

L^AT_EX for linguists

Sylvia Blaho & Patrik Bye

The logo for the Center for Advanced Study in Theoretical Linguistics (CASTL) features the letters C, A, S, T, and L in a white, serif font, spaced out across a dark green rectangular background.

CASTL

Center for Advanced Study
in Theoretical Linguistics

University of Tromsø

Spring 2005

Aims for today

Aims for today

1. Phonetic symbols using TIPA.

Aims for today

1. Phonetic symbols using TIPA.
2. Drawing trees with `qtree`.

1. Typesetting IPA symbols: The TIPA Package

1. Typesetting IPA symbols: The TIPA Package

- TIPA (=Tokyo IPA, by Fukui Rei) has all of the symbols in the IPA, including a number which are obsolete after the 1989 and 1996 revisions. You will also find symbols for typesetting Indo-European and Germanic linguistics as well as American transcription symbols.

1. Typesetting IPA symbols: The TIPA Package

- TIPA (=Tokyo IPA, by Fukui Rei) has all of the symbols in the IPA, including a number which are obsolete after the 1989 and 1996 revisions. You will also find symbols for typesetting Indo-European and Germanic linguistics as well as American transcription symbols.
- Load it by including the following line in the preamble:

```
\usepackage{tipa}
```

- To change to TIPA within your source file, type

```
\textipa{ }
```

hɪəz ən ɪg'zɑ:mpɫ əv ə
fən'etɪk ˌtræn'skrɪpʃn

```
\textipa{hI@z @n Ig"zA:mp\s{1} @v @  
f@n"EtIk ""tr\ae n"skrIpS\s{n}}
```

1. Typesetting IPA symbols: The TIPA Package

- TIPA (=Tokyo IPA, by Fukui Rei) has all of the symbols in the IPA, including a number which are obsolete after the 1989 and 1996 revisions. You will also find symbols for typesetting Indo-European and Germanic linguistics as well as American transcription symbols.
- Load it by including the following line in the preamble:

```
\usepackage{tipa}
```

- To change to TIPA within your source file, type

```
\textipa{ }
```

hɪəz ən ɪg'zɑ:mpɫ əv ə
fən'etɪk ˌtræn'skrɪpʃn

```
\textipa{hI@z @n Ig"zA:mp\s{1} @v @  
f@n"EtIk ""tr\ae n"skrIpS\s{n}}
```

- TIPA also comes with slanted and sans serif fonts. To access these,

```
\textsl{\textipa{ ... }}
```

```
\textsf{\textipa{ ... }}
```

```
\textsf{\textipa{ ... }}
```

- You can also switch to IPA using the declaration

```
\tipaencoding
```

```
\textsf{\textipa{ ... }}
```

- You can also switch to IPA using the declaration

```
\tipaencoding
```

or the environment

```
\begin{IPA}
```

```
⋮
```

```
\end{IPA}
```

- Most of the common symbols are accessible with a single key stroke.

- Most of the common symbols are accessible with a single key stroke.

SYMBOL	TYPESET AS	SYMBOL	TYPESET AS	SYMBOL	TYPESET AS
α	A	η	N	$\hbar$	0
β	B	$\varnothing$	O	$\dot{\imath}$	1
$\mathcal{C}$	C	$\mathcal{P}$	P	Λ	2
$\mathcal{D}$	D	$\mathcal{Q}$	Q	3	3
ε	E	$\mathcal{R}$	R	$\mathcal{U}$	4
Φ	F	$\int$	S	$\mathcal{V}$	5
$\mathcal{Y}$	G	θ	T	$\mathcal{D}$	6
$\mathcal{H}$	H	$\mathcal{U}$	U	$\mathcal{V}$	7
$\mathcal{I}$	I	$\mathcal{V}$	V	$\emptyset$	8
$\mathcal{J}$	J	$\mathcal{W}$	W	$\mathcal{O}$	9
$\mathcal{K}$	K	$\mathcal{X}$	X	$\mathcal{I}$	:
$\mathcal{L}$	L	$\mathcal{Y}$	Y	$\mathcal{V}$	;
$\mathcal{M}$	M	$\mathcal{Z}$	Z	$\mathcal{I}$	"

- TIPA provides macros to access many of the other symbols with just a few key strokes. The `\*` macro typesets mainly turned symbols and a few others.

- TIPA provides macros to access many of the other symbols with just a few key strokes. The `\*` macro typesets mainly turned symbols and a few others.

SYMBOL	TYPESET AS
J	<code>*f</code>
ŋ	<code>*k</code>
ɹ	<code>*r</code>
ɿ	<code>*t</code>
ʌ	<code>*w</code>
ʃ	<code>*j</code>
ɲ	<code>*n</code>
ħ	<code>*h</code>
ɺ	<code>*l</code>
ɸ	<code>*z</code>

- The `\:` macro typesets the symbols for retroflex consonants.

- The `\:` macro typesets the symbols for retroflex consonants.

