The `synttree` package for typesetting syntactic trees.*

Matijs van Zuijlen†

June 28, 2009

Abstract

The `synttree` package provides a simple way to typeset syntactic trees as used in Chomsky's Generative Grammar.

Contents

*Package version 1.4.2

†e-mail: `mvz@xs4all`; web: `http://www.matijs.net/`

1 Introduction

The `synttree` package provides a macro for creating syntactic tree structures such as those used in Noam Chomsky's Generative Grammar. It is designed to create a tree that looks nice, without manual tweaking, and with as little use of 'special effects', such as PostScript, as possible.

Since the application is very specific, there is no need for a very complex set of options: `synttree` is designed to do one thing, and do it well.

Using `synttree`, you can create trees of any depth, and an unlimited number of branches (up to the memory limit of your version of TEX, of course).

2 Usage

2.1 Drawing Options

The current implementation can use either emTEX `\special`'s or LATEX 2_ε's `\qbezier` macro to draw the lines in the tree, although the latter option is probably technically not very nice. Still, the behavior defaults to the LATEX 2_ε version.

In order to switch between the two modes, the following options are supported:

specials Use emTEX special commands.

nospecials Don't use emTEX special commands. This is the default.

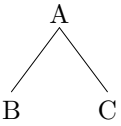
2.2 Defined Macros

`\synttree` The main macro defined by this package is the `\synttree` macro. It takes as arguments first an optional parameter indicating the maximum depth of the tree, and second a parameter describing the structure of the tree, delimited by square brackets (`[` and `]`).

A tree consists of the parent node's label, followed by the child trees. In principle, an unlimited number of children can be specified, but to signal runaway trees, a warning will be issued when the number is more than ten. Each of the child trees is again surrounded by square brackets. For example, the line

```
\synttree[A[B][C]]
```

creates the following tree:



Note that leading and trailing spaces in the label will be ignored, so the following has the same result:

```
\synttree[ A [ B ][ C ]]
```

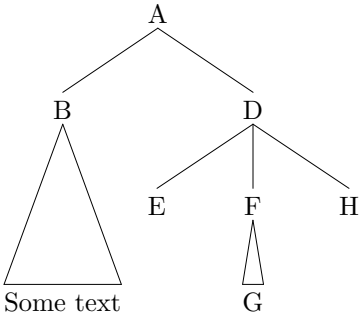
Before the label, a parameter may be added. This is specified by appending a period (`.`) to the opening bracket, and appending a letter that modifies the appearance of the tree. Append a `t` to create a triangle instead of a line going from the label to the level above it, and a `b` to specify that the node has to appear

at the bottom of the tree, on a line with the lowest leaves. To use `b`, a maximum tree depth has to be specified. To combine the two features, use `x`.

Using the optional parameters, the code

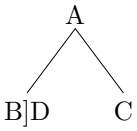
```
\synttree{4}
[A
  [B [.x Some text] ]
  [D [E] [F[.t G]] [H] ]
]
```

creates this tree:



If brackets are included in the label, surround them (or the entire label) by braces:

```
\synttree[A[{B]D}][C]]
```



`\branchheight` The other defined macros affect the look and sizing of the trees. The distance between levels can be adjusted by using `\branchheight`. It takes a length as its argument, and sets the distance to that length. The default distance is half an inch. This value is the distance between the baselines of the labels, so setting it to zero will cause everything to be on one line. The minimum horizontal separation between edges of two children is set by `\childsidesep`. The default is `1em`. The length `\childattachsep` indicates the minimum distance between the attachment points of two children. Here, the default is `0.5in`. Finally, the balancing of the triangles can be adjusted by using `\trianglebalance`. It takes a number from 0 to 100, indicating the percentage of the triangle on the right side of its attachment point to the parent. The default is 50, resulting in isosceles triangles.

`\childsidesep`

`\childattachsep`

`\trianglebalance`

3 Wish List and Bugs

It is important to note that this package may not prevent you from doing certain things 'wrong'. For example, triangles on non-leaf nodes are not forbidden, but may not give the desired result.

Sometimes, the label of a parent may be a phrase, and it may be desirable to attach the children to a particular element in that phrase (e.g., to elaborate

on the structure of that particular element). This is currently not possible: The children are always attached to the center of the label. Support for this may be added sooner or later.

