# jTree
for linguists

Tex macros for typesetting complex trees

User's Guide

John Frampton
j.frampton@neu.edu

17 September 2006
Version 2.3

Contents

1. Introduction

Complex trees that linguists often need to display are difficult to represent linearly in a fashion that allows a human to readily grasp the intended hierarchical structure. This makes it difficult to write Tex code in such a way that it can be easily debugged or modified a year after it was first written, or that portions of it can be copied for use in a different environment. jTree was designed to overcome these problems.
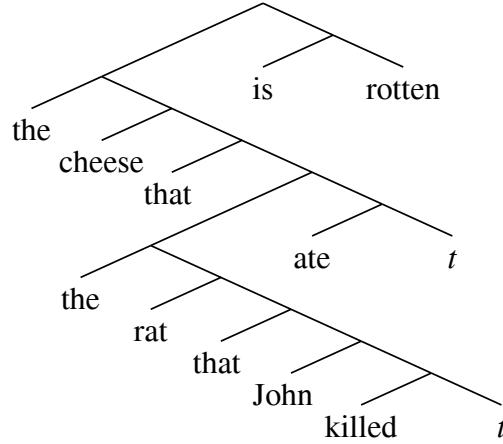
As usual, it is much easier to design a good tool for a task if the task is fairly narrow. So my aim was to write a tree formatter that would be particularly good for the kind of trees that I most often encounter. In the syntax that most interests me, trees are often fairly deep, with a depth of embedding at least 5. Example 9 in Section 14 has depth 19. Using parentheses or brackets to encode embedding is not a viable option if the Tex code is to have the desired characteristics. In spite of being deep, however, the trees I usually want to typeset are simple in some other ways; they are generally binary branching and tend strongly to have their complex branching to the right. jTree is designed to be particularly good at typesetting such trees. Obviously, this represents a particular point of view about syntactic theory.

jTree is based on the widely used *pstricks* macro package. Many of the more advanced features of jTree require a willingness to learn a certain amount of PSTricks. Linguists will find much in PSTricks that is useful, even outside the context of displaying and annotating trees. Most of the commands take parameters which are defined and explained in the PSTricks documentation. For information about PSTricks, see *http://www.tug.org/applications/PSTricks/*. jTree also requires the *xkeyval* package, which has become the standard mechanism for handling parameters in *pstricks* based packages. See Appendix A for information on installing the PSTricks, XKeyVal, and jTree packages.

Some tree formatters attempt to make many decisions about placement automatically, in order to make the formater easy and foolproof to use. The downside is that complex trees are generally quite difficult to typeset because some effort is required to undo the automatic help provided by the formatter. The approach of jTree is to do very little automatically, but to make what is done very transparent and very easy to adjust. Many features of tree construction in jTree are controlled by parameters, whose default settings usually suffice, but which are available for fine-grained control when necessary. The core of jTree is a tree description language. We begin by discussing that language.
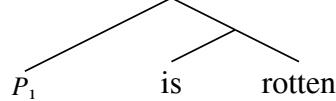
2. The jTree description language (JTDL)

There are two components: 1. a simple way to describe right branching trees (which I take to be trees whose left branches do not branch); and 2. a mechanism for adjoining one tree onto another. Consider the following tree, for example:

the cheese that the rat that John killed t ate t is rotten
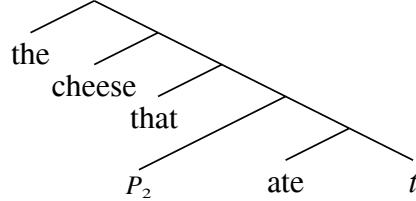
This tree can be described by a sequence of three right branching trees and instructions for how to combine them.

(1) 1.

P_1 is   rotten

2. adjoin at P_1:

the cheese that P_2 ate t

3. adjoin at P_2:

the rat that John killed t
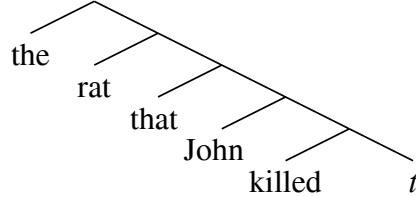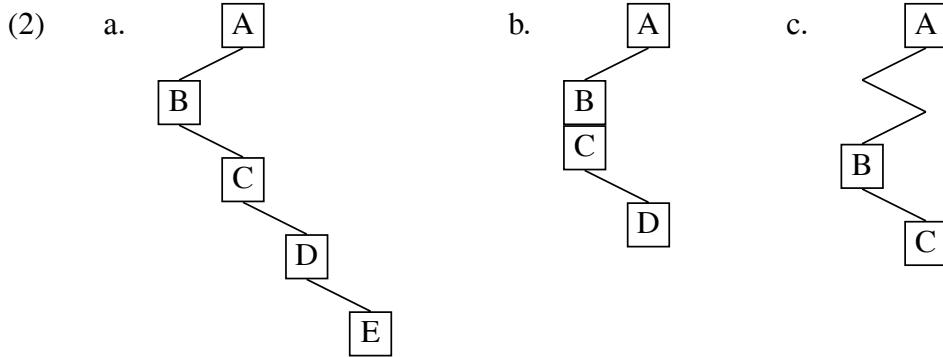
We first address the question of describing right branching trees, then a mechanism for combining them.

2.1. The description of right branching trees
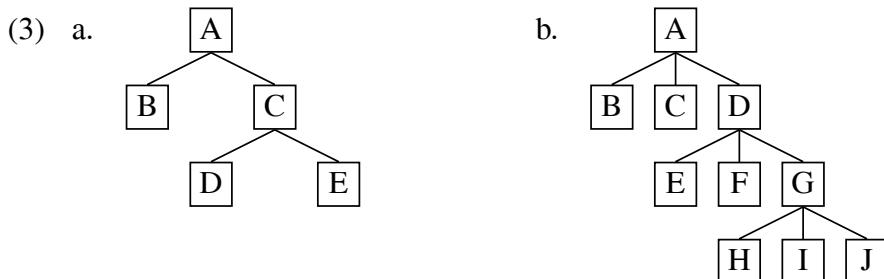
Some trees are essentially a linear sequence of branches and nodes.

(2)   a.

```
            A
       B
           C
             D
               E
```

b.

```
            A
       B
       C
         D
```

c.

```
            A

        B
          C
```

Such a tree can be described as a sequence of branches and labels, each one simply attached to the one before it in the obvious way.

   a.   A ⟨left⟩ B ⟨right⟩ C ⟨right⟩ D ⟨right⟩ E .

   b.   A ⟨left⟩ B C ⟨right⟩ D .

   c.   A ⟨left⟩ ⟨right⟩ ⟨left⟩ B ⟨right⟩ C .

If this language is extended by adding an operator ⌃ which shifts the attachment point to the start of the previous branch, then right branching trees can be described. The trees (3) are described by the sequences (4).

(3)   a.

```
            A
       B       C
             D   E
```

b.

```
            A
      B  C  D
           E F G
               H I J
```

(4)   a.   A

       ⟨left⟩ B ⌃ ⟨right⟩ C

       ⟨left⟩ D ⌃ ⟨right⟩ E .

b. $\boxed{A}$
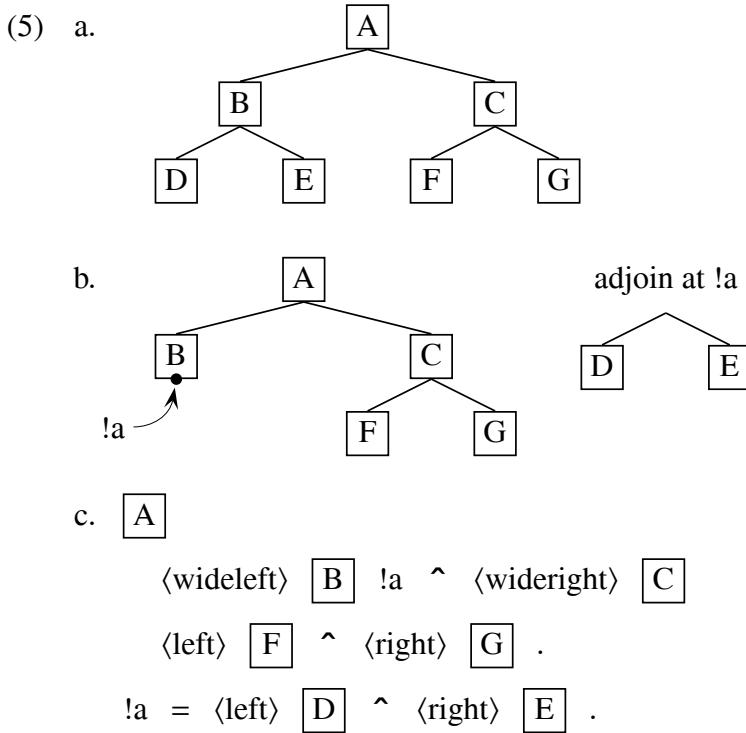
⟨left⟩ $\boxed{B}$ ^ ⟨vert⟩ $\boxed{C}$ ^ ⟨right⟩ $\boxed{D}$

⟨left⟩ $\boxed{E}$ ^ ⟨vert⟩ $\boxed{F}$ ^ ⟨right⟩ $\boxed{G}$

⟨left⟩ $\boxed{H}$ ^ ⟨vert⟩ $\boxed{I}$ ^ ⟨right⟩ $\boxed{J}$.

A parser for such descriptions needs to keep track of just two positions. At the start, $\mathcal{P}$ and $\mathcal{Q}$ are both set to some default starting point. After a label or branch is parsed, $\mathcal{P}$ is updated to be the current point. When a branch is parsed, $\mathcal{Q}$ is updated to be its starting point. When ^ is parsed, $\mathcal{P}$ is set to $\mathcal{Q}$.

2.2. Tree adjunction

This language can be extended by adding the notion of an adjunction point, and the syntax for adjoining one tree to another at a specified adjunction point. The decomposition (5b) of the tree (5a) is described by (5c).

(5) a.

b.

adjoin at !a

c. $\boxed{A}$

⟨wideleft⟩ $\boxed{B}$!a ^ ⟨wideright⟩ $\boxed{C}$

⟨left⟩ $\boxed{F}$ ^ ⟨right⟩ $\boxed{G}$.

!a = ⟨left⟩ $\boxed{D}$ ^ ⟨right⟩ $\boxed{E}$.
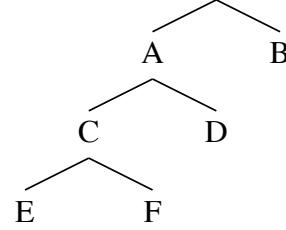
When the parser encounters an adjunction point, it records $\mathcal{P}$ and names it so that it can be accessed in the future. In the jTree implementation of this tree description language, the names of adjunction points, called !tags, are character strings beginning with ! and followed by a space. The tree that is described is a physical tree, not an abstract tree. Branches are objects which have dimensions and are identified by a name. The branch denoted by `<wideleft>` above, for example, has different dimensions than the branch denoted by `<left>`.

4

jTree is based on this tree description language. Facilities are provided for defining and naming branches of various kinds and for drawing trees specified in JTDL format. How a tree is rendered will depend not only on its description, but on numerous parameter settings, font choices, etc. The Tex code for rendering a tree has the form:

> \jtree
> *preliminary definitions. parameter settings*
> \! = *simple tree description.*
> *definitions, parameter settings, dimensionless graphics*
> \!a = *simple tree description.*
> *definitions, parameter settings, dimensionless graphics*
> \!b = *simple tree description.*
> *dimensionless graphics*
> \endjtree
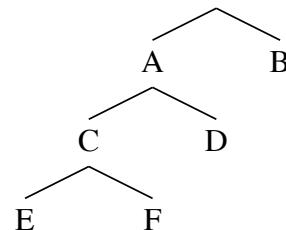
When \jtree is executed, it establishes an adjunction point named !. The macro \! is defined so that executing \!xxx = sets $\mathcal{P}$ and $\mathcal{Q}$ to the point named !xxx (xxx can be empty, so the point can be named !) and parsing the tree description which follows is initiated, terminating when a period is encountered. For example:

```
\jtree
\! = <left>{A}!a ^<right>{B}.
\!a = <left>{C}!b ^<right>{D}.
\!b = <left>{E} ^<right>{F}.
\endjtree
```
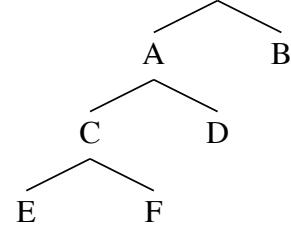

The following code produces identical results.

```
\jtree
\! = <left>{A}!a ^<right>{B}.
\!a = <left>{C}!a ^<right>{D}.
\!a = <left>{E} ^<right>{F}.
\endjtree
```


The name !a created in the main tree, is no longer needed (in this example) after the first subtree is initiated and can therefore be reassigned. It is even possible to say:

```
\jtree
\! = <left>{A}! ^<right>{B}.
\! = <left>{C}! ^<right>{D}.
\! = <left>{E} ^<right>{F}.
\endjtree
```
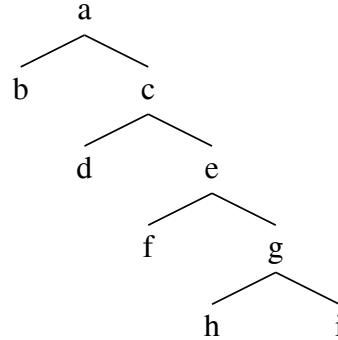
There is nothing special about the name ! that \jtree assigns other than it is assigned before tree construction commences.

2.3. The colon construction

The combination <left>*label*^<right> occurs frequently in the descriptions of binary trees. JTDL provides a shortcut for this. :*label* is interpreted as <left>*label*^<right>. Using this:

```
\jtree
\! = {a}
   :{b} {c}
   :{d} {e}
   :{f} {g}
   :{h} {i}.
\endjtree
```
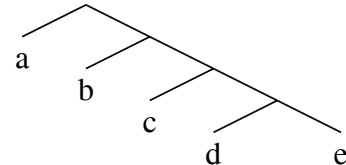
The formatting is for readability. The following code produces the same tree:

```
\jtree\! ={a}:{b}{c}:{d}{e}:{f}{g}:{h}{i}.\endjtree
```

One more example:
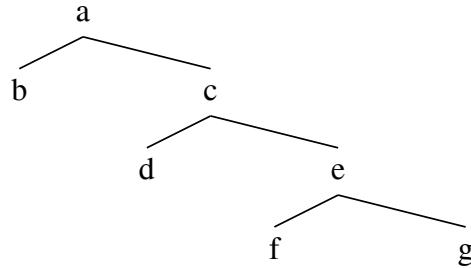
```
\jtree
\! = :{a} :{b} :{c} :{d} {e}.
\endjtree
```

For expository reasons, the description above was oversimplified. Actually, :*label* is shorthand for <colonA>*label*^<colonB>. *pst-jtree* defines the branches <colonA> and <left> identically, and the branches <colonB> and <right>

identically. The user, however, is free to redefine any of these branches. For example.

```
\jtree                                          a
\defbranch<colonB>(1)(-.5)
\! = {a}                          b                     c
   :{b} {c}
   :{d} {e}                                        d           e
   :{f} {g}.
\endjtree                                                   f       g
```
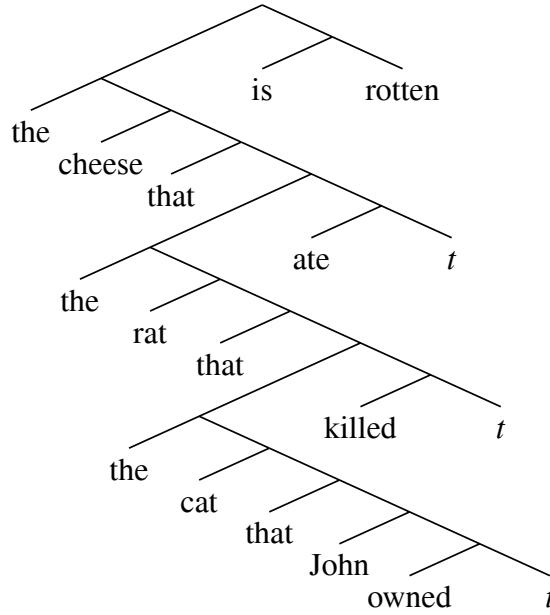
A later section will give the details of how branches are defined. Suffice it to say here that a branch is defined by giving its height and slope. Labels are recognized by the jTree parser as { *stuff* }, where the "stuff" is anything that can go in a Tex hbox. We also defer to a later section automatic space that is inserted over and under labels and how this is specified and adjusted.

The example below uses the colon shortcut extensively.
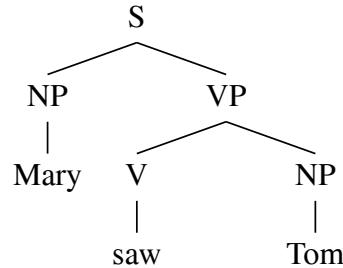
```
(6)  \jtree
     \defbranch<Left>(2.3)(1)
     \! = <Left>!a ^<right> :{is} {rotten}.
     \!a = :{the} :{cheese} :{that}
        <Left>!b ^<right> :{ate} {\it t}.
     \!b = :{the} :{rat} :{that}
        <Left>!c ^<right> :{killed} {\it t}.
     \!c = :{the} :{cat} :{that} :{John} :{owned} {\it t}.
     \endjtree
```

2.4. Inline adjunction

In addition to the **:** operator, ȷTree provides one other shortcut. It provides an alternative to adjunction in the code below.

```
\jtree[xunit=2.8em,yunit=1em]
\! = {S}
   :{NP}!a {VP}
   :{V}!b {NP}
   <vert>{Tom}.
\!a = <vert>{Mary}.
\!b = <vert>{saw}.
\endjtree
```

This can be written:

```
    \jtree[xunit=2.8em,yunit=1em]
    \! = {S}
       :{NP}(<vert>{Mary})  {VP}
       :{V}(<vert>{saw})  {NP}<vert>{Tom}.
    \endjtree
```
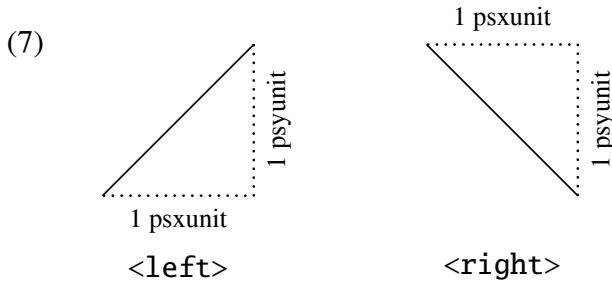
Effectively, interpreting (. . .) carries out *inline adjunction*, without the need to explicitly name the adjunction point. In practice, description by inline adjunction should be avoided for all but very simple nonterminals (as in this example) since it can quickly lead to opaque code and the hornet's nest of parentheses that JTDL was designed to avoid.

