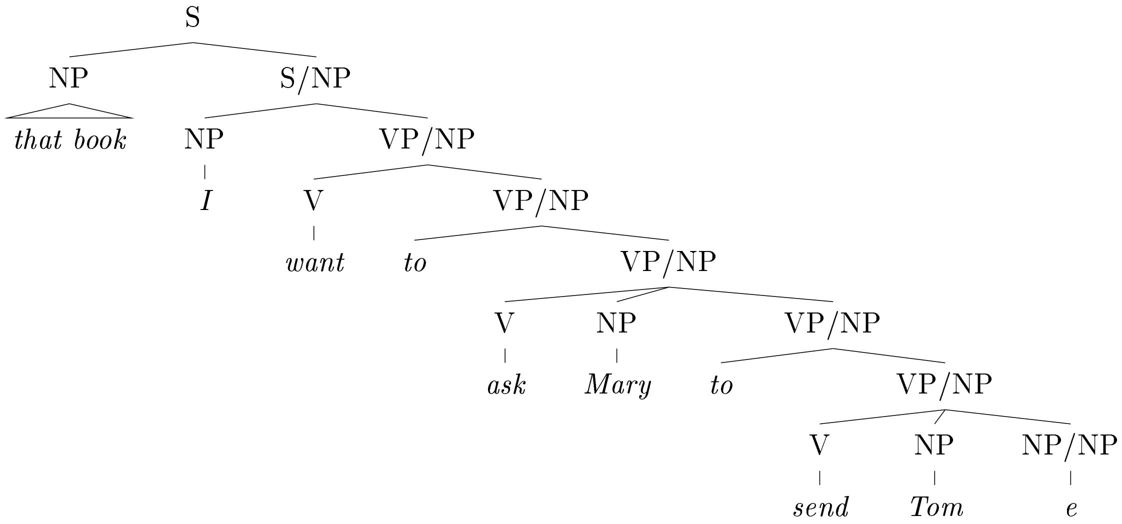
Avery Andrews'

Trees Preprocessor

Consider the tree shown in (1):

(1)



Making trees like this in TeX or LaTeX can be a cause for dread. But it can become quite simple and fun with the trees preprocessor.

trees is a preprocessor designed to work together with the `trees.sty` macro package and Emma Pease's `tree-dvips.sty` package (and the associated postscript input files),[1] to help with tree-formatting. The system will run equally well under TeX or LaTeX, but from now on, we will give LaTeX examples, but refer to the processor as simply TeX. `trees.sty` gets TeX to lay out the nodes of a tree on a page. `tree-dvips.sty` gets TeX to put commands in the `.dvi` file that cause Postscript to draw lines, triangles and various other exotica. trees takes easy to write user-friendly input and converts it into input that `trees.sty` and `tree-dvips.sty` can understand. trees inherits all the printing restrictions of `tree-dvips.sty`. In particular, you must convert your `.dvi` file with `dvips` and print it on a Postscript printer.

1 The preprocessor

1.1 Usage

One needs to run the trees preprocessor on a file to produce a TeX or LaTeX input file. So one needs to start off with another file extension, for example `.txp`. At the moment, trees runs as a filter,[2] so one gives a command like:

```
trees <nose.txp >nose.tex
```

trees will probably be modified soon so it can also accept regular command line parameters. Errors are written to stderr.

The file needs to include `trees.sty` and `tree-dvips.sty` among the style files that are loaded. For example, it might start out:

[1] `tree-dvips.sty` is not described here, but documentation is available separately. I would imagine things work equally well using `tree.sty` and `dvi3ps`, but this hasn't been tested.

[2] A command that reads from standard input and writes to standard output.

```
\documentstyle[tree-dvips,trees]{article}
```

Inside the input file, the signal for trees to kick into action is a line beginning with `.[`. The tree is then presented as an indented list, with a first daughter appearing (directly) after and more deeply embedded than its mother, and subsequent sisters indented equally to the first. The tree is ended with `.]` at the beginning of the line. So a simple sample tree is:

```
.[
S
   NP
      Det
         a
      Nom
         A
            savage
         Nom
            N
               puppy
   VP
      V
         destroyed
      NP
         Det
            the
         Nom
            shoe
.]
```

What comes out of the preprocessor is input to be handled by `trees.sty` (which does tree layout) and `tree-dvips.sty` (which does the line-drawing). So the output of the above example is the plethora of commands shown below:

```
\tree{\ntnode{Za}{S},
  {\ntnode{Zb}{NP},
    {\ntnode{Zc}{Det},
      {\tnode{Zd}{a}}
    },
    {\ntnode{Ze}{Nom},
      {\ntnode{Zf}{A},
        {\tnode{Zg}{savage}}
      },
      {\ntnode{Zh}{Nom},
        {\ntnode{Zi}{N},
          {\tnode{Zj}{puppy}}
        }
      }
    }
  },
  {\ntnode{Zk}{VP},
```

```
    {\ntnode{Zl}{V},
      {\tnode{Zm}{destroyed}}
    },
    {\ntnode{Zn}{NP},
      {\ntnode{Zo}{Det},
        {\tnode{Zp}{the}}
      },
      {\ntnode{Zq}{Nom},
        {\tnode{Zr}{shoe}}
      }
    }
  }
}
\nodeconnect{Za}{Zb}
\nodeconnect{Za}{Zk}
\nodeconnect{Zb}{Zc}
\nodeconnect{Zb}{Ze}
\nodeconnect{Zc}{Zd}
\nodeconnect{Ze}{Zf}
\nodeconnect{Ze}{Zh}
\nodeconnect{Zf}{Zg}
\nodeconnect{Zh}{Zi}
\nodeconnect{Zi}{Zj}
\nodeconnect{Zk}{Zl}
\nodeconnect{Zk}{Zn}
\nodeconnect{Zl}{Zm}
\nodeconnect{Zn}{Zo}
\nodeconnect{Zn}{Zq}
\nodeconnect{Zo}{Zp}
\nodeconnect{Zq}{Zr}
```