4 Related Packages

A more complex package is the TreeTeX system[2]. This system, however, produces nodes consisting of a node symbol *and* a label, whereas in syntactic trees the label *is* the node symbol. Additionally, the method of specifying the tree structure itself makes the source code hard to read.

The `xyling` package[4] is a very powerful package, allowing many things that `synttree` will not do. However, the nodes have to be laid out meticulously in an array. Once that's done, however, the result looks very good.

The "LaTeX for Linguists" website[1] lists other tree-drawing packages, with comments on their usefulness.

5 Author and License

6 Implementation

6.1 Utility macros

Very useful for putting stuff after a `\fi` (as the name implies) [3].

```
\def\@AfterFi#1\fi{\fi#1}
\def\@AfterElseFi#1\else#2\fi{\fi#1}
```

6.2 Drawing commands

These command are used to draw the lines between the nodes. There are two versions: one that uses the LaTeX 2_ε `\qbezier` command, and one that is an adaptation of the unsupported `eepic` package, and uses specials. Currently, the package just uses the LaTeX 2_ε version.

First, the line drawing macros themselves. These simply draw a line between the two points given. Arguments are counters.

```
\def\MTr@latexdrawline(#1,#2)(#3,#4){%
  {%
    \count0=#1 \advance\count0 by #3 \divide\count0 2
    \count1=#2 \advance\count1 by #4 \divide\count1 2
    \qbezier(#1,#2)(\count0,\count1)(#3,#4)%
  }%
}
\def\MTr@etexdrawline(#1,#2)(#3,#4){%
```

```
  {%
    \count0=\@wholewidth \divide\count0 by 4736
    \special{pn \the\count0}%
    \count0= #1\advance \count0 2368 \divide \count0 4736
    \count1=-#2\advance \count1 -2368 \divide \count1 4736
    \special{pa \the\count0 \space \the\count1}%
    \count0= #3\advance \count0 2368 \divide \count0 4736
    \count1=-#4\advance \count1 -2368 \divide \count1 4736
    \special{pa \the\count0 \space \the\count1}%
    \special{fp}%
  }%
}
```

Options to select either version:

```
\DeclareOption{specials}{
  \let\MTr@drawline\MTr@etexdrawline%
}
\DeclareOption{nospecials}{
  \let\MTr@drawline\MTr@latexdrawline%
}
\ExecuteOptions{nospecials}%
\ProcessOptions%
```

6.3 Definitions

Some counters etc. are defined: The current level, the number of children the current node has, the maximum level specified, also, the current `branchmult`, and whether the current node should be typeset with a triangle. We need a separate counter to hold the current `branchmult`, because we calculate it before parsing the children, which will each set `branchmult` globally to save it into the child registers.

```
\newcount\MTr@level
\newcount\MTr@numchildren
\newcount\MTr@maxlevel
\newcount\MTr@mybranchmult
\newif\ifMTr@mytriangle
```

A boolean to specify whether read tokens that are not `[` or `]` should be used as a label for a node, or just ignored.

```
\newif\ifMTr@uselabel
```

Two `saveboxes`, one for the label, one to put the child in while it is being defined.

```
\newbox\MTr@labelbox
\newbox\MTr@treebox
```

6.4 Storage for information on (sub)trees

Define storage space to store contents of and information about a tree's children. Distances are measured from the child's label's virtual attachment point to the line to its parent — in reality, for aesthetic reasons, the line is drawn up to a point that lies slightly higher. Drawing a box around the whole child tree, we find the distance from the left side to the central point, v, the distance from the right side, w. Then, we have the height (equal to the height of the label), and

the depth (the height of the box minus the height of the label). There are also two external dimensions: the vertical distance from the parent label, y, and the horizontal distance from the parent tree's box, x. Usually, a child node is typeset one level below its parent. However, we may set it lower, e.g., to set the child on the bottom level. `branchmult` indicates how many levels below the parent the child should be typeset.

```
\def\MTr@makechildcounter#1{%
    \expandafter\newcount\csname MTr@child#1\endcsname%
}
\def\MTr@makechildstoreage#1{%
    \expandafter\newsavebox\csname MTr@child#1box\endcsname%
    \MTr@makechildcounter{#1x}%
    \MTr@makechildcounter{#1y}%
    \MTr@makechildcounter{#1v}%
    \MTr@makechildcounter{#1w}%
    \MTr@makechildcounter{#1height}%
    \MTr@makechildcounter{#1depth}%
    \MTr@makechildcounter{#1branchmult}%
    \MTr@makechildcounter{#1picheight}%
    \MTr@makechildcounter{#1triangle}%
}
```