SYMBOL	TYPESET AS
ṭ	<code>\:t</code>
ḍ	<code>\:d</code>
ṇ	<code>\:n</code>
ḷ	<code>\:l</code>
ṛ	<code>\:r</code>
Ṛ	<code>\:R</code>
ṣ	<code>\:s</code>
ṡ	<code>\:z</code>

- The `\!` macro typesets the symbols for implosives, as well as the odd click.

- The `\!` macro typesets the symbols for implosives, as well as the odd click.

SYMBOL	TYPESET AS
ɓ	<code>\!b</code>
ɗ	<code>\!d</code>
ɠ	<code>\!g</code>
ɢ	<code>\!G</code>
ʘ	<code>\!o</code>

- To get superscript symbols, type

`\super{ }`

$p^h g^w t^f$

`p\super{h} g\super{w} t\super{Q}`

- If you want to access tone letters, include the following line in your preamble.
This loads both the tipa core and the extra tone letter macros.

- If you want to access tone letters, include the following line in your preamble. This loads both the tipa core and the extra tone letter macros.

```
\usepackage[tone]{tipa}
```

Then you can typeset things like $\bar{\text{ma}}$ `\tone{55}ma` and $\surd\text{ma}$ `\tone{214}ma`.

- If you want to access tone letters, include the following line in your preamble. This loads both the tipa core and the extra tone letter macros.

```
\usepackage[tone]{tipa}
```

Then you can typeset things like $\bar{\text{ma}}$ `\tone{55}ma` and $\surd\text{ma}$ `\tone{214}ma`.

- TIPA offers a rich array of simple and complex accents. Refer to `tipaman` for details.

- If you want to access tone letters, include the following line in your preamble. This loads both the tipa core and the extra tone letter macros.

```
\usepackage[tone]{tipa}
```

Then you can typeset things like $\top$ ma `\tone{55}ma` and $\surd$ ma `\tone{214}ma`.

- TIPA offers a rich array of simple and complex accents. Refer to `tipaman` for details.

DESIRED OUTPUT	CTRL SEQUENCE 1	CTRL SEQUENCE 2
à	<code>\tipaupperaccent{0}{a}</code>	<code>\‘a</code>
á	<code>\tipaupperaccent{1}{a}</code>	<code>\’a</code>
â	<code>\tipaupperaccent{2}{a}</code>	<code>\^a</code>
ã	<code>\tipaupperaccent{3}{a}</code>	<code>\~a</code>
ä	<code>\tipaupperaccent{4}{a}</code>	<code>\"a</code>
ǎ	<code>\tipaupperaccent{5}{a}</code>	<code>\H{a}</code>
ȁ	<code>\tipaupperaccent{6}{a}</code>	<code>\r{a}</code>
ǣ	<code>\tipaupperaccent{7}{a}</code>	<code>\v{a}</code>
ǿ	<code>\tipaupperaccent{8}{a}</code>	<code>\u{a}</code>
ā	<code>\tipaupperaccent{9}{a}</code>	<code>\=a</code>

- `\tipaupperaccent` takes an `[adjust]` option which allows you to specify the vertical distance between the accent and the main character.

- `\tipaupperaccent` takes an [adjust] option which allows you to specify the vertical distance between the accent and the main character.

DESIRED OUTPUT	CTRL SEQUENCE
á	<code>\tipaupperaccent[.1ex]{1}{a}</code>
á	<code>\tipaupperaccent[.2ex]{1}{a}</code>
á	<code>\tipaupperaccent[.3ex]{1}{a}</code>
á	<code>\tipaupperaccent[.4ex]{1}{a}</code>
á	<code>\tipaupperaccent[.5ex]{1}{a}</code>
á	<code>\tipaupperaccent[.6ex]{1}{a}</code>
á	<code>\tipaupperaccent[.7ex]{1}{a}</code>
á	<code>\tipaupperaccent[.8ex]{1}{a}</code>
á	<code>\tipaupperaccent[.9ex]{1}{a}</code>
á	<code>\tipaupperaccent[1ex]{1}{a}</code>

There is a command `\tipaloweraccent` which works identically.

- Two other commands, useful in typesetting historical linguistics, are `\tipaUpperaccent` and `\tipaLoweraccent`. Notice the capital! — L^AT_EX commands are case-sensitive! Use these if you want to use other symbols in accent function, e.g.

`\tipaUpperaccent{e}{u}` → ũ

- Two other commands, useful in typesetting historical linguistics, are `\tipaUpperaccent` and `\tipaLoweraccent`. Notice the capital! — L^AT_EX commands are case-sensitive! Use these if you want to use other symbols in accent function, e.g.

`\tipaUpperaccent{e}{u}` → ŭ

This doesn't look so great, so to adjust the size and positioning of the *e*-accent, you might try

`\tipaUpperaccent[.3ex]{\scriptsize{e}}{u}` → ^eu

- Two other commands, useful in typesetting historical linguistics, are `\tipaUpperaccent` and `\tipaLoweraccent`. Notice the capital! — L^AT_EX commands are case-sensitive! Use these if you want to use other symbols in accent function, e.g.

