ADPOSITIONAL TREES IN LaTeX

MARCO BENINI AND FEDERICO GOBBO

1. Introduction

Adpositional trees (adtrees, for short) are a formal way to represent expressions in natural language. They have been defined, used, and discussed in F. Gobbo and M. Benini, *Constructive Adpositional Grammars: Foundations of Constructive Linguistics*, Cambridge Scholar Press (2011). Since then, they have been used without theoretical modifications in a number of applications.

This document describes the LaTeX package to draw adtrees. It is written so that a user may start typesetting adtrees as soon as possible: all the fundamental commands and macros are presented in Section 2. At the first reading, the subsequent sections may be freely skipped.

Section 3 describes the next obvious step: how to put an adtree inside your document, controlling the way to align it with other objects. Section 4 explains how to control the length of branches via the `\unitlength` value. It also introduces the variants of the fundamental commands to modify the lengths of the branches in the adtree, the angle between them, or both, either globally, locally, or recursively.

Section 5 describes the low-level definitions that affect the construction of the various pieces composing adtrees. Changing these definitions is intended for advanced users, who want to modify the standard appearance of adtrees. Section 6 concludes the description of the main graphical package, showing some special techniques to deal with adtrees, and a couple of very low-level primitives of the package.

Sections 7 and 8 describe two alternatives way to compose adtrees. the former shows a *path-like* format which is useful when keeping the order of morphemes in the original sentence is important; the latter is a textual rendering of an adtree as a piece of indented text.

The final Section 9 describes the way to render adtrees in a linear format. This is a minor feature which we discourage to use, except for very special purposes.

2. Simple Adtrees

The installation of the package is standard: see the documentation on your particular TEX system for the details. It reduces to put the `.sty` file in a sensible place. Using the package in a document requires it to be invoked with `\usepackage{adtrees}` in the preamble.

The simplest adtree is composed by a single morpheme:

Liza
O

which has been typeset by the command

```
\ATm{Liza}{O}
```

Sometimes, a morpheme requires to specify attributes:

Liza
O
[proper noun]
[animated]

this behaviour is accomplished by the command

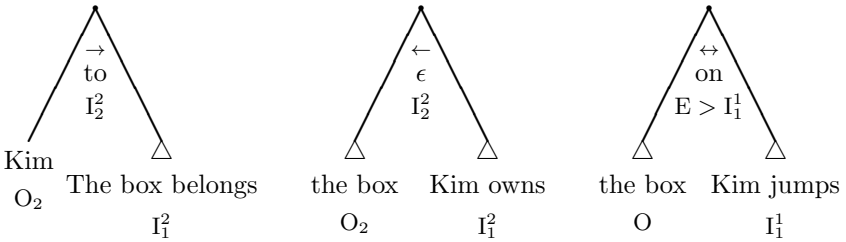
```
\ATm{Liza}{O}[proper noun][animated]
```

The general form of the `\ATm` command is

$$\texttt{\textbackslash ATm\{}m\texttt{\}\{}g\texttt{\}[}a_1\texttt{]}\cdots\texttt{[}a_n\texttt{]}$$

with m the morpheme, g its grammar character, and $a_1, \ldots, a_n$ its attributes, if any. The attributes are optional, and there can be any number of them. Also, notice how all the commands in the package start with `\AT`: the final `m` stands for *morpheme*. This is a general naming rule: all the fundamental command have the form `\AT` followed by a single letter which reminds its function. Variants of the fundamental commands use a two-letter code, the first letter being the same as the fundamental command, and the second letter reminding the variant.

2.1. **Complex Adtrees.** A complex adtree is made by two adtrees, composed via an *adposition*, which is described by a morpheme together with the grammar character of the resulting adtree, and the *trajectory*. The trajectory is an arrow, and it can be $\leftarrow$, $\rightarrow$, or $\leftrightarrow$. The following three examples are very similar to the ones in Figure 2.11 of Gobbo and Benini (2011):



These adtrees are typeset, respectively, by the commands

```
\ATr{to}{I_2^2}
{\ATm{Kim}{O_2}}
{\ATs{The box belongs}{I_1^2}}
```

```
\ATl{$\epsilon$}{I_2^2}
{\ATm{the box}{O_2}}
{\ATs{Kim owns}{I_1^2}}

\ATb{on}{E>I_1^1}
{\ATs{the box}{O_2}}
{\ATs{Kim jumps}{I_1^1}}
```

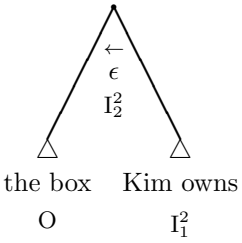
The general format of these commands is

$$c\{a\}\{g\}[x_1]\dots[x_n]\{L\}\{R\}$$

with c among `\ATl`, `ATr`, `\ATb`; a the adposition: g the grammar character of the whole construction; $x_1, \dots, x_n$ the optional attributes of the construction; L the left adtree; and R the right adtree. The adposition is typeset as normal text, the grammar character in math mode with the roman typeface, so that super- and sub-scripts can be freely used, and the trajectory is defined by the command: `\ATl` for the *left* arrow, `\ATr` for the *right* arrow, and `\ATb` for the arrow in *both* direction, following the general naming rule. The attributes are optional and they are typeset as in the morpheme construction.

Actually, the `\ATm` command is used to typeset proper morphemes, while the `\ATs` command is designed to typeset expressions which can be expanded into an adtree. In fact, the `s` letter stands for *summary*, following the general naming rule. The `\ATs` command has the same syntax as the `\ATm` command, and it differs just in the graphical appearance. So, it may take an arbitrary number of attributes, and whatever applies to `\ATm` holds for `\ATs`, too.

2.2. **Epsilon Adpositions.** Since writing adtrees with the empty adposition, represented by the ϵ symbol, is very common, the following commands are provided: `\ATle`, `\ATre`, and `\ATbe`, with `e` standing for *empty* in the general naming convention. They behave as the command without the trailing `e`, but the adposition is automatically typeset. Also, they share the same syntax. So, the previous example

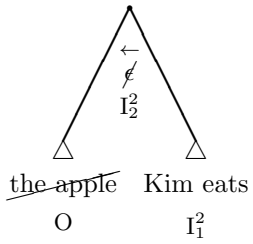


can be typeset in a simplified way as

```
\ATle{I_2^2}
{\ATs{the box}{O}}
{\ATs{Kim owns}{I_1^2}}
```

Similarly, because the so-called epsilon-transformations are common, and they require to forget about adpositions and morphemes, which is graphically marked by cancelling them with a stroke, the package automatically includes the `cancel` package by Donald Arseneau, (available in CTAN, at `http://mirror.ctan.org/macros/latex/contrib/cancel`). In addition, to simplify the cancellation of empty

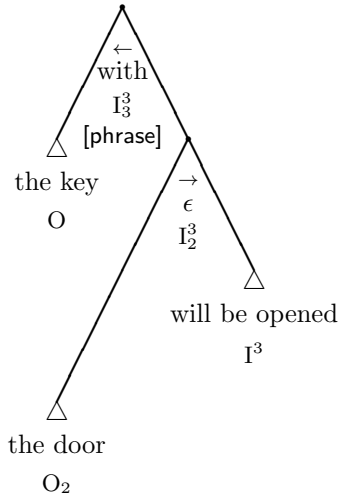
adpositions, the commands `\ATrc` `\ATlc` `\ATbc` are provided, with `c` standing for *cancel ed*. They work as their counterparts with the trailing `e`, sharing the same syntax. For example,



has been typeset by

```
\ATlc{I_2^2}
{\ATs{\cancel{the apple}}{O}}
{\ATs{Kim eats}{I_1^2}}
```

2.3. **Overlapping subtrees.** When one has to write complex adtrees, it is often the case that there are overlapping subtrees. To cope with these situations, the simplest way is to prolong the left or the right branch of a node. This behaviour can be accomplished with the `\ATxl` and `\ATxr` commands, where `x` stands for *extends* and `l` and `r` for *left* and *right*, respectively. They both take a single argument which is the adtree being appended at the end of the extended branch. For example, the following adtree, similar to the one in Figure 2.16 (Gobbo and Benini 2011):