`\MTr@childparam` We also need an easy way to retrieve each child's parameters. This way, we don't have to use the `\csname` construct all the time.

```
\def\MTr@childparam#1#2{\csname MTr@child#1#2\endcsname}
```

Store info for current subtree: v, w (x and y are external, thus not important), height, depth, `branchmult`, triangle status.

```
\newcount\MTr@treev
\newcount\MTr@treew
\newcount\MTr@treeheight
\newcount\MTr@treedepth
\newcount\MTr@branchmult
\MTr@branchmult 1
\newif\ifMTr@triangle
```

Next, the depth, height and half the width of the label.

```
\newcount\MTr@labeldepth
\newcount\MTr@labelheight
\newcount\MTr@labelhalfwidth
```

When drawing, we need `morex`, to store extra shift to the right when drawing the children, and `parenty`, the vertical position of the point where the line from the parent to the child starts. Also, we need the width and height of the picture being drawn.

```
\newcount\MTr@morex
\newcount\MTr@parenty
\newcount\MTr@picwidth
\newcount\MTr@picheight
```

Finally, we need a temporary length, and four temporary counters.

```
\newlength{\MTr@templength}
\newcount\MTr@loopcnta
\newcount\MTr@tempcnta
```

```
\newcount\MTr@tempcntb
\newcount\MTr@tempcntc
```

6.5 Adjustable and other parameters

`\branchheight` The user can set up several parameters. First, `\branchheight` will set the distance between levels. Default value is half an inch.

```
\newcount\MTr@branchheight%
\newcommand{\branchheight}[1]{%
  \setlength{\MTr@templength}{#1}%
  \MTr@branchheight\MTr@templength%
}
\branchheight{.5in}%
```

`\trianglebalance` Next, `\trianglebalance` will set the balancing of the triangle. Default value is 50.

```
\newcount\MTr@trianglemultright%
\newcount\MTr@trianglemultleft%
\newcommand{\trianglebalance}[1]{%
  \MTr@trianglemultleft100%
  \MTr@trianglemultright#1%
  \advance\MTr@trianglemultleft-#1%
}
\trianglebalance{50}%
```

Distance between the labels and the lines.

```
\newcount\MTr@lineoffset
\setlength{\MTr@templength}{2pt}%
\MTr@lineoffset\MTr@templength%
```

Minimum label height.

```
\newlength{\MTr@minheight}
\setlength{\MTr@minheight}{8pt}%
```

`\childsidesep` Minimum separation between edges of two children.

```
\newcount\MTr@childsidesep
\newcommand{\childsidesep}[1]{%
  \setlength{\MTr@templength}{#1}%
  \MTr@childsidesep\MTr@templength%
  \ignorespaces%
}
\childsidesep{1em}
```

`\childattachsep` How far apart are the attachment points of two children?

```
\newcount\MTr@childattachsep
\newcommand{\childattachsep}[1]{%
    \setlength{\MTr@templength}{#1}%
    \MTr@childattachsep\MTr@templength%
    \ignorespaces%
}
\childattachsep{0.5in}
```

6.6 Main macro

`\synttree` `\synttree` is the main macro. If the user has not provided a maximum depth, set it to 0. There will be no messages concerning depth, except when the bottomlevel modifier is used. Control is passed on to `\MTr@synttree`

```
\def\synttree{%
    \@ifnextchar[{\MTr@synttree{0}}{\MTr@synttree}%]
}
```

`\MTr@synttree` This macro sets maximum depth, end sets picture coordinates to scaled points. Next, initial values of some variables are set, and the first real parsing macro, `\MTr@parserightbracket`, is called.

```
\def\MTr@synttree#1{%
  \MTr@maxlevel#1%
  \unitlength 1sp%
  \MTr@level=0%
  \MTr@numchildren=0%
  \MTr@uselabelfalse%
  \MTr@parserightbracket%
}
```

6.7 Parsing

Parsing will only work if at least one `]` is present.