3. Parameters
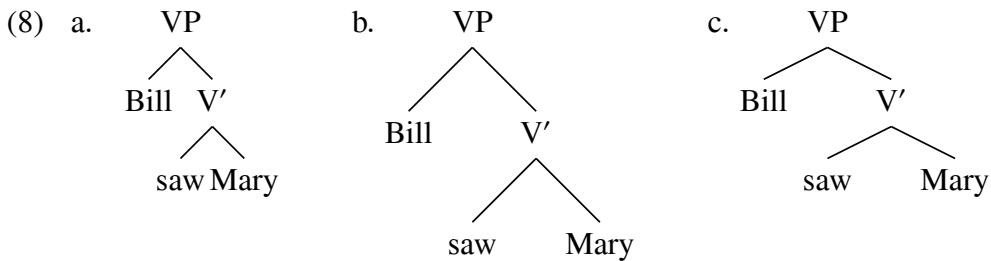
Many jTree items take which control various aspects of typesetting. *pst-jtree* contains the definitions:

```
\defbranch<left>(1)(1)
\defbranch<right>(1)(-1)
```

(Quotations from *pst-jtree* will be displayed in a box, as above.) This defines `<left>` to be a branch with height 1 psyunit and slope is 1, and `<right>` to be a branch with height 1 psyunit and slope −1. The specification of branches will be discussed in more detail in Section 4, but for now it is sufficient to know that `<left>` and `<right>` are drawn as shown below:

(7)
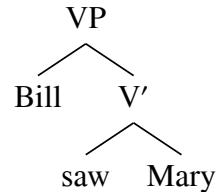
The fact that the dimensions of branches can be specified in psunits, with the xunits independent of the yunits, coupled with the fact that branches are rendered in physical (Tex) units, produces a great deal of flexibility. The three trees below were produced with exactly the same jTree code, but with `\psset{unit=1em}`, `psset{unit=2em}`, and `psset{xunit=2em,yunit=1em}`.

(8)
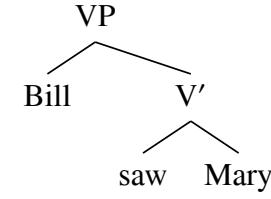
`\jtree` accepts parameters directly, so you can say:

```
\jtree[xunit=1.5em,yunit=1em]
\! = {VP}
   <left>{Bill} ^<right>{V$'$}
   <left>{saw} ^<right>{Mary}.
\endjtree
```


Individual branches also take parameters, so you can say:

9

```
\jtree[xunit=1.5em,yunit=1em]
\! = {VP}
   <left>{Bill} ^<right>[xunit=3em]{V$'$}
   <left>{saw} ^<right>{Mary}.
\endjtree
```

VP

Bill   V′

saw   Mary

The parameters unit, xunit, and yunit are defined by PSTricks. jTree defines some of its own parameters. A parameter scaleby is provided. \psset{scaleby=x y} causes branches to be drawn as if
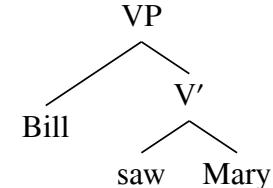
$$\psset{xunit=x\psxunit,yunit=y\psyunit}$$

had been executed (but the psunits are not actually changed).
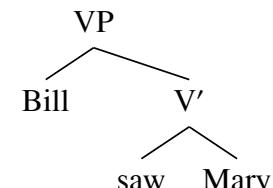
\psset{scaleby=x} is equivalent to \psset{scaleby=x x}.

The following are then possible:

```
\jtree[xunit=1.5em,yunit=1em]
\! = {VP}
   :[scaleby=2]{Bill} {V$'$}
   :{saw} {Mary}.
\endjtree
```

VP

       V′
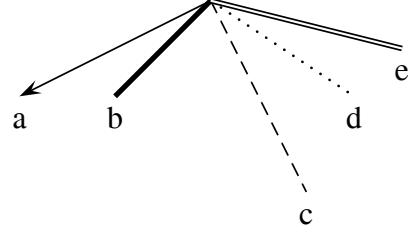
Bill   saw   Mary

```
\jtree[xunit=1.5em,yunit=1em]
\! = {VP}
   <left>{Bill} ^<right>[scaleby=2 1]{V$'$}
   :{saw} {Mary}.
\endjtree
```

VP

Bill   V′

saw   Mary

jTree defines a dozen or so parameters in all. They can be set by \psset, but more importantly, can be attached to particular instances of \jtree and the branches and labels that appear inside \jtree environments.

Modifying branches by standard PSTricks parameters is often useful.
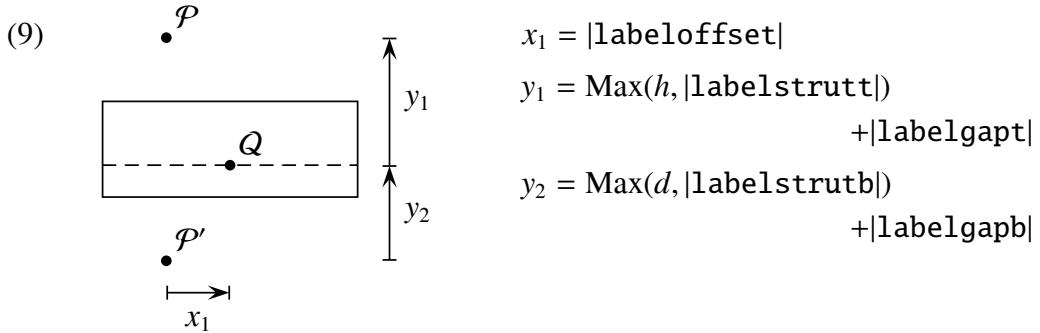
```
\jtree[xunit=3em,yunit=3em]
\! = <left>[scaleby=2 1,
          arrows=->]{a}
  ^<left>[linewidth=2pt]{b}
  ^<right>[scaleby=1 2,
     linestyle=dashed]{c}
  ^<right>[scaleby=1.5 1,
     linestyle=dotted,linewidth=1pt]{d}
  ^<right>[scaleby=2 .5,doubleline=true]{e}.
\endjtree
```


The PSTricks documentation can be consulted for many other line and arrow parameters which offer further control over how branches are drawn.

4. Labels

In tree descriptions, material in {...} is typeset in an hbox, called a *label box*. After the label box is added to a structure with current point $\mathcal{P}$, a new point $\mathcal{P}'$ becomes the current point. The issues addressed in this section are the relative positions of $\mathcal{P}$, the label box, and $\mathcal{P}'$. The position of the label box is specified by giving the position of the center of its baseline, called $\mathcal{Q}$ here. The relative positions depend upon the height h and depth d of the label box, and five different parameters: `labelgapt`, `labelgapb`, `labelstrutt`, `labelstrutb`, and `labeloffset`. The beginning user should not be discouraged by the apparent complexity. Various defaults are set in *pst-jtree* and the issue can be largely ignored unless some special effect is desired. When the time comes that a complex positioning problem arises, the flexibility will be both understandable and welcome. The positioning rules are given below, with |`parameter`| representing the value of the parameter.

(9)

$$x_1 = |\texttt{labeloffset}|$$
$$y_1 = \text{Max}(h, |\texttt{labelstrutt}|)$$
$$+ |\texttt{labelgapt}|$$
$$y_2 = \text{Max}(d, |\texttt{labelstrutb}|)$$
$$+ |\texttt{labelgapb}|$$

A consequence of these rules is that the top of a label box will be at a distance of at least |`labelgapt`| below the terminus of the branch or label that it follows, and the baseline of a label box will be at a distance |`labelstrutt`| + |`labelgapt`| below that terminus unless the label box is unusually high (i.e. $h > |\texttt{labelstrutt}|$). This
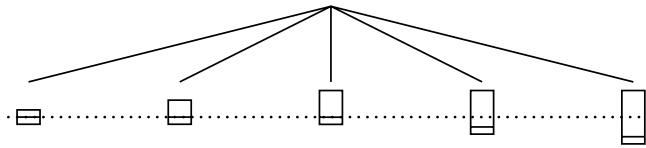
means that label boxes never get too close to the branches or labels they follow, and the baselines of label boxes on different branches of the same height will be aligned, unless the label boxes are exceptionally high. The same considerations apply to the relative positions of the label box and $\mathcal{P}'$.

There is also a parameter `normallabelstrut` which can be set to *true* or *false*. If set to *true*, every time that `\jtree` is executed the dimensions of the label strut are set to the dimensions of the current Tex strut. Specifically,
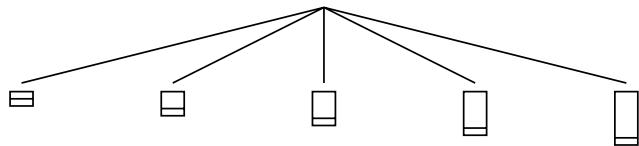
> `\psset{labelstrut={\the\ht\strutbox} {\the\dp\strutbox}}`

is executed when `\jtree` is invoked. `\psset{labelstrut=`x y`}` is equivalent to `\psset{labelstrutt=`x`,labelstrutb=`y`}`. Users do not have to be concerned with setting label strut unless they desire some special effect since *pst-jtree* sets `normallabelstrut` to *true*. The default settings for the top and bottom label gaps are .35 ex. `\psset{labelgap=`x`}` is equivalent to `\psset{labelgapt=`x`,labelgapb=`x`}`.

The effect of this scheme is shown below with `normallabelstrut` set to *true* and `labelgapt` and `labelgapb` set to .6 ex. The size of the labels is as shown, with the baselines shown. If the height of the label does not exceed |`labelstrutt`|, the baselines of the labels are aligned, otherwise the top of the label box is at a distance |`labelgapt`| below the terminus of the branch it follows.
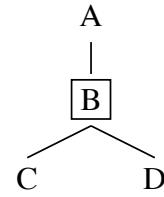
Sometimes it is useful to set `labelstrutt` to 0. In that case, the result is:
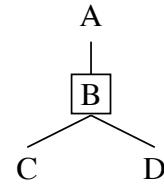

Many users will probably leave `normallabelstrut` set to *true* and forget about it. But most users will want to change the label gaps from time to time. Contrast the following, for example. In some applications, the second might be preferable.

```
\jtree
\! = {A}
   <vert>{\psframebox{B}}
   :{C}{D}.
\endjtree
```

A

B

C   D

```
\jtree
\! = {A}
   <vert>{\psframebox{B}}[labelgap=0]
   :{C}{D}.
\endjtree
```

A

B
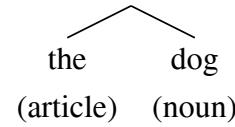
C   D
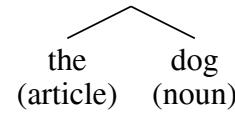
This is used in typesetting (15) in Section 11.

The following contrast is also interesting, in part because it uses a negative label gap.

```
\jtree
\! = :({the}{(article)}) {dog}{(noun)}.
\endjtree
```

the   dog
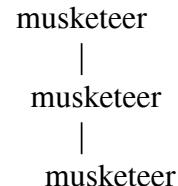
(article)   (noun)

```
\jtree
\! = :({the}{(article)}[labelgapt=-3pt])
   {dog}{(noun)}[labelgapt=-3pt].
\endjtree
```

the   dog

(article)   (noun)

See examples 3 and 4 in Section 14 for applications of this trick, which often eliminates the need for complex multiline labels.

The label box can be positioned horizontally using `labeloffset`.
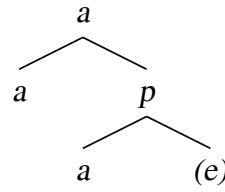
```
\jtree
\! = {musketeer}
   <vert>{musketeer}[labeloffset=1ex]
   <vert>{musketeer}[labeloffset=2ex].
\endjtree
```

musketeer

musketeer

musketeer

See example 1 in Section 14 for an illustration of how changing `labeloffset` can solve certain spacing problems between labels.
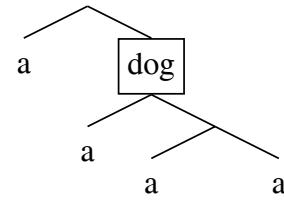
There is one other parameter that is relevant to labels. The material specified by `everylabel` is put into a token list and the token list is inserted at the beginning of every label.

```
\jtree[everylabel=\sl]
\! = {a}
   :{a} {p}
   :{a} {(e)}.
\endjtree
```
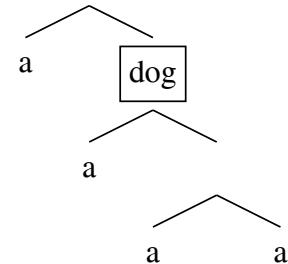
If a label is {\omit...} or {\pnode...}, then the vertical positioning algorithm discussed above is bypassed and the label is positioned with its top edge at the level of $\mathcal{P}$ and $\mathcal{P}'$ is positioned directly under $\mathcal{P}$ at the level of the bottom edge of the label box. `labeloffset` still operates. Contrast the following:

```
\jtree[everylabel=\strut,labelgap=3pt]
\! = :{a} {\omit\psframebox{dog\strut}}
   :{a} {\pnode{A1}}
   :{a} {a}.
\endjtree
```



```
\jtree[everylabel=\strut,labelgap=3pt]
\! = :{a} {\psframebox{dog\strut}}
   :{a} {}
   :{a} {a}.
\endjtree
```


---
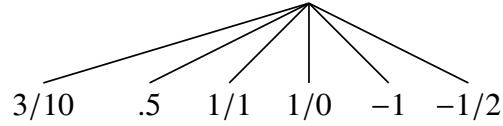
5. Branches

Most tree formatting schemes format the array of labels, then connect them with branches. The dimensions of the branches are determined by the location of the labels they must connect. A jTree branch, on the other hand, has dimensions specified in its definition and determines the location of the material it points to. Branches are defined by giving their height and slope. The syntax is:

\defbranch<*name*>(*height*)(*slope*)

The height can have Tex dimensions (pt, ex, em, in, cm, etc.), or it can be purely numerical. If numeric, it is taken to be the height measured in psyunits. The slope is the ratio of the vertical dimension measured in psyunits to the horizontal dimension measured in psxunits. Lines going from the lower left to the upper right have positive slopes and lines going from the upper left to the bottom right have negative slopes. Usually, the slope can be expressed as a decimal number. If the horizontal dimension is zero, as it is with a vertical line, a decimal slope is impossible (because division by zero is) and the slope must be expressed as a ratio.
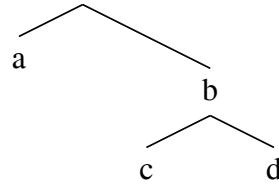
```
\jtree[unit=2.5em]
\defbranch<1;3/10>(1)(3/10)
\defbranch<1;.5>(1)(.5)
\defbranch<1;1/1>(1)(1/1)
\defbranch<1;1/0>(1)(1/0)
\defbranch<1;-1>(1)(-1)
\defbranch<1;-1/2>(1)(-1/2)
\! = <1;3/10>{$3/10$}
   ^<1;.5>{$.5$}
   ^<1;1/1>{$1/1$}
   ^<1;1/0>{$1/0$}
   ^<1;-1>{$-1$}
   ^<1;-1/2>{$-1/2$}.
\endjtree
```


$$3/10 \qquad .5 \qquad 1/1 \quad 1/0 \quad -1 \quad -1/2$$

Note that the characters appearing in a branch name are not restricted to alphabetical characters.
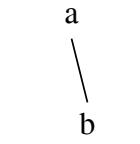
I have never found a use for defining branches with the height specified in physical units, but someone might.

```
\jtree[xunit=2em,yunit=1em]
\defbranch<Right>(2em)(-1)
\! = <left>{a} ^<Right>{b}
   <left>{c} ^<right>{d}.
\endjtree
```
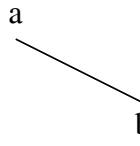
Finally, note that the slope specification of a branch will not match the slope of the line which is drawn on paper unless the xunits and yunits are equal. One is the ratio of psyunits to psxunits, the other is the ratio of physical units. The branch `<right>` is specified to have slope -1, but:

```
\jtree[xunit=.5em,
   yunit=2em]
\! = {a}<right>{b}.
\endjtree
```

```
\jtree[xunit=4em,
   yunit=2em]
\! = {a}<right>{b}.
\endjtree
```

jTree specializes in binary trees, but *pst-jtree* predefines some branches which make it relatively easy to make bushier trees. The entire inventory of predefined branches is:

```
\defbranch<left>(1)(1)
\defbranch<right>(1)(-1)

\defbranch<4wideleft>(1)(2/3)
\defbranch<4left>(1)(2)
\defbranch<4right>(1)(-2)
\defbranch<4wideright>(1)(-2/3)

\defbranch<wideleft>(1)(1/2)
\defbranch<wideright>(1)(-1/2)

\defbranch<bigleft>(2)(1)
\defbranch<bigright>(2)(-1)

\defbranch<vert>(1)(1/0)
\defbranch<shortvert>(.5)(1/0)

\defbranch<colonA>(1)(1)
\defbranch<colonB>(1)(-1)
```
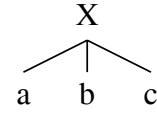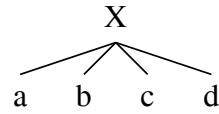
Of course, users can define whatever branches they find need for. `<colonA>` and `<colonB>` are the branches used by the colon macro. They are defined to be the same as `<left>` and `<right>`, but can be redefined to be whatever is convenient.

```
\jtree
\! = {X}
   <left>{a} ^<vert>{b} ^<right>{c}.
\endjtree
```
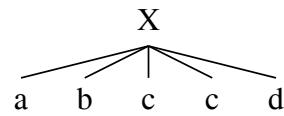
```
\jtree
\! = {X}
   <4wideleft>{a} ^<4left>{b}
      ^<4right>{c} ^<4wideright>{d}.
\endjtree
```

```
\jtree
\! = {X}
   <wideleft>{a} ^<left>{b} ^<vert>{c}
      ^<right>{c} ^<wideright>{d}.
\endjtree
```

