependency}
```

It may be worth repeating that even though you will generally want to set this properties at the `dependency` level, you can very well set them individually for each edge. And of course, you can combine them with all the properties that we saw before (e.g., `/depgraph/edge below`):
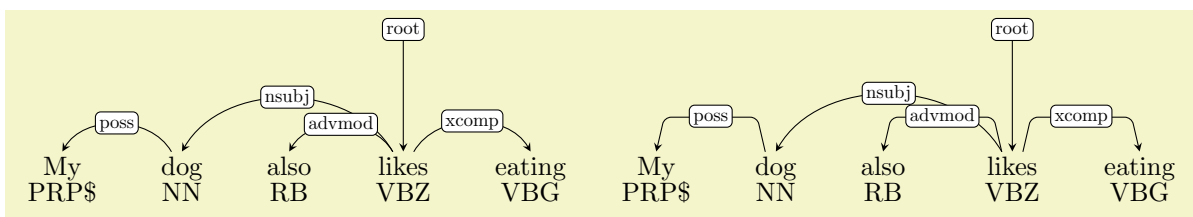


```
\begin{dependency}[edge vertical padding=.5ex]
    \begin{deptext}[column sep=.7cm]
        My \& dog \& also \&[.3cm] likes \& eating \\
        PRP\$ \& NN \& RB \& VBZ \& VBG \\
    \end{deptext}
    \deproot{4}{root}
    \depedge[edge vertical padding=0]{2}{1}{poss}
    \depedge[edge below, edge vertical padding=.3cm]{4}{2}{nsubj}
    \depedge[edge horizontal padding=10pt]{4}{3}{advmod}
    \depedge[edge slant=0]{4}{5}{xcomp}
\end{dependency}
```

`/depgraph/arc edge=⟨boolean⟩`   (default `false`)
`/depgraph/segmented edge=⟨boolean⟩`   (default `true`)

This key allows you to switch between the default, fancier edge shape (`/depgraph/segmented edge`) and the traditional arc-shaped edges used in dependency parsing literature. Can be used either within `dependency` (affect all the edges in the graph) or individual `\depedge` commands. Local overrides are also possible (as shown in the rightmost example).
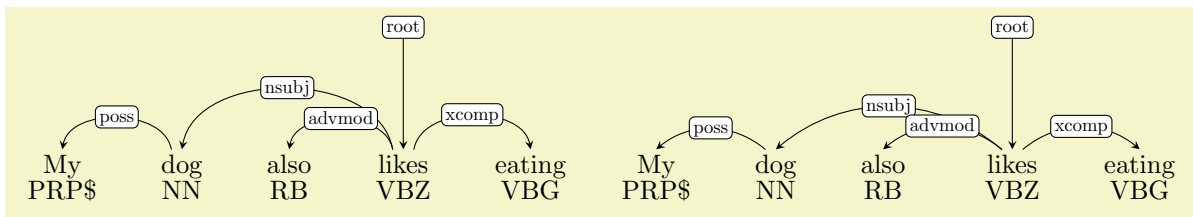

14

```
\begin{dependency}[arc edge]
    \begin{deptext}[column sep=.7cm]
        My \& dog \& also \& likes \& eating \\
        PRP\$ \& NN \& RB \& VBZ \& VBG \\
    \end{deptext}
    \deproot{4}{root}
    \depedge{2}{1}{poss}
    \depedge{4}{2}{nsubj}
    \depedge{4}{3}{advmod}
    \depedge{4}{5}{xcomp}
\end{dependency}
\quad
\begin{dependency}[segmented edge]
    \begin{deptext}[column sep=.7cm]
        My \& dog \& also \&[.2cm] likes \& eating \\
        PRP\$ \& NN \& RB \& VBZ \& VBG \\
    \end{deptext}
    \deproot{4}{root}
    \depedge{2}{1}{poss}
    \depedge[arc edge]{4}{2}{nsubj}
    \depedge{4}{3}{advmod}
    \depedge{4}{5}{xcomp}
\end{dependency}
```

/depgraph/arc angle=⟨*decimal*⟩   (default `60`)

When using **/depgraph/arc edge**, this key controls the angle with which edges enter and exit the nodes.
It has no effect when the edge is drawn using **/depgraph/segmented edge**.
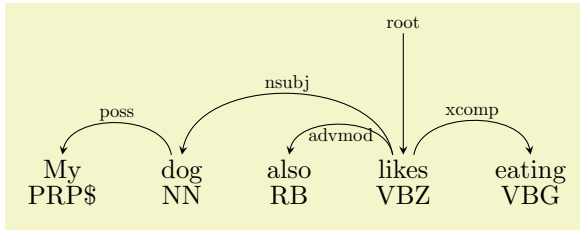


```
\begin{dependency}[arc edge, arc angle=80]
    \begin{deptext}[column sep=.7cm]
        My \& dog \& also \& likes \& eating \\
        PRP\$ \& NN \& RB \& VBZ \& VBG \\
    \end{deptext}
    \deproot{4}{root}
    \depedge{2}{1}{poss}
    \depedge{4}{2}{nsubj}
    \depedge{4}{3}{advmod}
    \depedge{4}{5}{xcomp}
\end{dependency}
\quad
\begin{dependency}[arc edge, arc angle = 40]
    \begin{deptext}[column sep=.7cm]
        My \& dog \& also \&[.2cm] likes \& eating \\
        PRP\$ \& NN \& RB \& VBZ \& VBG \\
    \end{deptext}
    \deproot{4}{root}
    \depedge{2}{1}{poss}
    \depedge{4}{2}{nsubj}
    \depedge{4}{3}{advmod}
    \depedge{4}{5}{xcomp}
\end{dependency}
```

/depgraph/text only label=⟨*boolean*⟩   (default `false`)

At least in my opinion, heavily drawn and filled labels don't look very well with **arc edge**. This flag
allows you to show only the text of the label, without drawing the node around it. To prevent the label
to be drawn just over the edge line, you may want to add either `above` or `below` to the **label style**,
as in this example:

```latex
    \begin{dependency}[arc edge, arc angle=80, text only label, label style={above}]
    \begin{deptext}[column sep=.7cm]
        My \& dog \& also \& likes \& eating \\
        PRP\$ \& NN \& RB \& VBZ \& VBG \\
    \end{deptext}
    \deproot{4}{root}
    \depedge{2}{1}{poss}
    \depedge{4}{2}{nsubj}
    \depedge[label style={below}]{4}{3}{advmod}
    \depedge{4}{5}{xcomp}
\end{dependency}
```
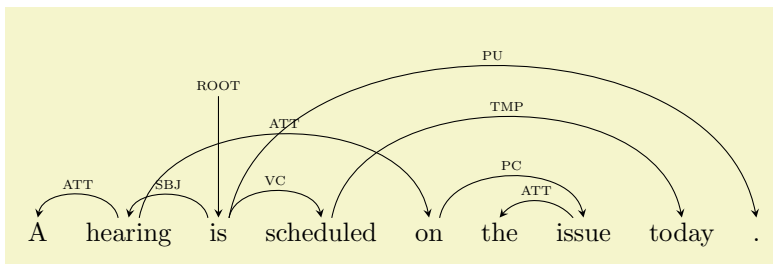
/depgraph/edge start offset x=⟨*length*⟩   (default 0)
/depgraph/edge end offset x=⟨*length*⟩   (default 0)

When using arc edges, some edges may overlap as in the following example:


```latex
\begin{dependency}[theme = simple]
    \begin{deptext}[column sep=1em]
        A \& hearing \& is \& scheduled \& on \& the \& issue \& today \& . \\
    \end{deptext}
    \deproot{3}{ROOT}
    \depedge{2}{1}{ATT}
    \depedge{2}{5}{ATT}
    \depedge{3}{2}{SBJ}
    \depedge{3}{9}{PU}
    \depedge{3}{4}{VC}
    \depedge{4}{8}{TMP}
    \depedge{5}{7}{PC}
    \depedge[arc angle=50]{7}{6}{ATT}
\end{dependency}
```
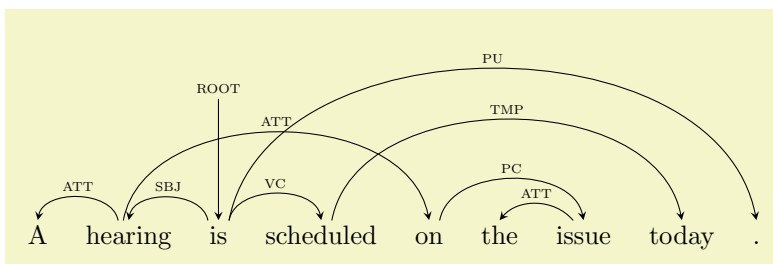
These two keys allow you to shift the start/end point of the edge horizontally by the specified amount, so as to untangle the graph:

16

```
\begin{dependency}[theme = simple]
   \begin{deptext}[column sep=1em]
      A \& hearing \& is \& scheduled \& on \& the \& issue \& today \& . \\
   \end{deptext}
   \deproot{3}{ROOT}
   \depedge{2}{1}{ATT}
   \depedge[edge start x offset=-6pt]{2}{5}{ATT}
   \depedge{3}{2}{SBJ}
   \depedge{3}{9}{PU}
   \depedge{3}{4}{VC}
   \depedge{4}{8}{TMP}
   \depedge{5}{7}{PC}
   \depedge[arc angle=50]{7}{6}{ATT}
\end{dependency}
```
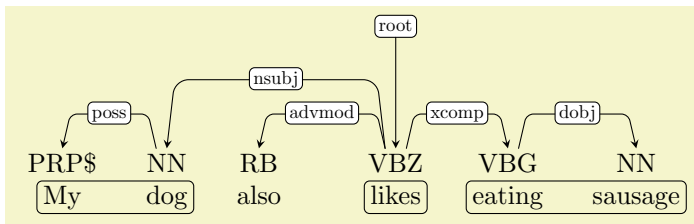
5   Node groups and group linking

Another thing that you may want to do is to annotate relations between groups of nodes. I added this functionality thinking about semantic parsing, so that it would be possible to annotate on the same graph the syntactic and semantic layers of a sentence. You can think of many other ways of exploiting the functionality and use it for your own specific needs. Two new commands can be used to define word groups and to connect them.

\wordgroup[⟨*key-value-pairs*⟩]{⟨*row-offset*⟩}{⟨*word-offset-beg*⟩}{⟨*word-offset-end*⟩}{⟨*gid*⟩}

This command creates a group spanning the words in the {⟨*row-offset*⟩}-th row of the matrix from position {⟨*word-offset-beg*⟩} to position {⟨*word-offset-end*⟩}. {⟨*gid*⟩} is a unique identifier for the group, that will be used to connect groups. From TikZ' point of view, it is just the name of the node enclosing the group, so you can directly use it as you would use any other node.
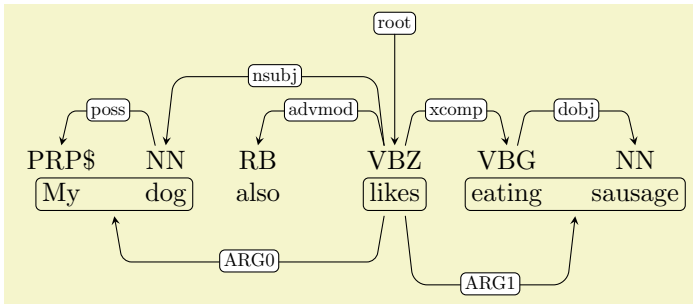


```
\begin{dependency}
   \begin{deptext}[column sep=.5cm, row sep=.1ex]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \deproot{4}{root}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
   %  group on 2nd row, from word 4 to word 4, labeled "pred"
   \wordgroup{2}{4}{4}{pred}
   %  group on 2nd row, from word 1 to word 2, labeled "a0"
   \wordgroup{2}{1}{2}{a0}
   %  group on 2nd row, from word 5 to word 6, labeled "a1"
   \wordgroup{2}{5}{6}{a1}
\end{dependency}
```

\groupedge[⟨*key-value-pairs*⟩]{⟨*from-gid*⟩}{⟨*to-gid*⟩}{⟨*label*⟩}{⟨*height*⟩}

This command draws an edge with label {⟨*label*⟩} from the group identified with {⟨*from-gid*⟩} to the group identified {⟨*to-grid*⟩}. The horizontal segment of the edge will be {⟨*height*⟩} distant from the words. Just as \depedge, also \groupedge honors the global edge positioning set via dependency. Also in this case, you can use the keys /depgraph/edge below and /depgraph/edge above to override the directive locally. More generally, all the key-value pairs that can be passed to \depedge (e.g., edge slant) can also be used with \groupedge. The two macros actually do the same thing, with

17

the difference that `\depedge` automatically calculates a default height for the edge based on the word-distance between its endpoints.


```
\begin{dependency}
\begin{deptext}[column sep=.5cm, row sep=.1ex]
    PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
    My \& dog \& also \& likes \& eating \& sausage \\
\end{deptext}
\deproot{4}{root}
\depedge{2}{1}{poss}
\depedge{4}{2}{nsubj}
\depedge{4}{3}{advmod}
\depedge{4}{5}{xcomp}
\depedge{5}{6}{dobj}
\wordgroup{2}{4}{4}{pred}
\wordgroup{2}{1}{2}{a0}
\wordgroup{2}{5}{6}{a1}
\groupedge[edge below]{pred}{a0}{ARG0}{4ex} %  pred -> a0 (ARG0, 4ex)
\groupedge[edge below]{pred}{a1}{ARG1}{6ex} %  pred -> a1 (ARG1, 6ex)
\end{dependency}
```
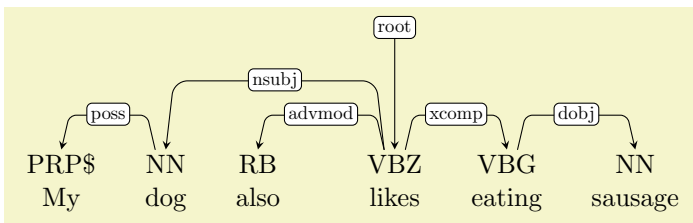
6   Styling text, edges and labels

This section covers the ways in which you can customize the look of the building blocks of the dependency graph: words, groups, edges and edge labels. Styling requires some basic knowledge of TikZ styling facilities. For those who really don't want to read to read the glorious manual of TikZ[1] (you really should, for your own sake), in Appendix A I do my best to try to summarize a minimum of information that should be enough to get you started.

6.1   Using themes

The simplest way to change the look and feel of your dependency graphs is to use a pre-defined *theme*.

/depgraph/theme=⟨*theme name*⟩   (default `default`)

By passing this option to `dependency`, you can completely change the appearence of the graphs. At the moment of writing, the following five themes have been defined: `default`, `simple`, `night`, `brazil`, `grassy`, `iron`, `copper`. They look like this:
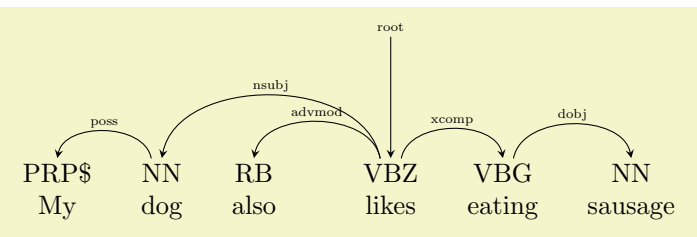

---

[1]http://ftp.gui.uva.es/sites/ctan.org/graphics/pgf/base/doc/generic/pgf/pgfmanual.pdf