`\tipaUpperaccent{e}{u}` → ũ

This doesn't look so great, so to adjust the size and positioning of the *e*-accent, you might try

`\tipaUpperaccent[.3ex]{\scriptsize{e}}{u}` → ^eũ

Here we've included the [adjust] option and reduced the size of the accent using the scriptsize declaration.

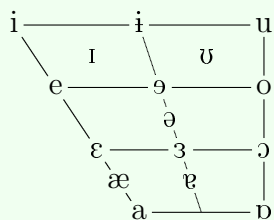
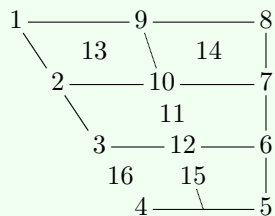
- Accents may be stacked simply by nesting, e.g.

```
\textipa { \'  
          {\~  
           {\=  
            {\\"a  
             }  
          }  
        }  
      }
```

This generates $\tilde{\tilde{a}}$.

- Included in TIPA is the **vowel** package, which allows you to draw vowel quadrilaterals à la Daniel Jones (as well as several other styles). It is useful know something of the proportions of this diagram. If the bottom line (a–ɑ) is 2, then the right hand side (ɑ–u) is 3 and the top line (u–i) is 4.

- Included in TIPA is the **vowel** package, which allows you to draw vowel quadrilaterals à la Daniel Jones (as well as several other styles). It is useful know something of the proportions of this diagram. If the bottom line (a–ɑ) is 2, then the right hand side (ɑ–u) is 3 and the top line (u–i) is 4.



```
\begin{vowel}
  \putcvowel{i}{1}
  \putcvowel{e}{2}
  \putcvowel{\textipa{E}}{3}
  \putcvowel{a}{4}
  ...
\end{vowel}
```

- Another useful feature of `vowel` is the ability to plot vowel realizations in the quadrilateral. The relevant commands are `\vowelhunit` (h = horizontal) and `\vowelvunit` (v = vertical).

- Another useful feature of **vowel** is the ability to plot vowel realizations in the quadrilateral. The relevant commands are `\vowelhunit` (h = horizontal) and `\vowelvunit` (v = vertical). Notice that the origin of the graph in this case is the *top left* corner, *not* the bottom left. This makes sense in this context, since *O* corresponds to the location of Cardinal Vowel 1 [i].

- Another useful feature of **vowel** is the ability to plot vowel realizations in the quadrilateral. The relevant commands are `\vowelhunit` (h = horizontal) and `\vowelvunit` (v = vertical). Notice that the origin of the graph in this case is the *top left* corner, *not* the bottom left. This makes sense in this context, since *O* corresponds to the location of Cardinal Vowel 1 [i]. So, `\vowelhunit` specifies the displacement to the right of *O*. `\vowelvunit` specifies the displacement *down* from *O*.

- Another useful feature of **vowel** is the ability to plot vowel realizations in the quadrilateral. The relevant commands are `\vowelhunit` (h = horizontal) and `\vowelvunit` (v = vertical). Notice that the origin of the graph in this case is the *top left* corner, *not* the bottom left. This makes sense in this context, since *O* corresponds to the location of Cardinal Vowel 1 [i]. So, `\vowelhunit` specifies the displacement to the right of *O*. `\vowelvunit` specifies the displacement *down* from *O*. Thinking back to the proportions for the quadrilateral, CV 8 [u] is at `4\vowelhunit` and `0\vowelvunit`, CV 5 [ɑ] is at `4\vowelhunit` and `3\vowelvunit` and CV 4 [a] is at `2\vowelhunit` and `3\vowelvunit`.

- Another useful feature of **vowel** is the ability to plot vowel realizations in the quadrilateral. The relevant commands are `\vowelhunit` (h = horizontal) and `\vowelvunit` (v = vertical). Notice that the origin of the graph in this case is the *top left* corner, *not* the bottom left. This makes sense in this context, since *O* corresponds to the location of Cardinal Vowel 1 [i]. So, `\vowelhunit` specifies the displacement to the right of *O*. `\vowelvunit` specifies the displacement *down* from *O*. Thinking back to the proportions for the quadrilateral, CV 8 [u] is at `4\vowelhunit` and `0\vowelvunit`, CV 5 [ɑ] is at `4\vowelhunit` and `3\vowelvunit` and CV 4 [a] is at `2\vowelhunit` and `3\vowelvunit`. With these as reference points, here's how you might plot the vowels for English RP:

```
\begin{vowel}  
  \putvowel{\ipa{\normalsize i:}}{0.5\ vowelhunit}{0.2\ vowelvunit}  
  \putvowel{\ipa{\normalsize I}}{1.4\ vowelhunit}{0.7\ vowelvunit}  
  \putvowel{\ipa{\normalsize e}}{1.3\ vowelhunit}{1.6\ vowelvunit}  
  \putvowel{\ipa{\normalsize 3:}}{2.3\ vowelhunit}{1.7\ vowelvunit}  
  \putvowel{\ipa{\normalsize \ae}}{1.9\ vowelhunit}{2.5\ vowelvunit}  
  \putvowel{\ipa{\normalsize 2}}{2.6\ vowelhunit}{2.4\ vowelvunit}  
  \putvowel{\ipa{\normalsize A:}}{3.5\ vowelhunit}{2.8\ vowelvunit}  
  \putvowel{\ipa{\normalsize 6}}{3.8\ vowelhunit}{2.6\ vowelvunit}  
  \putvowel{\ipa{\normalsize 0:}}{3.7\ vowelhunit}{1.5\ vowelvunit}  
  \putvowel{\ipa{\normalsize U}}{3.4\ vowelhunit}{0.7\ vowelvunit}  
  \putvowel{\ipa{\normalsize u:}}{3.3\ vowelhunit}{0.2\ vowelvunit}  
\end{vowel}
```

