Implementation of FOREST, a PGF/Ti*k*Z-based package for drawing linguistic trees

v2.1.5

Sašo Živanović*

July 14, 2017

This file contains the documented source of FOREST. If you are searching for the manual, follow this link to forest-doc.pdf.

The latest release of the package, including the sources, can be found on CTAN. For all versions of the package, including any non-yet-released work in progress, visit FOREST's GitHub repo. Contributions are welcome.

A disclaimer: the code could've been much cleaner and better-documented ...

Contents

---

*e-mail: saso.zivanovic@guest.arnes.si; web: http://spj.ff.uni-lj.si/zivanovic/

1 Identification

```
\ProvidesPackage{forest}[2017/07/14 v2.1.5 Drawing (linguistic) trees]

\RequirePackage{tikz}[2013/12/13]
\usetikzlibrary{shapes}
\usetikzlibrary{fit}
\usetikzlibrary{calc}
\usepgflibrary{intersections}

\RequirePackage{pgfopts}
\RequirePackage{etoolbox}[2010/08/21]
\RequirePackage{elocalloc}% for \locbox
\RequirePackage{environ}
\RequirePackage{xparse}

\RequirePackage{inlinedef}
\newtoks\ID@usercommands{}
\newcommand\NewInlineCommand[3][0]{%
  \newcommand#2[#1]{#3}%
  \ID@usercommands\xa{%
    \the\ID@usercommands
    \ifx\@foo#2%
      \def\next{\ID@expandunsafe#2}%
    \fi
  }%
}
\def\@ExpandIfTF#1{%
  \csname
    % I'm not 100% sure if this plays well in every situation
    \csname if#1\endcsname
      @firstoftwo%
    \else
      @secondoftwo%
    \fi
  \endcsname
}
\patchcmd{\ID@switch}
  {\ifcat\noexpand\@foo\space}
  {\the\ID@usercommands\ifcat\noexpand\@foo\space}
  {%
    \NewInlineCommand[2]\ExpandIfT{%
      \MultiExpand{3}{%
        \@ExpandIfTF{#1}{#2}{}%
      }%
    }
    \NewInlineCommand[2]\ExpandIfF{%
      \MultiExpand{3}{%
        \@ExpandIfTF{#1}{}{#2}%
      }%
    }
    \NewInlineCommand[3]\ExpandIfTF{%
      \MultiExpand{3}{%
        \@ExpandIfTF{#1}{#2}{#3}%
      }%
    }%
    \newcommand\InlineNoDef[1]{%
      \begingroup
      % Define a few ‘‘quarks’’
      \def\Expand{\Expand}\def\Super{\Super}%
      \def\UnsafeExpand{\UnsafeExpand}\def\MultiExpand{\MultiExpand}%
      \def\Recurse{\Recurse}\def\NoExpand{\NoExpand}%
      \def\Q@END{\Q@END}%
      % Define a toks register
      \ID@toks{}%
      % Signal that we need to look for a star
      \@testtrue\ID@starfalse\ID@starstarfalse\ID@bangfalse
      % Start scanning for \def or \gdef
      \ID@scan#1\Q@END{}%
      \expandafter\endgroup
      %\expandafter\@firstofone
      \the\ID@toks
    }%
  }%
  {%
    \PackageWarning{forest}{Could not patch inlinedef! Disabling it. Except in some special situations (neste
   \let\Inline\relax
   \def\Expand#1{#1}%
   \def\MultiExpand#1#2{#2}%
   \def\InlineNoDef#1{#1}%
   \def\ExpandIfT#1#2{\@ExpandIfTF{#1}{#2}{}}%
   \def\ExpandIfF#1#2{\@ExpandIfTF{#1}{}{#2}}%
   \def\ExpandIfTF#1#2#3{\@ExpandIfTF{#1}{#2}{#3}}%
  }
```

`/forest` is the root of the key hierarchy.

```
\pgfkeys{/forest/.is family}
\def\forestset#1{\pgfqkeys{/forest}{#1}}
```

2 Package options

```
\newif\ifforest@external@
\newif\ifforesttikzcshack
\newif\ifforest@install@keys@to@tikz@path@
\newif\ifforestdebugnodewalks
```

```
\newif\ifforestdebugdynamics
\newif\ifforestdebugprocess
\newif\ifforestdebugtemp
\newif\ifforestdebug
\def\forest@compat{}
\forestset{package@options/.cd,
  external/.is if=forest@external@,
  tikzcshack/.is if=foresttikzcshack,
  tikzinstallkeys/.is if=forest@install@keys@to@tikz@path@,
  compat/.code={\appto\forest@compat{,#1}},
  compat/.default=most,
  .unknown/.code={% load library
    \eappto\forest@loadlibrarieslater{%
      \noexpand\useforestlibrary{\pgfkeyscurrentname}%
      \noexpand\forestapplylibrarydefaults{\pgfkeyscurrentname}%
    }%
  },
  debug/.code={\forestdebugtrue\pgfqkeys{/forest/package@options/debug}{#1}},
  debug/.default={nodewalks,dynamics,process},
  debug/nodewalks/.is if=forestdebugnodewalks,
  debug/dynamics/.is if=forestdebugdynamics,
  debug/process/.is if=forestdebugprocess,
}
\forest@install@keys@to@tikz@path@true
\foresttikzcshacktrue
\def\forest@loadlibrarieslater{}
\AtEndOfPackage{\forest@loadlibrarieslater}
\NewDocumentCommand\useforestlibrary{s O{} m}{%
  \def\useforestlibrary@@##1{\useforestlibrary@{#2}{##1}}%
  \forcsvlist\useforestlibrary@@{#3}%
  \IfBooleanT{#1}{\forestapplylibrarydefaults{#3}}%
}
\def\useforestlibrary@#1#2{%
  \RequirePackage[#1]{forest-lib-#2}%
  \csuse{forest@compat@libraries@#2}%
}
\def\forestapplylibrarydefaults#1{\forcsvlist\forestapplylibrarydefaults@{#1}}
\def\forestapplylibrarydefaults@#1{\forestset{libraries/#1/defaults/.try}}
\NewDocumentCommand\ProvidesForestLibrary{m O{}}{%
  \ProvidesPackage{forest-lib-#1}[#2]%
  \csdef{forest@libraries@loaded@#1}{}%
}
\def\forest@iflibraryloaded#1#2#3{\ifcsdef{forest@libraries@loaded@#1}{#2}{#3}}
\ProcessPgfPackageOptions{/forest/package@options}
```