```
\begin{dependency}[theme=default]
   \begin{deptext}[column sep=.5cm, row sep=.1ex]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \deproot{4}{root}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
\end{dependency}
```
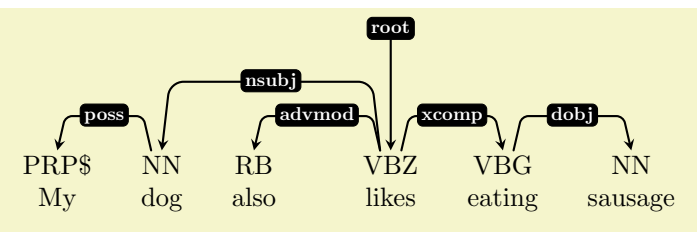


```
\begin{dependency}[theme=simple]
   \begin{deptext}[column sep=.5cm, row sep=.1ex]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \deproot{4}{root}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
\end{dependency}
```



```
\begin{dependency}[theme=night]
   \begin{deptext}[column sep=.5cm, row sep=.1ex]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \deproot{4}{root}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
\end{dependency}
```
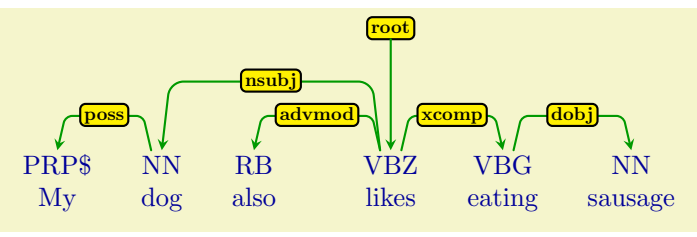
```
\begin{dependency}[theme=brazil]
   \begin{deptext}[column sep=.5cm, row sep=.1ex]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \deproot{4}{root}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
\end{dependency}
```
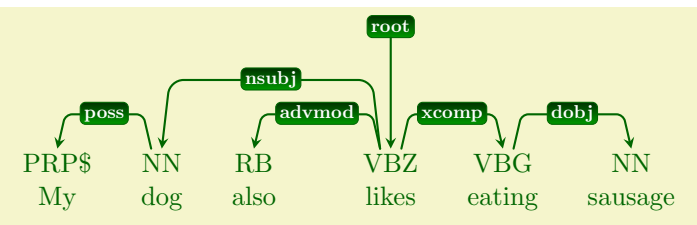


```
\begin{dependency}[theme=grassy]
   \begin{deptext}[column sep=.5cm, row sep=.1ex]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \deproot{4}{root}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
\end{dependency}
```
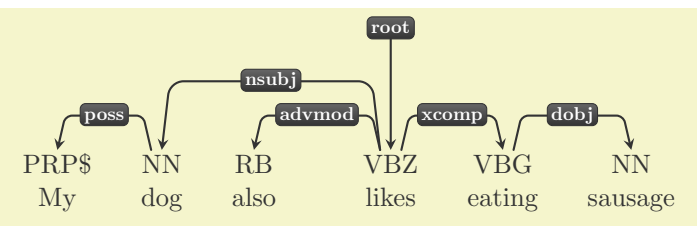


```
\begin{dependency}[theme=iron]
   \begin{deptext}[column sep=.5cm, row sep=.1ex]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \deproot{4}{root}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
\end{dependency}
```

```
\begin{dependency}[theme=copper]
    \begin{deptext}[column sep=.5cm, row sep=.1ex]
        PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
        My \& dog \& also \& likes \& eating \& sausage \\
    \end{deptext}
    \deproot{4}{root}
    \depedge{2}{1}{poss}
    \depedge{4}{2}{nsubj}
    \depedge{4}{3}{advmod}
    \depedge{4}{5}{xcomp}
    \depedge{5}{6}{dobj}
\end{dependency}
```
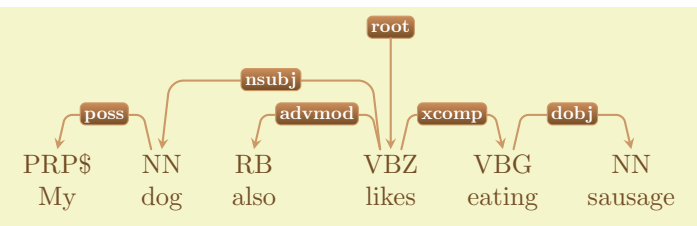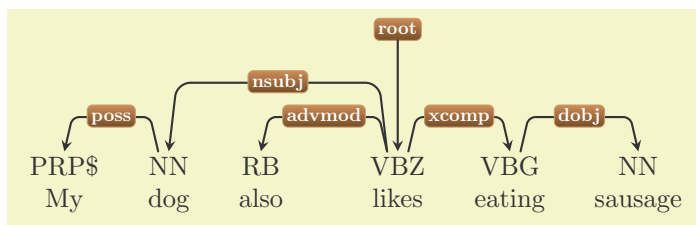
/depgraph/edge theme=⟨*theme name*⟩   (default `default`)
/depgraph/label theme=⟨*theme name*⟩   (default `default`)
/depgraph/text theme=⟨*theme name*⟩   (default `default`)

Each of the aforementioned themes is actually a combination of three themes with the same name: one for edges, one for edge labels and one for the sentence text. These three themes can also be applied independently. So, for example, you could apply the `iron` theme globally and then use `copper` to draw the labels:
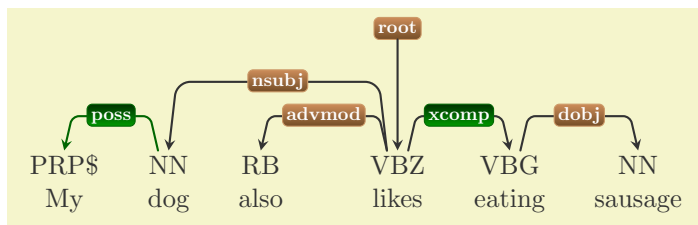
```
\begin{dependency}[theme=iron, label theme=copper]
    \begin{deptext}[column sep=.5cm, row sep=.1ex]
        PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
        My \& dog \& also \& likes \& eating \& sausage \\
    \end{deptext}
    \deproot{4}{root}
    \depedge{2}{1}{poss}
    \depedge{4}{2}{nsubj}
    \depedge{4}{3}{advmod}
    \depedge{4}{5}{xcomp}
    \depedge{5}{6}{dobj}
\end{dependency}
```

Edges and labels can be styled locally, by passing the desired theme, label theme or edge theme to specific edges. In the following example, I will use `iron` as the base theme, globally overriding the label theme with `copper` and using `grassy` as the theme and label theme two special edges:

```
\begin{dependency}[theme=iron, label theme=copper]
   \begin{deptext}[column sep=.5cm, row sep=.1ex]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \deproot{4}{root}
   \depedge[theme=grassy]{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge[label theme=grassy]{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
\end{dependency}
```
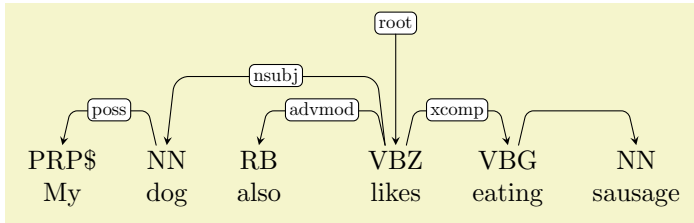
6.2 Styling elements individually

/depgraph/hide label=⟨*boolean*⟩ (default `false`)

/depgraph/show label=⟨*boolean*⟩ (default `true`)

By default, all labels are visible. Add `hide label` to the `[`⟨*key-value-pairs*⟩`]` of `\depedge` or `\groupedge` to prevent the bounding box and the text of the label to be drawn.
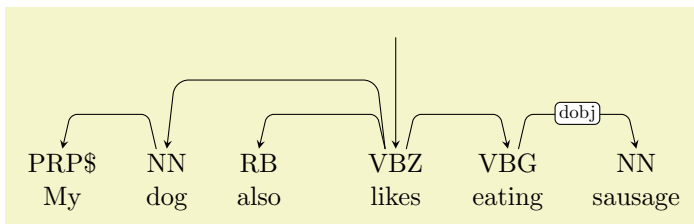


```
\begin{dependency}
   \begin{deptext}[column sep=.5cm, row sep=.1ex]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \deproot{4}{root}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge[hide label]{5}{6}{dobj}
\end{dependency}
```

If added to the options of `dependency`, all the labels will be suppressed. Local overriding is still possible by adding the /depgraph/show label key:


```
\begin{dependency}[hide label]
   \begin{deptext}[column sep=.5cm, row sep=.1ex]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \deproot{4}{root}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge[show label]{5}{6}{dobj}
\end{dependency}
```

Please, note that the labels are still there, only they are transparent.

/depgraph/edge style=⟨*style*⟩ (style, default {})

This style is applied to all the edges drawn by Ti*k*Z-dependency. By setting its value you can alter the appearance of the dependency edges. The default style applied to every edge is:

```
\pgfkeys{%
   /depgraph/reserved/edge style/.style = {%
      ->, >=stealth, % arrow properties
      black, solid, line cap=round, % line properties
      rounded corners=2, % make corners round
   },%
}
```

By setting `edge style` at the `dependency` level, all the edges in the graph are affected:

22

```
\begin{dependency}[edge style={red,densely dotted}]
   \begin{deptext}[column sep=.5cm, row sep=.1ex]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \deproot{4}{root}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
\end{dependency}
```
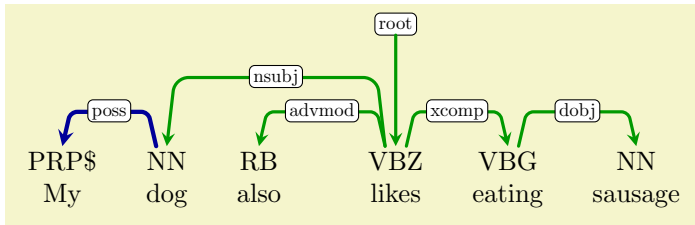
As always, local settings at the edge level take precedence over global settings:


```
\begin{dependency}[edge style={green!60!black,very thick}]
   \begin{deptext}[column sep=.5cm, row sep=.1ex]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \deproot{4}{root}
   \depedge[edge style={blue!60!black,ultra thick}]{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
\end{dependency}
```
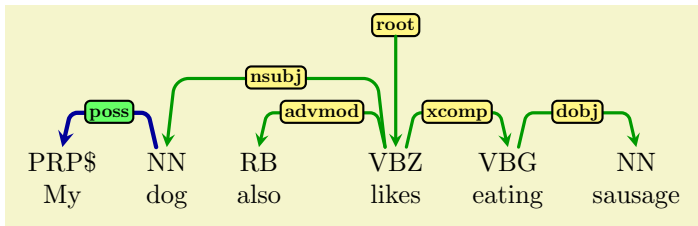
/depgraph/label style=⟨*style*⟩   (style, default {})

The default style applied to every label is the following:

```
\pgfkeys{%
   /depgraph/reserved/label style/.style = {%
      draw, solid, black, % the outline of the label
      scale=.7, % scale down the text, to make it smaller than the sentence text
      text=black, % color of the text
      text height=1.5ex, % needed to center text vertically
      text depth=0.25ex, % needed to center text vertically
      inner sep=.5ex, % padding between the text and the border of the node
      fill=white, % background color
      outer sep=0pt,
      rounded corners=2pt,
      anchor=mid,
   },%
}
```

The look of labels can be changed by defining the `label style` style, either locally to each edge or globally:
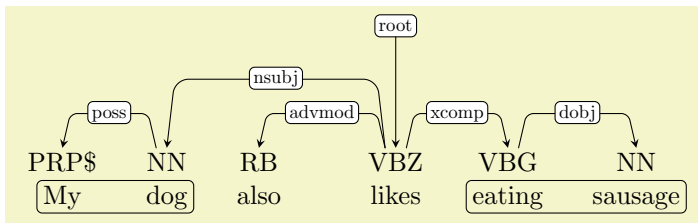
```
\begin{dependency}[edge style={green!60!black,very thick},
   label style={fill=yellow!60,font=\bfseries,thick}]
   \begin{deptext}[column sep=.5cm, row sep=.1ex]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \deproot{4}{root}
   \depedge[edge style={blue!60!black,ultra thick},
      label style={fill=green!60,font=\bfseries,text=black}]{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
\end{dependency}
```

/depgraph/group style=⟨*style*⟩   (style, default {})

By default, when you create a group you will only see a box with rounded corners around the selected words:


```
   \begin{dependency}
   \begin{deptext}[column sep=.5cm, row sep=.1ex]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \deproot{4}{root}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
   \wordgroup{2}{1}{2}{a0}
   \wordgroup{2}{5}{6}{a1}
\end{dependency}
```

The group style property allows you to redefine this style according to your taste and needs:
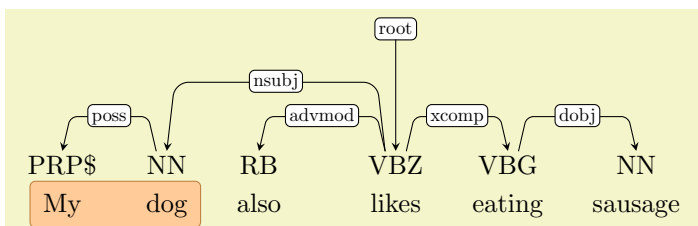

24