iː		uː
ɪ		ʊ
e	ɜː	ɔː
æ	ʌ	ɑː
		ɒ

2. Typesetting syntactic trees using **qtree**

2. Typesetting syntactic trees using **qtree**

- Invoke **qtree** in the preamble with

```
\usepackage{qtree}
```

2. Typesetting syntactic trees using **qtree**

- Invoke **qtree** in the preamble with

```
\usepackage{qtree}
```

- Those of you who are using Gerben Wierda's T_EXShop for Mac OS X must obtain the style file **qtree.sty** and place this somewhere T_EX can find it, e.g. in Home/Library/texmf/tex/latex/.

- Basic tree-drawing with the `qtree` involves generating a tree from a bracketed structure like this:

- Basic tree-drawing with the `qtree` involves generating a tree from a bracketed structure like this:

```
\Tree [.S_This_ [.VP_ [.V_is_]_ \qroof{a simple tree}.NP_]_]
```

- Basic tree-drawing with the `qtree` involves generating a tree from a bracketed structure like this:

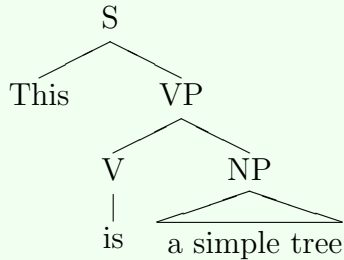
```
\Tree [.S_This_ [.VP_ [.V_is_]_ \qroof{a simple tree}.NP_]_]
```

To be explicit about the location of the spaces, I've rendered these visible as `_`.

- Basic tree-drawing with the `qtree` involves generating a tree from a bracketed structure like this:

```
\Tree [.S_This_.VP_.V_is_.\qroof{a simple tree}.NP_.]
```

To be explicit about the location of the spaces, I've rendered these visible as `_.` The command for this is `\textvisiblespace`. The code for the tree should produce



- **qtree** automatically loads another package called **eeepic**, which is used to generate PostScript pictures. If you are using T_EXShop, which by default compiles T_EX code directly to pdf-format using pdfL^AT_EX, you will not be able to see or print the node-connecting edges unless you do one of two things. Either:

- **qtree** automatically loads another package called **eeepic**, which is used to generate PostScript pictures. If you are using **T_EXShop**, which by default compiles **T_EX** code directly to pdf-format using **pdfL^AT_EX**, you will not be able to see or print the node-connecting edges unless you do one of two things. Either:
 1. In the **Typeset** menu, instead of the default **Pdftex**, select **T_EX** and **Ghostscript**.

- **qtree** automatically loads another package called **eeepic**, which is used to generate PostScript pictures. If you are using T_EXShop, which by default compiles T_EX code directly to pdf-format using pdfL^AT_EX, you will not be able to see or print the node-connecting edges unless you do one of two things. Either:
 1. In the **Typeset** menu, instead of the default **Pdftex**, select **TeX** and **Ghostscript**.
 2. In your preamble, load **qtree** with the **noeepic** option as

```
\usepackage[noeepic]{qtree}
```

This will give you lines of slightly poorer quality, but directly generates pdf-output. If you want to include movement arrows (see further below), there is no way around using **eeepic**, so use the first option.

- By default, trees are centered. To turn this *off*, type the following declaration:

```
\qtreetocenterfalse
```

- By default, trees are centered. To turn this *off*, type the following declaration:

```
\qtreetcenterfalse
```

If you want this setting to apply to the entire document, include the line in your preamble. Other times, you might just want to switch off centering for a single example (e.g. to stop the tree from spilling over into the right margin), in which case you issue it at the appropriate location. To turn centering *on*, issue the declaration

```
\qtreetcentertrue
```

- Tree labels are written immediately adjacent to the relevant bracket preceded by a full stop, e.g.

[.CP_

- Tree labels are written immediately adjacent to the relevant bracket preceded by a full stop, e.g.

[.CP_

White space tells $\text{\TeX}$ that it's reached the end of the label. If you want to include labels which have spaces in them, or if a node label ends in a control sequence, such as $\text{\LaTeX}$, $\text{\ldots}$, and so on, you need to enclose it in parentheses.