---

6. Triangles and vartriangles

Triangles can be defined. The definition syntax is:
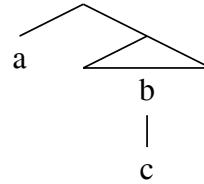
$$\texttt{\textbackslash deftriangle} < name > (height) (slopeA) (slopeB)$$

One triangle comes predefined.

```
\deftriangle<tri>(1)(1)(-1)
```
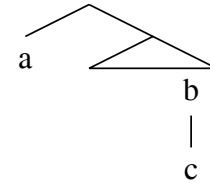
`<tri>` can be used like any other branch.

```
\jtree
\! = :{a} <tri>{b} <vert>{c}.
\endjtree
```

---

pst-jtree defines the parameter `triratio`, which can be used to adjust where the new current point is positioned along the bottom edge of the triangle. If *width* is the width of the triangle, the new current point is at distance $x = triratio \times width$ to the right of the left corner of the triangle. The display above is set with `triratio=.5`, the default, which puts the current point in the center of the bottom edge.

```
\jtree
\! = :{a} <tri>[triratio=.8]{b} <vert>{c}.
\endjtree
```
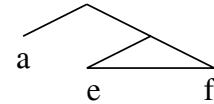
When a triangle is evaluated, its width is computed and a macro `\triwd` is defined which evaluates to this width. *pst-jtree* contains:

```
\def\triline#1{\hbox to\triwd{#1}}
```
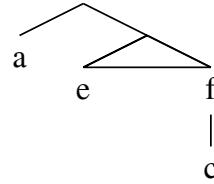
A construction like the following is sometimes useful:

```
\jtree
\! = :{a} <tri>{\triline{e\hfil f}}.
\endjtree
```

Remember as well that branches can be overwritten. Something along the lines of the following is sometimes useful.

```
\jtree
\! = :{a} <tri> ^:{e} {f} <vert>{c}.
\endjtree
```

6.1. Vartriangles
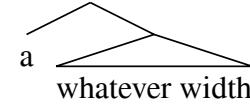
Vartriangles adjust their width to fit the label that they point to. A vartriangle is specified by giving a height. The definition syntax is:

$$\text{\textbackslash defvartriangle}{<}\textit{name}{>}(\textit{height})$$

jTree predefines one vartriangle.

```
\defvartriangle<vartri>(1)
```

```
\jtree
\! = :{a} <vartri>{whatever width}.
\endjtree
```
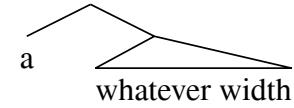
a
whatever width
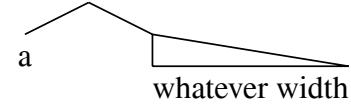
The parameter `triratio` has a different meaning for vartriangles than it has for ordinary triangles. It determines where the center of the bottom edge of the vartriangle is with respect to the top vertex. The center of the bottom edge is a distance *triratio × width* to the right of its left edge. It is easier to illustrate (done below) than to give the formula.
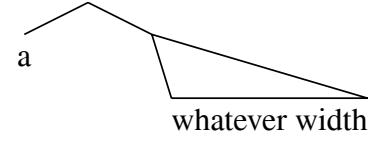
```
\jtree
\! = :{a} <vartri>
   [triratio=.3]{whatever width}.
\endjtree
```

a
whatever width

```
\jtree
\! = :{a}
   <vartri>[triratio=0]{whatever width}.
\endjtree
```

a
whatever width
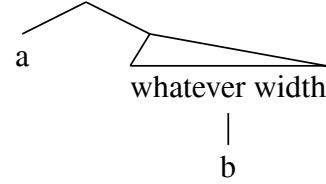
```
\jtree
\! = :{a}
   <vartri>[triratio=-.1,scaleby=2]
      {whatever width}.
\endjtree
```

a
whatever width

Generally, there is no branching from the label following a vartriangle, so no provision is made for adjusting the termination point of the label that follows vartriangles. It is at the center of the base.

```
\jtree
\! = :{a} <vartri>
      [triratio=.1]{whatever width}
   <vert>{b}.
\endjtree
```

a
whatever width
|
b
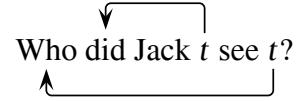
Another branch can follow a branch, but since the width of a vartriangle depends on the width of the label which follows it, a vartriangle (with optional parameters) must be followed immediately by a label. An error results if it is not.

7. @tags

jTree includes direct support for PSTricks' powerful *pst-node* package. Even in linear structures, *pst-node* is very useful to linguists. The ability to define nodes and to connect them with arrowed curves of various kinds allows the user to easily code things like:
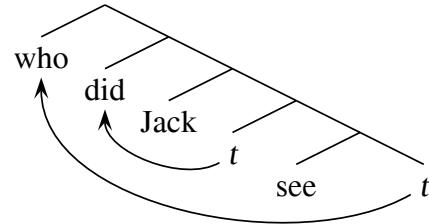
```
\rnode{A1}{Who} \rnode{B1}{did}
   Jack \rnode{B2}{\sl t\/}
   see \rnode{A2}{\sl t\/}?}
\psset{linearc=2pt}
\ncbar[angle=-90]{->}{A2}{A1}
\ncbar[angle=90]{->}{B2}{B1}
```

Who did Jack *t* see *t*?

In tree structures, the *pst-node* macro `\nccurve` is particularly useful. **Careful**: Nodes refer to named structures defined by *pst-node* commands, not to tree nodes.
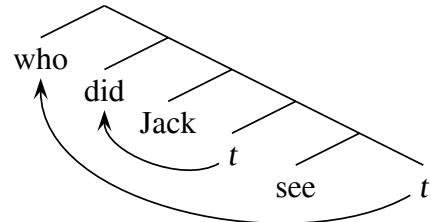
```
\jtree
\! = :{\rnode{A1}{who}}
   :{\rnode{B1}{did}} :{Jack}
   :{\rnode{B2}{\sl t}} :{see}
   {\rnode{A2}{\sl t}}.
\psset{arrows=->,angleA=-150,
   angleB=-90}
\nccurve{A2}{A1}
\nccurve{B2}{B1}
\endjtree
```

who
  did
    Jack
      t
        see   *t*

---

jTree facilitates the use of *pst-node* by means of what are called @tags, as illustrated below.

```
\jtree
\! = :{who}@A1 :{did}@B1 :{Jack}
   :{\sl t}@B2 :{see} {\sl t}@A2 .
\psset{arrows=->,angleA=-150,
   angleB=-90}
\nccurve{A2}{A1}
\nccurve{B2}{B1}
\endjtree
```

who
  did
    Jack
      t
        see   *t*
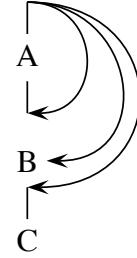
A jTree @tag consists of @ followed by any sequence of characters that is a valid PSTricks node name, followed by a space. If a @tag follows a label (parameters
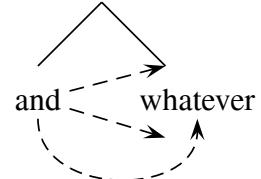
are part of the label), then three nodes are created. If the @tag is @P2, for example, point nodes named P2:t, P2, and P2:b are created. P2:t is at the point that was current before the label was added to the structure, and P2:b is at the point that becomes the current point after the label is added to the structure. P2 is a box node consisting of the label box, with the reference point at its center. With respect to P2, {*stuff*}@P2 is equivalent to {\rnode{P2}{*stuff*}}. If a @tag does not follow a label, then a single point node is created at the current position $\mathcal{P}$.

These ideas are illustrated below:

```
\jtree
\! = @AA
   <vert>{A}
   <vert>{B}[labelgapt=12pt]@P2
   <vert>{C}.
\psset{arrows=->,angleA=0,angleB=0,
   ncurv=1.4}
\nccurve[nodesep=0]{AA}{P2:t}
\nccurve[nodesepA=0]{AA}{P2}
\nccurve[nodesep=0,ncurv=1.6]{AA}{P2:b}
\endjtree
```

```
\jtree[scaleby=1 2,labelgap=1.2ex]
\! = :{and}@A
   {whatever}[labeloffset=1em]@AA .
\psset{linestyle=dashed,arrows=->}
\ncline{A}{AA:t}
\ncline{A}{AA:b}
\nccurve[angleA=-90,angleB=-90,ncurv=1]{A}{AA}
\endjtree
```

There is some further information about nodes and the use of \nccurve in Section 13.

8. The colon construction in more detail

There are some details of the syntax of the colon construction that have been left vague to this point. The idea of the colon construction is simple enough; the colon is replaced by the branch <colonA> and then, after some material has been processed, ^<colonB> is inserted. But some precision is needed in specifying exactly where this later insertion takes place. For example, where is ^<colonB> inserted in the expression below?

    :{a} [labelgap=0] @AA !a @BB {c}
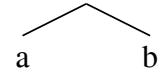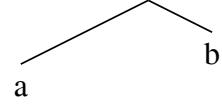
The rule is this. The template below is filled out a fully as possible, then : is replaced by <colonA> and ^<colonB> is inserted after the target.

$$
(10) \qquad : \quad [\textit{pars}] \quad \{\textit{stuff}\} \quad [\textit{pars}] \quad @\textit{tag} \quad \underbrace{\begin{pmatrix} (\dots) \\ !\textit{tag} \end{pmatrix}}
$$

$$
\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{target}}
$$

There are two restrictions. First, the target cannot be completely empty. :<left>... or ::..., for example, is impossible. Second, since parameters must be parameters of something, the [*pars*] term in the target can only be present if there is a {*stuff*} term.
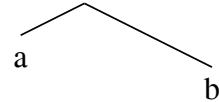
Consider the following examples:

(11) a. \jtree
          \! = :[scaleby=2]{a} {b}.
          \endjtree

      b. \jtree
          \! = :{a} [scaleby=2]{b}.
          \endjtree

You might expect the right branch to be scaled in (11b). The reason that it is not is that [scaleby=2] goes into the target. It is parsed, according to (10), as label parameters and <colonB> is inserted after the target. As label parameters, it has no effect. In Section 14, the following idiom is frequently used to ensure that parameters are not interpreted as label parameters.

(12)   \jtree
          \! = :{a}() [scaleby=2]{b}.
          \endjtree
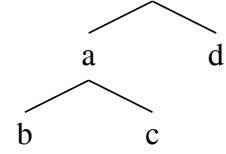
The template (10) is filled out with target {a}(), so <colonB> is inserted before [scaleby=2].
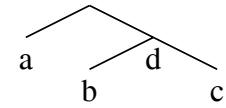
The following contrast is interesting. In (13a), the target is {a}(:{b}{c}), so inline adjunction occurs on the left branch. In (13b), on the other hand, !a fills the

adjunction slot in the template, so the target is {a}!a and the adjunction is on the right branch.

(13) a. \jtree
          \! = :{a} (:{b}{c}) {d}.
          \endjtree

      b. \jtree
          \! = :{a}!a (:{b}{c}){d}.
          \endjtree

Note in (13b) that the inline adjunction leaves the current point $\mathcal{P}$ unchanged, so it is overwritten by the label which follows.

A target can consist entirely of a @tag.

(14) \jtree
      \! = :@A @B .
      \nccurve[angleA=-60,
          angleB=240]{->}{A}{B}
      \endjtree

For the sake of completeness, a full account of the tree description grammar follows.

8.1. The syntax of the tree description language

The well formed tree descriptions are specified by a context free grammar and some assumptions about how the parser operates. We first consider the grammar. The term *string* which occurs in a number of expansions is a string of characters which can be put into a Tex control sequence. Various conditions on the strings which can appear in different contexts are discussed below.

$\mathit{tree\_description} \;\rightarrow\; \mathit{simple\_tree\_description}\;(\,\mathit{tree\_description}\,)$

$\mathit{simple\_tree\_description} \;\rightarrow\; \boxed{\backslash!}\,\mathit{string}\,\boxed{\ }\,\boxed{=}\;(\,\mathit{tree\_body}\,)\;\boxed{.}$

$\mathit{tree\_body} \;\rightarrow\; \mathit{tree\_item}\;(\,\mathit{tree\_body}\,)$

$\mathit{tree\_item} \;\rightarrow\; \mathit{sprout},\;\mathit{target},\;\mathit{vartri\_group},\;\mathit{colon\_group},\;\mathit{operator}$

$\mathit{sprout} \;\rightarrow\; \boxed{<}\mathit{string}\boxed{>}\;(\,\boxed{[}\,\mathit{pars}\,\boxed{]}\,)$

$\mathit{target} \;\rightarrow\; \left(\begin{array}{c}\mathit{label}\\ \mathit{@tag}\end{array}\right)\left(\begin{array}{c}!\mathit{tag}\\ \mathit{inline\_adjunction}\end{array}\right)$

$\mathit{label} \;\rightarrow\; \boxed{\{}\,\mathit{stuff}\,\boxed{\}}\;(\,\boxed{[}\,\mathit{pars}\,\boxed{]}\,)\;(\,\mathit{@tag}\,)$

$\mathit{inline\_adjunction} \;\rightarrow\; \boxed{(}\,\mathit{tree\_body}\,\boxed{)}$

$\mathit{vartri\_group} \;\rightarrow\; \boxed{<}\mathit{string}\boxed{>}\;(\,\mathit{parameters}\,)\;\mathit{label}$

$\mathit{colon\_group} \;\rightarrow\; \boxed{:}\;(\,\boxed{[}\,\mathit{pars}\,\boxed{]}\,)\;\mathit{target}\;(\,\boxed{[}\,\mathit{pars}\,\boxed{]}\,)$

$$@tag \quad \rightarrow \quad \boxed{@}\mathit{string}\,\square$$

$$!tag \quad \rightarrow \quad \boxed{!}\mathit{string}\,\square$$

$$operator \quad \rightarrow \quad \boxed{\char94}$$

$$pars \quad \rightarrow \quad \text{valid PSTricks parameter settings}, \;\; \emptyset$$

The boxed characters above are symbols in the tree description language, not the grammar description language. "Stuff" is anything that will go in a Tex \hbox. Note that an empty parameter specification [] is permitted, and is actually useful at times. This is an extension of the PSTricks parameter system, which disallows empty parameters.

The string which follows \! must be such that *!string* is the name of an adjunction point that has already been defined, either by \jtree or by a !tag in a previously parsed target. The *<string>* which occurs in a sprout must be the name of a branch or triangle, and the *<string>* which occurs in a vartri group must be the name of a vartriangle. A name with an empty string or a string containing spaces is permitted, but not advised. The string which occurs in a @tag must be a valid PSTricks node name, with the additional restriction that it cannot contain a space. The string which occurs in a !tag, preceded by !, becomes the name of an adjunction point. It cannot contain a space. Note that !tags and @tags must be followed by a space. Otherwise, jTree is tolerant of spaces or their absence between items.

Parsing is unique under the assumption that targets and labels are always maximized in left to right parsing and are never empty. Labels are never empty in any event since they contain { and }.

9. Expansion and evaluation of control sequences in tree parsing
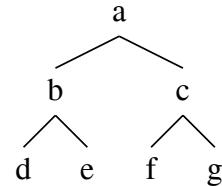
9.1. The " escape from tree parsing

If the parser encounters ", it evaluates the next token or group and continues the parse. Evaluation should contribute no material, since it is not parsed. But evaluation can change parameter settings, which affects how the remaining material is typeset. For example:

```
\jtree
\! = {a} :{b}!! {c}
   "{\psset{scaleby=.5 1}} :{f} {g}.
\!! = :{d} {e}.
\endjtree
```
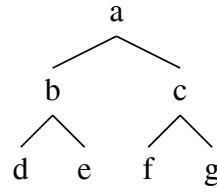
\! ... . does not establish an implicit group, so the change in scale persists to the subsequent subtree.