```
   \begin{dependency}
   \begin{deptext}[column sep=.5cm, row sep=.5ex]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \deproot{4}{root}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
   \wordgroup[group style={fill=orange!40, draw=brown, inner sep=.6ex}]{2}{1}{2}{a0}
\end{dependency}
```
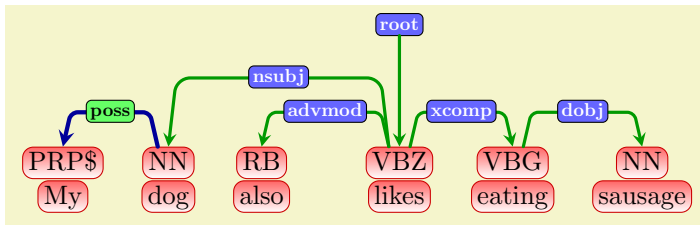
Please, keep in mind that a group is just a box drawn *underneath* the word nodes. They have no text themselves, as the text that they surround is actually the text of some word node. Therefore, you cannot use `group style` to change in any way the rendering of the words they group.

/tikz/nodes=⟨*style*⟩   (style, default {})

This is an TikZ style key used to render the nodes within a given scope. By using it within a `deptext`, you can assign a style to all the nodes in the matrix, i.e. the words in the sentence:
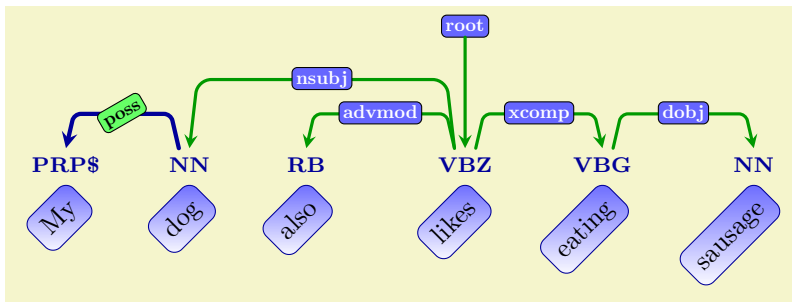


```
\begin{dependency}[edge style={green!60!black,very thick},
      label style={fill=blue!60,font=\bfseries,text=white}]
   \begin{deptext}[column sep=.5cm, row sep=.1ex,
         nodes={draw=red!80!black, shade, top color=red!60, rounded corners}]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \deproot{4}{root}
   \depedge[edge style={blue!60!black,ultra thick},
      label style={fill=green!60,font=\bfseries,text=black}]{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
\end{dependency}
```

The style can be overridden on a per-node basis, by using a special syntax understood by matrix that allows to set style properties of each node individually. To to so, you need to add `|[<style>]|` before every word that you want to restyle, where ⟨*style*⟩ is a style definition. The following example shows how you would apply a different style to all the nodes in the second row:

```
\begin{dependency}[edge style={green!60!black,very thick},
    label style={fill=blue!60,font=\bfseries,text=white}]
    \tikzstyle{POS}=[font=\footnotesize\bfseries,text=blue!60!black]
    \tikzstyle{word}=[draw=blue!60!black, shade, text=black,
                        rotate=45, anchor=north east, inner sep=1ex,
                        font=\normalsize, top color=blue!60, rounded corners]
    \begin{deptext}[column sep=.5cm, row sep=0, nodes={POS}]
        PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
        |[word]| My \& |[word]| dog \& |[word]| also \&
        |[word]| likes \& |[word]| eating \& |[word]| sausage \\
    \end{deptext}
    \deproot{4}{root}
    \depedge[edge style={blue!60!black,ultra thick},
        label style={fill=green!60,font=\bfseries,text=black, rotate=10}]{2}{1}{poss}
    \depedge{4}{2}{nsubj}
    \depedge{4}{3}{advmod}
    \depedge{4}{5}{xcomp}
    \depedge{5}{6}{dobj}
\end{dependency}
```
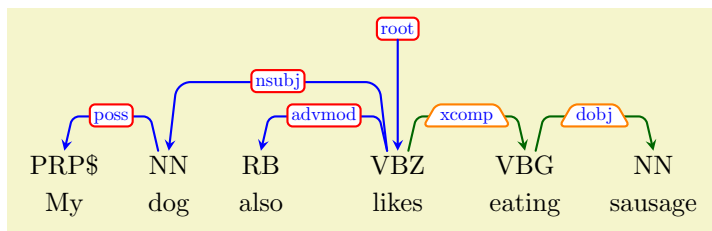
When using this syntax, you should always consider that the `nodes` style is in any case applied to *every node* in the `matrix`, then the style specified for each node (if any) is appended. This fact has the subtle implication that all the nodes inherit all the properties set via `nodes`, and per-node styles must explicitly overwrite all the properties that they don't need to use. As a rule of thumb, you should use `nodes` to set properties that all the nodes will share, and then use per-node options to add or override options.

6.3   Defining your own styles

`\depstyle`{⟨*style-name*⟩}{⟨*style-definition*⟩}

Defining your own styles is a convenient way of using a consistent look for all your graphs across a document, at the same time avoiding a lot of unnecessary typing. `\depstyle` is a wrapper around `\tikzset` (see Section A) with the advantage that you do not have to prepend `/depgraph` to all the styling keys that are defined by TikZ-dependency. `\depstyle` automatically forwards to TikZ all the keys that it does not how to handle.

In the following example I will define and use two styles. The first style results in thick blue edges, labels with a thick red outline, blue text and white background; the second style also uses a predefined themes.


```
\depstyle{red and blue}{edge style = {thick, blue},
    label style = {thick, draw=red, text=blue, fill=white}}
\depstyle{collage}{edge theme = grassy, label style={draw=orange,trapezium}}
\begin{dependency}[red and blue]
    \begin{deptext}[column sep=.5cm, row sep=.5ex]
        PRP\$ \& NN \& RB \&[.5cm] VBZ \&[.2cm] VBG \& NN \\
        My \& dog \& also \& likes \& eating \& sausage \\
    \end{deptext}
    \deproot{4}{root}
    \depedge{2}{1}{poss}
    \depedge{4}{2}{nsubj}
    \depedge{4}{3}{advmod}
    \depedge[collage]{4}{5}{xcomp}
    \depedge[collage]{5}{6}{dobj}
\end{dependency}
```

Note how the labels in the style `collage` inherit the background and the text color of `red and blue` labels. This happens because the `red and blue` style is applied at the `dependency` level, meaning that it is applied to all the labels. Then, the `collage` style is applied to the labels of the last two edges.
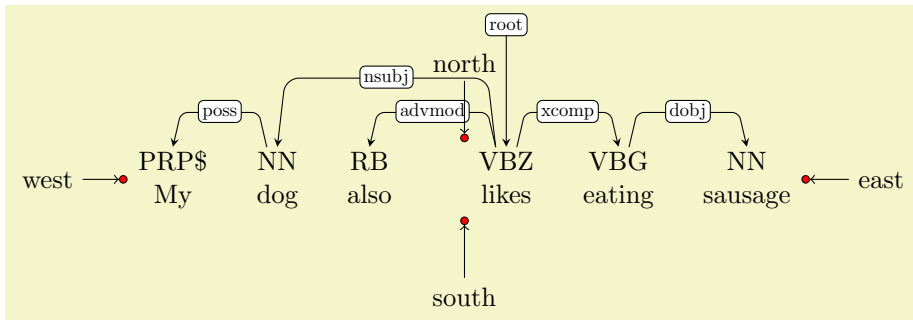
Since this style only overrides the color of the outline (`draw`), all the other properties are inherited from `red and blue`.

7 Interaction with Ti*k*Z

Some function make it possible to reference the nodes in the graph so that it is easier to extend the graph with Ti*k*Z drawing capabilities. Understanding what is going on here requires at least some basic knowledge of how to draw nodes and paths in Ti*k*Z.
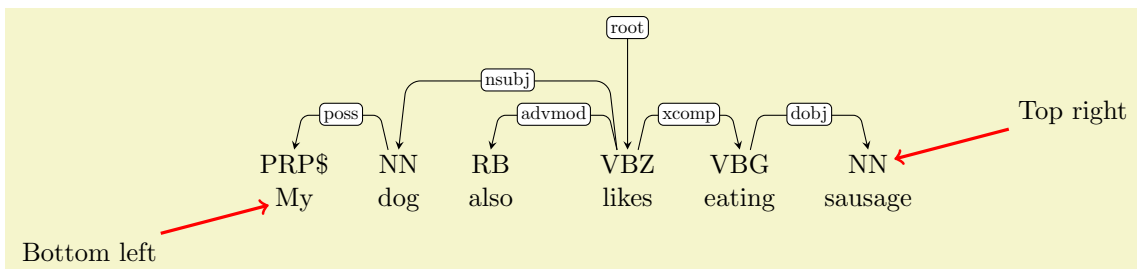
`\matrixref`

Returns a reference to the `\matrix` used to lay out the sentence. You can use it to position other elements in the picture relative to it.