3 Patches

This macro implements a fairly safe patching mechanism: the code is only patched if the original hasn't changed. If it did change, a warning message is printed. (This produces a spurious warning when the new version of the code fixes something else too, but what the heck.)

```
\def\forest@patch#1#2#3#4#5{%
  % #1 = cs to be patched
  % %2 = purpose of the patch
  % #3 = macro arguments
  % #4 = original code
  % #5 = patched code
  \csdef{forest@original@#1}#3{#4}%
  \csdef{forest@patched@#1}#3{#5}%
  \ifcsequal{#1}{forest@original@#1}{%
    \csletcs{#1}{forest@patched@#1}%
  }{%
    \ifcsequal{#1}{forest@patched@#1}{% all is good, the patch is in!
    }{%
      \PackageWarning{forest}{Failed patching '\expandafter\string\csname #1\endcsname'. Purpose of the patch
    }%
  }%
}
```

Patches for PGF 3.0.0 — required version is [2013/12/13].

```
\forest@patch{pgfgettransform}{fix a leaking space}{#1}{%
  \edef#1{{\pgf@pt@aa}{\pgf@pt@ab}{\pgf@pt@ba}{\pgf@pt@bb}{\the\pgf@pt@x}{\the\pgf@pt@y}}
}{%
  \edef#1{{\pgf@pt@aa}{\pgf@pt@ab}{\pgf@pt@ba}{\pgf@pt@bb}{\the\pgf@pt@x}{\the\pgf@pt@y}}%
}
```

4 Utilities

This is handy.

```
\def\forest@empty{}
```

Escaping `\ifs`.

```
\long\def\@escapeif#1#2\fi{\fi#1}
\long\def\@escapeifif#1#2\fi#3\fi{\fi\fi#1}
\long\def\@escapeififif#1#2\fi#3\fi#4\fi{\fi\fi\fi#1}
```

```
\def\forest@repeat@n@times#1{% #1=n, #2=code
  \expandafter\forest@repeat@n@times@\expandafter{\the\numexpr#1}}
\def\forest@repeat@n@times@#1{%
  \ifnum#1>0
    \@escapeif{%
      \expandafter\forest@repeat@n@times@@\expandafter{\the\numexpr#1-1}%
    }%
  \else
    \expandafter\@gobble
  \fi
}
\def\forest@repeat@n@times@@#1#2{%
  #2%
  \forest@repeat@n@times@{#1}{#2}%
}
```

A factory for creating `\...loop...` macros.

```
\def\newloop#1{%
  \count@=\escapechar
  \escapechar=-1
  \expandafter\newloop@parse@loopname\string#1\newloop@end
  \escapechar=\count@
}%
{\lccode`7=`l \lccode`8=`o \lccode`9=`p
  \lowercase{\gdef\newloop@parse@loopname#17889#2\newloop@end{%
      \edef\newloop@marshal{%
        \noexpand\csdef{#1loop#2}####1\expandafter\noexpand\csname #1repeat#2\endcsname{%
          \noexpand\csdef{#1iterate#2}{####1\relax\noexpand\expandafter\expandafter\noexpand\csname#1iterate#
          \expandafter\noexpand\csname#1iterate#2\endcsname
          \let\expandafter\noexpand\csname#1iterate#2\endcsname\relax
        }%
      }%
      \newloop@marshal
    }%
  }%
}%
```

Loop that can be arbitrarily nested. (Not in the same macro, however: use another macro for the inner loop.) Usage: `\safeloop_code_\if..._code_\saferepeat`. `\safeloopn` expands to the current repetition number of the innermost group.

```
\def\newsafeloop#1{%
  \csdef{safeloop@#1}##1\saferepeat{%
    \forest@temp@toks{##1}%
    \csedef{safeiterate@#1}{%
      \the\forest@temp@toks\relax
      \noexpand\expandafter
      \expandonce{\csname safeiterate@#1\endcsname}%
      \noexpand\fi
    }%
    \csuse{safeiterate@#1}%
    \advance\noexpand\safeloop@depth-1\relax
    \cslet{safeiterate@#1}\relax
  }%
  \expandafter\newif\csname ifsafebreak@\the\safeloop@depth\endcsname
}%
\newcount\safeloop@depth
\def\safeloop{%
  \advance\safeloop@depth1
  \ifcsdef{safeloop@\the\safeloop@depth}{}{\expandafter\newsafeloop\expandafter{\the\safeloop@depth}}%
  \csdef{safeloopn@\the\safeloop@depth}{0}%
  \csuse{safeloop@\the\safeloop@depth}%
  \csedef{safeloopn@\the\safeloop@depth}{\number\numexpr\csuse{safeloopn@\the\safeloop@depth}+1}%
}
\let\saferepeat\fi
\def\safeloopn{\csuse{safeloopn@\the\safeloop@depth}}%
```