- Labels may be appended to either the left or right bracket, or both. All of the following produce the following tree.

- Labels may be appended to either the left or right bracket, or both. All of the following produce the following tree.

```
\Tree [ .CP [ .C when ] [ .IP [ .NP hell ] [ .VP freezes ] ] ]
```

- Labels may be appended to either the left or right bracket, or both. All of the following produce the following tree.

```
\Tree [.CP[.C_when_] [.IP[.NP_hell_] [.VP_freezes_] ] ]
```

```
\Tree [.CP[.C_when_] [.IP[ hell_] .NP[.VP_freezes_] ] ]
```

- Labels may be appended to either the left or right bracket, or both. All of the following produce the following tree.

```
\Tree [.CP_.C_when_.] [.IP_.NP_hell_.] [.VP_freezes_.] ] ]
```

```
\Tree [.CP_.C_when_.] [.IP_. hell_.] .NP_.VP_freezes_. ] ]
```

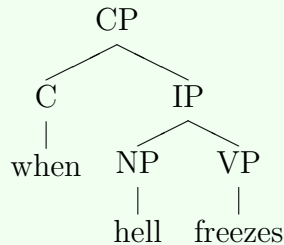
```
\Tree [.CP_.C_when_.] [.IP_.NP_hell_.] .NP_.VP_freezes_. ]  
]
```

- Labels may be appended to either the left or right bracket, or both. All of the following produce the following tree.

```
\Tree [.CP_ [.C_ when_] [.IP_ [.NP_ hell_] [.VP_ freezes_] ] ]
```

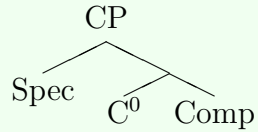
```
\Tree [.CP_ [.C_ when_] [.IP_ [ hell_] .NP_ [.VP_ freezes_] ] ]
```

```
\Tree [.CP_ [.C_ when_] [.IP_ [.NP_ hell_] .NP_ [.VP_ freezes_] ] ]
```



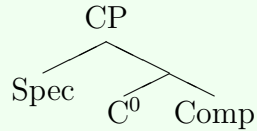
- Labels can also be left off where we don't want to be specific about category.

```
\Tree [ .CP Spec [ C0 Comp ] ]
```



- Labels can also be left off where we don't want to be specific about category.

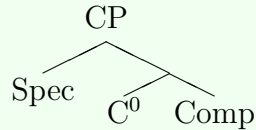
```
\Tree [.CP Spec [ C0 Comp ] ]
```



- Where we don't want to be specific about the internal structure of a phrase, we can use the `\qroof{ }` command.

- Labels can also be left off where we don't want to be specific about category.

```
\Tree [.CP Spec [ C0 Comp ] ]
```

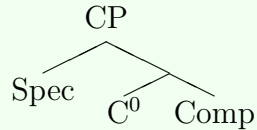


- Where we don't want to be specific about the internal structure of a phrase, we can use the `\qroof{ }` command.

```
\Tree [.IP Roofs [.IP are \qroof{great for lazy syntacticians}.AP
] ]
```

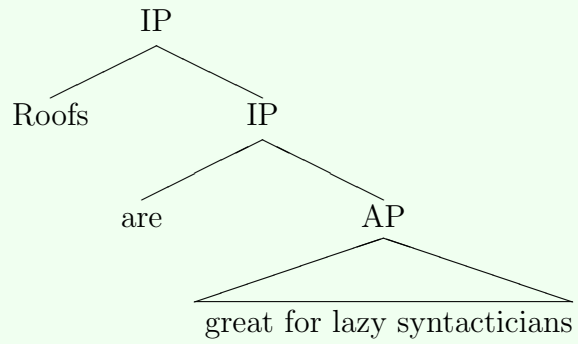
- Labels can also be left off where we don't want to be specific about category.

```
\Tree [.CP Spec [ C0 Comp ] ]
```



- Where we don't want to be specific about the internal structure of a phrase, we can use the `\qroof{ }` command.

```
\Tree [.IP Roofs [.IP are \qroof{great for lazy syntacticians}.AP
] ]
```



- Primes may be typeset using either $X^{\prime}$ or X' .

- Primes may be typeset using either $X\prime$ or X_1 .

```
\Tree [ .CP { \sc Spec } [ .C\1 C \qroof{\ldots\ldots}.XP ] ]
```

- Primes may be typeset using either $X\prime$ or X_1 .

```
\Tree [.CP {\sc Spec} [.C\prime C \qroof{\ldots\ldots}.XP ] ]
```

or

```
\Tree [.CP {\sc Spec} [.C\prime C \qroof{\ldots\ldots}.XP ] ]
```

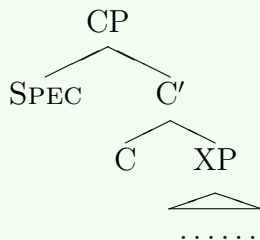
- Primes may be typeset using either $X\$'$ or $X\1$.