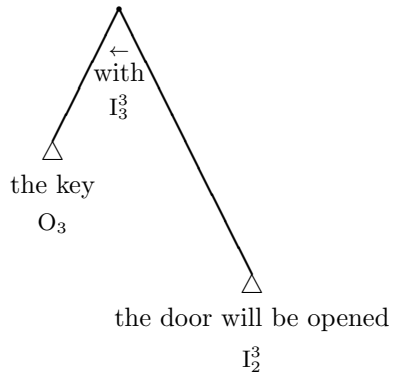


has been typeset by the following LaTeX code

```
\ATl{with}{I^3_3}[phrase]
{\ATs{the key}{O_3}}
{\ATre{I^3_2}
  {\ATxl{\ATs{the door}{O_2}}}
  {\ATs{will be opened}{I^3}}}
```

The example also shows how to put an attribute in the top adposition. Normally, for reasons of space, adtrees use extensively summaries to put in evidence only the

linguistic phenomenon needed in that moment. For example, the following adtree:

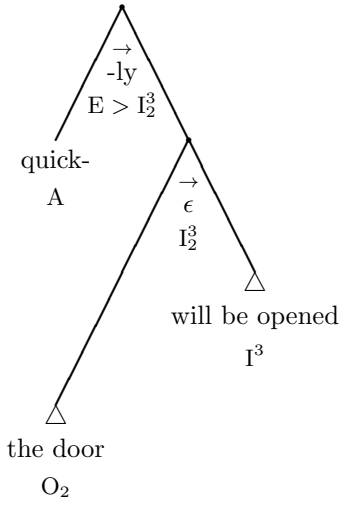


has been typeset by the following LaTeX code

```
\ATl{with}{I^3_3}
  {\ATs{the key}{O_3}}
  {\ATxr{\ATs{the door will be opened}{I^3_2}}}
```

The example also shows a right-branch extension. Overlapping can be also avoided in complex adtrees using advanced commands. See Section 4 for details.

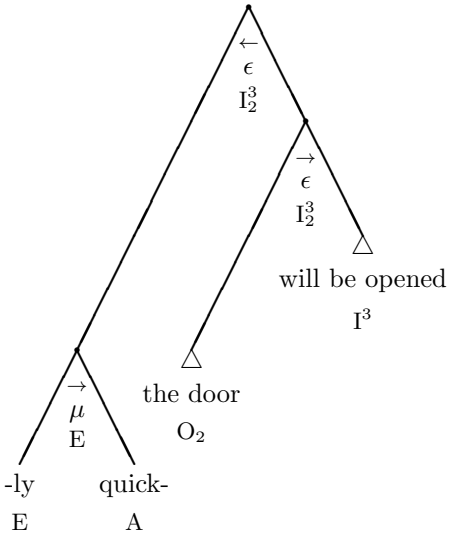
2.4. **Morphological adpositions.** Sometimes, for the sake of clarity, morphological relations should be stated explicitly. For example, the following adtree:



which has been typeset by the following LaTeX code

```
\ATr{-ly}{E>I^3_2}
{\ATm{quick-}{A}}
{\ATre{I^3_2}
  {\ATxl{\ATs{the door}{O_2}}}
  {\ATs{will be opened}{I^3}}}
```

could be made explicit, where the μ adposition indicates a morphological relation. For example, the following adtree is linguistically equivalent to the previous one:



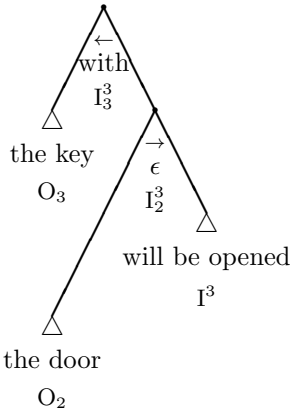
and it has been typeset by the following LaTeX code

```
\ATle{I^3_2}
{\ATxl{\ATxl{\ATrmu{E}
      {\ATm{-ly}{E}}
      {\ATm{quick-}{A}}}}}
{\ATre{I^3_2}
  {\ATxl{\ATs{the door}{O_2}}}
  {\ATs{will be opened}{I^3}}}
```

The package provides the commands `\ATlmu`, `\ATrmu`, and `\ATbmu`, analogous to the ϵ variants, to simplify the typesetting of adtrees using the μ adposition.

3. Alignments and Centring

Drawing an adtree in a display is the simplest and most common way:



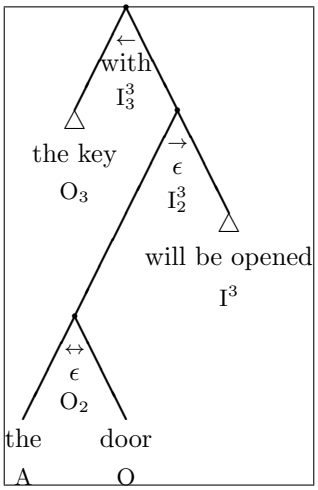
is generated by the following code

```
\begin{equation*}
  \unitlength.18ex
  \ATl{with}{I^3_3}
    {\ATs{the key}{O_3}}
    {\ATre{I^3_2}
      {\ATxl{\ATs{the door}{O_2}}}
      {\ATs{will be opened}{I^3}}}
\end{equation*}
```

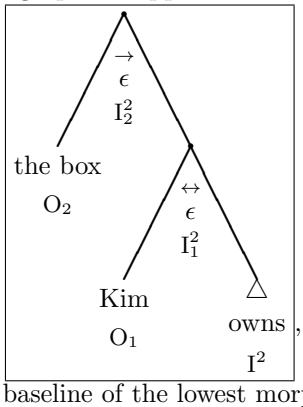
Putting an adtree in a figure environment reduces to write the corresponding LaTeX display inside the content of the `figure` environment.

In general, we strongly encourages the user to use a robust display environment, like `equation*` or `displaymath`, and to avoid the `center` environment. Advanced displays to align mathematical material, as in the $\mathcal{AMS}$ suite, are another good and reliable option to place adtrees in a page.

An adtree fits exactly its enclosing box, that is

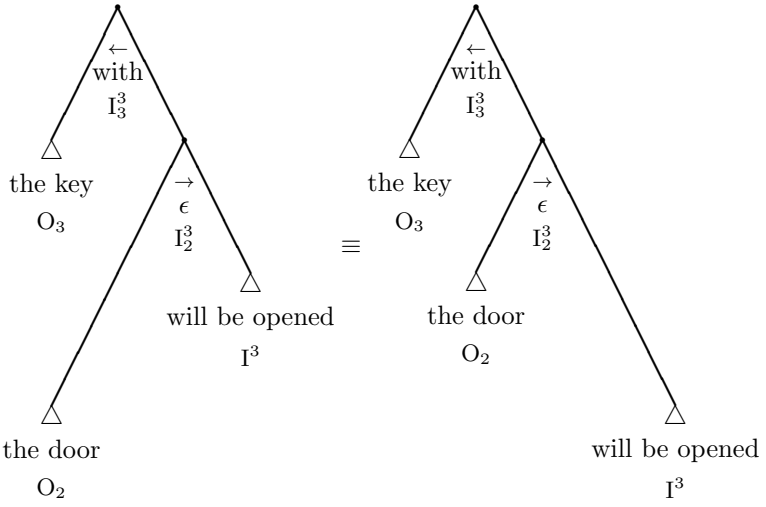


If one looks closely (and that is why the adtree is so big), there is half of the point in the root which lies outside the box. This is done on purpose, to make easier to collate together adtrees.

Writing a morpheme or a summary directly inside the text, like $\boxed{\begin{array}{c}\text{Gargoyle}\\ \text{O}\end{array}}$, produces a box whose baseline is exactly the baseline of the morpheme. This makes the behaviour predictable, even if the graphical appearance is quite terrible (like in this paragraph). An adtree, like 

, behaves similarly, because the baseline of the box is the baseline of the lowest morpheme or summary in the whole tree, the one corresponding to the *owns* morpheme in the example.