Another safeloop for usage with “repeat” / “while” // “until” keys, so that the user can refer to loop *ns* for outer loops.

```
\def\newsafeRKloop#1{%
  \csdef{safeRKloop@#1}##1\safeRKrepeat{%
    \forest@temp@toks{##1}%
    \csedef{safeRKiterate@#1}{%
      \the\forest@temp@toks\relax
      \noexpand\expandafter
      \expandonce{\csname safeRKiterate@#1\endcsname}%
      \noexpand\fi
    }%
    \csuse{safeRKiterate@#1}%
    \advance\noexpand\safeRKloop@depth-1\relax
    \cslet{safeRKiterate@#1}\relax
  }%
  \expandafter\newif\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
}%
\newcount\safeRKloop@depth
\def\safeRKloop{%
  \advance\safeRKloop@depth1
  \ifcsdef{safeRKloop@\the\safeRKloop@depth}{}{\expandafter\newsafeRKloop\expandafter{\the\safeRKloop@depth}}
  \csdef{safeRKloopn@\the\safeRKloop@depth}{0}%
  \csuse{safeRKbreak@\the\safeRKloop@depth false}%
  \csuse{safeRKloop@\the\safeRKloop@depth}%
  \csedef{safeRKloopn@\the\safeRKloop@depth}{\number\numexpr\csuse{safeRKloopn@\the\safeRKloop@depth}+1}%
}
\let\safeRKrepeat\fi
\def\safeRKloopn{\csuse{safeRKloopn@\the\safeRKloop@depth}}%
```

Additional loops (for embedding).

```
\newloop\forest@loop
```

New counters, dimens, ifs.

```
\newdimen\forest@temp@dimen
\newcount\forest@temp@count
\newcount\forest@n
\newif\ifforest@temp
\newcount\forest@temp@global@count
\newtoks\forest@temp@toks
```

Appending and prepending to token lists.

```
\def\etotoks#1#2{\edef\pot@temp{#2}\expandafter#1\expandafter{\pot@temp}}
\def\apptotoks#1#2{\expandafter#1\expandafter{\the#1#2}}
\long\def\lapptotoks#1#2{\expandafter#1\expandafter{\the#1#2}}
\def\eapptotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter#1\expandafter\expandafter\expandaf
\def\pretotoks#1#2{\toks@={#2}\expandafter\expandafter\expandafter#1\expandafter\expandafter\expandafter{\exp
\def\epretotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter#1\expandafter\expandafter\expandaf
\def\gapptotoks#1#2{\expandafter\global\expandafter#1\expandafter{\the#1#2}}
\def\xapptotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter\global\expandafter\expandafter\exp
\def\gpretotoks#1#2{\toks@={#2}\expandafter\expandafter\expandafter\global\expandafter\expandafter\expandafte
\def\xpretotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter\global\expandafter\expandafter\exp
```

Expanding number arguments.

```
\def\expandnumberarg#1#2{\expandafter#1\expandafter{\number#2}}
\def\expandtwonumberargs#1#2#3{%
  \expandafter\expandtwonumberargs@\expandafter#1\expandafter{\number#3}{#2}}
\def\expandtwonumberargs@#1#2#3{%
  \expandafter#1\expandafter{\number#3}{#2}}
\def\expandthreenumberargs#1#2#3#4{%
  \expandafter\expandthreenumberargs@\expandafter#1\expandafter{\number#4}{#2}{#3}}
\def\expandthreenumberargs@#1#2#3#4{%
  \expandafter\expandthreenumberargs@@\expandafter#1\expandafter{\number#4}{#2}{#3}}
\def\expandthreenumberargs@@#1#2#3#4{%
  \expandafter#1\expandafter{\number#4}{#2}{#3}}
```

A macro converting all non-alphanumerics (and an initial number) in a string to `__`. `#1` = string, `#2` = receiving macro. Used for declaring pgfmath functions.

```
\def\forest@convert@others@to@underscores#1#2{%
  \def\forest@cotu@result{}%
  \forest@cotu@first#1\forest@end
  \let#2\forest@cotu@result
}
\def\forest@cotu{%
  \let\forest@cotu@have@num\forest@cotu@have@alpha
  \futurelet\forest@cotu@nextchar\forest@cotu@checkforspace
}
\def\forest@cotu@first{%
  \let\forest@cotu@have@num\forest@cotu@haveother
  \futurelet\forest@cotu@nextchar\forest@cotu@checkforspace
}
\def\forest@cotu@checkforspace{%
  \expandafter\ifx\space\forest@cotu@nextchar
    \let\forest@cotu@next\forest@cotu@havespace
  \else
    \let\forest@cotu@next\forest@cotu@nospace
  \fi
  \forest@cotu@next
}
\def\forest@cotu@havespace#1{%
  \appto\forest@cotu@result{_}%
  \forest@cotu#1%
}
\def\forest@cotu@nospace{%
  \ifx\forest@cotu@nextchar\forest@end
    \@escapeif\@gobble
  \else
    \@escapeif\forest@cotu@nospaceB
  \fi
}
\def\forest@cotu@nospaceB#1{%
  \ifcat#1a%
    \let\forest@cotu@next\forest@cotu@have@alpha
  \else
    \if!\ifnum9<1#1!\fi
      \let\forest@cotu@next\forest@cotu@have@num
    \else
      \let\forest@cotu@next\forest@cotu@haveother
    \fi
  \fi
  \forest@cotu@next#1%
}
\def\forest@cotu@have@alpha#1{%
  \appto\forest@cotu@result{#1}%
  \forest@cotu
}
\def\forest@cotu@haveother#1{%
  \appto\forest@cotu@result{_}%
  \forest@cotu
}
```

Additional list macros.

```
\def\forest@listedel#1#2{% #1 = list, #2 = item
  \edef\forest@marshal{\noexpand\forest@listdel\noexpand#1{#2}}%
  \forest@marshal
}
\def\forest@listcsdel#1#2{%
  \expandafter\forest@listdel\csname #1\endcsname{#2}%
}
\def\forest@listcsedel#1#2{%
  \expandafter\forest@listedel\csname #1\endcsname{#2}%
}
\edef\forest@restorelistsepcatcode{\noexpand\catcode`|\the\catcode`|\relax}%
\catcode`\|=3
\gdef\forest@listdel#1#2{%
  \def\forest@listedel@A##1|#2|##2\forest@END{%
    \forest@listedel@B##1|##2\forest@END%|
  }%
  \def\forest@listedel@B|##1\forest@END{%|
    \def#1{##1}%
  }%
  \expandafter\forest@listedel@A\expandafter|#1\forest@END%|
}
\forest@restorelistsepcatcode
```