`\Tree [.CP {\sc Spec} [.C\1 C \qroof{\ldots\ldots}.XP ] ]`

or

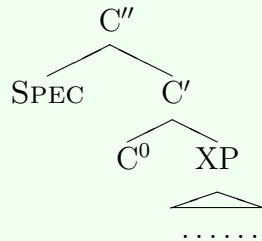
`\Tree [.CP {\sc Spec} [.C\$' C \qroof{\ldots\ldots}.XP ] ]`

gives

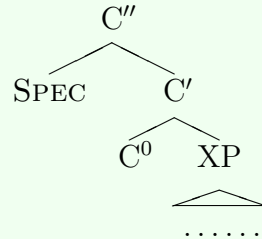


For X-bar style double primes, type x'' , for heads x'

For X-bar style double primes, type $\mathbf{x\backslash2}$, for heads $\mathbf{x\backslash0}$

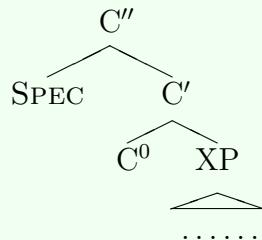


For X-bar style double primes, type $\text{X}^{\prime\prime}$, for heads X^0

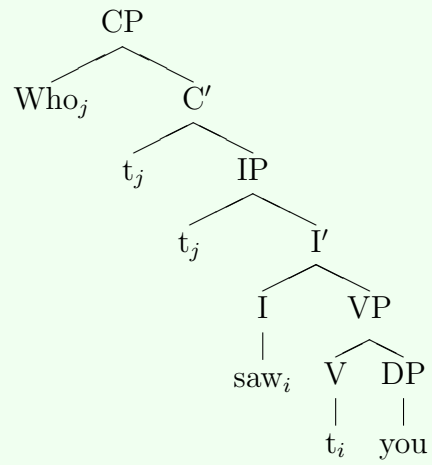


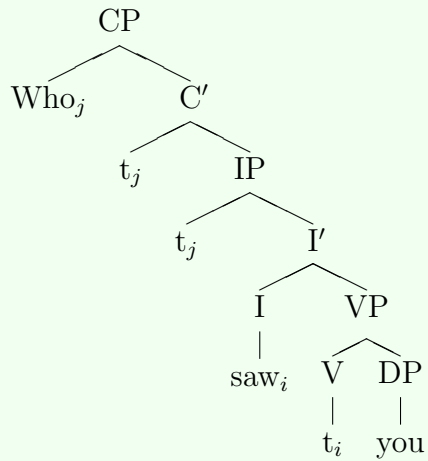
- Indices are typeset *in trees* as X_i . For superscript, use X^0 .

For X-bar style double primes, type $\text{x}\backslash 2$, for heads $\text{x}\backslash 0$



- Indices are typeset *in trees* as x_i . For superscript, use x^0 .





- If you want subscripts and superscripts in your texts, type x_i and x^0 .

- For larger trees, it is possible to tweak the distance between sister nodes to improve the aesthetics and fit the tree on the page. The commands for this are `\qsetw{<length>}` and `\faketreewidth{<text>}`.

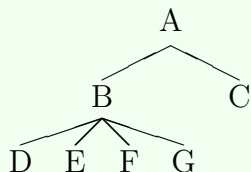
- For larger trees, it is possible to tweak the distance between sister nodes to improve the aesthetics and fit the tree on the page. The commands for this are `\qsetw{<length>}` and `\faketreewidth{<text>}`.

```
\Tree [.A [ D E F G ].B [.C ] ]
```

- For larger trees, it is possible to tweak the distance between sister nodes to improve the aesthetics and fit the tree on the page. The commands for this are `\qsetw{<length>}` and `\faketreewidth{<text>}`.

```
\Tree [.A [ D E F G ].B [.C ] ]
```

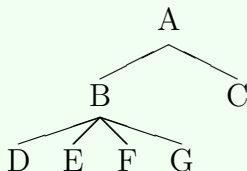
gives



- For larger trees, it is possible to tweak the distance between sister nodes to improve the aesthetics and fit the tree on the page. The commands for this are `\qsetw{<length>}` and `\faketreewidth{<text>}`.

```
\Tree [.A [ D E F G ].B [.C ] ]
```

gives



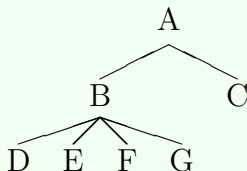
Compare this with

```
\Tree [.A [ D E F G ].B !\qsetw{1cm} [.C ] ]
```

- For larger trees, it is possible to tweak the distance between sister nodes to improve the aesthetics and fit the tree on the page. The commands for this are `\qsetw{<length>}` and `\faketreewidth{<text>}`.

```
\Tree [.A [ D E F G ].B [.C ] ]
```

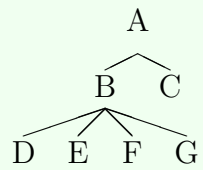
gives

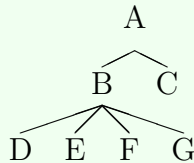


Compare this with

```
\Tree [.A [ D E F G ].B !\qsetw{1cm} [.C ] ]
```

which gives





Notice that `\qsetw{ }` is preceded by an exclamation mark, which tells the parser to override the default calculation of width with whatever follows. If you omit the exclamation mark, the parser will interpret the material as an extra node.