So, at least in principle, since the user can rely on the size of the enclosing box, and on the position of the baseline, it is possible to align an adtree in whatever way one may desire. In our experience, the only really common way one requires to move adtrees in the surrounding text is to centre them: the package provides three commands to centre an adtree horizontally, vertically, or both. These are `\ATvcentre`, `\AThcentre`, and `\ATcentre`, respectively. For example



has been generated by

```
\ATvcentre{\ATl{with}{I^3_3}
  {\ATs{the key}{O_3}}
  {\ATre{I^3_2}
```

```
    {\ATxl{\ATs{the door}{O_2}}}
    {\ATs{will be opened}{I^3}}}}
\mathbin{\equiv}
\ATvcentre{\ATl{with}{I^3_3}
  {\ATs{the key}{O_3}}
  {\ATre{I^3_2}
    {\ATs{the door}{O_2}}
  {\ATxr{\ATs{will be opened}{I^3}}}}}
```

More sophisticated ways to place an adtree in the page are discussed in Section 6.

4. Dealing with Complex Adtrees

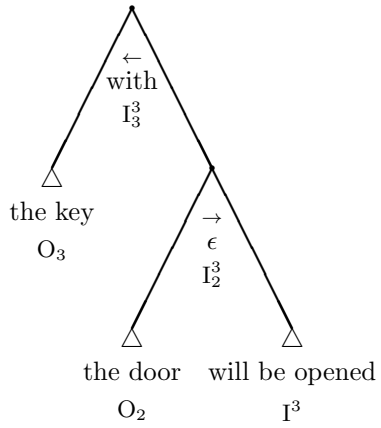
Simple adtrees are easily typeset using the previously described commands. But, sooner or later, one has to typeset a complex adtree, which does not fit into the page, or whose branches overlap, no matter how we extend branches, or which looks awful because of a poor choice of dimensions. In all these cases, there are a number of strategies one may adopt.

The simplest and crudest one is to change the length of branches. In this respect, branches are drawn using the LaTeX `picture` environment, so the length of each branch is 25`\unitlength`.

The value of `\unitlength` may be changed globally, affecting the way all subsequent adtrees in a document are typeset. Also, the value may be changed inside a surrounding group, which affects only the adtree to be typeset, but not the rest of the document.

Finally, the value of `\unitlength` can be changed within the adtree, to affect the point where the change appears and all the subtrees. Notice how there is no way to directly change the value of `\unitlength` in a single node: we will return on this point later.

For example,



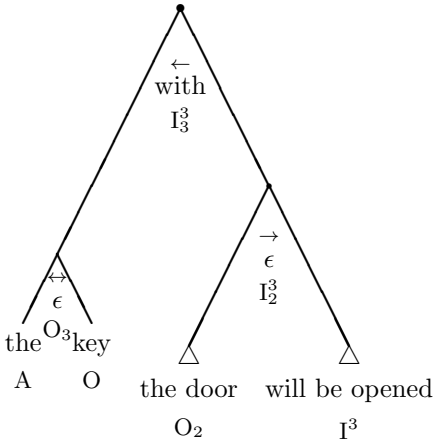
has been typeset adopting the strategy to affect all the branches in the adtree:

```
{\unitlength.3ex
  \ATl{with}{I^3_3}{\ATs{the key}{O_3}}
  {\ATre{I^3_2}
    {\ATs{the door}{O_2}}
    {\ATs{will be opened}{I^3}}} }
```

On the contrary,

```
\unitlength.31ex
\ATl{with}{I^3_3}
{\unitlength.12ex\ATxl{\ATbe{O_3}
  {\ATm{the}{A}}
  {\ATm{key}{O}}}}
{\unitlength.28ex\ATre{I^3_2}
  {\ATs{the door}{O_2}}
  {\ATs{will be opened}{I^3}}}
```

modifies the length of all the branches in the left subtree to `3ex` (since $3 = 25 \cdot 0.12$), and to `7ex` in the right subtree, while the outer branches have a length of `7.75ex`, obtaining



The adposition, and the related arrow and grammar character are moved accordingly, which is not pleasant in this example, although it is what one usually wants when changing the value of `\unitlength` in the whole adtree.

The right way to have the adposition always in the same place with respect to the branches in a node, is to fix a suitable `\unitlength` for the whole adtree, and to extend locally, just in the single node, its value. The exact technique is a bit complex, and it will be explained by an example later. Usually, there is no need to cope with these minor details, except for maniacs or for high precision works.

The length of branches can be changed locally to a single node, without recursively affecting the subtrees, as it happens when we change the value of `\unitlength` inside a node, like in the latest example. This can be achieved using the length variant of the adtree drawing commands. There are also an angle variant, and a variant affecting both the length and the angle. The complete list of complex adtree drawing commands, with all the variants is in Table 1

The names of these commands are easy to remember: the length variant has a trailing `L`, the angle variant a trailing `A`, and the combined variant a trailing `LA`.

The syntax of the length variant is the same as the base command except that there is an additional argument, the first one, which contains the value for the local `\unitlength`. In fact, the length we specify has exactly the same effect of changing `\unitlength` just before the command, but the change does not apply recursively.

It is worth remarking that, e.g., `\ATlL{\unitlength}` is equivalent to `\ATl`, so to reduce the length of branches to 90% of their current value, it suffices the write something like `\ATrL{.9\unitlength}`.

Similarly, the syntax of the angle variant requires an additional first argument which can be 60, 90, or 120, representing the angle between the branches: the default is 60 degrees. Specifying any other values resorts to the default.

The combined variant requires two additional arguments, the length and the angle, in this order, before any other parameter.

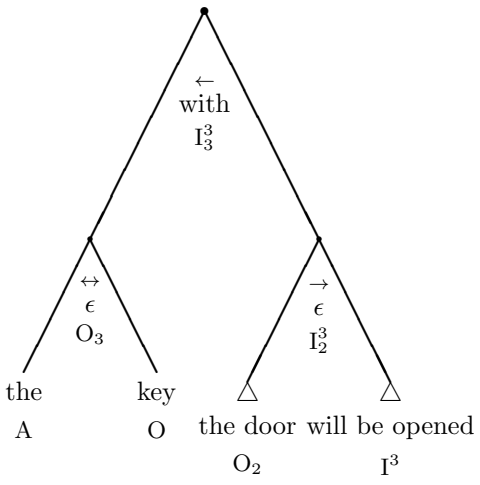
A hidden, very technical, feature of the package is that, whatever follows the length in the `L` argument gets evaluated inside the group which generates the

base command	length variant	angle variant	combined variant
	plain adtrees		
\ATl	\ATlL	\ATlA	\ATlLA
\ATr	\ATrL	\ATrA	\ATrLA
\ATb	\ATbL	\ATbA	\ATbLA
	extensions		
\ATxl	\ATxlL	\ATxlA	\ATxlLA
\ATxr	\ATxrL	\ATxrA	\ATxrLA
	adtrees with special adpositions		
\ATle	\ATleL	\ATleA	\ATleLA
\ATlc	\ATlcL	\ATlcA	\ATlcLA
\ATlmu	\ATlmuL	\ATlmuA	\ATlmuLA
\ATre	\ATreL	\ATreA	\ATreLA
\ATrc	\ATrcL	\ATrcA	\ATrcLA
\ATrmu	\ATrmuL	\ATrmuA	\ATrmuLA
\ATbe	\ATbeL	\ATbeA	\ATbeLA
\ATbc	\ATbcL	\ATbcA	\ATbcLA
\ATbmu	\ATbmuL	\ATbmuA	\ATbmuLA

TABLE 1. Complex adtrees drawing command.

branches in the node. So, for those well inside LaTeX mysteries, this feature can be used to tweak the behaviour of the graphical engine.