The same effect can be achieved without using " if rescaling is done outside tree parsing.
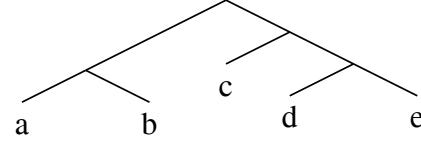
```
\jtree
\! = {a} :{b}!a {c}!b .
\psset{scaleby=.5 1}
\!a = :{d} {e}.
\!b = :{f} {g}.
\endjtree
```

9.2. Control sequence expansion
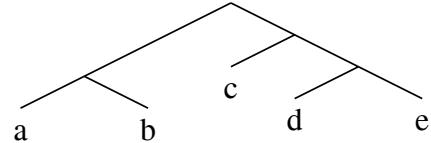
If the parser encounters a control sequence or active character in a tree description, it is replaced by its expansion before the parse continues. The expansion is parsed without further expansion of the initial token, which prevents an infinite loop if the expansion yields an unexpandable control sequence.

```
\jtree
\def\Colon{:[scaleby=2.3]}%
\! = \Colon !a :{c} :{d} {e}.
\!a = :{a} {b}.
\endjtree
```

---

pst-jtree contains the definition \def\jtlong{[scaleby=2.3]}, so we could also write:

```
\jtree
\! = :\jtlong !a :{c} :{d} {e}.
\!a = :{a} {b}.
\endjtree
```

This technique makes some code much more transparent. (6) on page 8, for example, can be written:

```
\jtree[xunit=2.2em,yunit=1em]
\! =  :\jtlong !a :{is} {rotten}.
\!a = :{the} :{cheese} :{that} :\jtlong !b :{ate} {\it t}.
\!b = :{the} :{rat} :{that} :\jtlong !c :{killed} {\it t}.
\!c = :{the} :{cat} :{that} :{John} :{owned} {\it t}.
\endjtree
```

The complete list of parameter changes that are encoded in macros in *pst-jtree* is:
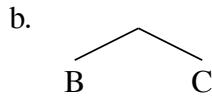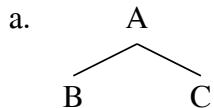
```
\def\jtlong{[scaleby=2.3]}
\def\jtshort{[scaleby=.5]}
\def\jtwide{[scaleby=2 1]}
\def\jtbig{[scaleby=2]}
\def\jtjot{[scaleby=1.3]}
```

Of course, users should define whatever similar macros are useful in their own work.
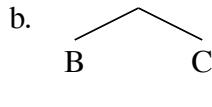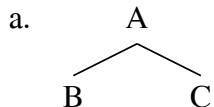
10. Bells and whistles

10.1. *baretopadjust*

The baseline of the box created by `\jtree...\endjtree` is normally the baseline of the root label. If, however, the root label is empty, that puts the baseline too low, at least for my taste. Contrast the trees below:

a.   A   b.
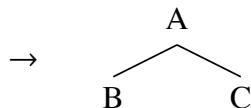
jTree takes corrective action by raising the tree diagram if the root label is empty by the amount specified by the parameter `baretopadjust`. The default setting is 1.4ex, but you can set it to whatever you want. With the default setting:

a.   A   b.

10.2. *treevshift*

jTree also provides the parameter `treevshift`, which is set to 0 by default. It is independent of `baretopadjust` and can be used to move the tree diagram up and down, if desired.

```
$\to$\qquad                                          A
\jtree[treevshift=1.2em]              →
\! = {A} :{B} {C}.                             B        C
\endjtree
```

10.3. *everytree*

If you say, for example, `\psset{everytree=\psset{style=treestyle}}`, and if you have defined `treestyle` using a PSTricks style definition, then your
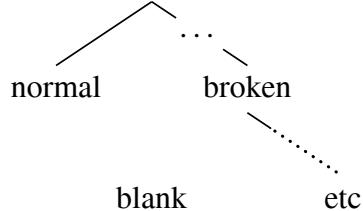
trees will done in that style. `\psset{everytree=x}` puts *x* into the token list `\jteverytree` which is inserted at the beginning of every `\jtree...\endjtree` construction. It is grouped, so that its scope is limited to the tree construction. It is inserted before the `\jtree` parameters take effect, so that it will be overridden by parameter settings.

If you want, you can set `\jteverytree` directly. The same is true, by the way, for `everylabel`, which uses the token list `\jteverylabel`.

10.4. Custom branches

jTree ordinarily draws branches using the PSTricks macro `\psline`, but it does this in an indirect way. It actually calls the macro `\branch@type` to do the drawing; and jTree contains the line `\let\branch@type=\psline`. If the user says `\psset{branch=\customline}`, then `\branch@type` is made `\customline` instead of `\psline` with the usual locality. If `\customline` is suitably defined then it will be used to draw branches. A few alternative macros for drawing branches are included in jTree. The enterprising user might want to define appropriate zigzag or coil macros, following the PSTricks model. jTree provides `\blank` (see example 10 in Section 14 for an illustration of its use), `\brokenbranch` (see example 14 in Section 14), and `\etcbranch` (see example 13 in Section 14).

```
\jtree[xunit=3em,yunit=2em]
\! = :{normal}()
        [branch=\brokenbranch]{broken}
   :[branch=\blank]{blank}()
      [branch=\etcbranch]{etc}.
\endjtree
```

The proportion of the "etc branch" that is dotted is controlled by a parameter `etcratio`. *pst-jtree* contains `\psset{etcratio=.75}`, but the user is free to change this if desired. The style for the dotted portion is defined by

```
\newpsstyle{etc}{nodesepB=0,nodesepA=1pt,linestyle=dotted,
    linewidth=1.2pt,dotsep=2pt}
```
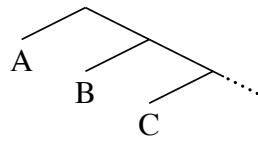
The user can also overwrite the specification of this style with a new specification, if desired.

pst-jtree contains the macro definition:

```
\def\etc{[branch=\etcbranch,scaleby=.7]}
```

So you can write:

```
\jtree
\! = :{A} :{B} :{C}() \etc.
\endjtree
```
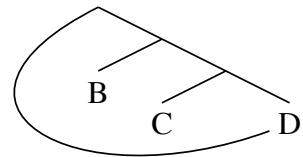
10.5. The pseudo-parameters *dirA* and *dirB*

Suppose that you wish to use `\nccurve` to draw a curve which leaves a node A in the same direction that a standard left branch would leave A. You need to set `angleA` to the appropriate value. The appropriate angle can be calculated using some simple trigonometry, but the calculation depends on the ratio of the psxunits to the psyunits. It is desirable both to avoid having to do a trig calculation on the side and to code a tree so that unit changes do not alter the geometry in an undesirable fashion. `dirA` and `dirB` were introduced to solve this problem. `\psset{dirA=(-1:-1)}` will cause `angleA` to be set so that a path drawn by `\nccurve` will leave the starting node in the direction of the vector $(-1, -1)$. It is called a pseudo-parameter (my terminology) because `\psset{dirA=`x`}` is executed for its effect on the parameter `angleA`, not `dirA`. Note that a colon is used, so that the `\psset` parser is not confused. `dirB` works almost the same way for `angleB`, but the vector which sets `dirB` should point backwards along the curve, which is the direction which `angleB` measures.

Use of `dirA` below ensures that the "curved branch" lines up with the straight ones. Note that the curve is made fairly stiff (a high value of `ncurv`) so that it bows out sufficiently.

```
\jtree
\! = @A1
   <right>
   :{B}
   :{C} {D}@A2 .
\nccurve[dirA=(-1:-1),angleB=200,
   ncurv=2,nodesepA=0]{A1}{A2}
\endjtree\kern1em
```
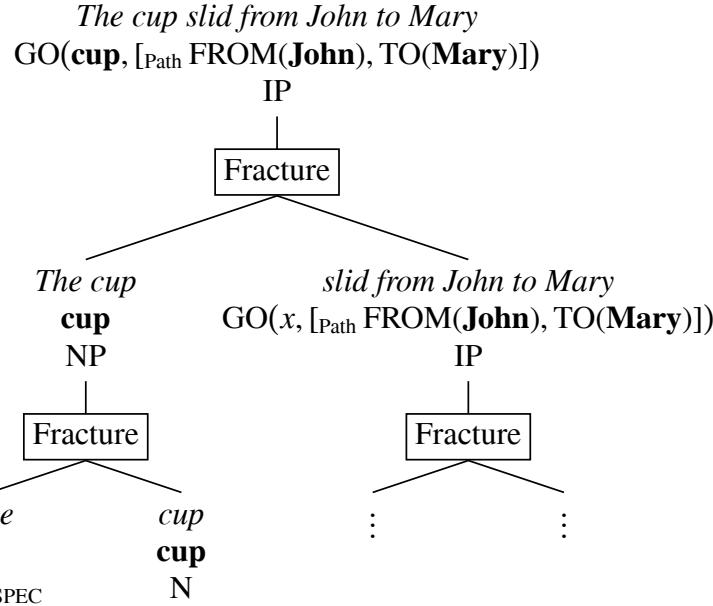
The example in 14 is a good illustration of the use of this parameter.

11. How to build complex trees

jTree has several features which make it easy to build trees incrementally. In complex trees, this is a big advantage, because the software does not make it easy to pinpoint the source of errors in the code. Consider, for example, the tree that is used to illustrate the features of `qtree`, a popular tree-drawing macro package (available on CTAN).

(15)　　　　　*The cup slid from John to Mary*
　　　　　GO(**cup**, [$_{\text{Path}}$ FROM(**John**), TO(**Mary**)])

$$\begin{array}{c}\text{IP}\\|\\\boxed{\text{Fracture}}\end{array}$$

The cup
cup
NP

slid from John to Mary
GO(x, [$_{\text{Path}}$ FROM(**John**), TO(**Mary**)])
IP

$\boxed{\text{Fracture}}$ 　　　 $\boxed{\text{Fracture}}$

The
$\perp$
N_{SPEC}

cup
cup
N

$\vdots$ 　 $\vdots$
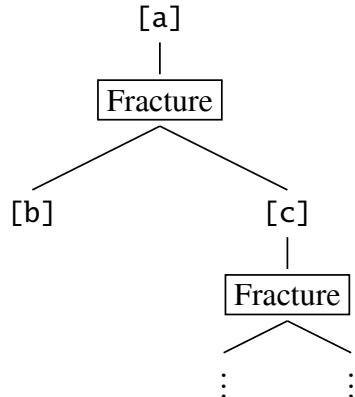
We proceed, as usual, by starting with the right branching spine of the tree, putting in adjunction points for complex material on left branches that must be added. Similar to adjunction points, jTree allows for tags to be inserted for "stuff" that can be filled in later. Macros `\stuff` and `\defstuff` are defined in *pst-jtree* and used as shown below.

```
\jtree
\def\fracture{\psframebox{Fracture}}
\! = {\stuff[a]}
   <vert>{\fracture}
   :\jtbig{\stuff[b]}!a
      \jtbig{\stuff[c]}
   <vert>{\fracture}
   :{$\vdots$} {$\vdots$}.
\endjtree
```

[a]
|
$\boxed{\text{Fracture}}$

[b]　　　　　[c]
|
$\boxed{\text{Fracture}}$

$\vdots$　　$\vdots$
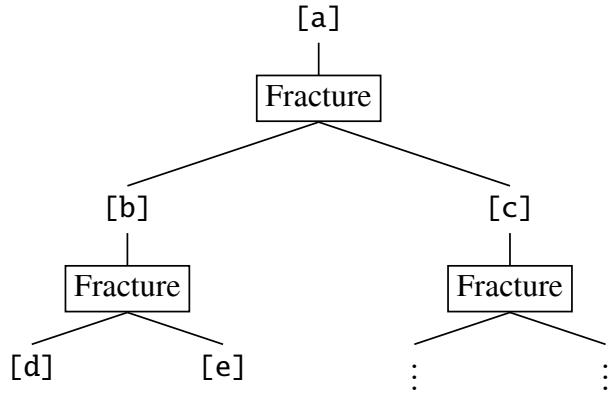
We then proceed to complete the structure at the tagged positions, fill in the missing stuff, and to make whatever other changes are needed to get a good looking tree. Some things are already apparent: there is too much whitespace under "Fracture" inside the box; the tree branches do not meet the "fracture box", and the tree needs to be stretched in the x-direction. We delay filling in most of the missing stuff until the end, and concentrate on fixing the problems noted above and getting the tree structure right. This produces:

```
\jtree[xunit=3em,yunit=1em]
\def\Fracture{<vert>{\omit\psframebox[boxsep=.4ex]
   {Fracture$\vphantom j$}}}%
\! = {\stuff[a]}
   \Fracture
   :\jtbig{\stuff[b]}!a
      \jtbig{\stuff[c]}
   \Fracture
   :{$\vdots$}
      {$\vdots$}.
\!a = \Fracture
   :{\stuff[d]}
      {\stuff[e]}.
\endjtree
```

We now are ready to fill in the stuff. Since the go-path construction is complicated, appears twice, and is probably useful for this kind of work, it has been generalized to a macro. *pst-jtree* contains the \multiline and \endmultiline macros to help construct complex multiline labels. \multiline starts a vbox and begins \halign{\hfil#\hfil\cr. \endmultiline closes up the construction. Some care is taken with the vertical spacing. The full code for (15) is:

```
\def\GO#1#2#3{$\rm GO\bigl({#1},[_{Path}\,
   FROM({#2}),TO({#3})]\bigr)$}
```

```
\jtree[xunit=3em,yunit=1em]
\defstuff[a]{\multiline
   \it The cup slid from John to Mary\cr
   \GO{\bf cup}{\bf John}{\bf Mary}\cr
   IP\endmultiline}
```

```
\defstuff[b]{\multiline
   \it The cup\cr
   \bf cup\cr
   NP\endmultiline}
\defstuff[c]{\multiline
   \it slid from John to Mary\cr
   \GO{\mit x}{\bf John}{\bf Mary}\cr
   IP\endmultiline}
\defstuff[d]{\multiline
   \it The\cr
   $\perp$\cr
  \quad N$\rm _{SPEC}$\endmultiline}
\defstuff[e]{\multiline
   \it cup\cr
   \bf cup\cr
   N\endmultiline}
\def\Fracture{<vert>{\omit\psframebox[boxsep=.4ex]
   {Fracture$\vphantom j$}}}%
\! = {\stuff[a]}
   \Fracture
   :\jtbig{\stuff[b]}!a \jtbig{\stuff[c]}
   \Fracture
   :{$\vdots$} {$\vdots$}.
\!a = \Fracture
   :{\stuff[d]} {\stuff[e]}.
\endjtree
```
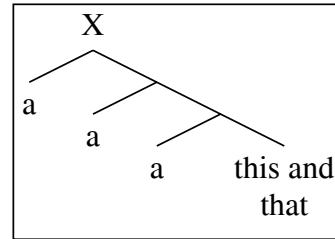
12. The bounding box

PSTricks creates dimensionless graphics, but jTree goes to a lot of trouble to figure out the sizes of the trees that it generates and to put them in appropriately sized boxes. For example:

```
\psframebox[boxsep=0]{\jtree
\! = {X} :{a} :{a} :{a}
   {\multiline
      this and\cr
      that\endmultiline}.
\endjtree}
```
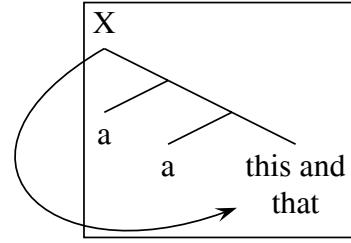
The sizing is not perfect. jTree is not clever enough to recognize the white space due to `labelgapt`, `labelgapb`, `labelstrutt`, and `labelstrutb`. But it is not bad.
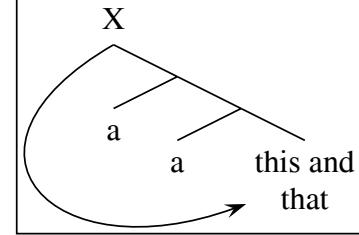
If PSTricks is used to draw arrows, they often extend outside the jTree bounding box.

```
\psframebox[boxsep=0]{\jtree
\! = {X}@A1
   <right>
   :{a}
   :{a}
      {\multiline
         this and\cr
         that\endmultiline}@A2 .
\nccurve[angleA=210,angleB=200,
   ncurv=2,nodesepA=0]{->}{A1:b}{A2}
\endjtree}
```

This has to be fixed by hand by inserting appropriate kerning.

```
\psframebox[boxsep=0]{\kern2.4em
\jtree
\! = {X}@A1
   <right>
   :{a}
   :{a}
      {\multiline
         this and\cr
         that\endmultiline}@A2 .
\nccurve[angleA=210,angleB=200,
   ncurv=2,nodesepA=0]{->}{A1:b}{A2}
\endjtree}
```

---

13. Nodes and connections between them

Nodes have a shape, a reference point, and a name. *pst-node* allows the user to define box, elliptical, circular, and point nodes. In addition to a shape, nodes have a reference point. It is at the center of an elliptical, circular, or point node. It can be at the center of a box node, but there are other options for box nodes: the ends and centers of the edges and the baseline. *pst-node* has a number of commands for drawing curves of various kinds between nodes. Various features of how these curves are rendered (linewidth, linestyle, arrows, etc.) can be specified. The most useful curve drawing command for annotating tree structures is \nccurve. The main point of this section is to explain how \nccurve works.
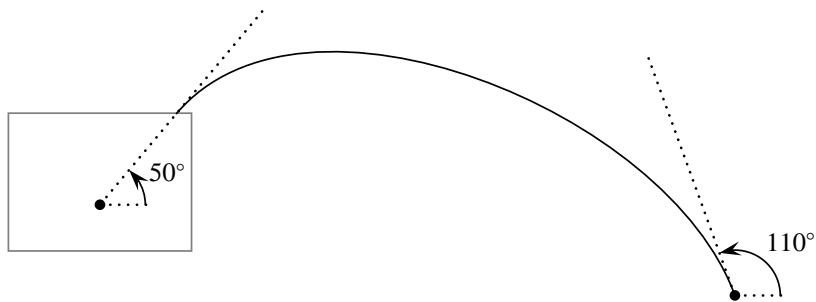
Suppose there is a box node and a point node as shown below. The dots are the reference points.

A

B

The diagram below illustrates how
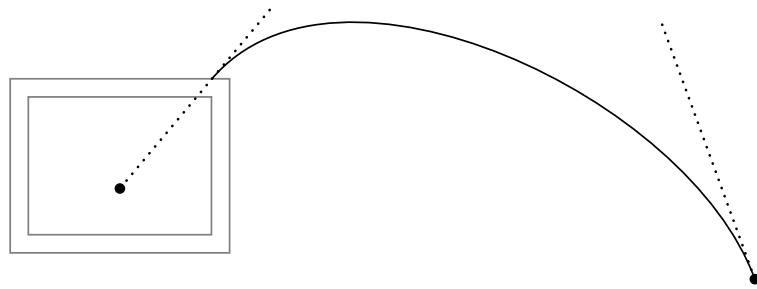
\nccurve[angleA=50,angleB=110]{A}{B}

is drawn.