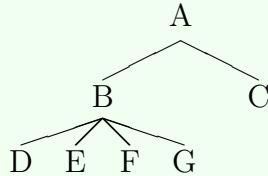
- `\faketreewidth` works by inserting the length of whatever text you type in the argument of the command.

- `\faketreewidth` works by inserting the length of whatever text you type in the argument of the command.

```
\Tree [.A [ D E F G ].B !\faketreewidth{my dog has fleas} [.C  
] ]
```

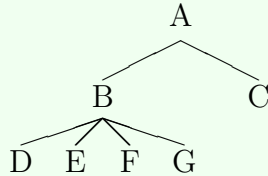
- `\faketreewidth` works by inserting the length of whatever text you type in the argument of the command.

```
\Tree [.A [ D E F G ].B !\faketreewidth{my dog has fleas} [.C  
 ] ]
```



- `\faketreewidth` works by inserting the length of whatever text you type in the argument of the command.

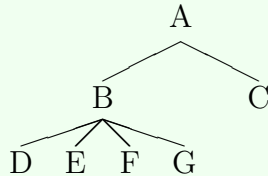
```
\Tree [.A [ D E F G ].B !\faketreewidth{my dog has fleas} [.C  
] ]
```



Now look what happens if you skip the exclamation mark:

- `\faketreewidth` works by inserting the length of whatever text you type in the argument of the command.

```
\Tree [.A [ D E F G ].B !\faketreewidth{my dog has fleas} [.C  
] ]
```

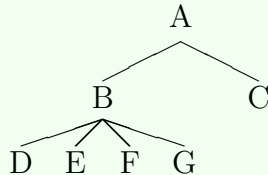


Now look what happens if you skip the exclamation mark:

```
\Tree [.A [ D E F G ].B \faketreewidth{my dog has fleas} [.C  
] ]
```

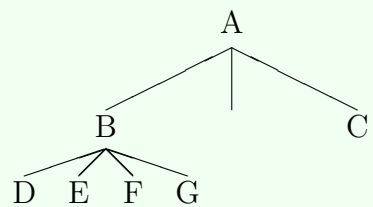
- `\faketreewidth` works by inserting the length of whatever text you type in the argument of the command.

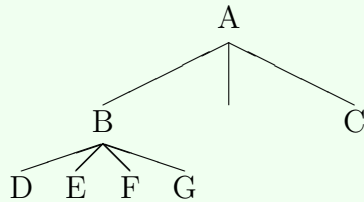
```
\Tree [.A [ D E F G ].B !\faketreewidth{my dog has fleas} [.C  
] ]
```



Now look what happens if you skip the exclamation mark:

```
\Tree [.A [ D E F G ].B \faketreewidth{my dog has fleas} [.C  
] ]
```





- If these methods don't help you fit the tree on the page, you can always change the text size, e.g. by enclosing the entire tree in `{\small ...}`.

- To draw movement arrows between nodes, you need another package called `tree-dvips`.

- To draw movement arrows between nodes, you need another package called **tree-dvips**. This package can create trees too, although it doesn't make it easy. Still, the package has very useful line- and arrow-drawing commands, and these are fully compatible with **qtree**.

- To draw movement arrows between nodes, you need another package called **tree-dvips**. This package can create trees too, although it doesn't make it easy. Still, the package has very useful line- and arrow-drawing commands, and these are fully compatible with **qtree**. **tree-dvips** was designed for the previous version of $\text{\LaTeX}$, $\text{\LaTeX}2.09$, so when you try and load the package in your document, you might find $\text{\TeX}$ Shop warns you of a possible incompatibility. Ignore it: **tree-dvips** is upwardly compatible with $\text{\LaTeX}2_{\epsilon}$.

- To draw movement arrows between nodes, you need another package called **tree-dvips**. This package can create trees too, although it doesn't make it easy. Still, the package has very useful line- and arrow-drawing commands, and these are fully compatible with **qtree**. **tree-dvips** was designed for the previous version of $\text{\LaTeX}$, $\text{\LaTeX}2.09$, so when you try and load the package in your document, you might find $\text{\TeX}$ Shop warns you of a possible incompatibility. Ignore it: **tree-dvips** is upwardly compatible with $\text{\LaTeX}2_{\epsilon}$. So, in your preamble:

```
\usepackage{tree-dvips}
```