`\MTr@parserightbracket` First, scan until the first occurrance of `]`. Argument `#1` will contain no `]`, but may contain some `[`, which are detected by `\MTr@parseleftbracket`. After `#1` has been processed, lower the nesting level. If we're back at 0, we're done. Otherwise, continue looking for `]`.

```
\def\MTr@parserightbracket#1]{%
  \MTr@parseleftbracket#1[:\END%
  \advance\MTr@level by -1%
  \MTr@dorightbracket%
  \ifnum\MTr@level=0%
    \unhbox\MTr@childibox{}%
  \else
    \@AfterFi{\MTr@parserightbracket}%
  \fi%
}
```

`\Mtr@parseleftbracket` Scan until the first occurrance of `[`. Argument `#1` will contain no `]` or `[`. `#2` may contain more `]`. Possibly, `#1` is a label. If `#2=:`, the `[` was placed by `\MTr@parserightbracket`, and we're done.

```
\def\MTr@parseleftbracket#1[#2\END{%
  \ifMTr@uselabel%
    \MTr@bottomnodefalse%
    \MTr@mytrianglefalse%
    \MTr@parsedot#1.: \END%
  \fi%
  \ifx:#2%
  \else%
    \MTr@doleftbracket%
    \advance\MTr@level by 1%
```

```
    \@AfterFi{\MTr@parseleftbracket#2\END}%
  \fi%
}
```

\Mtr@parsedot Parse the optional node argument.

```
\def\MTr@parsedot#1.#2 #3\END{%
  \ifx:#2%
    \setbox\MTr@labelbox\hbox{\ignorespaces#1\unskip}%
  \else
    \ifx#2b\MTr@bottomnodetrue\else%
    \ifx#2x\MTr@bottomnodetrue\MTr@mytriangletrue\else%
    \ifx#2t\MTr@mytriangletrue\else%
    \typeout{synttree Warning: unknown dot option #2 in tree}%
    \fi\fi\fi%
    \MTr@parsedot#3\END
  \fi
}
```

6.8 Action

The next macros implement the actions to be undertaken when encountering one of the square brackets.

\MTr@doleftbracket Upon encountering a `[`, we go one level deeper: Begin a group, and reset parameters.

```
\def\MTr@doleftbracket{%
  \bgroup%
  \MTr@numchildren=0%
  \MTr@uselabeltrue%
}
```

\MTr@dorightbracket Upon encountering a `]`, we typeset the current group's tree, and end the group. Until we encounter some brackets, we should not add tokens to any label. Save the just-typeset tree as the newest child in the parent group — now the current group.

```
\def\MTr@dorightbracket{%
  \MTr@maketreebox%
  \egroup%
  \MTr@uselabelfalse%
  \MTr@savecurrentchildbox%
}
```

\MTr@savecurrentchildbox Save the current picture, with all its data, in the box and registers for the next empty child.

```
\def\MTr@savecurrentchildbox{%
  \advance\MTr@numchildren by 1
  \ifnum\MTr@numchildren<1%
    \typeout{synttree internal warning: There is no child box to save.}%
  \else
    \ifnum\MTr@numchildren>10%
      \typeout{synttree warning: More than 10 child boxes.
      Can this be true?}%
    \fi
```

```
    \MTr@savechildbox{\romannumeral\MTr@numchildren}%
  \fi
}
```

`\MTr@savechildbox` Save current tree in a specific child box. Basically, we just copy from the tree registers to the child registers, except x is set to v initially. Used only in `\MTr@savecurrentchildbox`.

```
\def\MTr@savechildbox#1{%
  \expandafter
  \ifx\csname MTr@child#1box\endcsname\relax%
    \MTr@makechildstoreage{#1}%
  \fi%
  \setbox%
  \csname MTr@child#1box\endcsname%
  \hbox{\unhbox\MTr@treebox}%
  \csname MTr@child#1v\endcsname\MTr@treev%
  \csname MTr@child#1w\endcsname\MTr@treew%
  \csname MTr@child#1x\endcsname\MTr@treev%
  \csname MTr@child#1height\endcsname\MTr@treeheight%
  \csname MTr@child#1depth\endcsname\MTr@treedepth%
  \csname MTr@child#1branchmult\endcsname\MTr@branchmult%
  \ifMTr@triangle%
    \csname MTr@child#1triangle\endcsname 1%
  \else%
    \csname MTr@child#1triangle\endcsname 0%
  \fi
}
```

6.9 Bottom Nodes

For bottom nodes, we have to adapt the vertical position so that they become, indeed, bottom nodes. This is done by the macros `\MTr@bottomnodetrue` and `\MTr@bottomnodefalse`.