50°

110°
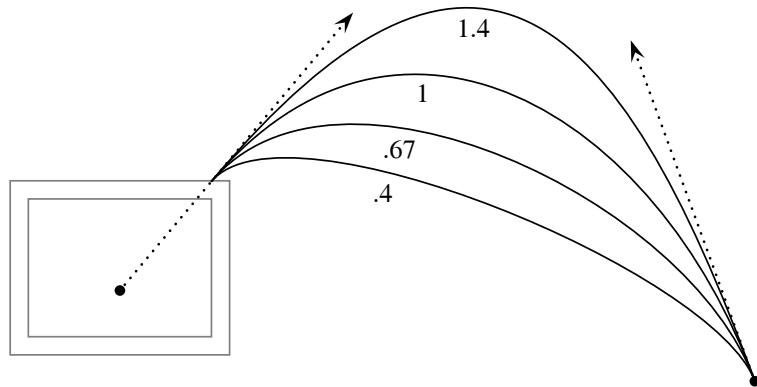
This simple picture can be modified by a number of parameters. In addition to the usual parameters like linewidth, linestyle and arrows which determine how a geometrical curve is rendered, there are six parameters which modify the geometry of the curve itself: ncurvA, ncurvB, nodesepA, nodesepB, offsetA, and offsetB. The parameters can be set individually or in pairs. \psset{nodesep=x} induces \psset{nodesepA=x,nodesepB=x}. ncurv and offset work the same way. We will examine the parameters in turn.

\psset{nodesepA=x} causes the dimensions of the node box to be adjusted by x before the curve is determined.
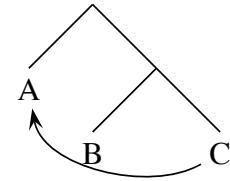
The effect of ncurv is the most subtle. One can imagine that the curve is "pulled" in the directions of the arrows below. You can think of ncurv as setting the strength of the pulls. The default is .67. The picture below shows the curves that are drawn with various settings of ncurv.
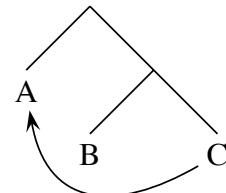
Here is a simple illustration of the use of ncurv. There is a typesetting problem with the pointer below which must be fixed.

```
\jtree[unit=2em]
\! = :{A}@A :{B} {C}@C .
\nccurve[angleA=210,angleB=-80]{->}{C}{A}
\endjtree
```
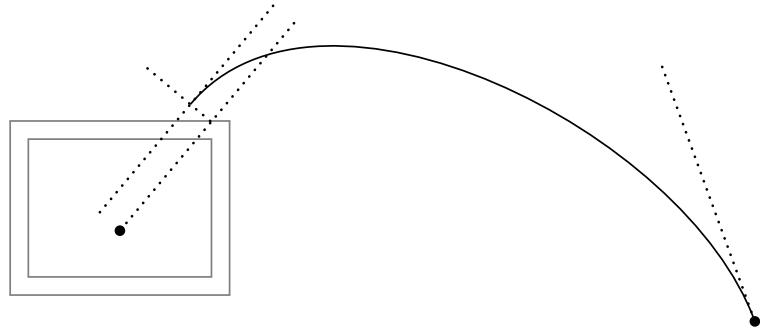
One way to fix the problem is to increase the value of the ncurv parameter.

```
\jtree[unit=2em,nodesep=.6ex]
\! = :{A}@A :{B} {C}@C .
\nccurve[angleA=210,angleB=-80,
   ncurv=1.1]{->}{C}{A}
\endjtree
```

\psset{ncurv=*x*} has the effect \psset{ncurvA=*x*,ncurvB=*x*}. Sometimes it is advantageous to set the parameters to different values. Experimentation will quickly give you a feel for how this works.
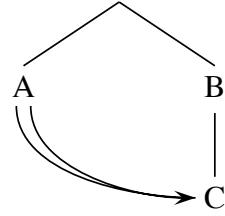
Finally, we come to the two offset parameters. \psset{offsetA=*x*} causes the starting point to be offset a distance *x* perpindicular to the starting direction. The direction is to the left (looking in the starting direction) if *x* is positive.

\psset{offsetB=*x*} causes the finishing point to be displaced a distance *x* perpindicular to the finishing direction, to the left. Remember that the finishing direction points toward the node, not back along the curve.

Here is a simple application that might be appropriate for some special emphasis.

```
\jtree[xunit=3em,yunit=2em]
\! = :{A}@A {B} <vert>{C}@C .
\psset{angleA=-90,angleB=180}
\nccurve[offsetA=.5ex]{->}{A}{C}
\nccurve[offsetA=-.5ex]{A}{C}
\endjtree
```
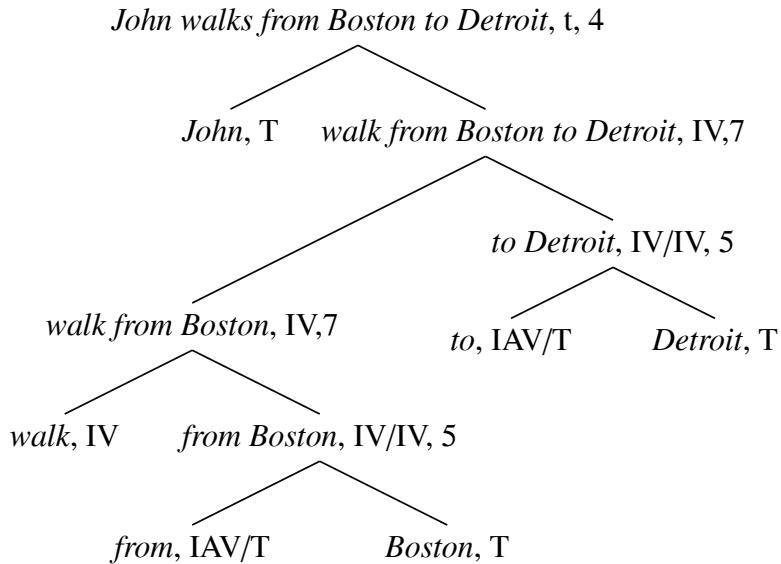
This construction is used several times in example 6 in Section 14.

14. Examples

This section, the last, presents a series of examples with complete code, which illustrate techniques for solving various problems that arise in typesetting complex trees. Examples 1 and 2 illustrate various techniques for solving spacing problems. The remainder of the examples illustrate the cooperation between *pst-jtree* and *pst-node*, with its suite of macros for creating nodes and connecting them in various ways.

Example 1

John walks from Boston to Detroit, t, 4

John, T   *walk from Boston to Detroit*, IV,7

to Detroit, IV/IV, 5

walk from Boston, IV,7

to, IAV/T   *Detroit*, T

walk, IV   *from Boston*, IV/IV, 5

from, IAV/T   *Boston*, T
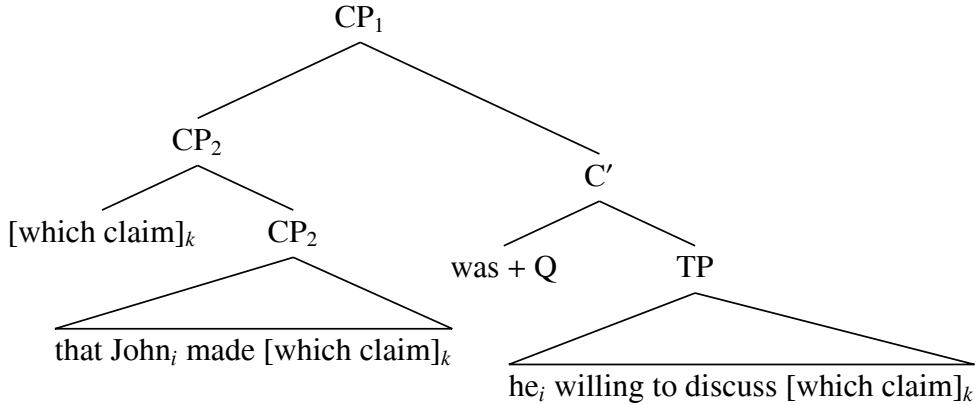
(Dowty, David. 1979. *Word Meaning and Montague Grammar*, p. 215.)

```
\jtree[xunit=4em,yunit=2em,everylabel=\strut\it]
\! = {John walks from Boston to Detroit\rm, t, 4}
   :{John\rm, T} {walk from Boston
                  to Detroit\rm, IV,7}[labeloffset=1.5em] ¹
   :\jtlong{walk from Boston\rm, IV,7}!a
      {to Detroit\rm, IV/IV, 5}
   :[scaleby=.8]{to\rm, IAV/T}() [scaleby=.8]{Detroit\rm, T}. ²
\!a = :{walk\rm, IV} {from Boston\rm, IV/IV, 5}
   :{from\rm, IAV/T} {Boston\rm, T}.
\endjtree
```

1. The label is offset to improve the spacing. The label is off center, but not so much as to be a distraction.

2. The null inline adjunction () keeps [scaleby=.8] from being interpreted as a parameter associated with the preceding label.

Example 2

CP$_1$

CP$_2$

[which claim]$_k$ CP$_2$

C′

was + Q   TP

that John$_i$ made [which claim]$_k$

he$_i$ willing to discuss [which claim]$_k$
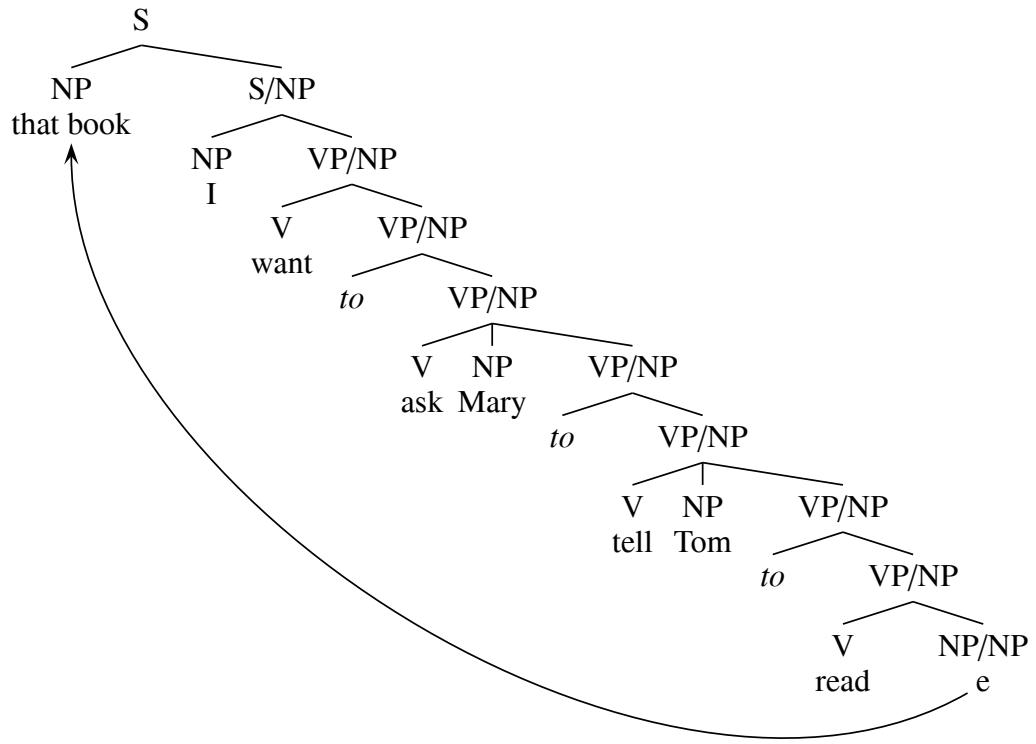
(Nunes, Jairo. 2004. *Linearization of Chains and Sideward Movement*, p. 149.)
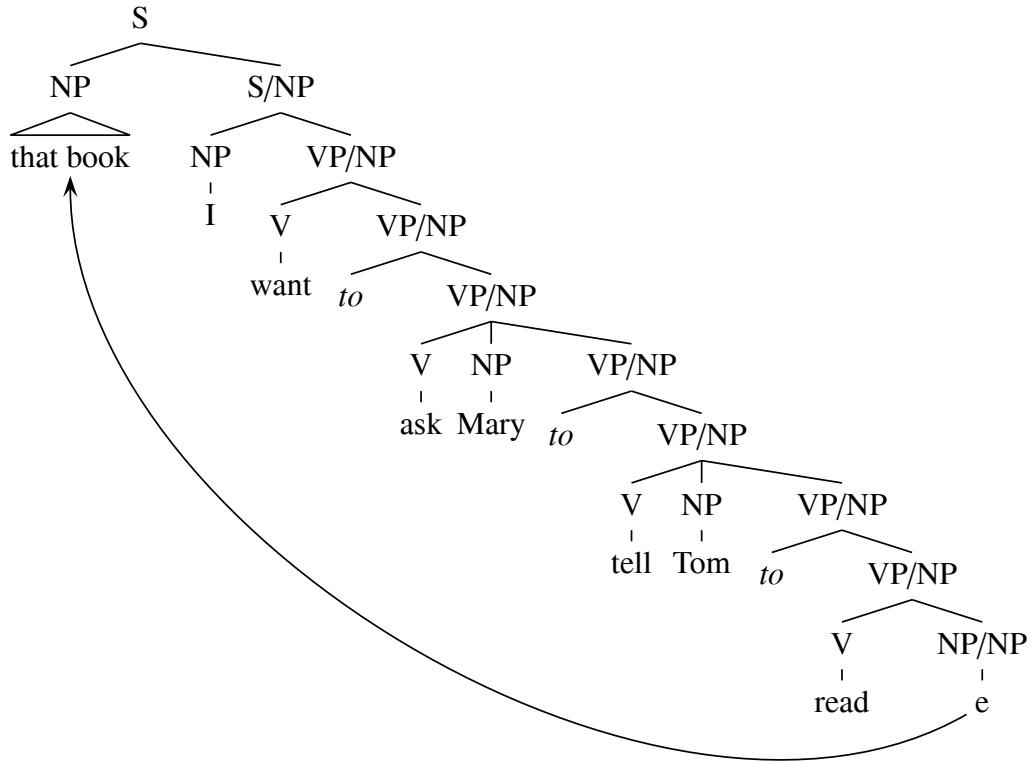
```
\jtree[xunit=3em,yunit=1.4em]
\def\what{[which claim]$_k$}%
\! = {CP$_1$}
   :[scaleby=1.7]{CP$_2$}!a [scaleby=2.5]{C$'$}¹
   :{$\rm was+Q$} {TP}
   <vartri>[scaleby=1.6,triratio=.4]²
      {he$_i$ willing to discuss \what}.
\!a = :{\what} {CP$_2$}
   <vartri>[scaleby=1.6,triratio=.6]²
      {that John$_i$ made \what}.
\endjtree
```

1. The two legs are not scaled equally because the structure is not (linguistically) symmetrical. Keeping the left branch shorter helps visually to identify the high C-specifier.

2. The settings of `triratio` allow for a much more compact display.

```
                    S
          NP                S/NP
       that book        NP      VP/NP
                         I     V      VP/NP
                              want   to      VP/NP
                                         V    NP    VP/NP
                                        ask  Mary  to      VP/NP
                                                      V    NP    VP/NP
                                                     tell  Tom  to      VP/NP
                                                                   V       NP/NP
                                                                  read       e
```
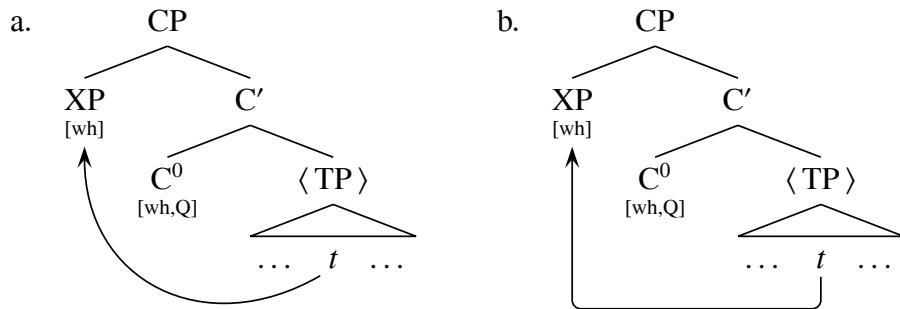
(This is an adaptation of an example in the documentation to Avery Andrew's tree formatting preprocessor.)

Some may prefer the following formatting style. The code is almost identical to the code for the style above. \\ has a different definition and <vartri> is used instead of \\ in one place.

S

NP   S/NP

that book   NP   VP/NP

I

V   VP/NP

want   *to*   VP/NP

V   NP   VP/NP

ask Mary   *to*   VP/NP

V   NP   VP/NP

tell Tom   *to*   VP/NP

V   NP/NP

read   e

Example 3

The contrast illustrates the two main varieties of pointers that PSTricks makes available for relating nodes in a tree. My preference is (a), since it more clearly distinguishes the relation expressed by the pointer from tree relations. Tastes may vary.

a.   CP   b.   CP

XP   C′   XP   C′
[wh]   [wh]

C^0 ⟨ TP ⟩ C^0 ⟨ TP ⟩
[wh,Q]   [wh,Q]

. . .   *t*   . . .   . . .   *t*   . . .

(The version on the right is from a paper by Jason Merchant.)