Strip (the first level of) braces from all the tokens in the argument.

```
\def\forest@strip@braces#1{%
  \forest@strip@braces@A#1\forest@strip@braces@preend\forest@strip@braces@end
}
\def\forest@strip@braces@A#1#2\forest@strip@braces@end{%
  #1\ifx\forest@strip@braces@preend#2\else\@escapeif{\forest@strip@braces@A#2\forest@strip@braces@end}\fi
}
```

Utilities dealing with pgfkeys.

```
\def\forest@copycommandkey#1#2{% copies command of #1 into #2
  \pgfkeysifdefined{#1/.@cmd}{}{%
    \PackageError{forest}{Key #1 is not a command key}{}%
  }%
  \pgfkeysgetvalue{#1/.@cmd}\forest@temp
  \pgfkeyslet{#2/.@cmd}\forest@temp
  \pgfkeysifdefined{#1/.@args}{%
    \pgfkeysgetvalue{#1/.@args}\forest@temp
    \pgfkeyslet{#2/.@args}\forest@temp
  }{}%
  \pgfkeysifdefined{#1/.@body}{%
    \pgfkeysgetvalue{#1/.@body}\forest@temp
    \pgfkeyslet{#2/.@body}\forest@temp
  }{}%
  \pgfkeysifdefined{#1/.@@body}{%
    \pgfkeysgetvalue{#1/.@@body}\forest@temp
    \pgfkeyslet{#2/.@@body}\forest@temp
  }{}%
  \pgfkeysifdefined{#1/.@def}{%
    \pgfkeysgetvalue{#1/.@def}\forest@temp
    \pgfkeyslet{#2/.@def}\forest@temp
  }{}%
}
\forestset{
  copy command key/.code 2 args={\forest@copycommandkey{#1}{#2}},
  autoforward/.code 2 args={\forest@autoforward{#1}{#2={#1={##1}}}{true}},
  autoforward’/.code 2 args={\forest@autoforward{#1}{#2-=#1,#2={#1={##1}}}{true}},
  Autoforward/.code 2 args={\forest@autoforward{#1}{#2}{true}},
  autoforward register/.code 2 args={\forest@autoforward{#1}{#2={#1={##1}}}{false}},
  autoforward register’/.code 2 args={\forest@autoforward{#1}{#2-=#1,#2={#1={##1}}}{false}},
  Autoforward register/.code 2 args={\forest@autoforward{#1}{#2}{false}},
  copy command key@if it exists/.code 2 args={%
    \pgfkeysifdefined{#1/.@cmd}{%
      \forest@copycommandkey{#1}{#2}%
    }{}%
  },
  unautoforward/.style={
    typeout={unautoforwarding #1},
    copy command key@if it exists={/forest/autoforwarded #1}{/forest/#1},
    copy command key@if it exists={/forest/autoforwarded #1+}{/forest/#1+},
    copy command key@if it exists={/forest/autoforwarded #1-}{/forest/#1-},
    copy command key@if it exists={/forest/autoforwarded #1*}{/forest/#1*},
    copy command key@if it exists={/forest/autoforwarded #1:}{/forest/#1:},
    copy command key@if it exists={/forest/autoforwarded #1’}{/forest/#1’},
    copy command key@if it exists={/forest/autoforwarded #1+’}{/forest/#1+’},
    copy command key@if it exists={/forest/autoforwarded #1-’}{/forest/#1-’},
    copy command key@if it exists={/forest/autoforwarded #1*’}{/forest/#1*’},
    copy command key@if it exists={/forest/autoforwarded #1:’}{/forest/#1:’},
    copy command key@if it exists={/forest/autoforwarded +#1}{/forest/+#1},
  },
  /handlers/.undef/.code={\csundef{pgfk@\pgfkeyscurrentpath}},
  undef option/.style={
    /forest/#1/.undef,
    /forest/#1/.@cmd/.undef,
    /forest/#1+/.@cmd/.undef,
    /forest/#1-/.@cmd/.undef,
    /forest/#1*/.@cmd/.undef,
    /forest/#1:/.@cmd/.undef,
    /forest/#1’/.@cmd/.undef,
    /forest/#1+’/.@cmd/.undef,
    /forest/#1-’/.@cmd/.undef,
    /forest/#1*’/.@cmd/.undef,
    /forest/#1:’/.@cmd/.undef,
    /forest/+#1/.@cmd/.undef,
    /forest/TeX={\patchcmd{\forest@node@init}{\forestoinit{#1}}{}{}{}},
  },
  undef register/.style={undef option={#1}},
}
\def\forest@autoforward#1#2#3{%
  % #1 = option name
  % #2 = code of a style taking one arg (new option value),
  %      which expands to whatever should be done with the new value
  %      autoforward(') adds to the keylist (arg#2)
