Connections: Arrows in bracketed syntactic structures

Radu Craioveanu

July 2015

This is a tiny package (more of a small set of custom commands) that draws connections between words written in a single line. It's built on TiKZ, and works by defining each thing you want to connect with lines as a node (which I've called a *target* in the package). The names of those nodes are remembered across the document. Note that because of the way TiKZ saves these nodes, you'll often need to compile *twice* for things to look right; so if the line placement looks totally crazy, just hit compile again and it should be fixed.

Targets are coded with the `\target[label]{text}` command. The argument in curly braces is the text that will be printed, while the optional argument defines an explicit node name. If the optional argument is absent, the package will attempt to use the node text as the label, which will be fine for most cases. The optional argument is useful if you have multiple nodes with the same text, or the node text is too long or contains characters that are illegal in node labels. In the examples below, simple nodes like T are defined with `\target{T}` (implicitly labelled as `(T)`), while *v*P is defined as `\target[v]{\textit{v}}` (explicitly labelled as `(v)`).

The `\connect[options]{node1}{node2}` command makes connections between targets. Optional arguments may be given in square brackets, to change the style of the line, add an arrowhead, &c. Any standard TiKZ options should work. I've coded an extra option, `over`, to quickly define a line that needs to pass over (okay, technically under) other lines.

```
\ex. [\subsc{CP} [\subsc{TP} T [\subsc{VoiceP} \target{Voice}
[\subsc{\textit{v}P} \target[v]{\textit{v}} [\subsc{KP} \target{K}
[\subsc{DP} \target{D} [\subsc{\textit{n}P} \textit{n} ]]]]]]]

\connect{v}{K}
\connect[over,densely dashed]{Voice}{D}
```

(1) [$_{\text{CP}}$ [$_{\text{TP}}$ T [$_{\text{VoiceP}}$ Voice [$_{v\text{P}}$ *v* [$_{\text{KP}}$ K [$_{\text{DP}}$ D [$_{n\text{P}}$ *n*]]]]]]]

If you need to connect multiple lines to the same target, you can mark that node separately as a `\multitarget{node}`, and do so after drawing the connections. (You may also give this the `over` style if you wish, but note that it will currently only work if the lines on *both* sides of the multitarget are raised. This seems like a fringe case, so I'm not implementing it more robustly now, but I can think about it if needed.)

```
\connect[<-]{T}{v}
\connect{v}{K}
\multitarget{v}
```

(2) [$_{\text{CP}}$ [$_{\text{TP}}$ T [$_{\text{VoiceP}}$ Voice [$_{v\text{P}}$ v [$_{\text{KP}}$ K [$_{\text{DP}}$ D [$_{n\text{P}}$ n]]]]]]]

You can define anything you like as the target, so it can be a morpheme or even just a letter. The package defines a little `tikzpicture` for each target and saves its position, and then draws lines between them as a separate operation, so defining smaller parts of the words as targets can help in getting the line looking the way you want it to.

```
\ex. nee=dii=n gya'a-\target{t}=t \target{Aidan}
\connect[->]{t}{Aidan}
```

(3) nee=dii=n gya'a-t=t Aidan

As long as you give `connect` commands after the diagram that sets up the targets, you shouldn't have problems reusing node labels between examples. The old node labels are just overwritten. Thus, it isn't necessary to create note names like `t1`, `t2`, `t3`, etc. In the examples below, I didn't define any custom names, and the second connect command only pays attention to the nodes most recently named `(t)` and `(Aidan)`. I've also defined a custom line depth for demonstration purposes.

```
\ex. nee=dii=n gya'a-\target{t}=t \target{Aidan}
\connect[->,depth=2]{t}{Aidan}
```

(4) nee=dii=n gya'a-t=t Aidan