The code for (a) is:
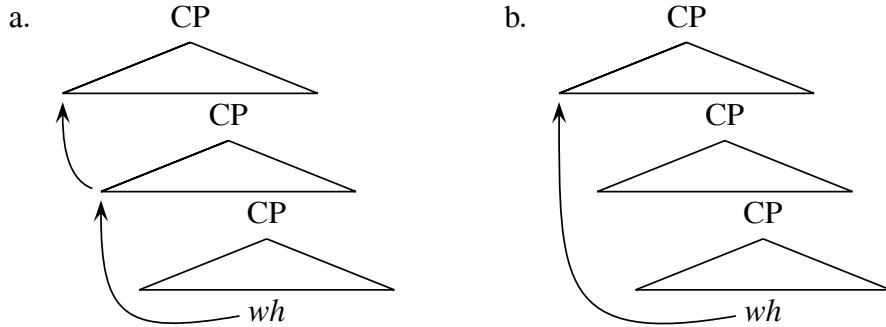
```
\jtree[xunit=2.6em,yunit=1em]
\def\\{[labelgapb=-1.2ex]}%¹
\everymath={\rm}%
\! = {CP}
   :({XP}\\{$\scriptstyle [wh]$}@A1 ) {$C'$}
   :({$C^0$}\\{$\scriptstyle [wh,Q]$})
       {$\langle\, TP\,\rangle$}
   <tri>{\dots\quad\rnode[b]{A2}{\it t}\quad\dots}.²
\nccurve[angleA=-150,angleB=-90,ncurv=1]{->}{A2}{A1}
\endjtree
```

1. \\ is used to close up the gap between category labels and features underneath them.

2. The node A2 is inserted via \rnode rather than by using @A2 because it is not a pointer from the whole label, but from the trace inside the label. \rnode[b] is used, which places the reference point at the bottom of the box that the trace is put in. This works better visually, since it keeps the pointer from coming too close to the ellipsis.

For (b), substitute the following for the \nccurve pointer.

```
\ncbar[angleA=-90,angleB=-90,armA=1em,
   armB=1em,linearc=.6ex]{->}{A2}{A1}
```

Example 4

a. CP   b. CP

[Two syntactic tree diagrams, labeled a. and b., each showing nested CP nodes drawn as triangles with curved movement arrows pointing to a wh element.]
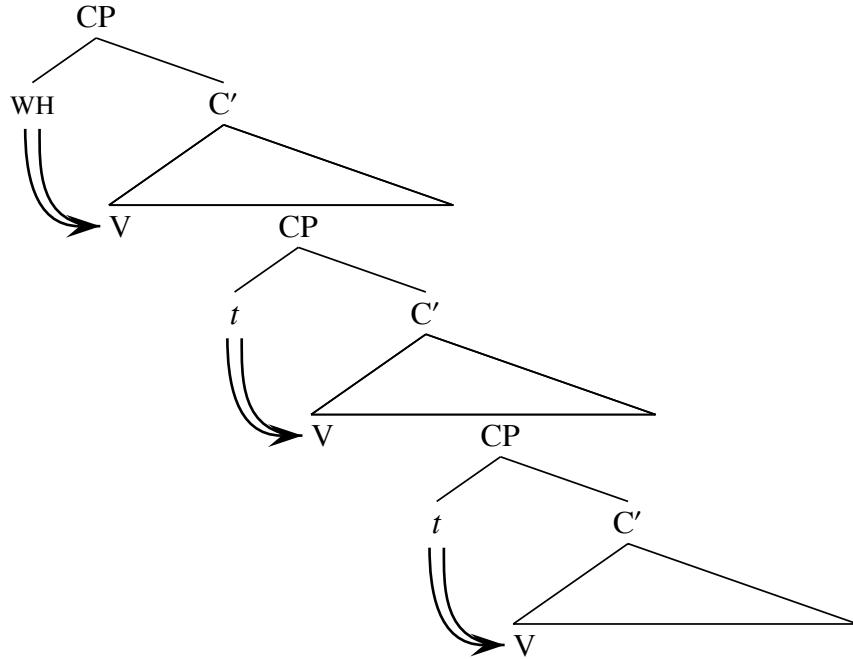
(Chung, Sandy. 1998. *The Design of Agreement*, p. 365.)

```
\jtree
\! = {CP}
   <left>@A1 ^<tri>[triratio=.65]{CP} ¹
   <left>@A2 ^<tri>[triratio=.65]{CP}
   <tri>{\it wh}@A3 .
\psset{angleB=-90,arrows=->}
\nccurve[angleA=190,ncurv=1.3]{A3}{A2}
\nccurve[angleA=160]{A2}{A1}
\endjtree
```

```
\jtree
\! = {CP}
   <left>@A1 ^<tri>[triratio=.65]{CP}
   <tri>[triratio=.65]{CP}
   <tri>{\it wh}@A3 .
\nccurve[angleA=190,angleB=-90,ncurv=1.3]{->}{A3}{A1}
\endjtree
```

1. The <left>...^<tri> construction is used so that the @tag following
<left> and the label following <tri> can be independently positioned. The
triangle simply overwrites the left branch. In other situations, something like
<left>[branch=\blank] might be appropriate.

Example 5

CP

WH   C′

V   CP

t   C′

V   CP

t   C′

V

(Chung, Sandy. 1998. *The Design of Agreement*, p. 246.)

```
\jtree[xunit=2em,yunit=1.4em,labelgapb=0,triratio=0] 1
\deftriangle<tri>(1.8)(1)(-.5)
\defbranch<colonB>(1)(-.5)
\! = {CP}
   :{\sc WH}@A {C$'$}
   <tri>{\rlap{V}}@AA ^<tri>[triratio=.55]{CP} 2
   :{\it t}@B {C$'$}
   <tri>{\rlap{V}}@BB ^<tri>[triratio=.55]{CP}
   :{\it t}@C {C$'$}
   <tri>{\rlap{V}}@CC .
\psset{linewidth=1pt,ncurvB=1.1,nodesepA=1ex,
   angleA=-90,angleB=180,offsetA=.5ex}
\nccurve{A}{AA}
\nccurve{B}{BB}
\nccurve{C}{CC}
\psset{offsetA=-.5ex,arrows=->}
\nccurve{A}{AA}
\nccurve{B}{BB}
\nccurve{C}{CC}
\endjtree
```

1. Since all the node labels are uppercase, and therefore do not descend below the baseline, spacing is improved by `labelgapb=0`. `triratio` is set to 0 in order to properly position the instances of V.
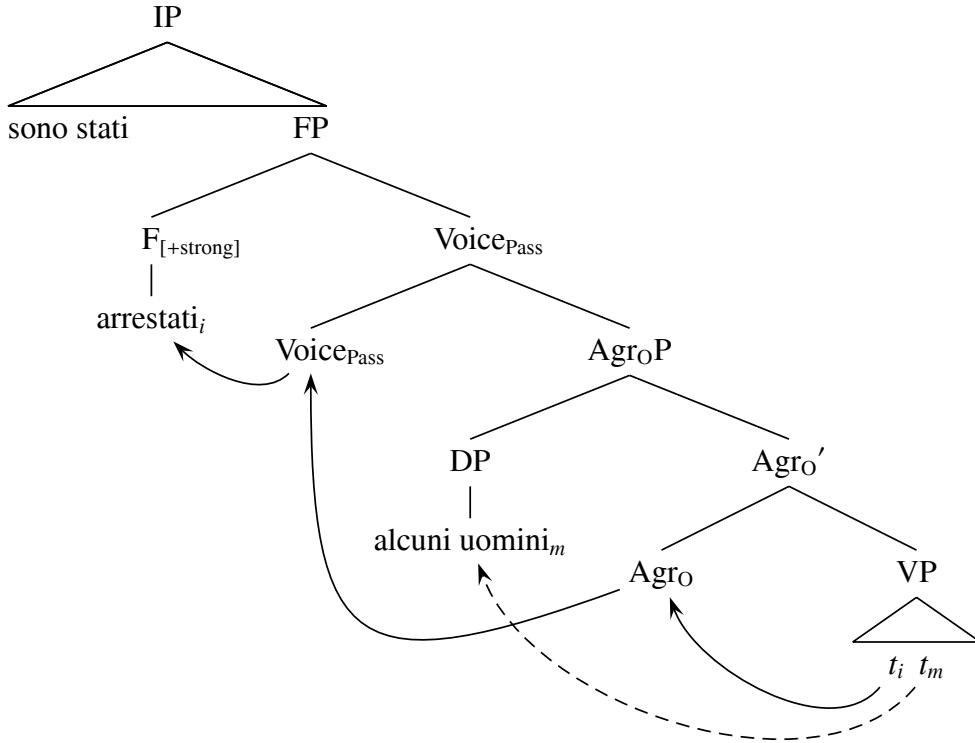
2. The triangle is overwritten so that both V and CP can be positioned independently, using different values of triratio.

Here is a less elegant approach which achieves the same effect without using an offset.

```
\jtree[xunit=2em,yunit=1.4em,labelgapb=0,triratio=0]
\deftriangle<tri>(1.8)(1)(-.5)
\defbranch<colonB>(1)(-.5)
\def\gap{\hskip1ex}%
\def\\{[labelstrutb=0]}%
\! = {CP}
   :{\sc WH}\\({\pnode{A1}\gap\pnode{A2}}) {C$'$}[1]
   <tri>{\rlap{V}}@A ^<tri>[triratio=.55]{CP}
   :{\it t}\\({\pnode{B1}\gap\pnode{B2}}) {C$'$}
   <tri>{\rlap{V}}@B ^<tri>[triratio=.55]{CP}
   :{\it t}\\({\pnode{C1}\gap\pnode{C2}}) {C$'$}
   <tri>{\rlap{V}}@C .
\psset{linewidth=1pt,ncurvB=1.1,nodesepA=1ex,
   angleA=-90,angleB=180}
\nccurve{A1}{A}
\nccurve{B1}{B}
\nccurve{C1}{C}
\psset{arrows=->}
\nccurve{A2}{A}
\nccurve{B2}{B}
\nccurve{C2}{C}
\endjtree
```

1. It is necessary to set labelstrutb to 0 (via \\) so that the inline adjunction is to the bottom of the label box. labelgapb is 0 throughout the tree.

Example 6

IP

sono stati   FP

F_{[+strong]}   Voice_{Pass}

arrestati_{i}   Voice_{Pass}   Agr_{O}P

DP   Agr_{O}′

alcuni uomini_{m}   Agr_{O}   VP

t_{i} t_{m}
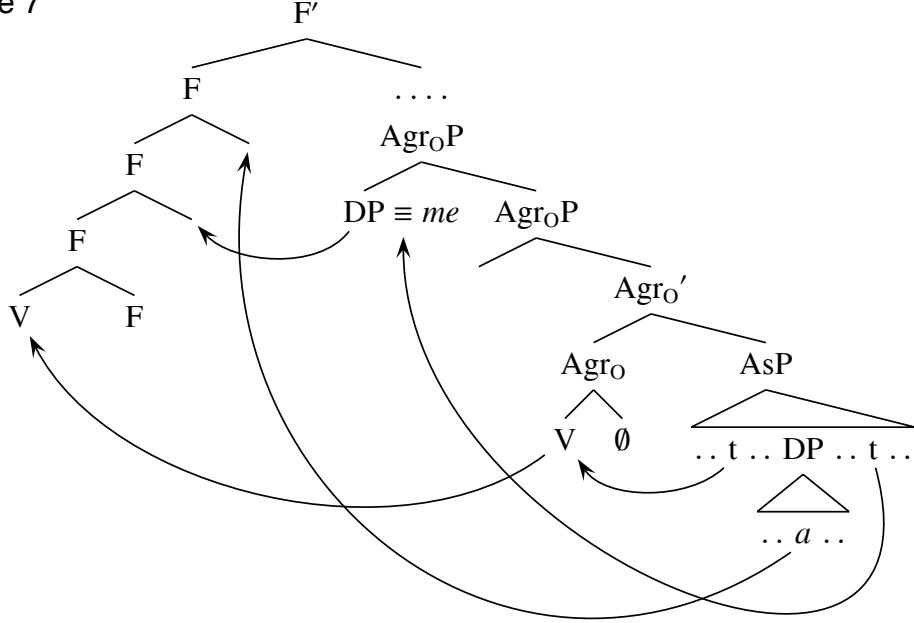
(Caponigro, Ivano and Carson Schütze. 2003. Parameterizing Passive Participle Movement, *Linguistic Inquiry* 34.2, p. 300.)

```
\jtree[xunit=5em,yunit=2em]
\! = {IP}
   <tri>{\triline{sono stati\hfil}}  ^<tri>[triratio=.95]{FP}
   :{F$_{\rlap{$\scriptstyle\rm [+strong]$}}$}!a
      {Voice$_{\rlap{$\scriptstyle\rm Pass$}}$}
   :{Voice\rlap{$_{\rm Pass}$}}@A2  {$\rm Agr_OP$}
   :{DP}!b  {${\rm Agr_O}'$}
   :[scaleby=.8 1]{$\rm Agr_O$}@A3  [scaleby=.8 1]{VP}
   <tri>[scaleby=.4 .7]
      {\rnode{A5}{$t_i$}\hskip1ex \rnode{A6}{$t_m$}}.
\!a = <shortvert>{arrestati$_i$}@A1 .
\!b = <shortvert>{alcuni uomini$_m$}@A4 .
\psset{arrows=->}
\nccurve[angleA=225,angleB=-45]{A2}{A1}
\nccurve[angleA=200,angleB=-90,ncurv=1.5]{A3}{A2}
\nccurve[angleA=-130,angleB=-70]{A5}{A3}
\nccurve[angleA=-130,angleB=-70,linestyle=dashed]{A6}{A4}
\endjtree
```

44

Example 7

F′

F . . . .

F Agr$_O$P

F DP ≡ *me* Agr$_O$P

V F Agr$_O$′

Agr$_O$ AsP

V ∅ . . t . . DP . . t . .

. . *a* . .
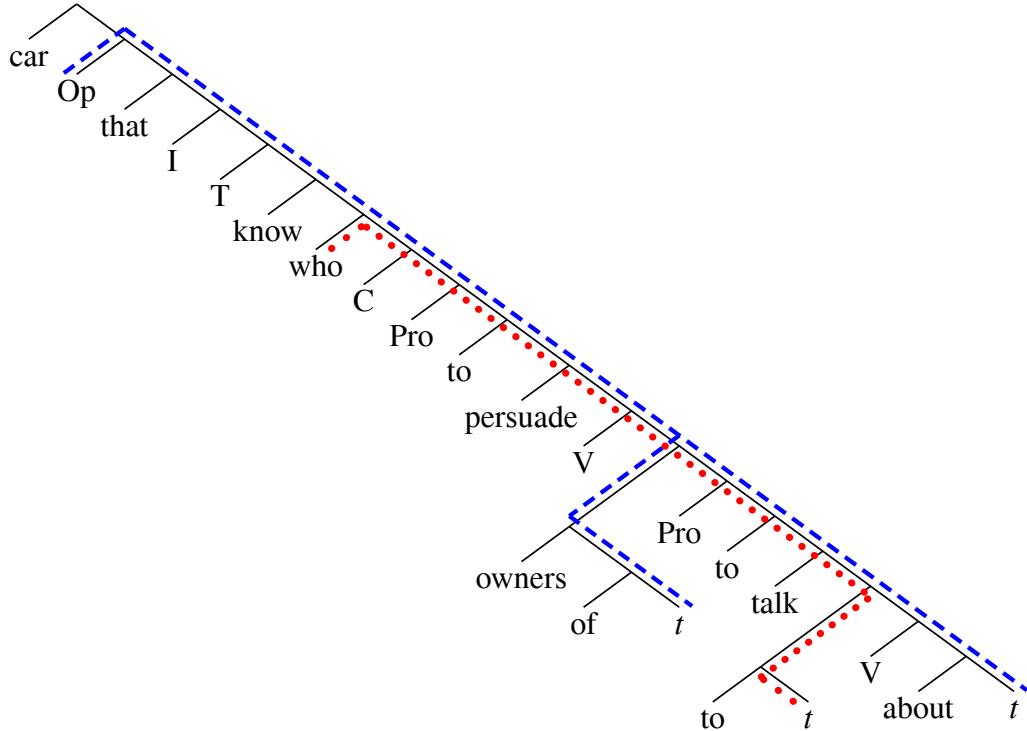
(Uriagereka, Juan. 1995. Syntax of Clitic Placement in Western Romance, *Linguistic Inquiry* 26.1:115.)

```
\jtree[xunit=1.8em,yunit=.9em]
\def\*{\xleaders\hbox{\kern1pt.\kern1pt}\hfil}%
\deftriangle<triA>(1.3)(1)(-1/2)
\! = {F$'$}
   <wideleft>{F}!a
      ^<wideright>{\hbox to 2em{\*}}{$\rm Agr_O P$}
   <left>{DP\rlap{\rnode{B2}{$\;\equiv\;$}\it me}}@B1
      ^<wideright>{$\rm Agr_O P$}
   <left>  ^<wideright>{$\rm {Agr_O}'$}
   <left>{$\rm Agr_O$}!b  ^<wideright>{AsP}
   <triA>{\triline
      {\* \rnode[b]{D1}{t} \* DP \* \rnode[b]{D2}{t} \*}}
   <tri>[scaleby=.8 1.3]{\triline{\* \rnode[b]{E1}{\it a} \*}}.
\!a = :{F}!a1 @A1 .
\!a1 = :{F}!a2 @A2 .
\!a2 = :{V}@A3 {F}.
\psset{scaleby=.5 1}
\!b = :{V}@C1 {$\emptyset$}.
\psset{arrows=->}
\ncarc[arcangle=50]{C1}{A3}
\ncarc[arcangle=50]{B1}{A2:t}
\ncarc[arcangle=50]{D1}{C1}
\nccurve[angleA=-75,angleB=-90,ncurv=1.2,nodesepB=1ex]{D2}{B2}
\nccurve[angleA=-145,angleB=-100,ncurv=1]{E1}{A1:t}
\endjtree
```

Example 8

(Richards, Norvin. 2001. *Movement in Language: Interactions and Architectures*, p. 262.)

```
\jtree[xunit=1.5em,yunit=1.1em,labelgap=0]
\def\A#1{\pnode(0,.3){A#1}}%
\def\B#1{\pnode(0,-.3){B#1}}%
\! = :{car} {\omit\A1}
   :({\omit\A0}{Op})
   :{that}
   :{I}
   :{T}
   :{know} {\omit\B1}
   :({\omit\B0}{who})
   :{C}
   :{Pro}
   :{to}() \jtjot
   :{persuade}() \jtjot
   :{V} {\omit\A2}
   :\jtlong{\omit\A4}!a
   :{Pro}
   :{to}
   :{talk} {\omit\B2}
   :\jtlong{\omit\B3}!b
```
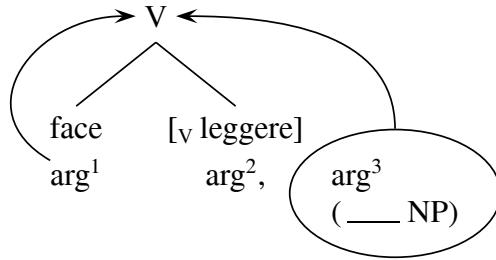
```
    :{V}
    :{about}() {\omit\A3}{\it t}.
\!a = :{owners}() \jtjot
    :{of} {\omit\A5}{\it t}.
\!b = :{to} {\omit\B4}{\it t}.
\psset{linestyle=dashed,linewidth=.3ex,
    linecolor=blue,nodesep=0}
\def\fudge{.5}%
\ncline[nodesepA=-\fudge]{A0}{A1}
\ncline[nodesepB=-\fudge]{A1}{A3}
\ncline{A2}{A4}
\ncline[nodesepB=-\fudge]{A4}{A5}
\psset{linestyle=dotted,linewidth=.5ex,linecolor=red}
\ncline[nodesepA=\fudge]{B0}{B1}
\ncline{B1}{B2}
\ncline{B2}{B3}
\ncline[nodesepB=\fudge]{B3}{B4}
\endjtree
```

Some adjustments were made after the first proof was examined. `xunit` was adjusted so that the display was as wide as possible, `\jtjot` was used to slightly lengthen a few branches in order to improve the spacing, and `\fudge` was introduced in order to adjust the end points of the dotted and dashed paths.