  % #3 = true=option, false=register
  \forest@autoforward@createforwarder{}{#1}{}{#2}{#3}%
  \forest@autoforward@createforwarder{}{#1}{+}{#2}{#3}%
  \forest@autoforward@createforwarder{}{#1}{-}{#2}{#3}%
  \forest@autoforward@createforwarder{}{#1}{*}{#2}{#3}%
  \forest@autoforward@createforwarder{}{#1}{:}{#2}{#3}%
  \forest@autoforward@createforwarder{}{#1}{'}{#2}{#3}%
  \forest@autoforward@createforwarder{}{#1}{+'}{#2}{#3}%
  \forest@autoforward@createforwarder{}{#1}{-'}{#2}{#3}%
  \forest@autoforward@createforwarder{}{#1}{*'}{#2}{#3}%
  \forest@autoforward@createforwarder{}{#1}{:'}{#2}{#3}%
  \forest@autoforward@createforwarder{+}{#1}{}{#2}{#3}%
}
\def\forest@autoforward@createforwarder#1#2#3#4#5{%
  % #1=prefix, #2=option name, #3=suffix, #4=macro code (#2 above), #5=option or register
  \pgfkeysifdefined{/forest/#1#2#3/.@cmd}{%
    \forest@copycommandkey{/forest/#1#2#3}{/forest/autoforwarded #1#2#3}%
    \pgfkeyssetvalue{/forest/autoforwarded #1#2#3/option@name}{#2}%
    \pgfkeysdef{/forest/#1#2#3}{%
      \pgfkeysalso{autoforwarded #1#2#3={##1}}%
      \def\forest@temp@macro####1{#4}%
      \csname forest@temp#5\endcsname
      \edef\forest@temp@value{\ifforest@temp\expandafter\forestOv\expandafter{\expandafter\forest@setter@node
      %\expandafter\expandafter\expandafter\pgfkeysalso\expandafter\expandafter\expandafter{\expandafter\fore
      \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\pgfkeysalso\expand
    }%
  }{}%
}
\def\forest@node@removekeysfromkeylist#1#2{% #1 = keys to remove, #2 = option name
  \edef\forest@marshal{%
    \noexpand\forest@removekeysfromkeylist{\unexpanded{#1}}{\forestov{#2}}\noexpand\forest@temp@toks}\forest@
  \forestoeset{#2}{\the\forest@temp@toks}%
}
\def\forest@removekeysfromkeylist#1#2#3{%
  % #1 = keys to remove (a keylist: an empty value means remove a key with any value)
  % #2 = keylist
  % #3 = toks cs for result
  \forest@temp@toks{}%
  \def\forestnovalue{\forestnovalue}%
  \pgfqkeys{/forest/remove@key@installer}{#1}%
  \let\forestnovalue\pgfkeysnovaluetext
  \pgfqkeys{/forest/remove@key}{#2}%
  \pgfqkeys{/forest/remove@key@uninstaller}{#1}%
  #3\forest@temp@toks
}
\def\forest@remove@key@novalue{\forest@remove@key@novalue}%
\forestset{
  remove@key@installer/.unknown/.code={% #1 = (outer) value
    \def\forest@temp{#1}%
    \ifx\forest@temp\pgfkeysnovalue@text
      \pgfkeysdef{/forest/remove@key/\pgfkeyscurrentname}{}%
    \else
      \ifx\forest@temp\forestnovalue
```

```
        \expandafter\forest@remove@key@installer@defwithvalue\expandafter{\pgfkeyscurrentname}{\pgfkeysnovalu
      \else
        \expandafter\forest@remove@key@installer@defwithvalue\expandafter{\pgfkeyscurrentname}{#1}%
      \fi
    \fi
  },
  remove@key/.unknown/.code={% #1 = (inner) value
    \expandafter\apptotoks\expandafter\forest@temp@toks\expandafter{\pgfkeyscurrentname={#1},}%
  },
  remove@key@uninstaller/.unknown/.code={%
    \pgfkeyslet{/forest/remove@key/\pgfkeyscurrentname/.@cmd}\@undefined},
}
\def\forest@remove@key@installer@defwithvalue#1#2{% #1=key name, #2 = outer value
  \pgfkeysdef{/forest/remove@key/#1}{% ##1 = inner value
    \def\forest@temp@outer{#2}%
    \def\forest@temp@inner{##1}%
    \ifx\forest@temp@outer\forest@temp@inner
    \else
      \apptotoks\forest@temp@toks{#1={##1},}%
    \fi
  }%
}
\forestset{
  show register/.code={%
    \forestrget{#1}\foresttemp
    \typeout{Forest register "#1"=\expandafter\detokenize\expandafter{\foresttemp}}%
  },
}
```