For example,



has been typeset by

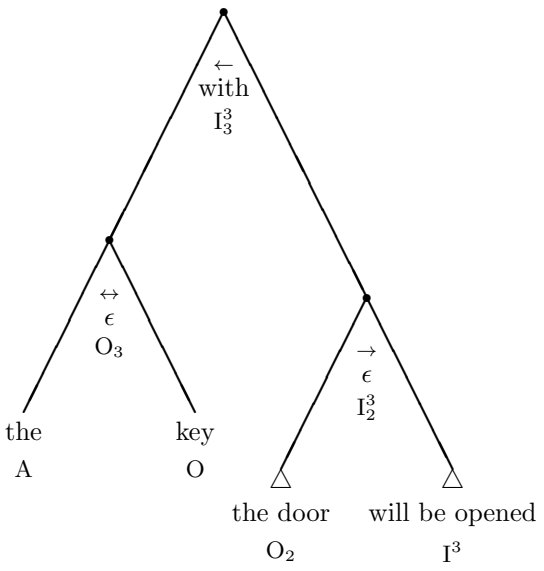
```
\ATlL{.4ex}{with}{I^3_3}
{\ATbe{O_3}
  {\ATm{the}{A}}
  {\ATm{key}{O}}}
{\ATreL{.25ex}{I^3_2}
```

```
{\ATs{the door}{O_2}}
{\ATs{will be opened}{I^3}}}
```

The *trick* to have the adposition always in the same place we speak before, is now easy to write, although slightly cumbersome. Specifically, the trick is to choose a reasonable global value for `\unitlength` which affects all the adtree, and then to locally extend the branches with the `\ATxlL` and `\ATxrL` commands whenever there is a need. The sensible choice for `\unitlength` minimises the number of extensions, and nicely place the adpositions. For example

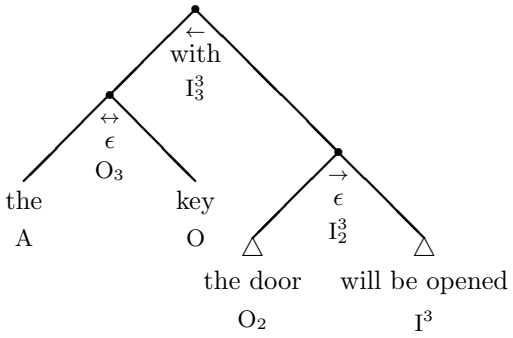
```
{\unitlength.32ex
  \ATl{with}{I^3_3}
  {\ATxlL{.1ex}
    {\ATbe{O_3}
      {\ATm{the}{A}}
      {\ATm{key}{O}}}}
  {\ATxrL{.2ex}
    {\ATre{I^3_2}
      {\ATs{the door}{O_2}}
      {\ATs{will be opened}{I^3}}}}}
```

generates



Another possibility to change the geometry of adtrees is to modify the angle between the branches. The commands to change the angle in the whole document, or within any group, affecting just the content of the group, are `\ATnormalangle`, which sets the angle to 60 degrees, the default, `\ATwideangle`, which sets the angle to 90 degrees, and `\ATextrawideangle`, which sets the angle to 120 degrees. These values are the only possible ones, also in the angle variant of the commands, as said before. Moreover, the value 60, 90, or 120 passed to the angle variant of a command (or to the combined variant) affects the node locally, as in the length variants' case.

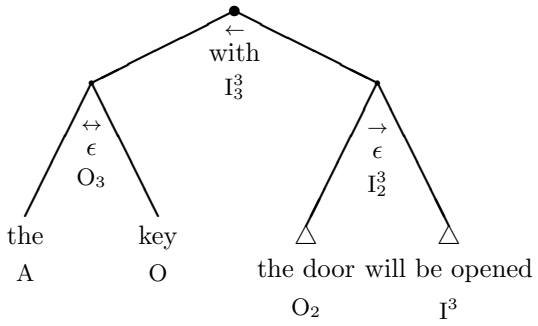
For example



has been typeset by

```
{\ATwideangle\unitlength.3ex
  \ATl{with}{I^3_3}
  {\ATbe{O_3}
    {\ATm{the}{A}}
    {\ATm{key}{O}}}
  {\ATxrL{.2ex}{\ATre{I^3_2}
    {\ATs{the door}{O_2}}
    {\ATs{will be opened}{I^3}}}} }
```

However, the following adtree



has been generated by

```
\ATlLA{.5ex}{120}{with}{I^3_3}
{\ATbe{O_3}
  {\ATm{the}{A}}
  {\ATm{key}{O}}}
{\ATreL{.25ex}{I^3_2}
  {\ATs{the door}{O_2}}
  {\ATs{will be opened}{I^3}}}
```

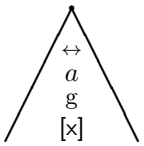
The placement of the adposition varies when the angle is changed: this has been done on purpose to cope with the reduction in height which is induced by a larger angle between the branches. Notice how the reduction in height may cause the adposition part of the adtree to fall beyond the base of the branches.

5. Internal Constructions

All the major parts of an adtree can be customised. This section describes a number of definitions which are used to control the spacing between the various parts of morpheme blocks, summaries, and adposition blocks. Also, the boxes which are deputed to provide a format to a single morpheme, a grammar character, or an attribute, are described.

The shape of the adposition block, which lies just below the root of a node in the adtree, is synthetically described as:

pre skip
t
first intermediate skip
a
second intermediate skip
$g + x$
post skip

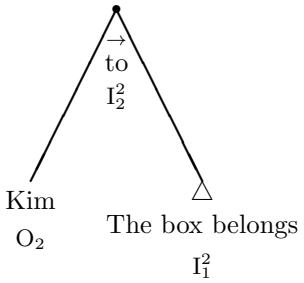


Here, t stands for the trajectory, one of $\leftarrow$, $\rightarrow$, or $\leftrightarrow$, as in the exemplifying adtree above; a is the adposition; g is the grammar character; x is an attribute. The various skips are LaTeX commands deputed to add vertical space between the components.

These commands, together with their definition, are:

```
\def\ATpreadpositionskip{\relax}
\def\ATfirstinteradpositionskip{\vskip.3ex}
\def\ATsecondinteradpositionskip{\vskip.3ex}
\def\ATpostadpositionskip{\relax}
```

They can be redefined globally, to affect all the adtrees following the redefinition, or locally, within a group which limits the scope. For example, the following adtree

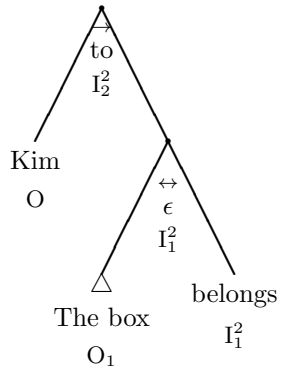


has been obtained by the code

```
{\def\ATpreadpositionskip{\vskip-2ex}
  \ATrL{.3ex}{to}{I_2^2}
  {\ATm{Kim}{O_2}}
  {\ATs{The box belongs}{I^2_1}}}
```

The `L` hidden feature of the package, already introduced, which injects code into the macro expansion, allows to redefine these commands within the scope of a single

node, as in the root of



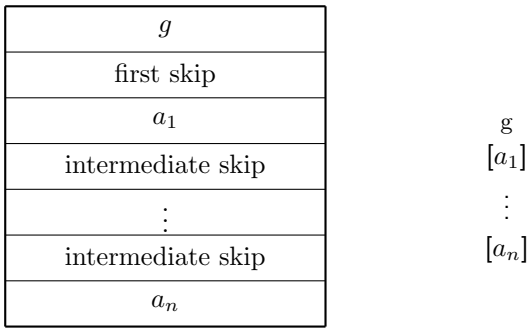
This effect is obtained by the following code

```
\ATrL{\unitlength\def\ATpreadpositionskip{\vskip-2ex}}
{to}{I_2^2}
{\ATm{Kim}{O_2}}
{\ATbe{I_1^2}{\ATs{The box}{O_1}}{\ATm{belongs}{I^2}}}
```

Attributes are managed so to form a unique block together with the grammar character. This block is constructed by the `\ATm` and `\ATs` commands, and inside the previously described adposition block.

It is important to remark that, although the size of text is normal in the case of the morpheme and summary blocks, and while it is small in the case of the adposition block, the values of skips are not affected by the size. Thus, we strongly suggest, if you want to change them, to express these values in the `ex` unit, which is the right unit to measure vertical space in terms of the *current* font size.