A solution using `offset` in drawing the paths is also possible, but it is much more difficult to ensure that the segments join smoothly.

Example 9
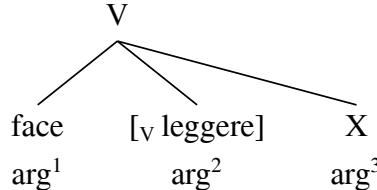
(Zubizaretta, Maria Luisa. 1985. Morphology and Morphosyntax: Romance Causatives, *Linguistic Inquiry* 16.2, p. 276.)

```
\jtree[xunit=2.5em,yunit=2em]
\def\ovalstuff{\vtop{\hbox{arg$^3$}%
   \hbox to 4em{(\thinspace \leaders\hrule\hfil\ NP)}}}%
\! = {V}@B1
   <left>{face}({arg$^1$}@B2 )
   ^<right>{$\rm [_V\,$leggere]}({arg$^2$,})
   ^<right>[scaleby=3 1,branch=\blank]
      {}{\ovalnode[framesep=1ex,boxsep=false]{K}{\ovalstuff}}.
\psset{arrows=->,nodesepA=0}
\nccurve[angleA=150,angleB=180,ncurv=1.2]{B2}{B1}
\nccurve[angleA=90,angleB=0,ncurv=.8]{K}{B1}
\endjtree
```

In order to see how this tree is put together, consider first:
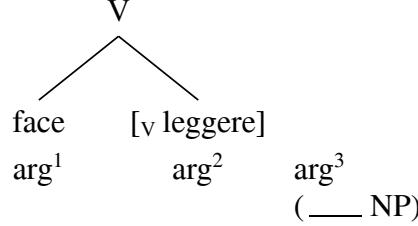

```
\jtree[xunit=2.5em,yunit=2em]
\! = {V}
   <left>{face}({arg$^1$})
   ^<right>{$\rm [_V\,$leggere]}({arg$^2$})
   ^<right>[scaleby=3 1]{X}
   {arg$^3$}.
\endjtree
```

Then:

$$V$$

face　　[$_V$ leggere]

arg^1　　　arg^2　　　arg^3

　　　　　　　　　　(―― NP)

```
\jtree[xunit=2.5em,yunit=2em]
\def\ovalstuff{\vtop{\hbox{arg$^3$}%
   \hbox to 4em{(\thinspace \leaders\hrule\hfil\ NP)}}}%
\! = {V}
   <left>{face}({arg$^1$})
   ^<right>{$\rm [_V\,$leggere]}({arg$^2$})
   ^<right>[scaleby=3 1,branch=\blank]{}
      {\ovalstuff}.
\endjtree
```
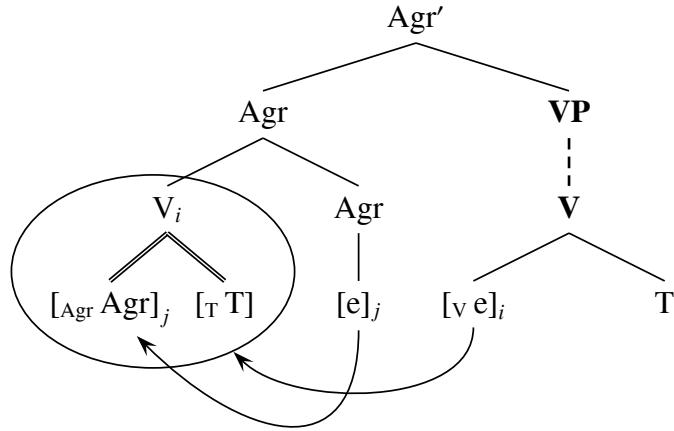
Finally, the oval node is built around \ovalstuff using \ovalnode. The syntax is:

$$\text{\ovalnode}[\textit{pars}]\{\textit{name}\}\{\textit{stuff}\}$$

with the parameters optional. The box is enlarged on all sides by |framesep| and an oval drawn through the four corners of the resulting box. If |boxsep| is *false*, the node construction is invisible to Tex, otherwise \ovalnode creates a Tex box the size of the oval.

The parameter settings in the use of \ovalnode in the example are crucial. If |boxsep| were not *false*, the alignment would be disrupted. |*framesep*| = 1 ex means that the oval is drawn around the box with a separation of 1 ex, leaving a visually important gap between the oval and the box it surrounds.

Example 10

(Koopman, Hilda. 1995. On Verbs That Fail to Undergo V-Second, *Linguistic Inquiry* 26.1:150.)
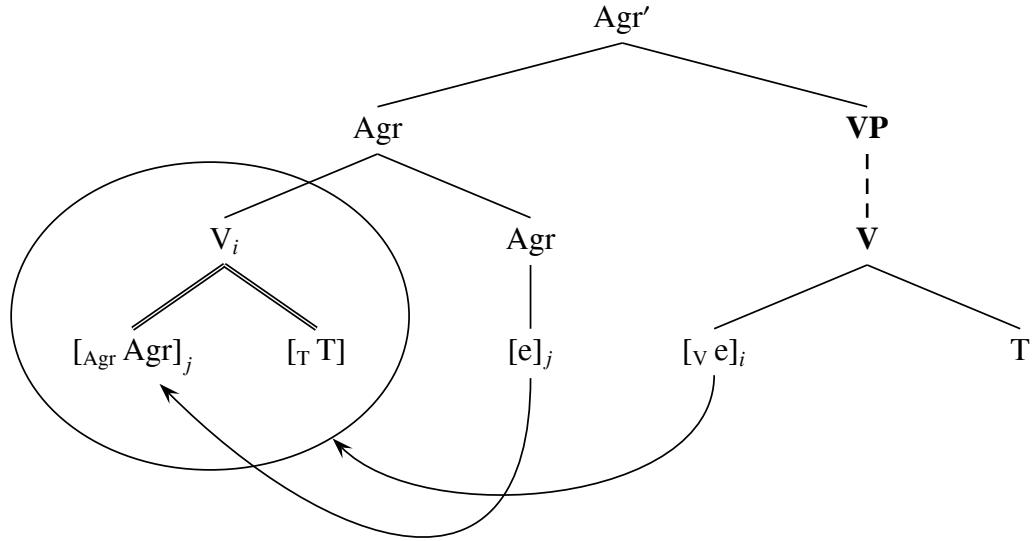
```
\jtree[xunit=3em,yunit=1.5em]
\def\scaleA{[scaleby=1.6 1]}%
\def\scaleB{[scaleby=.6 1,doubleline=true,doublesep=.1ex]}%
\def\mkovalnode{\rput(-1ex,-.8)
   {\ovalnode[framesep=\psxunit]{K}{\hskip2em}}}%
\! = {$\rm Agr'$}
   :\scaleA{Agr}!a \scaleA{\bf VP}
   <vert>[linestyle=dashed,linewidth=1pt]{\bf V}
   :{${\rm [_V\,e]}_i$}@A1 {T}.
\!a = :{V$_i$}!b {Agr}
   <vert>{[e]$_j$}@A2 .
\!b = {\omit\mkovalnode}
   :\scaleB{${\rm [_{Agr}\,Agr]}_j$}@A3 \scaleB{$\rm [_T\,T]$}.
\psset{angleA=-90,angleB=-45,arrows=->}
\nccurve[nodesepB=0]{A1}{K}
\nccurve[ncurv=1.3]{A2}{A3}
\endjtree
```

The difficulty in this problem is drawing the oval and making it a node, so that a node connection can point to the oval. \mkovalnode, which is evaluated at the apex of the !b subtree, does all the work. \rput(-1ex,-.8) positions the center of the oval slightly to the left of the apex and 80% of the way to the bottom of the two branches from the apex (since they have height 1). The size of the oval is determined by |framesep| and box \hbox{\hskip2em}. It required some trial and error to get the various settings in \mkovalnode adjusted to values which produced a suitable oval.
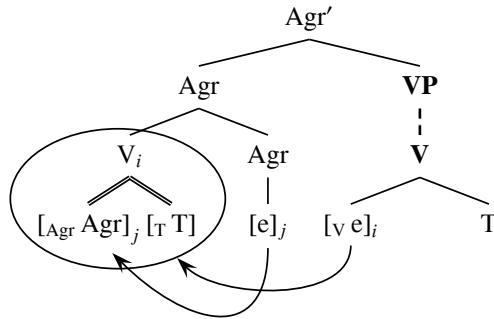
It is worth noting that this tree formatting can be adjusted to different sizes and fonts without difficulty. This makes it fairly easy to resize the tree to make the

transition from a draft paper to a book manuscript, or to move the tree from the text to a footnote with a smaller font.

`\twelvepoint\jtree[xunit=4.8em,yunit=2em]`, without altering any other code, produces

Agr′

Agr   **VP**

V$_i$ Agr   **V**

[$_{\mathrm{Agr}}$ Agr]$_j$ [$_{\mathrm{T}}$ T]   [e]$_j$ [$_{\mathrm{V}}$ e]$_i$ T

`\tenpoint\jtree[xunit=2.6em,yunit=1em]`, again without altering any other code, produces:

Agr′

Agr   **VP**

V$_i$ Agr **V**

[$_{\mathrm{Agr}}$ Agr]$_j$ [$_{\mathrm{T}}$ T] [e]$_j$ [$_{\mathrm{V}}$ e]$_i$ T
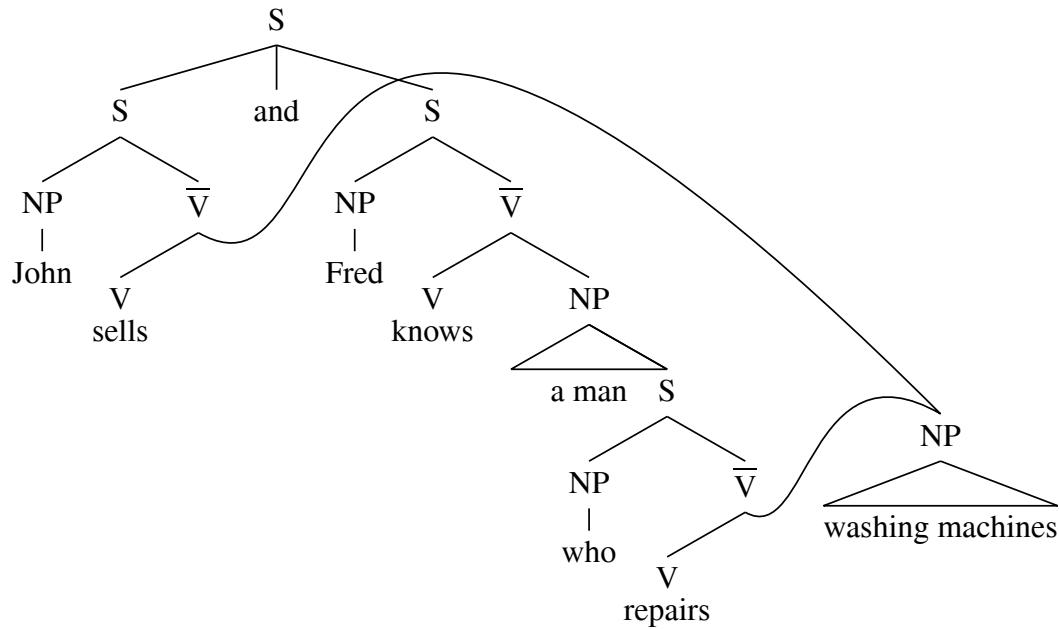
The oval is somewhat too small, but that is easily fixed by increasing `framesep`.

Example 11

This example, and the following four, illustrate some techniques for displaying multidominance structures. I have argued that movement should be seen as creating multidominance trees with shared structure. My suspicion is that the difficulty in portraying the resulting structures in a typeset diagram makes the idea less appealing than it might otherwise be. It would be unfortunate for deficiencies in typesetting technology to have a significant influence in shaping theoretical work. What made writing *pst-jtree* worth the trouble was that I see it as a possible contribution to the development of theories of mental computation.

The influence of typesetting technology on linguistic theory should not be underestimated. The widespread inattention to autosegmental structure in phonology is at least partially a reflection of the state of typesetting technology.

Since McCawley (1982) seems to be the first linguist to attempt to use multidominance to solve a complex syntax problem, we begin with one of his examples (right node raising).

```
\jtree[xunit=2.45em,yunit=1.4em,dirA=(1:-1),nodesep=0]
\def\\{[labelgapb=-4pt]}%
\def\V{$\rm \overline V$}%
\! = {S}
   <wideleft>{S}!a ^<vert>{and} ^<wideright>{S}
   :({NP}<shortvert>{Fred}) {\V}
   :({V}\\{knows}) {NP}
   <tri>{a man} ^<right>
   <right>[scaleby=3.5 1,branch=\blank]{NP}@A3 !b ^{S}
   :({NP}<shortvert>{who}) {\V}@A2
   <left>({V}\\{repairs}).
\!a = :({NP}<shortvert>{John}) {\V}@A1
   <left>{V}\\{sells}.
\!b = <vartri>{washing machines}.
\nccurve[angleB=150,ncurvB=1.4]{A2:b}{A3:t}
\nccurve[angleB=135,ncurvA=.5,ncurvB=2.6]{A1:b}{A3:t}
\endjtree
```
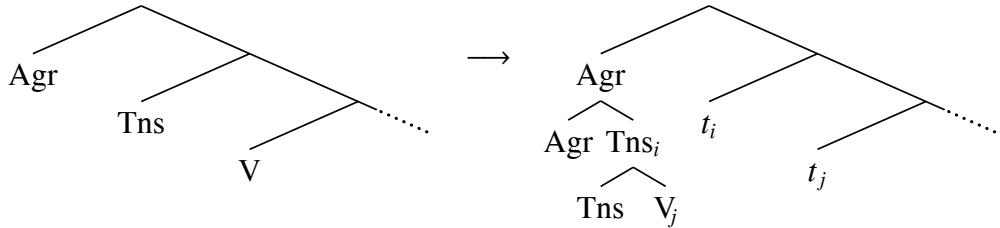
1. A blank branch is used to position the node which is shared between the two conjuncts.