```
\begin{dependency}
   \begin{deptext}[column sep=.5cm, row sep=.1ex]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \deproot{4}{root}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
   \foreach \a/\pos in {south/below of,north/above of,east/right of,west/left of} {
      \node (tmp) at (\matrixref.\a) [draw, circle, fill=red, inner sep=1pt] {};
      \node (tmp2) [\pos = tmp] {\a} -- (tmp);
      \draw [->] (tmp2) -- (tmp);
   };
\end{dependency}
```

`\wordref{⟨row-offset⟩}{⟨word-offset⟩}`

Returns the reference to the matrix node in row {⟨*row-offset*⟩} and column {⟨*word-offset*⟩}.

```
\begin{dependency}
    \begin{deptext}[column sep=.5cm, row sep=.1ex]
        PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
        My \& dog \& also \& likes \& eating \& sausage \\
    \end{deptext}
    \deproot{4}{root}
    \depedge{2}{1}{poss}
    \depedge{4}{2}{nsubj}
    \depedge{4}{3}{advmod}
    \depedge{4}{5}{xcomp}
    \depedge{5}{6}{dobj}
    \node (silly1) [above right of = \wordref{1}{6}, xshift = 2cm] {Top right};
    \node (silly2) [below left of = \wordref{2}{1}, xshift = -2cm] {Bottom left};
    \draw [->, very thick, red] (silly1) -- (\wordref{1}{6});
    \draw [->, very thick, red] (silly2) -- (\wordref{2}{1});
\end{dependency}
```

\rootref

> Returns a reference to the node holding the root label.


```
\begin{dependency}
    \begin{deptext}[column sep=.5cm, row sep=.1ex]
        PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
        My \& dog \& also \& likes \& eating \& sausage \\
    \end{deptext}
    \deproot{4}{root}
    \depedge{2}{1}{poss}
    \depedge{4}{2}{nsubj}
    \depedge{4}{3}{advmod}
    \depedge{4}{5}{xcomp}
    \depedge{5}{6}{dobj}
    \node (silly) [above right of = \rootref, xshift = 3cm] {This is the root node!};
    \draw [->, very thick, red] (silly) -- (\rootref);
    \draw [thick, blue] (\rootref)--(\wordref{1}{2})--(\wordref{2}{6})--(\rootref);
\end{dependency}
```
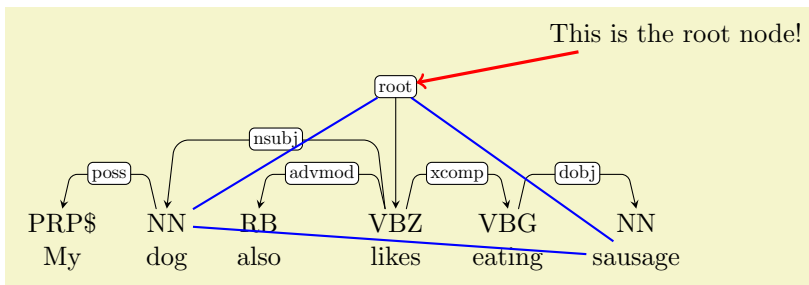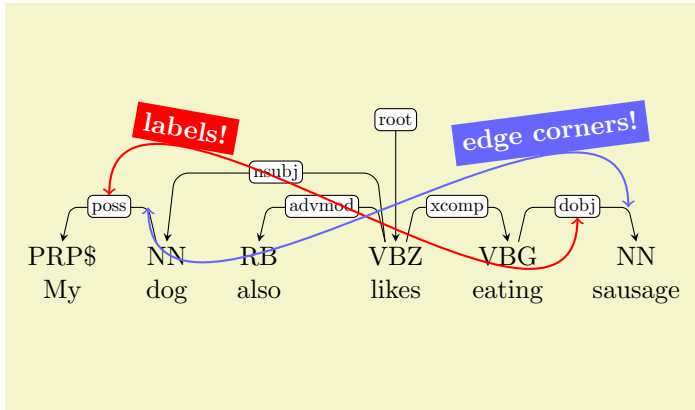
\storelabelnode{⟨*macro*⟩}
\storefirstcorner{⟨*macro*⟩}
\storesecondcorner{⟨*macro*⟩}

> After adding a new edge with \depedge or \groupedge, three commands allow you to store the name of the nodes used to draw the label end the corners of the dependency edge. \storelabelnode can be used to store the name of the label in the macro ⟨*macro*⟩. You can then use ⟨*macro*⟩ just as you would use any node, to draw something involving the label node. \storefirstcorner and \storesecondcorner work exactly in the same way, but they hold they point to the nodes holding the coordinates of the edge corners.

```
\begin{dependency}
   \begin{deptext}[column sep=.5cm, row sep=.1ex]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \& VBG \& NN \\
      My \& dog \& also \& likes \& eating \& sausage \\
   \end{deptext}
   \deproot{4}{root}
   \depedge{2}{1}{poss}
   \storelabelnode\firstlab
   \storefirstcorner\firstcorner
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
   \depedge{4}{5}{xcomp}
   \depedge{5}{6}{dobj}
   \storelabelnode\secondlab
   \storesecondcorner\secondcorner
   \draw [<->,out=90,in=-90, thick, red] (\firstlab) to
      node [fill=red, text=white, font=\bfseries, above, near start, sloped]
      {labels!} (\secondlab);
   \draw [<->, out=-90,in=90, thick, blue!60] (\firstcorner) to
      node [fill=blue!60, text=white, font=\bfseries, above, near end, sloped]
      {edge corners!} (\secondcorner);
\end{dependency}
```
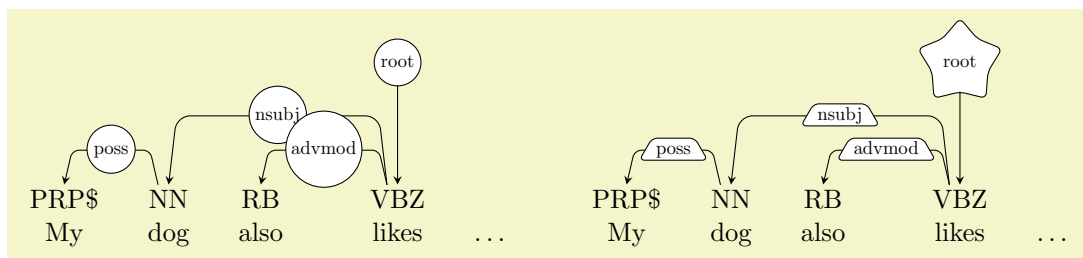
8 Answers to frequent or interesting questions

This section shows practical examples of how to achieve specific results with Ti*k*Z-dependency that may not seem that obvious.

8.1 How do I change the shape of label nodes?

By default, Ti*k*Z-dependency uses Ti*k*Z's rounded corners shape to draw the labels, but you can use every shape that you want. By default, Ti*k*Z defines the rectangle, rounded corners and circle shapes. More shapes are available via the shapes library, which Ti*k*Z-dependency already loads for you. As always, for more information you can refer to Ti*k*Z manual.

To change the shape of labels, you simply add the shape name to, edge style, as in the following examples:

```
\begin{dependency}[label style = {circle}]
   \begin{deptext}[column sep=.5cm, row sep=.1ex]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \\
      My \& dog \& also \& likes \& \dots \\
   \end{deptext}
   \deproot{4}{root}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
\end{dependency}
\qquad
\begin{dependency}[label style = {trapezium}]
   \begin{deptext}[column sep=.5cm, row sep=.1ex]
      PRP\$ \& NN \& RB \&[.5cm] VBZ \\
      My \& dog \& also \& likes \& \dots \\
   \end{deptext}
   \deproot[label style = {star}]{4}{root}
   \depedge{2}{1}{poss}
   \depedge{4}{2}{nsubj}
   \depedge{4}{3}{advmod}
\end{dependency}
```
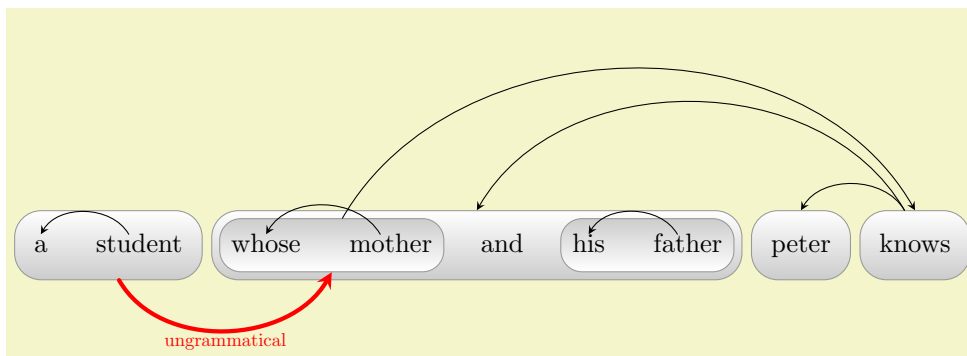
8.2   How do I draw bubble parses?

Until a user of TikZ-dependency asked me how to draw them, I was not even aware that these things existed. Anyway, it turns out that is quite simple to draw bubble parses with TikZ-dependency, and that \wordgroup is basically all you need.

Until I or someone else writes some facilities to explicitly support these beauties, you will have to take care of the details explicitly. What you have to do is:

- use \wordgroup to create bubbles;

- use inner sep and minimum height with group style to create bubbles of the desired size;

- increase the values of inner sep and minimum height for more external bubbles;

- use \groupedge to create edges between bubbles.

The follow example should give you a good starting point:

```
\depstyle{inner bubble}{draw=gray!80, minimum height=20pt, rounded corners=8pt,
   inner sep=2pt, top color=gray!40, bottom color=white}
\depstyle{outer bubble}{draw=gray!80, minimum height=26pt, rounded corners=10pt,
   inner sep=5pt, top color=white, bottom color=gray!40}
\depstyle{ungrammatical}{edge below, edge style = {ultra thick, red},
   label style = {below, text=red}, text only label}
\begin{dependency}[hide label, arc edge,
      edge vertical padding=-2pt,
      group style=outer bubble]
   \begin{deptext}[column sep=1.4em]
      a \& student \& whose \& mother \& and \& his \& father \& peter \& knows \\
   \end{deptext}
   \wordgroup{1}{3}{7}{chubby}
   \wordgroup{1}{1}{2}{first} \depedge{2}{1}{}
   \wordgroup[inner bubble]{1}{3}{4}{second} \depedge{4}{3}{}
   \wordgroup[inner bubble]{1}{6}{7}{third} \depedge{7}{6}{}
   \wordgroup{1}{8}{8}{fourth}
   \wordgroup{1}{9}{9}{fifth}
   \groupedge{fifth}{fourth}{}{0}
   \groupedge[ungrammatical]{first}{second}{ungrammatical}{.5cm}
   \groupedge[edge above]{second}{fifth}{}{1.5cm}
   \groupedge[edge above]{fifth}{chubby}{}{1cm}
\end{dependency}%
```

When using `fill` or `shade` with `\wordgroup`, as in the example, be aware of the fact that the nodes are drawn in the order that you write them down. For this reason, larger bubbles should always be drawn first, and the contained bubbles afterwards. Otherwise, the inner bubbles will be covered by the outer one.

A Ti*k*Z crash course

Disclaimer. *I am making a lot of rough simplifications here to make it possible to explain everything in as little space as possible. Really invite to read the full version of the manual, if you haven't already. It is the only way to really appreciate the elegance and beauty of this Ti*k*Z and its incredible potential.*

PGF/Ti*k*Z is a set of lower/higher level macros that allow you to draw extremely refined and sophisticated graphics directly in LaTeX. If you don't know anything about it, then you are really missing on something, and you should run and check the manual[2]. I am repeating myself, but you really should.

A.1 `\pgfkeys` and `\tikzset`

Ti*k*Z is basically built on top of `pgfkeys`, which is a very flexible and sophisticated system to store key-value pairs. Ti*k*Z (and derived packages), such as Ti*k*Z-`dependency`, use this system for setting local or global variables, and to pass optional arguments to commands (macros) that can take a large number of configuration option.

The definition of virtually all Ti*k*Z macros follows the template:

`\somecommand` [⟨*key-value-pairs*⟩]{⟨*first*⟩}...{⟨*last*⟩}

The first argument, optional, is a set of key-value pairs that can be used to set a bunch of options. It is thus possible to reduce to a minimum the number of mandatory arguments {⟨*first*⟩}...{⟨*last*⟩}, and keep macro's interface lighter and simpler to use. All commands, first parse the key-value pairs, then do what they have to do based on the updated state of the variables.

Key names and values can contain spaces, values can be assigned via the = sign and commas can be used to separate assignments. Braces can be used for grouping values. So, for example, the assignment `variable one = \{a, b\}, vartwo = c` would result in `a, b` being assigned to `variable one` and `c` to `vartwo`.

The function which is responsible for parsing key-value pairs is `\pgfkeys`:

---

[2]http://ftp.gui.uva.es/sites/ctan.org/graphics/pgf/base/doc/generic/pgf/pgfmanual.pdf

`\pgfkeys{⟨keys⟩}`

> The `\pgfkeys` command parses a string containing a sequence of keys and stores their value, e.g.:
>
> ```
> \pgfkeys{a=1, b=2, c={d=3,e=4}}
> ```
>
> In this case, the value of the key `a` is set to `a` and `b` to 2. `c` contains two keys, `d` and `e`, whose values are 3 and 4, respectively. How this values can be accessed is of no interest for the scope of this document. We only want to be able to set those keys. Please, note that you can only set the values for keys that already exist. How a key can be created is also of no interest for us, as we only want to be able to change the values of existing keys.

Keys can be arranged hierarchically, as in a directory structure. This property allows every sub-package to define its own protected space of keys, making it less likely to cause interaction problems. All Ti*k*Z keys are stored under the root key `/tikz/`. So, for example, `/tikz/fill`=blue assigns the value `blue` to the `fill` key within the namespace `/tikz/`. Ti*k*Z-dependency defines its own namespace, which is `/depgraph/`, and defines its own subset of keys that interact with Ti*k*Z' own.

As a convenience function, Ti*k*Z also defines the following macro:

`\tikzset{⟨keys⟩}`

> `\tikzset` is just a wrapper around `\pgfkeys`, that prefixes all keys with `/tikz/`. The two following instructions are (for us) equivalent:
>
> ```
> \pgfkeys{/tikz/text=red, /tikz/fill=white}
> \tikzset{text=red, fill=white}
> ```

Some keys only have a name and no value. Without going into details, it won't hurt to know that such keys are either *binary* flags or one of a set of alternative options (*choice* keys), so that activating one automatically deactivates the other alternatives in the set.

Conveniently, `\pgfkeys` also define an inheritance mechanism that allows sub-packages to consume the keys that they can understand, and then pass on to Ti*k*Z all the keys that haven't been handled. This fact also makes it possible for a user to specify a key without using its fully qualified name. When introducing a key for the first time, I will always use its fully qualified name, just to make it clear where it belongs. In the examples, though, I will generally prefer to use a more compact notation and only use the "base name" of the keys.

A.2   Basic styling notions

A style is a set of key-value pairs (we have seen many already in the previous pages) that define the properties of some graphical object. As an example, `{draw=red, thick}` is a style (we will soon come to the meaning of these properties).

The simplest way to define a new style is with the `\tikzstyle` command:

`\tikzstyle{⟨name⟩}=[⟨key-value-pairs⟩]`

> Here, `{⟨name⟩}` is just an identifier for your style (such as `my style`, or `very cool`, whatever). `[⟨key-value-pairs⟩]` is a sequence of comma separated key-value pairs, e.g.:
>
> ```
> \tikzstyle{my cool style} = [draw=black, fill=red]
> ```

Once a style is defined, you can "add" things to the style in at least two ways:

- By using the same syntax as before, but adding a `+` between `{⟨name⟩}` and the `=` sign, e.g.,:

  ```
  %  before: my cool style = {draw=black, fill=red}
  \tikzstyle{my cool style} += [thick]
  %  after: my cool style = {draw=black, fill=red, thick}
  ```

- By setting (via `\tikzset`) the special sub-property `.append style` of an already existent style, e.g.:

```
%  before: my cool style = {draw=black, fill=red, thick}
\tikzset{my cool style/.append style = {dotted}}
%  after: my cool style = {draw=black, fill=red, thick, dotted}
```

A dependency graph is built using two TikZ primitives: *nodes* and *paths*. Dependency edges are paths. Edge labels and words in the sentence are nodes containing text. Word groups are also nodes, which are drawn on the layer *underneath* the word nodes. They have no text themselves, as the text that they surround is actually the text of some word node.

Nodes have an *outline* (i.e., a contour) and an *area* (the space inside the contour), and (generally) contain some text. All these properties can be styled independently, and the styling can be done either globally (in a way that affects all the graphics within a given scope) or locally (i.e., for each node individually). The same applies to paths, even though in this case you only have the outline to play with.

Let's briefly dive through the most relevant key-value pairs that can be used to configure these properties. My purpose here is just to provide a compact reference to understand the examples in the documentation.

A.2.1 Outline properties

These describe the properties of lines. They apply to edges and to the contour of nodes (such as labels and words in the sentence)

`/tikz/draw`   (no value)

    This key tells tikz to actually draw the outline of a shape. If draw is not set, then the outline is not drawn. By default, outlines are not drawn.

`/tikz/circle`   (no value)
`/tikz/ellipse`   (no value)
`/tikz/rectangle`   (no value)

    Selecting one of this properties allows you to use a different shape to draw a node. When `draw` is selected, by default the node will be drawn as a `rectangle`.

`/tikz/rounded corners`=⟨*length*⟩   (no default)

    This property allows you to make corners round. The value that you assign to it is the amount of roundness.

`/tikz/very thin`   (no value)
`/tikz/thin`   (no value)
`/tikz/thick`   (no value)
`/tikz/very thick`   (no value)

    Can be used to set the width of a line. `thin` is the default value (corresponding to `0.4pt`).

`/tikz/color`=⟨*color-spec*⟩   (no default)

    Sets the color used to draw the line. ⟨*color-spec*⟩ can be a color name, such as `red`, `green` or `blue`, or a more complicated expression, such as `red!60!black`, which means "make a new color made of red for 60% of red, of black for the rest". Colors can be specified in many other ways, which we won't cover here.

`/tikz/solid`   (no value)
`/tikz/dotted`   (no value)
`/tikz/loosely dotted`   (no value)
`/tikz/densely dotted`   (no value)
`/tikz/dashed`   (no value)
`/tikz/loosely dashed`   (no value)
`/tikz/densely dashed`   (no value)

    Set the kind of line to be used to draw the outline. `solid` is the default (i.e. a continuous line).

A.2.2   Area properties

These describe the properties of areas. They apply to the background of nodes (such as labels, words in the sentence and word groups).

`/tikz/fill=`⟨*color*⟩   (no default)

    This key tells tikz to fill the inside of a shape (or a node) with a solid color. Color specification is the same as for `/tikz/draw`.

`/tikz/shade`   (no value)

    Instead of filling the area with a solid color, fill it with a gradient.

`/tikz/top color=`⟨*color*⟩   (no default)
`/tikz/bottom color=`⟨*color*⟩   (no default)

    If `shade` is set, you can use these two keys to set up a vertical gradient.

`/tikz/left color=`⟨*color*⟩   (no default)
`/tikz/right color=`⟨*color*⟩   (no default)

    If `shade` is set, you can use these two keys to set up a horizontal gradient.

`/tikz/inner sep=`⟨*length*⟩   (no default)

    This parameter controls the distance between the outline of a node and its contents (e.g., the text inside the node).

`/tikz/minimum height=`⟨*length*⟩   (no default)
`/tikz/minimum width=`⟨*length*⟩   (no default)

    By default, the size of a node is determined based on its content, i.e., it is just as big as to fit what's inside (plus padding and spacing, e.g., `inner sep`). By setting these keys, you can force a node to be at least as tall and/or wide as requested.

A.2.3   Text properties

These describe the properties of text boxes. They apply to the text of the sentence (the rows if the `matrix` wrapped by `deptext`) and to labels.

`/tikz/text=`⟨*color*⟩   (no default)

    This key sets the color to be used to render the text.

`/tikz/font=`⟨*fontmacros*⟩   (no default)

    Set the font size/family/series/shape to be user to render the text. Every combination of these standard LaTeX macros is a valid value for ⟨*fontmacros*⟩: `\normalsize`, `\small`, `\footnotesize`, `\tiny`, `\large`, `\huge`, `\rmfamily`, `\ttfamily`, `\sffamily`, `\mdseries`, `\bfseries`, `\upshape`, `\itshape`, `\scshape`, `\slshape`.

Index