The structure of the attribute block is

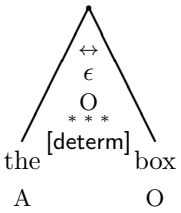


Here, g is the grammar character, while $a_1, \ldots, a_n$ are the various attributes. The commands defining the skips in this block are

```
\def\ATfirstattrskip{\vskip.7ex}
\def\ATinterattrskip{\vskip.5ex}
```

These definitions can be changed globally or locally, within the scope of a group. Also, in the case of attributes in the adposition block, they can be redefined within the single node, with the same technique as before.

It is worth remarking that any kind of material can be generated by these commands, not only vertical space. For example

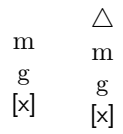


has been generated by

```
{\makeatletter
  \def\ATfirstattrskip{\vskip.5ex
    \hbox to\@ATlen{\hfil\tiny * * *\hfil}\nointerlineskip
    \vskip.3ex}
  \makeatother
  \ATbe{O}[determ]
  {\ATm{the}{A}}
  {\ATm{box}{O}}}
```

Finally, morpheme and summary blocks are as follows:

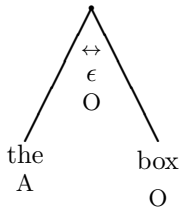
△, if a summary
pre skip
m
intermediate skip
$g + x$
post skip



Here, m is the morpheme or expression, and g is the grammar character together with its attributes x. The commands defining the skips in this block are

```
\def\ATpremorphemeskip{\vskip.5ex}
\def\ATintermorphemeskip{\vskip1ex}
\def\ATpostmorphemeskip{\relax}
```

Again, these definitions can be modified globally, or locally, within a group that limits the scope of the change. It is worth remarking that nesting an `\ATm` or `\ATs` command inside a complex adtree provides a natural group. For example



has been typeset by

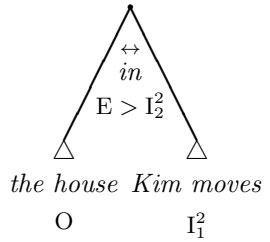
```
\ATbe{O}
{\def\ATpremorphemeskip{\vskip0ex}
  \def\ATintermorphemeskip{\vskip.2ex}
```

```
  \ATm{the}{A}}
{\ATm{box}{O}}
```

Actually, the face of morphemes, adpositions, and expressions in summaries is controlled by the following command:

```
\def\ATMorphemeBox#1{#1\strut}
```

The `\strut` forces all the instances to have the same minimal height and depth. So, to typeset all the morphemes and summaries in italic, as in



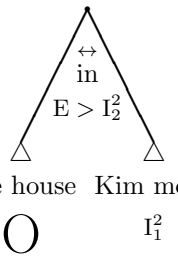
the following code suffices

```
{\def\ATMorphemeBox#1{\textit{#1}\strut}
  \ATb{in}{E>I_2^2}
  {\ATs{the house}{O}}
  {\ATs{Kim moves}{I^2_1}} }
```

Similarly, the face of grammar characters, wherever they appear, is controlled by the command

```
\def\ATGrammarCharacterBox#1{$\mathrm{#1}$}
```

The same scoping rules apply as in the case of `\ATMorphemeBox`. So, for example, we can make the grammar character of the left subtree to be `\Huge`, as in



by typing the following code

```
\ATb{in}{E>I_2^2}
{\def\ATGrammarCharacterBox#1{\Huge$\mathrm{#1}$}
  \ATs{the house}{O}}
{\ATs{Kim moves}{I_1^2}}
```

Also attributes are typeset according to a command:

```
\def\ATAttributeBox#1{\textsf{[#1]}}
```

So, for example, the illustrating attribute block on the previous page has been generated by the following code

```
{\def\ATAttributeBox#1{\strut\textsf{#1}}
  \raisebox{20ex}{\ATm{}{g}[\lbrack$a_1$\rbrack]
    [\vdots]
```

```
[\lbrack$a_n$\rbrack]} }
```

Finally, the symbol denoting a summary can be customised by changing

```
\def\ATSummarySymbol{$\triangle$}
```

It is important to remark that the top vertex of the triangle lies in the middle of the top of the $\triangle$ character, providing a natural point to join with the branches of adtrees. This fact should be taken in account when changing the definition.

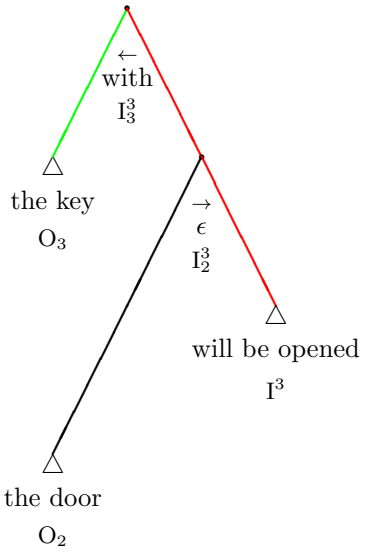
6. Hints and Tricks

There are a few last features which can be exploited in the `adtrees` package. These are the low-level commands to draw the branches:

```
\def\ATleftbranch#1#2{\line(#1,#2)}
\def\ATrightbranch#1#2{\line(#1,#2)}
\def\ATcircle{\circle*}
```

The `\ATleftbranch` and `\ATrightbranch` commands draw the left and the right branch of an adtree, respectively. Also, they draw the extensions, when present. They can be redefined globally, locally within the scope of a group, or locally within a node, using the implicit scope of the length variant, as previously described by the trick to equalise the position of adposition blocks.

For example,



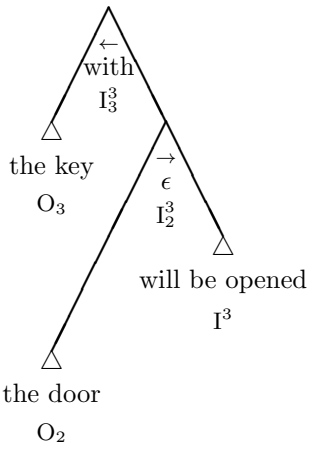
is obtained by making all the right branches red, and the topmost left branch green:

```
{\def\ATrightbranch#1#2{\color{red}\line(#1,#2)}
  \ATlL{\unitlength
    \def\ATleftbranch##1##2{\color{green}\line(##1,##2)}}
  {with}{I^3_3}
  {\ATs{the key}{O_3}}
  {\ATre{I^3_2}
    {\ATxl{\ATs{the door}{O_2}}}
    {\ATs{will be opened}{I^3}}} }
```

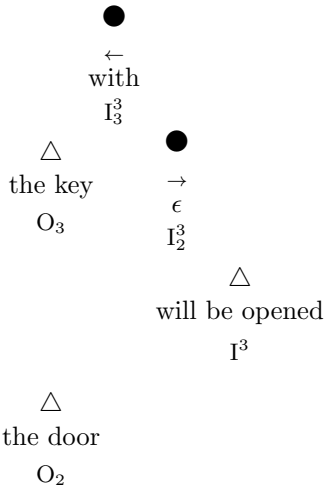
Also, the following definition

```
{\def\ATcircle#1{\relax}
  \ATl{with}{I^3_3}
  {\ATs{the key}{O_3}}
  {\ATre{I^3_2}
    {\ATxl{\ATs{the door}{O_2}}}
    {\ATs{will be opened}{I^3}}} }
```

suppresses the points in the whole adtree



As a rather extreme and useless example, one can typeset the previous adtree without showing the branches and making the points really huge

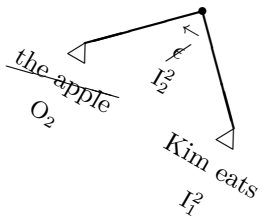


by the following code

```
{\def\ATcircle#1{\circle*{8}}
  \def\ATleftbranch#1#2#3{\relax}
  \def\ATrightbranch#1#2#3{\relax}
  \unitlength.22ex
  \ATl{with}{I^3_3}
  {\ATs{the key}{O_3}}
  {\ATre{I^3_2}
    {\ATxl{\ATs{the door}{O_2}}}
    {\ATs{will be opened}{I^3}}} }
```

Another important aspect of the package is that every adtree lies in a TeX box. It means that all the standard commands to manipulate boxes are available.