The preprocessor has converted the visually convenient indented list notation into appropriate TEX grouping for the `\tree` macro (defined in `tree.sty`), and has put the node labels into `\ntnode` (non-terminal node) and `\tnode` (terminal node) commands), which feed the `\node` command (from `tree-dvips.sty`) as discussed briefly below. Each node has also been assigned a tag starting with 'Z', and linedrawing commands (from `tree-dvips.sty`) have been issued as appropriate.

1.1.1 Options

Options are separated from a node label by a vertical bar (`|`). There are currently two options.

- The `tri` option causes a node to be connected to its mother with a triangle instead of a line.
- The `tag TAG` option, where `TAG` is any alphabetic string, sets `TAG` as the tag of the node. These tags are the node names used by `tree-dvips.sty`. For simple trees, you need not know anything about `tree-dvips.sty`, because trees will generate the appropriate commands for you. But if you want to do anything complicated, you will need to issue your own `tree-dvips.sty` commands, and then you'll need to give nodes explicit names.

Both of these options are illustrated in the example below:

```
.[
S
  NP
     the puppy|tri
  VP|tag vp
     barked
.]
\nodecircle{vp}
```

`\nodecircle` is a `tree-dvips.sty` command that puts a circle around a node.

1.2 Installation

trees has so far run under Sun `cc`, and Turbo C 2. It is compiled from the files `trees.c, slurp.c, words.c, ansi.c`. The module `ansi.c` is required for Sun `cc` to supply some library functions that appear to be missing. There are some bits of code for the VAX/VMS C version, but they are commented out rather than being done as conditional compilation statements. The following command will compile trees:

```
cc -o trees trees.c slurp.c words.c ansi.c -lm
```

The file `setup.src`, when sourced, defines command `go` to compile and link. So you can run trees as described previously, the executable file trees has to be in a directory which is in your path. Normally the current directory is in your path. See the Unix Manual entry for `csh` to find out how to see or alter your path.

2 Trees.sty

Two additional useful options are handled directly by the `trees.sty` definitions. First, one sometimes wants to specify the space between the mother and daughters (if the tree is rather wide, for example, the default spacing at the top might be too small). This can be done by putting a command such as `\daughterskip{4ex}` somewhere in the mother node, for example:

```
.[
S\daughtergap{4ex}
   NP
      the annoying child|tri
   VP
      ran away from the crazed puppy|tri
.]
```

It is also possible to specify the distance between a node and its previous sister with a command such as `\sistergap{-2em}`. This can be especially useful in packing 'wide trees' that wouldn't otherwise fit. The default spacings for sisters and daughters can also be altered, as discussed in `trees.sty`.

The `\ntnode` and `\tnode` commands do some moderately complicated things that are basically adapted to make it easier to write an LFG paper. Both commands operate on node specifications where a bunch of annotations are specified as sitting on top of the main node label. For example:

```
\ntnode{a}{(\uparrow SUBJ)=\downarrow/NP}
```

puts the annotation on top of the node-label. The levels are separated by a slash, and there can be as many as you like. The difference between `\ntnode` and `\tnode` is that the latter sets the last level in italics, as is the norm for terminals. To set the terminal nodes in a different type style, use a command such as the following:

```
\def\tstyle{\sl}   % slanted terminals
\def\tstyle{\rm}   % roman terminals
```

By default, as shown above, all other levels but the last are automatically set in math-mode. This can be convenient, but can also be a nuisance. It can be turned off with the command:

```
\def\nodeannotation#1{#1}
```

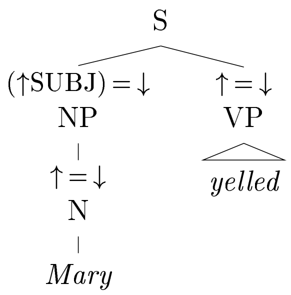
or alternatively `\nodeannotation` can be redefined to do other useful things such as set the annotations in a smaller font.

As a line of input to the preprocessor is fed to these commands, it may also be such a slash separated list, as shown in the sample LFG tree below:[3]

```
.[
S
   (\uparrow SUBJ) = \downarrow/NP
      \uparrow = \downarrow/N
         Mary
   \uparrow = \downarrow/VP
      yelled|tri
.]
```

and the output comes out like this:

(2)



Unfortunately, this means that to set a GPSG-style slashed category, one must either enclose it in braces (`{NP/NP}`) or redefine the operation of `\ntnode`. One can just switch off all of this stuff by redefining these commands to do nothing:

```
\def\ntnode#1#2{\node{#1}{#2}}
\def\tnode#1#2{\node{#1}{#2}}
```

For further information on altering the operation of `\ntnode` or `\tnode`, or for information about various other parameters and options that are available see the comments at the start of `trees.sty`.

[3]In practice one would want to also define some other macros that make it easier to type functional equations.