4.1 Arrays

```
\def\forest@newarray#1{%
  \forest@tempfalse % non-global
  {%
    \escapechar=-1
    \expandafter\escapechar\expandafter\count@\expandafter
  }%
  \expandafter\forest@newarray@\expandafter{\string#1}%
}
\def\forest@newglobalarray#1{%
  \forest@temptrue % global
  {%
    \escapechar=-1
    \expandafter\escapechar\expandafter\count@\expandafter
  }%
  \expandafter\forest@newarray@\expandafter{\string#1}%
}
\def\forest@array@empty@error#1{%
  \PackageError{forest}{Cannot pop from empty array "#1".}{}}%
\def\forest@array@oub@error#1#2{%
  \PackageError{forest}{#2 is out of bounds of array "#1"
    (\the\csuse{#1M}--\the\csuse{#1N}).}{}}%
```

Define array macros. For speed, we define most of them to be "direct", i.e. cointain the resolved control sequences specific to this array.

```
\def\forest@newarray@#1{%
  % array bounds: M <= i < N
  \expandafter\newcount\csname#1M\endcsname
  \expandafter\newcount\csname#1N\endcsname
  \csedef{#1clear}{%
    \ifforest@temp\global\fi\expandonce{\csname#1M\endcsname}0
    \ifforest@temp\global\fi\expandonce{\csname#1N\endcsname}0
  }%
  \csedef{#1ifempty}{%
    \noexpand\ifnum\expandonce{\csname#1M\endcsname}<\expandonce{\csname#1N\endcsname}\relax
      \unexpanded{\expandafter\@secondoftwo
    \else
      \expandafter\@firstoftwo
    \fi}%
  }%
  \csedef{#1length}{% a numexpr
    \noexpand\numexpr\expandonce{\csname#1N\endcsname}-\expandonce{\csname#1M\endcsname}\relax
  }%
  \csedef{#1checkrange}##1##2{% args can be \numexprs
    \noexpand\forest@tempfalse
    \noexpand\ifnum\numexpr##1<\expandonce{\csname#1M\endcsname}\relax
      \noexpand\forest@temptrue
    \noexpand\fi
    \noexpand\ifnum\numexpr##2>\expandonce{\csname#1N\endcsname}\relax
      \noexpand\forest@temptrue
    \noexpand\fi
    \noexpand\ifforest@temp
      \noexpand\forest@array@oub@error{#1}{Range "\noexpand\number\noexpand\numexpr##1\relax--\noexpand\numbe
    \noexpand\fi
  }%
  \csedef{#1checkindex}##1{% arg can be a \numexpr
    \noexpand\forest@tempfalse
    \noexpand\ifnum\numexpr##1<\expandonce{\csname#1M\endcsname}\relax
      \noexpand\forest@temptrue
    \noexpand\fi
    \noexpand\ifnum\numexpr##1<\expandonce{\csname#1N\endcsname}\relax
    \noexpand\else
      \noexpand\forest@temptrue
    \noexpand\fi
    \noexpand\ifforest@temp
      \noexpand\forest@array@oub@error{#1}{Index "\noexpand\number\noexpand\numexpr##1\relax"}%
    \noexpand\fi
  }%
  \csedef{#1get}##1##2{% ##1 = index, ##2 = receiving cs
    \expandonce{\csname#1checkindex\endcsname}{##1}%
    \noexpand\letcs##2{#1##1}%
  }%
  \csedef{#1get@}##1##2{% ##1 = index, ##2 = receiving cs (don't check bounds)
    \noexpand\letcs##2{#1##1}%
  }%
  \csedef{#1toppop}##1{% ##1 = receiving cs
    \expandonce{\csname#1ifempty\endcsname}{%
      \noexpand\forest@array@empty@error{#1}%
    }{%
      \ifforest@temp\global\fi\advance\expandonce{\csname#1N\endcsname}-1
      \noexpand\letcs\noexpand##1{#1\noexpand\the\expandonce{\csname#1N\endcsname}}%
    }%
  }%
  \InlineNoDef{\csdef{#1bottompop}##1{% ##1 = receiving cs
    \Expand{\csname#1ifempty\endcsname}{%
      \forest@array@empty@error{#1}%
    }{%
      \letcs##1{#1\the\Expand{\csname#1M\endcsname}}%
      \ExpandIfT{forest@temp}\global\advance\Expand{\csname#1M\endcsname 1}%
    }%
  }}%
  % \csdef{#1bottompop}##1{}% we need this as \Inline chokes on \let\macro=\relax
  % \expandafter\Inline\expandafter\def\csname#1bottompop\endcsname##1{% ##1 = receiving cs
  %   \Expand{\csname#1ifempty\endcsname}{%
  %     \forest@array@empty@error{#1}%
  %   }{%
  %     \letcs##1{#1\the\Expand{\csname#1M\endcsname}}%
  %     \ExpandIfT{forest@temp}\global\advance\Expand{\csname#1M\endcsname 1}%
  %   }%
  % }%
  % \csedef{#1bottompop}##1{% ##1 = receiving cs
  %   \expandonce{\csname#1ifempty\endcsname}{%
  %     \noexpand\forest@array@empty@error{#1}%
  %   }{%
  %     \noexpand\letcs\noexpand##1{#1\noexpand\the\expandonce{\csname#1M\endcsname}}%
  %     \ifforest@temp\global\fi\advance\expandonce{\csname#1M\endcsname}1
  %   }%
  % }%
  \csedef{#1setappend}##1{% ##1 = definition
    \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi
      {#1\noexpand\the\expandonce{\csname#1N\endcsname}}%
      {\noexpand\unexpanded{##1}}%
    \ifforest@temp\global\fi\advance\expandonce{\csname#1N\endcsname}1
  }%
  \csedef{#1setappend@}##1##2{% ##1 = continue by, ##2 = definition
    \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi
      {#1\noexpand\the\expandonce{\csname#1N\endcsname}}%
      {\noexpand\unexpanded{##2}}%
    \ifforest@temp\global\fi\advance\expandonce{\csname#1N\endcsname}1
    ##1%
  }%
  \csedef{#1setprepend}##1{% ##1 = definition
    \ifforest@temp\global\fi\advance\expandonce{\csname#1M\endcsname}-1
    \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi
      {#1\noexpand\the\expandonce{\csname#1M\endcsname}}%
      {\noexpand\unexpanded{##1}}%
  }%
  \csedef{#1esetappend}##1{% ##1 = definition
    \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi{#1\noexpand\the\expandonce{\csname#1N\endcsname}}{
    \ifforest@temp\global\fi\advance\expandonce{\csname#1N\endcsname}1
  }%
  \csedef{#1esetprepend}##1{% ##1 = definition
    \ifforest@temp\global\fi\advance\expandonce{\csname#1M\endcsname}-1
    \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi{#1\noexpand\the\expandonce{\csname#1M\endcsname}}{
  }%
  \csedef{#1letappend}##1{% ##1 = cs
    \ifforest@temp\noexpand\expandafter\noexpand\global\fi\noexpand\expandafter\noexpand\let
      \noexpand\csname#1\noexpand\the\expandonce{\csname#1N\endcsname}\noexpand\endcsname
      ##1%
    \ifforest@temp\global\fi\advance\expandonce{\csname#1N\endcsname}1
  }%
  \csedef{#1letprepend}##1{% ##1 = cs
    \ifforest@temp\global\fi\advance\expandonce{\csname#1M\endcsname}-1
    \ifforest@temp\noexpand\expandafter\noexpand\global\fi\noexpand\expandafter\noexpand\let
      \noexpand\csname#1\noexpand\the\expandonce{\csname#1M\endcsname}\noexpand\endcsname
      ##1%
  }%
```