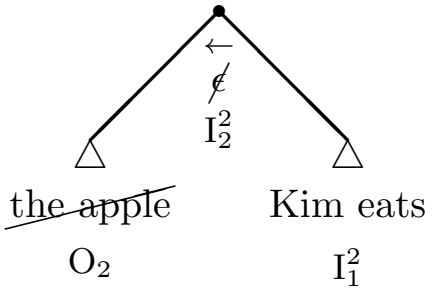
For example



has been generated by

```
\rotatebox{-30}
{\ATlcLA{.3ex}{90}{I_2^2}
  {\ATs{\cancel{the apple}}{O_2}}
  {\ATs{Kim eats}{I_1^2}}}
```

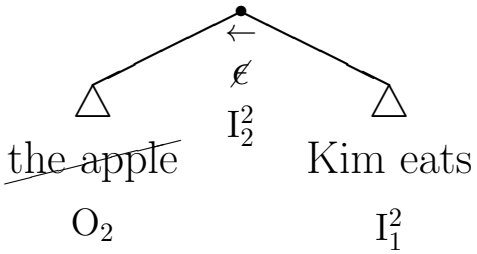
Sometimes, scaling a box is required:



has been generated by

```
\scalebox{1.5}
{\ATlcLA{.3ex}{90}{I_2^2}
  {\ATs{\cancel{the apple}}{O_2}}
  {\ATs{Kim eats}{I_1^2}}}
```

A final point is that, when the `\smaller` command is available, as in AMS-TEX, changing the font size automatically makes the adposition blocks to be `\smaller` than the size of the normal text, which is used to typeset the morpheme and summary blocks. For example,



which has been generated by

```
{\huge
  \ATlcLA{.3ex}{120}{I_2^2}
    {\ATs{\cancel{the apple}}{O_2}}
```

```
{\ATs{Kim eats}{I_1^2}}}
```

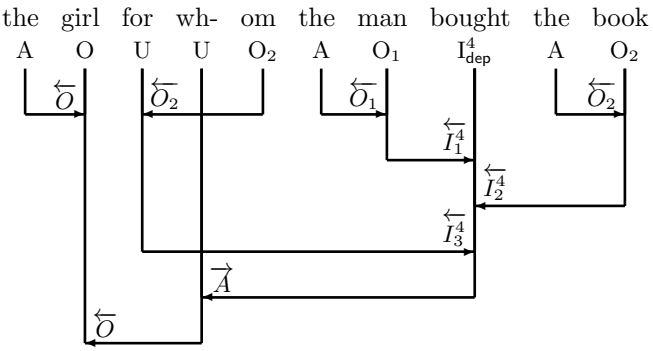
When the `smaller` command is not available, the size of the adposition block is fixed to be `\small`. This can be changed modifying the (internal) command `\@ATsmall`:

```
\makeatletter
\def\@ATsmall{\small}
\makeatother
```

Changing `\small` in the above code, modifies the size of the components of the adposition block. We suggest to avoid such a change, except in a local group which contains a complete adtree.

7. Path-like Adtrees

Adtrees are abstract representation of pieces of text. When the relation with an effective piece of text is relevant, a different representation is preferred: path-like adtrees. For example, the expression "the girl for whom the man bought the book" is rendered as



from the code

```
\begin{pathlikeadtree}
  the    [A]                  ( 1,1)[\overleftarrow{O}]     &
  girl   [O]                                                 &
  for    [U]                  ( 5,4)[\overleftarrow{I^4_3}] &
  wh-    [U]                  (-2,6)[\overleftarrow{O}]     &
  om     [O_2]                (-2,1)[\overleftarrow{O_2}]   &
  the    [A]                  ( 1,1)[\overleftarrow{O_1}]   &
  man    [O_1]                ( 1,2)[\overleftarrow{I^4_1}] &
  bought [I^4_{\mathsf{dep}}] (-4,5)[\overrightarrow{A}]    &
  the    [A]                  ( 1,1)[\overleftarrow{O_2}]   &
  book   [O_2]                (-2,3)[\overleftarrow{I^4_2}]
\end{pathlikeadtree}
```

To write a path-like adtree one encloses it into a `pathlikeadtree` environment. Then, each *cell* is written. Cells are separated by `&`. A cell is composed by a *morpheme*, a *grammar character*, and one or more *arcs*. The morpheme m comes first with his grammar character following in square brackets:

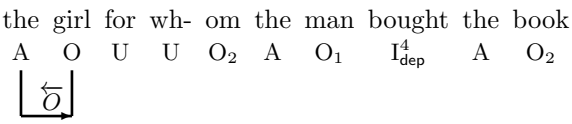
```
\begin{pathlikeadtree}
  morhpeme   [G]
\end{pathlikeadtree}
```

produces

morhpeme
G

An arc is specified as `(`t`,`h`)[`g`]` where t is the *target*, i.e., the node to which the arc is directed, h is the *height* of the arc, which **must** be a positive integer, and g is the *grammar character* of the composition of expressions which the arc intend to represent. It should be remarked that there **must** be no space between `)` and `[`.

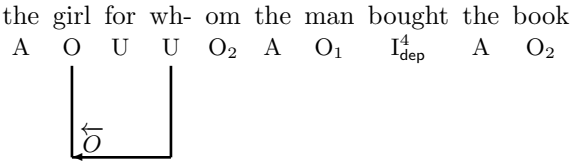
For example,



has been typeset by

```
\begin{pathlikeadtree}
  the[A](1,1)[\overleftarrow{O}] &
  girl[O]&for[U]&wh-[U]&om[O_2]&the[A]&man[O_1]&
  bought[I^4_{\mathsf{dep}}]&the[A]&book[O_2]
\end{pathlikeadtree}
```

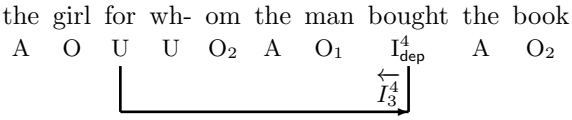
The target uses a *relative* specification: in the example, it is one step forward with respect to the node it has been written into. Of course, going backwards is possible, by using a negative value for the target:




```
\begin{pathlikeadtree}
  the[A]&girl[O]&for[U]&
  wh-[U] (-2,2)[\overleftarrow{O}] &
  om[O_2]&the[A]&man[O_1]&bought[I^4_{\mathsf{dep}}]&
  the[A]&book[O_2]
\end{pathlikeadtree}
```

However, it is possible to specify an *absolute* position prepending a `!` to the target:

```
\begin{pathlikeadtree}
  the[A]&girl[O]&
  for[U] (!8,1)[\overleftarrow{I^4_3}] &
  wh-[U]&om[O_2]&the[A]&man[O_1]&
  bought[I^4_{\mathsf{dep}}]&the[A]&book[O_2]
\end{pathlikeadtree}
```



The appearance of a path-like adtree is controlled by a number of parameters which the user may customise:

- `\ATpathinterskip` (default: `.5em`) is the distance between two cells;
- `\ATpathunitlength` (default: `4ex`) corresponds to the length of a line of height 1;
- `\ATpicskip` (default: `.2ex`) is the distance between the upper text and the arcs below;
- `\ATpathlinethickness` (default: `.1em`) specifies the thickness of the arcs;

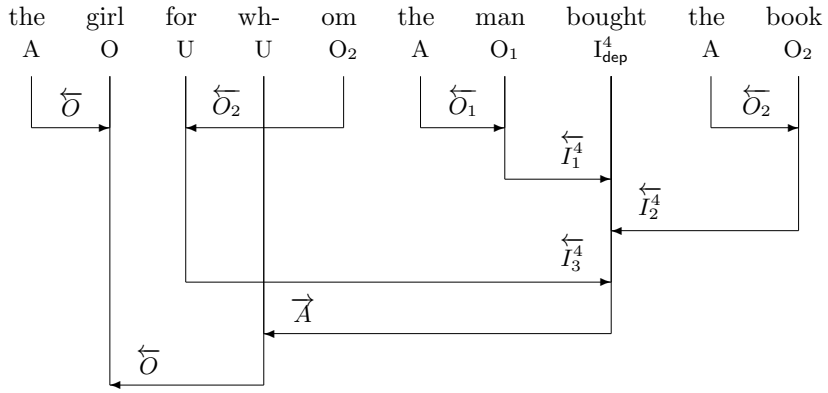
- `\ATpathlabelhspace` (default: `.3em`) defines the horizontal space between the grammar character in an arc and the vertical line of the arc to which it is closer;
- `\ATpathlabelvspace` (default: `1ex`) defines the vertical space between the baseline of the grammar character in an arc and the horizontal line of the arc.