2. Note the high value of ncurvB that is used to properly locate the curve.
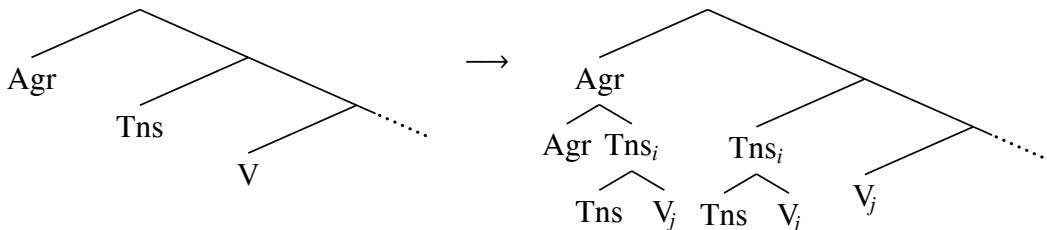
Example 12

Theories of verb raising

Trace theory

Agr
Tns
V

$\longrightarrow$

Agr
Agr Tns$_i$ t_i
Tns V$_j$
t_j

Copy theory

Agr
Tns
V

$\longrightarrow$

Agr
Agr Tns$_i$ Tns$_i$
Tns V$_j$ Tns V$_j$ V$_j$

Shared structure

Agr
Tns
V

$\longrightarrow$

Agr
Agr Tns
Tns V
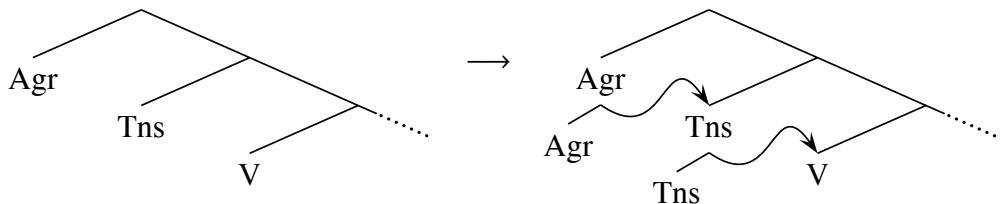
```
\def\Vj{V$\mskip-5mu _j$}
\psset{xunit=3.4em,yunit=1.5em,treevshift=1.3em}
```

Trace theory

```
\jtree
\! = :{Agr} :{Tns} :{V}() \etc.¹
\endjtree
\quad $\longrightarrow$\quad
\jtree
\! = :{Agr} !a
   :{$t_i$}
   :{$t_j$}() \etc.
\psset{scaleby=.3 .4}
\!a = :{Agr} {Tns$_i$}
   :{Tns} {\Vj}.
\endjtree
```

1. See page 27 for a discussion of \etc.
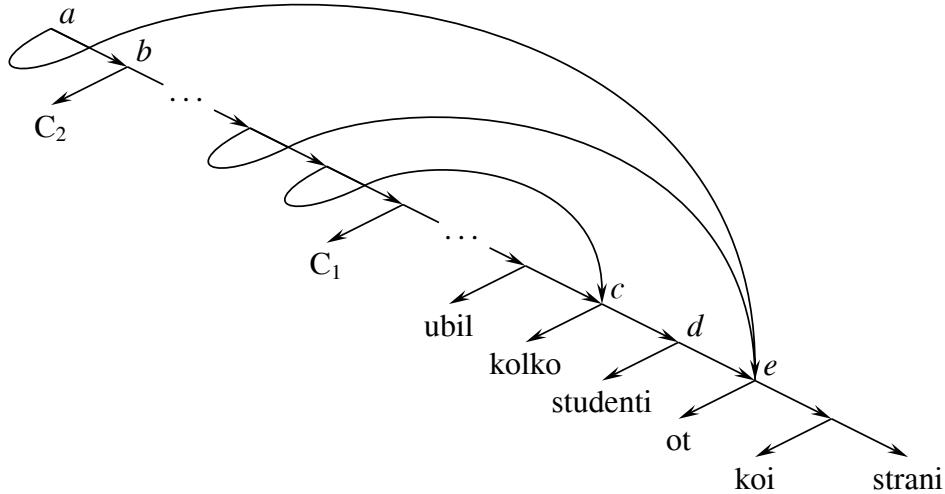
Copy theory

```
\jtree
\! = :{Agr} :{Tns} :{V}() \etc.
\endjtree
\quad $\longrightarrow$\quad
\jtree
\! = :{Agr}!a [scaleby=1.45]
   :{Tns$_i$}!b
   :{\Vj}() \etc.
\psset{scaleby=.3 .4}
\!a = :{Agr} {Tns$_i$}
   :{Tns} {\Vj}.
\!b = :{Tns} {\Vj}.
\endjtree
```

Shared structure

```
\jtree
\! = :{Agr} :{Tns} :{V}() \etc.
\endjtree
\quad $\longrightarrow$\quad
\jtree
\! = :{Agr}@A1 !a
   :{Tns}@A2 !b
   :{V}@A3 \etc.
\psset{scaleby=.3 .4,angleA=-35,angleB=125,ncurv=1.5,nodesep=0}
\!a = <left>{Agr}.
\!b = <left>{Tns}.
\nccurve{->}{A1:b}{A2:t}
\nccurve{->}{A2:b}{A3:t}
\endjtree
```

Example 13


```
\jtree[xunit=2.4em,yunit=1.2em,arrows=->,nodesep=0]
\def\broken{[branch=\brokenbranch,scaleby=1.6]}%¹
\def\stub{<right>[scaleby=.5,arrows=-]}%²
\def\\#1{\rput[bl](.6ex,.4ex){\it #1}}%³
\! = {\omit\\a}@A1
   \stub @K1  ^<right>{\omit\\b}
   :{C$_2$}()  \broken @A2
   \stub @K2  ^<right>@A3
   \stub @K3  ^<right>
   :{C$_1$}()  \broken
   :{ubil}  {\omit\\c}@A4
   :{kolko}  {\omit\\d}
   :{studenti}  {\omit\\e}@A5
   :{ot} :{koi} {strani}.
\psset{dirA=(1:1),angleB=90,ncurvA=.6,ncurvB=1}⁴
\nccurve{K1}{A5}
\nccurve{-}{K2}{A5}
\nccurve{K3}{A4}
\psset{dirA=(-1:-1),dirB=(-1:-1),ncurv=4,arrows=-}⁵
\nccurve{A1}{K1}
\nccurve{A2}{K2}
\nccurve{A3}{K3}
\endjtree
```

1. See Section 10.4 for information about the branch parameter.

2. \stub is used to position nodes halfway down certain right branches to facilitate drawing the complex (2 segment) pointers.

3. \\ is used to position the tags *a, b, c, . . .*

4. `dirA` is used in order to ensure that the complex pointers are parallel to left branches at the point that they cross the right branches.

5. A very high value of `ncurv` is used in order to get the loop to bow out sufficiently.

An alternate to using \stub to position the crossing points is \psinterpolate.
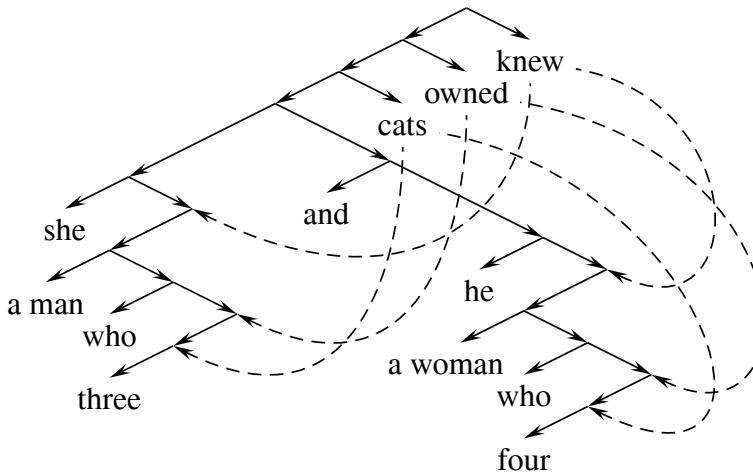
```
\jtree[xunit=2.4em,yunit=1.2em,arrows=->,nodesep=0,
    arrowlength=3.6,arrowsize=2pt,arrowinset=.4]
\def\broken{[branch=\brokenbranch,scaleby=1.6]}%
\def\\#1{\rput[bl](.6ex,.4ex){\it #1}}%
\! = {\omit\\a}@A1
    <right>{\omit\\b}@A1a
    :{C$_2$}()  \broken @A2
    <right>@A3
    <right>@A3a
    :{C$_1$}()  \broken
    :{ubil}  {\omit\\c}@A4
    :{kolko}  {\omit\\d}
    :{studenti}  {\omit\\e}@A5
    :{ot} :{koi} {strani}.
\psinterpolate(A1)(A1a){.5}{K1}
\psinterpolate(A2)(A3){.5}{K2}
\psinterpolate(A3)(A3a){.5}{K3}
```

. . . continues as above

Example 14

In order to show that right node raising does not in general apply to constituents, Chris Wilder gave the example (from German):

Er hat einen Mann, der drei, und sie hat eine Frau, die vier,

Katzen besitzt, gekannt.

In my own work on right node raising, I have had occasion to typeset the following. It may provide some useful things for jTree users, so it is included here.
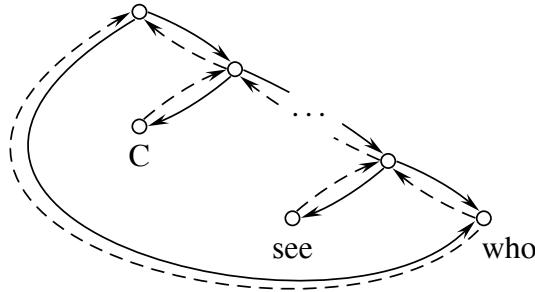


```
\jtree[dirA=(1:-1),nodesepA=0,nodesepB=.8ex,arrows=->,
   arrowlength=3.6,arrowsize=2pt,arrowinset=.4]
\! = :!a {\rnode{K1}{knew}}.
\!a = :!b {\rnode{O1}{owned}}.
\!b = :!c {\rnode{C1}{cats}}.
\!c =
   :\jtlong !d [scaleby=1.8]
   :{and}() [scaleby=2.4]
   :{he}() @K2
   <left>\jtjot !e .
\!d =
   :{she}() @K3
   <left>\jtjot !f .
\!e =
   :{a woman}[labeloffset=-1ex]
   :{who}() @O2
   <left>@C2
   <left>{four}.
\!f =
   :{a man}
   :{who}() @O3
   <left>@C3
   <left>{three}.
```

58

```
\psset{linestyle=dashed,arrows=<-}
\nccurve[angleB=-10,ncurvB=2,ncurvA=1.2]{O2}{O1}
\nccurve[angleB=-90,ncurvA=1.4]{O3}{O1}
\nccurve[angleB=-10,ncurvB=1.8,ncurvA=1.6]{K2}{K1}
\nccurve[angleB=-90,ncurvA=1.4]{K3}{K1}
\nccurve[angleB=-90,ncurvA=1.4]{C3}{C1}
\nccurve[angleB=-10,ncurvB=1.8,ncurvA=1.6]{C2}{C1}
\endjtree\kern6em
```

Example 15


```
\def\bilink(#1,#2)(#3,#4){%
   \pcarc(#1,#2)(#3,#4)%
   \pcarc[linestyle=dashed](#3,#4)(#1,#2)%
}
\jtree[xunit=3em,yunit=1.8em,style=arrows2,
   dirA=(-1:-1),branch=\bilink,nodesep=3pt,
   arcangle=10,offset=1pt,labelgapt=!3pt]
\def\@{\pscircle(0,0){3pt}}%
\! =
   {\pnode{A1}\@}
   <right>{\omit\@\pnode(.8,-.8){A3}}
   :({\omit\@}{C}) [scaleby=1.6,arcangle=7]{\omit\@}
   :({\omit\@}{see})
      ({\omit\@\pnode{A2}}{who}[labeloffset=.8em]).
\nccurve[angleB=225,ncurvA=1.95,ncurvB=1,offset=1.6pt]{A1}{A2}
\nccurve[angleB=227,ncurvA=2,ncurvB=1.02,offset=-1.6pt,
   linestyle=dashed,arrows=<-]{A1}{A2}
\rput(1.8,-1.8){\pscircle*[linecolor=white]{1em}}%
\rput(1.8,-1.8){\dots}
\endjtree
```

15. Compatibility issues

1. Earlier versions of jTree had the syntax in (16b), rather than the syntax (16a) which has been explained and used in this documentation.

(16)   a.   \jftree
      preliminary definitions. parameter settings
      \! = *simple tree description.*
      definitions, parameter settings, dimensionless graphics
      \!a = *simple tree description.*
      definitions, parameter settings, dimensionless graphics
      \!b = *simple tree description.*
      dimensionless graphics
      \endjtree

      b.   \jftree
      preliminary definitions. parameter settings
      \start *simple tree description.*
      definitions, parameter settings, dimensionless graphics
      \adjoin at !a *simple tree description.*
      definitions, parameter settings, dimensionless graphics
      \adjoin at !b *simple tree description.*
      dimensionless graphics
      \endjtree

The old syntax can still be used, if desired. It is even possible to mix the old and new syntax.

2. Tex uses the control sequence \! for negative spacing in math mode. jTree redefines \! inside \jtree ...\endjtree. This redefinition can be suppressed, if desired, by redefining the token list \jtEverytree. When \jtree is invoked, two token lists are expanded and evaluated. First, \jtEverytree and then \jteverytree. The first is intended for setup use, to be rarely altered, and the second to be modified as needed in the course of using jTree. \jteverytree can be modified by parameter setting. The first cannot be, but can be redefined by editing *pst-jtree* or by resetting it in a setup file that is loaded after *pst-jtree* is loaded. The default setting is:

$$\text{\jtEverytree=\{\let\!\adjoinop\}}$$

The \! will be suppressed if this is changed to:

$$\text{\jtEverytree=\{\}}$$

With the special use of \! suppressed, the syntax (16b) must be used.

In my personal file, I have:

$$\text{\jtEverytree=\{\everymath=\{\rm\}\let\!\adjoinop\}}$$

In typesetting linguistic trees, I more often prefer roman to math italic type in math mode.

3. The tree description parser assumes that characters in the set

$$\{., :, <, >, (,), ", @, !, \wedge\}$$

are not active and have the same category code that they had when *pst-jtree* was loaded. Normally, all but ˆ (with category code 7, "superscript") have category code 12, "other". Some macros sets (packages) alter these characters. This must be suppressed inside \jtree ... \endjtree. The Babel package, in particular, makes extensive use of active characters. Subfiles for particular languages, however, generally make available macros which return these characters to the normal state. The relevant commands can therefore be included in \jtEverytree or \jteverytree. Norerto Quiben, who reported the problem, has found that

```
\jteverytree={\spanishdeactivate{<>}}
```

cures the compatibility problem with Spanish Babel. Since then, \jtEverytree has been added to jTree and it seems appropriate for a regular user of Spanish Babel to put the deactivation command there rather than in the more fluid \jteverytree. Users who solve other Babel compatibility problems are encouraged to send me the solution for inclusion in future versions of the User's Guide.

4. Users with unsolved compatibility problems are also encouraged to report them to me at *j.frampton@neu.edu*.
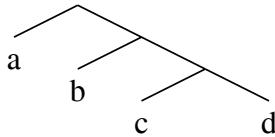
Appendix A: Installation and working environment

Assuming that you have already installed the PSTricks and XKeyVal packages, you have to put the file *pst-jtree.tex* in a place where it can be found. The directory that contains *pstricks.tex* is a natural place, but if you know how, it is probably better to make your own parallel Tex local subtree so that updating your Tex files with a new Tex distribution does not wipe out *pst-jtree.tex*. If you want to use jTree with LaTex, you need to do the same with *pst-jtree.sty*. Finally, if Tex file retrieval is done by an indexing method (as it almost certainly is), you have to run the indexing program so that the locations of *pst-jtree.tex* and *pst-jtree.sty* are properly indexed. The jTree dirstribution consists of only three files: *pst-jtree.tex*, *pst-jtree.sty*, and *pst-jtree-doc.pdf* (which you are now reading).

     Before you proceed, you should make sure that you can run and view a simple example. If you are a LaTex user, process (17a) and if a Tex user, process (17b).
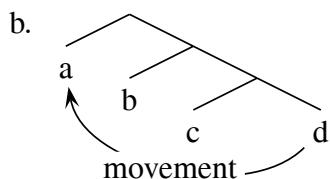
```
(17)  a.  \documentclass{article}     b.  \input pstricks
          \usepackage{pstricks}           \input pst-xkey
          \usepackage{pst-xkey}           \input pst-jtree
          \usepackage{pst-jtree}          \jtree
          \begin{document}                \! = :{a} :{b} :{c}{d}.
          \jtree                          \endjtree
          \! = :{a} :{b} :{c}{d}.         \bye
          \endjtree
          \end{document}
```

Your dvi viewer may understand enough Postscript code to properly display the dvi file that is produced. You should see the tree below, left aligned:

PSTricks does its tricks by using Tex \special commands to embed Postscript code in the dvi file that Tex produces. Some dvi viewers will simply ignore embedded Postscript code and you will have to use a dvi to ps translator (the program *dvips*, for example) to create a .ps file which can be viewed with a Postscript viewer like Ghostscript. Even if your dvi viewer can handle the dvi output from (17), which has very simple postscript inclusions, successful use of jTree using the full range of PSTricks tricks will require dvi to ps conversion for proper display, so it is a capability that you will soon have to acquire. If your dvi viewer successfuly displays (17), try it on (18a). If it can successfully understand the postscript inclusions, it will produce (18b).

(18) a.
```
\jtree
\! =
   :{a}@A1
   :{b}
   :{c} {d}@A2 .
\nccurve[angleA=225,angleB=-80]{->}{A2}{A1}
\mput*{movement}
\endjtree
```

  b.

I assume that the test has failed and your dvi viewer did not produce (18b). The point of this exercise was to drive home the point that you will need the capability of dvi to ps conversion and the capability of viewing Postscript files. In my own work, I often skip this step (saving a few seconds) and use my dvi viewer (Windvi), which does a good job at basic Postscript inclusions. But I have been using Postscript long enough so that I instantly recognize when the inclusions have gotten too complex for Windvi and switch to full conversion and viewing with Ghostscript, the standard Postscript viewer.

Convert the output of (18a) to a ps file, and view it with a Postscript viewer. You should now see (18b). Until you can convert the dvi file to a ps file and view it, you will not know if some baffling display is caused by a programming error or the inability of your dvi viewer to correctly display the postscript code which is embedded in the dvi file. If you cannot successfully run the test file and convert the

dvi file to a ps file and view it, get help with PSTricks and/or dvi to ps conversion before you proceed.

Working environment: Since jTree relies on adjustability rather than automatic sizing, it is important to create a good working Tex/LaTex environment that lets you see the effect of modifications quickly and with no fuss. A good interactive Tex environment minimizes the time between making a change in the editor and seeing the results on the screen. Your editor, dvi viewer, and postscript viewer should all remain active and you should be able to easily bring one or the other into the foreground. Your viewers should be configured so that they keep their place in the file they are viewing. If a new dvi or ps file is created, for example, your viewer should automatically load it and be positioned at the same place (page and xy-position) as it was in the old file. You want to reduce the cycle time between editing and viewing the result to a few seconds (on a fast PC).

Index of control sequences, parameters, and special characters

The characters =, <, >, ˆ, [,], {, }, (,), :, @, and ! have special meaning in parsing tree descriptions. See the discussion of the syntax of tree description on page 23.