`\MTr@bottomnodetrue` The node is a bottom node. Calculate the difference between this level and the bottom level as passed to `\synttree`, and use this to determine how many levels the node has to be advanced vertically to get it at the correct position. In effect, the distance between two levels is multiplied by the "branch multiplication." For non-bottom nodes this is set to 1.

```
\def\MTr@bottomnodetrue{%
  \MTr@branchmult\MTr@maxlevel%
  \advance\MTr@branchmult-\MTr@level%
  \advance\MTr@branchmult 1%
  \ifnum\MTr@branchmult<1%
    \typeout{synttree Warning: Tree has more levels than indicated.}%
    \typeout{>> Indicated: \the\MTr@maxlevel.}%
    \typeout{>> Level now: \the\MTr@level.}%
    \MTr@branchmult1%
  \fi%
  \MTr@mybranchmult\MTr@branchmult%
}
```

`\MTr@bottomnodefalse` It's not a bottom node: Just set the branch multiplication to 1.

```
\def\MTr@bottomnodefalse{%
  \MTr@mybranchmult1%
}
```

6.10 Utility macros for the output routines

`\MTr@setverticalchilddimens` Set vertical dimensions for a given child: y and `picheight`.

```
\def\MTr@setverticalchilddimens#1{%
    \MTr@tempcnta-\MTr@branchheight%
    \multiply\MTr@tempcnta\MTr@childparam{#1}{branchmult}%
    \MTr@tempcntb-\MTr@tempcnta%
    \advance\MTr@tempcntb\csname MTr@child#1depth\endcsname%
    \advance\MTr@tempcnta-\MTr@labelheight%
    \advance\MTr@tempcnta\csname MTr@child#1height\endcsname%
    \csname MTr@child#1y\endcsname\MTr@tempcnta%
    \csname MTr@child#1picheight\endcsname\MTr@tempcntb%
}
```

`\MTr@adjustdistance` Adjust the distance between two children. For the pair of neighboring children, `tempcnta` is used temporarily to store what amounts to the distance between the central points of the two children (the left child's w, plus the right child's v). If it is below the value `childattachsep`, adjust values so that is equal to it. `tempcnta` stores the resulting value. The right child's x is now set to put it in the correct position w.r.t. the left child.

```
\def\MTr@adjustdistance#1#2{%
  \MTr@tempcnta\MTr@childparam{#1}{w}%
  \advance\MTr@tempcnta\MTr@childsidesep%
  \advance\MTr@tempcnta\csname MTr@child#2v\endcsname%
  \ifnum\MTr@tempcnta<\MTr@childattachsep%
    \MTr@tempcnta\MTr@childattachsep%
  \fi%
  \csname MTr@child#2x\endcsname\MTr@childparam{#1}{x}%
  \advance\csname MTr@child#2x\endcsname\MTr@tempcnta%
}
```

`\MTr@setparentdimens` Set some parameters for the parent node. Parameters are the leftmost and rightmost child label.

```
\def\MTr@setparentdimens#1#2{%
```

Calculate the subtree's v and w:

```
    \MTr@tempcnta\MTr@childparam{#2}{x}%
    \advance \MTr@tempcnta -\MTr@childparam{#1}{x}%
    \divide\MTr@tempcnta 2%
    \MTr@treev\MTr@tempcnta%
    \MTr@treew\MTr@treev%
    \advance \MTr@treev \csname MTr@child#1x\endcsname%
    \advance \MTr@treew \csname MTr@child#2w\endcsname%
```

Calculate `morex`: The distance the (first) subtree has to be shifted to the right to accomodate a large parent label size.

```
    \MTr@morex\MTr@labelhalfwidth%
    \advance\MTr@morex-\MTr@treev%
    \ifnum\MTr@morex<0\MTr@morex0\fi%
```

Large label sizes are also incorporated into w and x.

```
    \ifnum\MTr@treew<\MTr@labelhalfwidth
        \MTr@treew\MTr@labelhalfwidth
    \fi%
    \ifnum\MTr@treev<\MTr@labelhalfwidth
        \MTr@treev\MTr@labelhalfwidth
    \fi%
```

Picture width.

```
    \MTr@picwidth\MTr@treev%
    \advance\MTr@picwidth\MTr@treew%
}
```