Finally, the way in which the grammar characters are written is customisable by changing their default boxes. They are defined as

```
\def\ATnGCBox#1{\@ATsmall$\mathrm{#1}$}
\def\ATlGCBox#1{\@ATsmall$\mathrm{#1}$}
```

with `\ATnGCBox` controlling the rendering of the part below each morpheme, and `\ATlGCBox` controlling the labels of the arcs.

As an example,



has been typeset by

```
\ATpathinterskip1.2em
\ATpathunitlength4.5ex
\ATpicskip1ex
\ATpathlinethickness.2pt
\ATpathlabelhspace1em
\ATpathlabelvspace2ex
\begin{pathlikeadtree}
  the    [A]                  ( 1,1)[\overleftarrow{O}]     &
  girl   [O]                                                 &
  for    [U]                  ( 5,4)[\overleftarrow{I^4_3}] &
  wh-    [U]                  (-2,6)[\overleftarrow{O}]     &
  om     [O_2]                (-2,1)[\overleftarrow{O_2}]   &
  the    [A]                  ( 1,1)[\overleftarrow{O_1}]   &
  man    [O_1]                ( 1,2)[\overleftarrow{I^4_1}] &
  bought [I^4_{\mathsf{dep}}] (-4,5)[\overrightarrow{A}]    &
  the    [A]                  ( 1,1)[\overleftarrow{O_2}]   &
  book   [O_2]                (-2,3)[\overleftarrow{I^4_2}]
\end{pathlikeadtree}
```

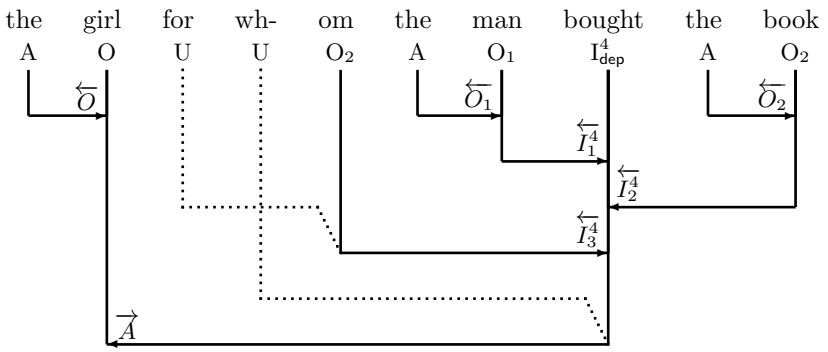
Path-like adtrees as explained so far are useful to represent adtrees having ϵ-only adpositions. Since every adtree, up to a syntactical transformation, can be

represented in this way, although losing some pieces of structural content, path-like adtrees drawn using the previously explained features are completely general.

However, there is less elegant way of writing general adtrees in the path-like form, in which adpositions are directly referred to in the arcs.

In short, the arc syntax is generalised to `(t,h)[g]<a>` where the a specifies the adposition, either using the relative or the absolute `!` syntax. Of course, when there is no explicit adpostion to refer, i.e., the adposition is ϵ, the previous syntax, without the parameter in angle brackets, is used.

As an illustration,



has been typeset by

```
\begin{displaymath}
  \ATpathinterskip1.2em
  \begin{pathlikeadtree}
    the    [A]                  ( 1,1)[\overleftarrow{O}]          &
    girl   [O]                                                     &
    for    [U]                                                     &
    wh-    [U]                                                     &
    om     [O_2]                ( 3,4)[\overleftarrow{I^4_3}]<!3> &
    the    [A]                  ( 1,1)[\overleftarrow{O_1}]        &
    man    [O_1]                ( 1,2)[\overleftarrow{I^4_1}]      &
    bought [I^4_{\mathsf{dep}}] (-6,6)[\overrightarrow{A}]<-4>     &
    the    [A]                  ( 1,1)[\overleftarrow{O_2}]        &
    book   [O_2]                (-2,3)[\overleftarrow{I^4_2}]
  \end{pathlikeadtree}
\end{displaymath}
```

There are a few hints and tricks about path-like adtrees: first, as in the case of adtrees, the `pathlikeadtree` environment produces a TEX box, thus it can be manipulated by, e.g., rotating, scaling, etc.

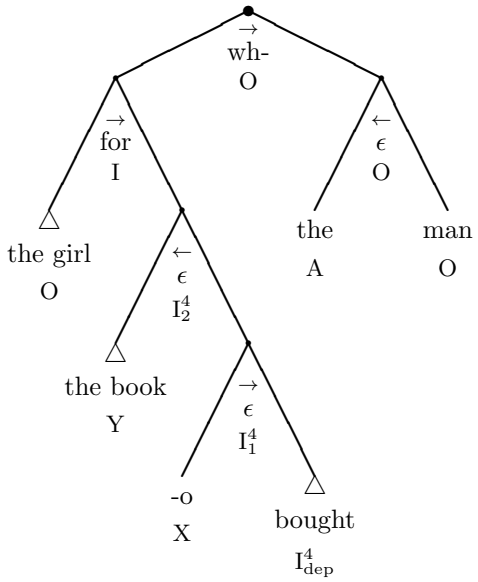
Second, while the syntax inside the `pathlikeadtree` is quite rigid, it is possible to locally use commands by putting them in the morpheme of a cell. The effects of this commands influence the cell from the point they are issued, but also all the subsequent cells. Beware that using commands with side effects inside a `pathlikeadtree` environment is an hack, which requires a deep understanding of how the code of the package operates.

The third way to radically affect the appearance of a path-like adtree is to redefine `\ATpathpichook`, which is `\def`ined to be empty by default. This command is issued

just before each arc is drawn. To code sophisticated manipulations of the drawing process requires to understand how the drawing engine works. However, for simple manipulations, it suffices to know that `\@ATpa` contains the index of the source cell of an arc, `\@ATpb` the index of the target cell, `\@ATpc` the height, `\@ATsa` is a box register containing the already formatted grammar character of an arc, and `\@ATpe` is either 0 or the index of the adposition cell. Needless to say, using this feature is for advanced users only.

8. Tabular Adtrees

Sometimes it is useful to have a purely textual representation of an adtree, which is more compact and fits, e.g., in a double column format. For example



can be rendered as

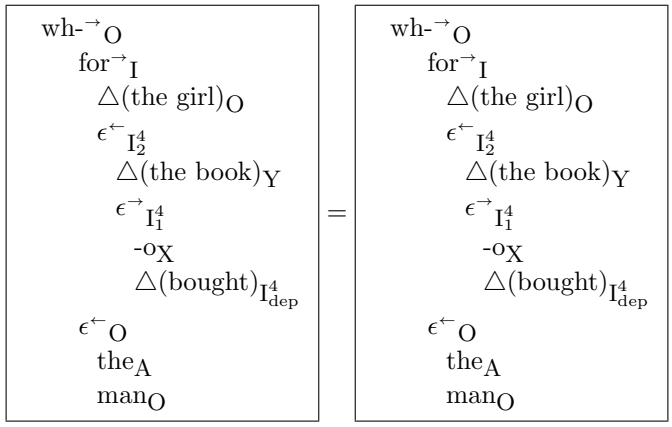
wh-$^{\rightarrow}$O
  for$^{\rightarrow}$I
    △(the girl)$_{O}$
    $\epsilon^{\leftarrow}_{I_2^4}$
      △(the book)$_{Y}$
      $\epsilon^{\rightarrow}_{I_1^4}$
        -o$_{X}$
        △(bought)$_{I^4_{dep}}$
  $\epsilon^{\leftarrow}$O
    the$_{A}$
    man$_{O}$

Obtaining this effect is as simple as enclosing the adtree into a `ATtabulardisplay` environment. The previous example has been typeset by

```
\begin{ATtabulardisplay}
  \ATrA{120}{wh-}{O}
  {\ATr{for}{I}
    {\ATs{the girl}{O}}
    {\ATle{I^4_2}
      {\ATs{the book}{Y}}
      {\ATre{I^4_1}
        {\ATm{-o}{X}}
        {\ATs{bought}{I^4_{\mathrm{dep}}}}}}}}
  {\ATle{O}{\ATm{the}{A}}{\ATm{man}{O}}}
\end{ATtabulardisplay}
```

The environment uses the `\ATtabskip` command to declare the amount of space to use for indenting the sub-trees. The default is `\def\ATtabskip{\hspace*{1em}}`. In addition, the whole environment is moved on the right by `\ATtabindent`, with default value `\def\ATtabindent{\hspace*{2em}}`.