I would love to define these only generically, as they will not be called often, but they need to be expandable. Argh. right?

```
\def\arrayvalues{% <-- \csedef{#1values}
  \expandafter\expandafter\expandafter\arrayvaluesfromrange %\arrayvaluesfromrange <-- \expandonce{\csname#
  \expandafter\expandafter\expandafter{%
  \expandafter\the
  \expandafter\arrayM %\arrayM <-- \expandonce{\csname#1M\endcsname}%
  \expandafter}%
 \expandafter{%
   \the\arrayN %\arrayN <-- \expandonce{\csname#1N\endcsname}%
  }%
}%
```

```
 \csedef{#1values}{%
  \noexpand\expandafter\noexpand\expandafter\noexpand\expandafter\expandonce{\csname#1valuesfromrange\endcs
   \noexpand\expandafter\noexpand\expandafter\noexpand\expandafter{%
    \noexpand\expandafter\noexpand\the
    \noexpand\expandafter\expandonce{\csname#1M\endcsname}%
    \noexpand\expandafter}%
   \noexpand\expandafter{\noexpand\the\expandonce{\csname#1N\endcsname}}%
 }%
```

```
\def\arrayvaluesfromrange##1##2{% ##1/##2 = lower/upper bounds (we receive them expanded) <-- \csedef{#1vue
 \ifnum##1<##2
  {\expandafter\expandonce\expandafter{\csname#1##1\endcsname}}% here we add braces (for the general case
  \expandafter\@escapeif\expandafter{\expandafter\arrayvaluesfromrange % <-- \expandonce{\csname#1valuesf
   \expandafter{\number\numexpr##1+1}{##2}}%
 \fi
}%
```

As we need this to be expandable, we cannot check the range within the macro. You need to to this on your own using `...checkrange` defined above.

```
 \csedef{#1valuesfromrange}##1##2{% ##1/##2 = lower/upper bounds (we receive them expanded)
  \noexpand\ifnum##1<##2
   {\noexpand\expandafter\noexpand\expandonce\noexpand\expandafter{\noexpand\csname#1##1\noexpand\endcsnam
   \noexpand\expandafter\noexpand\@escapeif\noexpand\expandafter{\noexpand\expandafter\expandonce{\csname#
    \noexpand\expandafter{\noexpand\number\noexpand\numexpr##1+1}{##2}}%
  \noexpand\fi
 }%
```

Puts all items until `\forest@eov` into the array. After that is done, execute `\forest@topextend@next` (Why this macro? So that we can extend the array by tokens never seen before.). This code is difficult and not run often, so it doesn't need specialized control sequences.

```
 \csdef{#1topextend}{\def\forest@array@currentarray{#1}\forest@array@topextend}%
}
\def\forest@array@topextend{\futurelet\forest@ate@next@token\forest@ate@checkforspace}
\def\forest@ate@checkforspace{%
 \expandafter\ifx\space\forest@ate@next@token
  \expandafter\forest@ate@havespace
 \else
  \expandafter\forest@ate@checkforgroup
 \fi
}
\def\forest@ate@havespace{\expandafter\forest@array@topextend\romannumeral-`0}%
\def\forest@ate@checkforgroup{%
 \ifx\forest@ate@next@token\bgroup
  \expandafter\forest@ate@appendgroup
 \else
  \expandafter\forest@ate@checkforeov
 \fi
}
\def\forest@ate@appendgroup{%
 \expandonce{\csname\forest@array@currentarray setappend@\endcsname}\forest@array@topextend
}
\def\forest@ate@checkforeov{%
 \ifx\forest@ate@next@token\forest@eov
  \expandafter\forest@ate@finish
  \else
    \expandafter\forest@ate@appendtoken
  \fi
}
\def\forest@ate@appendtoken#1{%
  \expandonce{\csname\forest@array@currentarray setappend\endcsname}{#1}%
  \forest@array@topextend
}
\def\forest@ate@finish\forest@eov{\forest@topextend@next}
\let\forest@topextend@next\relax
\forest@newarray\forest@temparray@
\forest@newglobalarray\forest@global@temparray@
```

4.2 Testing for numbers and dimensions

Test if the argument is an integer (only base 10) that can be assigned to a TEX count register. This is supposed to be a fast, not complete test, as anything not recognized as an integer will be passed on to `pgfmath` (by the code that uses these macros).

We support `+`s, `-`s and spaces before the number. We don't support count registers.

Dillema? Should `0abc` be interpreted as TEX style (decimal) or PGF style (octal)? We go for TEX style.

The return value will hide in TEX-style `\if`-macro `\forest@isnum` and counter `\forest@isnum@count`.

```
\def\forest@eon{ }
\newif\ifforest@isnum@minus
\newif\ifforest@isnum
\def\forest@isnum#1{%
  \forest@isnum@minusfalse
  \let\forest@isnum@next\forest@isnum@finish
```

Expand in advance, like pgfmath does.

```
  \edef\forest@isnum@temp{#1}%
```

Add two end-of-value markers. The first one might be eaten by count assignment: that's why there are two and they expand to a space.

```
  \expandafter\forest@isnum@a\forest@isnum@temp\forest@eon\forest@eon\forest@END
  \ifforest@isnum
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
}
\def\forest@isnum@a{\futurelet\forest@isnum@token\forest@isnum@b}
```

Test for three special characters: `-`, `+`, and space.

```
\def\forest@isnum@minustoggle{%
  \ifforest@isnum@minus\forest@isnum@minusfalse\else\forest@isnum@minustrue\fi
}
\def\forest@isnum@b{%
  \let\forest@next\forest@isnum@p
  \ifx-\forest@isnum@token
    \forest@isnum@minustoggle
    \let\forest@next\forest@isnum@c
  \else
    \ifx+\forest@isnum@token
      \let\forest@next\forest@isnum@c
    \else
      \expandafter\ifx\space\forest@isnum@token
        \let\forest@next\forest@isnum@s
      \fi
    \fi
  \fi
  \forest@next
}
```

Eat `+` and `-`.

```
\def\forest@isnum@c#1{\forest@isnum@a}%
```

Eat the space!

```
\def\forest@isnum@s#1{\forest@isnum@a#1}%
\newcount\forest@isnum@count
```

Check for `0`. Why? If we have one, we know that the initial argument started with a number, so we have a chance that it is a number even if our assignment will yield 0. If we have no `0` and the assignment yields 0, we know we don't have a number.

```
\def\forest@isnum@p{%
  \ifx0\forest@isnum@token
    \let\forest@next\forest@isnum@next
  \else
    \let\forest@next\forest@isnum@nz@
  \fi
  \forest@isnumtrue
  \afterassignment\forest@isnum@q\forest@isnum@count\ifforest@isnum@minus-\fi0%
}
\def\forest@isnum@q{%
  \futurelet\forest@isnum@token\forest@next
}
\def\forest@isnum@nz@{%
  \ifnum\forest@isnum@count=0
    \forest@isnumfalse
  \fi
  \forest@isnum@next
}
```

This is the end of testing for an integer. If we have left-over stuff (`#1`), this was not a number.

```
\def\forest@isnum@finish#1\forest@END{%
  \ifx\forest@isnum@token\forest@eon
  \else
    \forest@isnumfalse
  \fi
}
```

Is it a dimension? We support all TEX's units but `true` units. Also supported are unitless dimensions (i.e. decimal numbers), which are interpreted as `pt`s, as in pgfmath.