`\MTr@setpictureparameters` Set some parameters for the picture.

```
\def\MTr@setpictureparameters{%
    \global\MTr@treedepth\MTr@picheight%
    \advance\MTr@picheight\MTr@labelheight%
    \global\MTr@treeheight\MTr@labelheight%
    \MTr@parenty-\MTr@labelheight%
    \advance\MTr@parenty-\MTr@labeldepth%
    \advance\MTr@parenty-\MTr@lineoffset%
    \global\MTr@treev\MTr@treev%
    \global\MTr@treew\MTr@treew%
}
```

`\MTr@drawlabel`

```
\def\MTr@drawlabel{%
  \put(\MTr@treev,0){%
    \makebox(0,0)[t]{%
      \rule{0pt}{\MTr@minheight}%
      \usebox{\MTr@labelbox}}}%
}
```

`\MTr@drawchild` Draws the saved child node's picture.

```
\def\MTr@drawchild#1{%
  \MTr@tempcnta\MTr@childparam{#1}{x}
  \advance\MTr@tempcnta-\MTr@childparam{#1}{v}
  \put(\MTr@tempcnta,\MTr@childparam{#1}{y}){%
    \makebox(0,0)[tl]{%
      \usebox{\csname MTr@child#1box\endcsname}}}%
}
```

`\MTr@drawchildline` Draws the line or triangle from the parent to the given child.

```
\def\MTr@drawchildline#1{
```

Use child's y, but advance it to make the line stop just above the label.

```
  \MTr@tempcnta\MTr@childparam{#1}{y}
  \advance\MTr@tempcnta\MTr@lineoffset%
  \expandafter
  \ifnum\csname MTr@child#1triangle\endcsname=1%
    \MTr@drawchildlinetriangle{#1}%
  \else%
    \MTr@drawchildlineline{#1}%
  \fi%
```

```
}
\def\MTr@drawchildlineline#1{%
    \put(0,0){\MTr@drawline%
      (\MTr@treev,\MTr@parenty)%
      (\MTr@childparam{#1}{x},\MTr@tempcnta)}%
}
\def\MTr@drawchildlinetriangle#1{%
    \MTr@tempcntb\MTr@childparam{#1}{x}%
    \MTr@tempcntc\MTr@tempcntb%
    \advance\MTr@tempcntb \MTr@childparam{#1}{w}%
    \advance\MTr@tempcntc -\MTr@childparam{#1}{v}%
    \put(0,0){\MTr@drawline%
        (\MTr@treev,\MTr@parenty)%
        (\MTr@tempcntc,\MTr@tempcnta)}%
    \put(0,0){\MTr@drawline%
        (\MTr@treev,\MTr@parenty)%
        (\MTr@tempcntb,\MTr@tempcnta)}%
    \put(0,0){\MTr@drawline%
        (\MTr@tempcntc,\MTr@tempcnta)%
        (\MTr@tempcntb,\MTr@tempcnta)}%
}
```

6.11 Output routines

`\MTr@maketreebox` This is the main macro that starts the output.

```
\def\MTr@maketreebox{%
  \MTr@labelheight\ht\MTr@labelbox%
  \ifnum\MTr@labelheight<\MTr@minheight\MTr@labelheight\MTr@minheight\fi%%
  \MTr@labeldepth\dp\MTr@labelbox%
  \MTr@labelhalfwidth\wd\MTr@labelbox%
  \divide\MTr@labelhalfwidth 2%
  \ifnum\MTr@numchildren=0%
    \global\setbox\MTr@treebox\hbox{\MTr@outputlabel}%
  \fi%
  \ifnum\MTr@numchildren>0%
    \global\setbox\MTr@treebox\hbox{\MTr@outputchildren{\the\MTr@numchildren}}%
  \fi%
  \global\MTr@branchmult\MTr@mybranchmult%
  \ifMTr@mytriangle%
    \global\MTr@triangletrue%
  \else%
    \global\MTr@trianglefalse%
  \fi%
}
```

`\MTr@outputlabel` Output a tree that is just a label, with no children.

```
\def\MTr@outputlabel{%
```