- To draw movement arrows between nodes, you need another package called **tree-dvips**. This package can create trees too, although it doesn't make it easy. Still, the package has very useful line- and arrow-drawing commands, and these are fully compatible with **qtree**. **tree-dvips** was designed for the previous version of $\text{\LaTeX}$, $\text{\LaTeX}2.09$, so when you try and load the package in your document, you might find $\text{\TeX}$ Shop warns you of a possible incompatibility. Ignore it: **tree-dvips** is upwardly compatible with $\text{\LaTeX}2_{\epsilon}$. So, in your preamble:

```
\usepackage{tree-dvips}
```

tree-dvips works by defining nodes using the `\node{ }` command and connecting these nodes using `\anodecurve` and `\abarnodeconnect`. It is good practise, when writing your source file, to be generous with white space. Use tabbing to make the syntactic structure of your document more visible. This will save considerable time when you have to hunt down errors while debugging your document. In the source for the tree structure below, I've indented the left bracket of every new level. Closing right brackets are positioned with the same indentation as their left counterparts.

```

\Tree    [.CP \node{whoj}Who_j
          [.C\1 C
            [.IP \node{tj}t_j
              [.I\1
                [.I \node{sawi}saw_i ]
                [.VP
                  [.V \node{ti}t_i ]
                  [.DP you ]
                ]
              ]
            ]
          ]
]
\anodecurve[bl]{ti}[bl]{sawi}{.4in}
\anodecurve[bl]{tj}[bl]{whoj}{.4in}

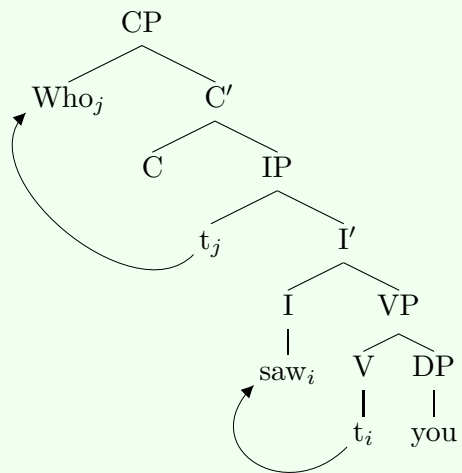
```

```

\Tree      [.CP \node{whoj}Who_j
            [.C\1 C
              [.IP \node{tj}t_j
                [.I\1
                  [.I \node{sawi}saw_i ]
                  [.VP
                    [.V \node{ti}t_i ]
                    [.DP you ]
                  ]
                ]
              ]
            ]
          ]
        ]
      ]
    ]
  ]
]
\anodecurve[bl]{ti}[bl]{sawi}{.4in}
\anodecurve[bl]{tj}[bl]{whoj}{.4in}

```

This gives



```

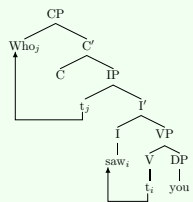
\Tree    [.CP \node{whoj}Who_j
          [.C\1 C
            [.IP \node{tj}t_j
              [.I\1
                [.I \node{sawi}saw_i ]
                [.VP
                  [.V \node{ti}t_i ]
                  [.DP you ]
                ]
              ]
            ]
          ]
        ]
\abarnodeconnect[-6pt]{ti}{sawi}
\abarnodeconnect[-6pt]{tj}{whoj}

```

```

\Tree    [.CP \node{whoj}Who_j
          [.C\1 C
            [.IP \node{tj}t_j
              [.I\1
                [.I \node{sawi}saw_i ]
                [.VP
                  [.V \node{ti}t_i ]
                  [.DP you ]
                ]
              ]
            ]
          ]
        ]
      ]
    ]
  ]
\abarnodeconnect[-6pt]{ti}{sawi}
\abarnodeconnect[-6pt]{tj}{whoj}

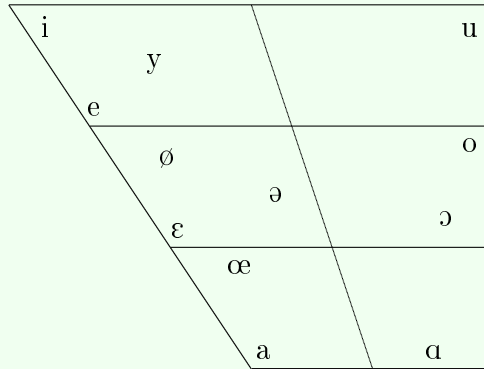
```



- For certain purposes, `qtree` may not be able to give you what you want. If you need to align terminal nodes on the same horizontal line, then you need to use other tree-drawing macros.

3. Exercises

1. Download the TIPA manual `tipaman.pdf` by Fukui Rei from the `LATEX` for Linguists website and use to help you transcribe the following expressions:
 - (a) [ˈhɪəz ɪ ɪɡˈzɑːmplɪ əv ə ˈbrɔːd fəˈnetɪk ˌtrænzˈskrɪpʃn]
 - (b) [hɪˈzɪŋzɪːfɪtəpənæɹəfɪnɛxtɪsɪvɪfɪʃn]
2. The quadrilateral below has the vowels of Parisian French plotted on it. Can you give the source? Extra help is available in `vowel.pdf` by Fukui Rei.



3. Give the source for the following tree. Compile it. (Hint: create the empty node in the very bottom VP using `\phantom{x}`. This will insert an invisible character 'x' as the terminal node.)