Alternatively, the environment `ATtabular` allows to typeset the adtree in the same way but inside a box, which can then be manipulated as one pleases:




```
\fbox{$\vcenter{
    \begin{ATtabular}
      \ATrLA{2\unitlength}{120}{wh-}{O}
      {\ATr{for}{I}
        {\ATs{the girl}{O}}
        {\ATle{I^4_2}
          {\ATs{the book}{Y}}
          {\ATre{I^4_1}
            {\ATm{-o}{X}}
            {\ATs{bought}{I^4_{\mathrm{dep}}}}}}}}
      {\ATle{O}
        {\ATm{the}{A}}
        {\ATm{man}{O}}}
    \end{ATtabular}}$} =
\fbox{$\vcenter{
    \begin{ATtabular}
      \ATrLA{2\unitlength}{120}{wh-}{O}
      {\ATr{for}{I}
        {\ATs{the girl}{O}}
        {\ATle{I^4_2}
          {\ATs{the book}{Y}}
          {\ATre{I^4_1}
            {\ATm{-o}{X}}
            {\ATs{bought}{I^4_{\mathrm{dep}}}}}}}}
      {\ATle{O}
        {\ATm{the}{A}}
        {\ATm{man}{O}}}
    \end{ATtabular}}$}
```

The various pieces of the adtree are rendered according to the following definitions, which could be modified on need, see also the next section, keeping in mind that they are not for the casual user.

```
\def\ATtabularadpositionblock#1#2#3{%
   \hbox{\ATtabindent\AT@loop@tab%
    {#2}\textsuperscript{#1}\textsubscript{#3}}}
\def\ATtabularmorphemeblock#1#2{
  \hbox{\ATtabindent\AT@loop@tab%
    {#1}\textsubscript{#2}}}
\def\ATtabularsummaryblock#1#2{
  \hbox{\ATtabindent\AT@loop@tab%
    \mbox{$\triangle($}%
    \mbox{#1}\mbox{$)$}\textsubscript{#2}}}
\def\ATtabularfirstattribute{\ATlinearfirstattribute}
\def\ATtabularnextattribute{\ATlinearnextattribute}
\def\ATtabularsubtrees#1#2{%
  \advance\@AT@tabcount1%
  \vbox{#1\relax#2}%
  \advance\@AT@tabcount-1}
```

Finally the `\ATTabular` command can be used to switch from the normal graphical rendering of adtrees to the tabular one. This is a low-level command which has been left accessible to end-users willing to write their own environments when `ATtabular` is not enough.

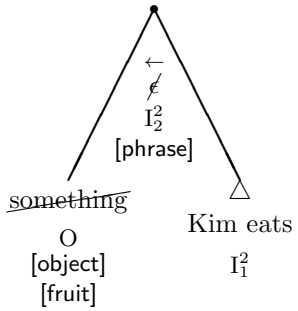
9. Linear Adtrees

Adtrees can be rendered in a linear format. This is not graphically pleasant, so we discourage users to adopt it.

Nevertheless, it may be useful to have a very compact representation of adtrees. This can be automatically obtained by prepending the `\ATlinearise` command to an adtree. For example

$$\not{\leftarrow}^{\mathrm{I}_2^2:\textsf{[phrase]}}(\cancel{\text{something}}^{\mathrm{O}:\textsf{[object]};\textsf{[fruit]}}, \triangle(\text{Kim eats})_{\mathrm{I}_1^2})$$

is the same as



which shows how all the features of adtrees are rendered in the linear format.

The linear presentation has been generated by

```
\ATlinearise{\ATlcL{.3ex}{I_2^2}[phrase]
  {\ATm{\cancel{something}}{O}[object][fruit]}
  {\ATs{Kim eats}{I_1^2}}}
```

which differs from the code to draw the graphics presentation just for `\ATlinearise`.

Declaring `\ATLinear` in some point of the text makes linear all the adtrees from that point on. To reestablish the standard behaviour of graphical adtrees, one issues the command `\ATNormal`.

Linear adtrees are composed using the following commands

```
\def\ATlinearadpositionblock#1#2#3%
  {{#2}\textsuperscript{#1}\textsubscript{#3}}
\def\ATlinearfirstattribute#1{{#1}:}
\def\ATlinearnextattribute#1{{#1};}
\def\ATlinearsubtrees#1#2{(#1,\linebreak[0] #2)}
\def\ATlinearmorphemeblock#1#2{{#1}\textsubscript{#2}}
\def\ATlinearsummaryblock#1#2{\mbox{$\triangle($}%
    \mbox{#1}\mbox{$)$}\textsubscript{#2}}
```

Their meaning should be intuitive after the explanations in Section 5.

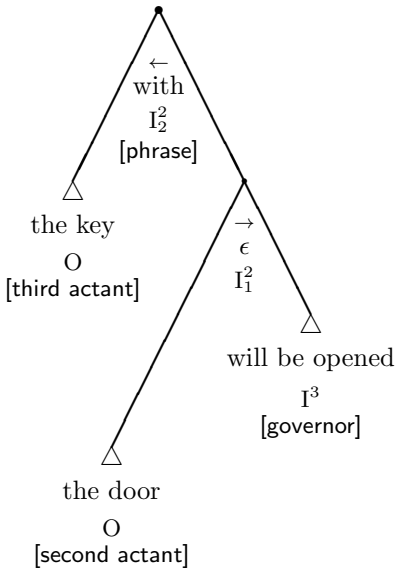
Also, it must be remarked that `\ATMorphemeBox`, `\ATGrammarCharacterBox`, and `\ATAttributeBox` are still used to write the corresponding elements.

Linear adtrees are useful when it is required to perform some computation on an adtree inside LaTeX, e.g., when trying to write a macro which manipulates an argument which is an adtree.

The reason is double: first, the linear format require far less computation than the graphical rendering; and, second, the building macros receive inputs which have

not been heavily preprocessed to prepare their graphical rendering, thus closer to what the user has written in the source code.

For example, to list all the morpheme attributes occurring inside



we can execute

```
{ \def\ATlinearadpositionblock#1#2#3{\relax}
  \def\ATlinearfirstattribute#1#2{#2}
  \def\ATlinearnextattribute#1#2{[{#2}]}
  \def\ATlinearsubtrees{\relax}
  \def\ATlinearmorphemeblock#1{\relax}
  \def\ATlinearsummaryblock#1{\relax}
  \ATlinearise{
    \ATl{with}{I^2_2}[phrase]
      {\ATs{the key}{O}[third actant]}
      {\ATre{I^2_1}
        {\ATxl{\ATs{the door}{O}[second actant]}}
        {\ATs{will be opened}{I^2}[governor]}}} }
```

which produces

[third actant][second actant][governor]

Needless to say, such computations require a real TeX magician, who is able to fully understand the code of the package! Nevertheless, this opens the door to LaTeX macros that operate on adtrees, treating them like data structures.

(Marco Benini) Dipartimento di Scienza e Alta Tecnologia, Università degli Studi dell'Insubria, via Valleggio 11, I-22100 Como, Italy

Email address: marco.benini@uninsubria.it

URL: http://marcobenini.wordpress.com

(Federico Gobbo) Faculty of Humanities, University of Amsterdam, Spuistraat 134,NL-1012VT Amsterdam, The Netherlands

Email address: F.Gobbo@uva.nl

URL: http://federicogobbo.name