First, set parameters: Height and depth of the subtree are equal to height and depth of the label. The x and w of the subtree each equal half the width of the label. Optionally, x may be zero and w may equal to the entire width of the label. The width and height of the picture to be drawn equal the width of the label and the height of the tree.

```
    \global\MTr@treeheight\MTr@labelheight%
```

```
    \global\MTr@treedepth\MTr@labeldepth%
    \ifMTr@mytriangle%
      \MTr@treew\MTr@labelhalfwidth%
      \MTr@treev\MTr@labelhalfwidth%
      \multiply\MTr@treew \MTr@trianglemultright%
      \multiply\MTr@treev \MTr@trianglemultleft%
      \divide\MTr@treew 50%
      \divide\MTr@treev 50%
      \global\MTr@treew\MTr@treew%
      \global\MTr@treev\MTr@treev%
    \else%
      \global\MTr@treew\MTr@labelhalfwidth%
      \global\MTr@treev\MTr@labelhalfwidth%
    \fi%
    \MTr@picwidth\wd\MTr@labelbox%
    \MTr@picheight\MTr@treeheight%
```

Second, draw the picture. The coordinates for the picture are nearly the same throughout. In any event, the label is centered in the picture, its baseline aligned with the bottom of the picture.

```
    \advance\MTr@picheight\MTr@treedepth%
    \begin{picture}%
        (\MTr@picwidth,\MTr@picheight)%
        (-\MTr@treev,-\MTr@picheight)%
      %\put(-\MTr@treev,-\MTr@picheight){\framebox(\MTr@picwidth,\MTr@picheight){}}%
      \put(-\MTr@treev,0){%
        \makebox(0,0)[tl]{%
          \rule{0pt}{\MTr@minheight}%
          \usebox{\MTr@labelbox}}}%
    \end{picture}%
}
```

`\MTr@outputchildren` Output a tree or subtree with at least one child tree.

```
\def\MTr@outputchildren#1{%
```

Calculate desired distance between the children. The cumulative value is put into `treev`.

```
    \ifnum#1>1
      \MTr@loopcnta 1
      \loop
\edef\MTr@temp{\romannumeral\MTr@loopcnta}%
\ifnum\MTr@loopcnta<#1
  \advance \MTr@loopcnta by 1
  \MTr@adjustdistance{\MTr@temp}{\romannumeral\MTr@loopcnta}%
      \repeat
    \fi
```

Calculate the subtree's v and w, based on the children, and the label's width.

```
    \MTr@setparentdimens{i}{\romannumeral#1}%
```

After the call to `setparentdimens`, all children's x values must be advanced by `morex`. In the same loop, also set vertical position and picture height for each child. The height of the picture is the height of the largest child.

```
    \MTr@picheight 0%
    \MTr@loopcnta #1
```

```
    \loop
      \edef\MTr@temp{\romannumeral\MTr@loopcnta}%
      \advance\MTr@childparam{\MTr@temp}{x}\MTr@morex%
      \MTr@setverticalchilddimens{\romannumeral\MTr@loopcnta}%
      \ifnum\MTr@childparam{\MTr@temp}{picheight}>\MTr@picheight%
\MTr@picheight\MTr@childparam{\MTr@temp}{picheight}%
      \fi%
    \advance \MTr@loopcnta by -1 \ifnum\MTr@loopcnta>0 \repeat
```

Set various other parameters.

```
  \MTr@setpictureparameters%
```

Start the picture and draw the label. Next, draw each child, and the line to that child. Then, end the picture.

```
  \begin{picture}(\MTr@picwidth,\MTr@picheight)(0,-\MTr@picheight)%
    %\put(0,-\MTr@picheight){\framebox(\MTr@picwidth,\MTr@picheight){}}%
    \MTr@drawlabel%
    \MTr@loopcnta #1
    \loop
      \expandafter\MTr@drawchild{\romannumeral\MTr@loopcnta}%
      \expandafter\MTr@drawchildline{\romannumeral\MTr@loopcnta}
    \advance \MTr@loopcnta by -1 \ifnum\MTr@loopcnta>0 \repeat
  \end{picture}%
}
```

References

[1] Doug Arnold *LaTeX for Linguists*
At `http://www.essex.ac.uk/linguistics/clmt/latex4ling/`

[2] A. Brüggemann-Klein and D. Wood. *Drawing trees nicely with TeX* File `tree_doc.tex` in the `treetex` package

[3] Jonathan Fine. *Some Basic Control Macros for TeX* TUGboat, Volume 13 (1992), No. 1

[4] Ralf Vogel *xyling — LaTeX macros for linguistic graphics*
At `http://www.ling.uni-potsdam.de/ rvogel/xyling/`