The return value will hide in TEX-style `\if`-macro `\forest@isdim` and counter `\forest@isdim@dimen`.

```
\newcount\forest@isdim@nonintpart
\newif\ifforest@isdim
\def\forest@isdim#1{%
  \forest@isdimfalse
  \forest@isnum@minusfalse
  \def\forest@isdim@leadingzeros{}%
  \forest@isdim@nonintpart=0
  \def\forest@isdim@unit{pt}%
  \let\forest@isnum@next\forest@isdim@checkfordot
  \edef\forest@isnum@temp{#1}%
```

4 end-of-value markers (`forest@eon`): one can be eaten by number (after the dot), two by a non-existing unit.

```
  \expandafter\forest@isnum@a\forest@isnum@temp\forest@eon\forest@eon\forest@eon\forest@eon\forest@END
  \ifforest@isdim
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
```

```
}
\def\forest@isdim@checkfordot{%
  \ifx.\forest@isnum@token
    \expandafter\forest@isdim@dot
  \else
    \ifx,\forest@isnum@token
      \expandafter\expandafter\expandafter\forest@isdim@dot
    \else
      \expandafter\expandafter\expandafter\forest@isdim@nodot
    \fi
  \fi
}
\def\forest@isdim@nodot{%
  \ifforest@isnum
```

No number, no dot, so not a dimension.

```
    \expandafter\forest@isdim@checkforunit
  \else
    \expandafter\forest@isdim@finish@nodim
  \fi
}
\def\forest@isdim@dot#1{% #1=. or ,
  \futurelet\forest@isnum@token\forest@isdim@collectzero
}
\def\forest@isdim@collectzero{%
  \ifx0\forest@isnum@token
    \expandafter\forest@isdim@collectzero@
  \else
    \expandafter\forest@isdim@getnonintpart
  \fi
}
\def\forest@isdim@collectzero@#1{% #1 = 0
  \appto\forest@isdim@leadingzeros{0}%
  \futurelet\forest@isnum@token\forest@isdim@collectzero
}
\def\forest@isdim@getnonintpart{%
  \afterassignment\forest@isdim@checkforunit\forest@isdim@nonintpart0%
}
```

Nothing else should be defined in `\forest@unit@` namespace.

```
\def\forest@def@unit#1{\csdef{forest@unit@#1}{#1}}
\forest@def@unit{em}
\forest@def@unit{ex}
\forest@def@unit{pt}
\forest@def@unit{pc}
\forest@def@unit{in}
\forest@def@unit{bp}
\forest@def@unit{cm}
\forest@def@unit{mm}
\forest@def@unit{dd}
\forest@def@unit{cc}
\forest@def@unit{sp}
\def\forest@isdim@checkforunit#1#2{%
  \lowercase{\edef\forest@isnum@temp{\detokenize{#1#2}}}%
  \ifcsname forest@unit@\forest@isnum@temp\endcsname
    \let\forest@isdim@next\forest@isdim@finish@dim
    \edef\forest@isdim@unit{\csname forest@unit@\forest@isnum@temp\endcsname}%
  \else
    \ifx#1\forest@eon
      \let\forest@isdim@next\forest@isdim@finish@dim
    \else
      \let\forest@isdim@next\forest@isdim@finish@nodim
    \fi
  \fi
  \forest@isdim@next
}
\def\forest@isdim@finish@dim{%
  \futurelet\forest@isnum@token\forest@isdim@finish@dim@a
}
\def\forest@isdim@finish@dim@a{%
  \expandafter\ifx\space\forest@isnum@token
    \expandafter\forest@isdim@finish@dim@b
  \else
    \expandafter\forest@isdim@finish@dim@c
  \fi
}
\expandafter\def\expandafter\forest@isdim@finish@dim@b\space{% eat one space
  \futurelet\forest@isnum@token\forest@isdim@finish@dim@c
}
\def\forest@isdim@finish@dim@c#1\forest@END{%
  \ifx\forest@isnum@token\forest@eon
    \forest@isdimtrue
    \forest@isdim@dimen\the\forest@isnum@count.\forest@isdim@leadingzeros\the\forest@isdim@nonintpart\forest@
  \else
    \forest@isdimfalse
  \fi
}
\def\forest@isdim@finish@nodim#1\forest@END{%
  \forest@isdimfalse
}
\newdimen\forest@isdim@dimen
 % \long\def\@firstofthree#1#2#3{#3} % defined by LaTeX
\long\def\@firstofthree#1#2#3{#1}
\long\def\@secondofthree#1#2#3{#2}
\def\forest@isnumdim#1{%
  \forest@isdim{#1}{%
    \forest@isnumdim@
  }{%
    \@thirdofthree
  }%
}
\def\forest@isnumdim@{%
  \ifforest@isnum
    \expandafter\@firstofthree
  \else
    \expandafter\@secondofthree
  \fi
}
```

4.3 forestmath

We imitate `pgfmath` a lot, but we remember the type of the result so that we can use TEX's primitives when possible.

```
\def\forestmathtype@generic{_} % generic (token list)
\def\forestmathtype@count{n} % integer
\def\forestmathtype@dimen{d} % a dimension: <decimal> pt
\def\forestmathtype@unitless{P} % <decimal> (a unitless dimension) (P because pgfmath returns such numbers)
\def\forestmathtype@textasc{t} % text (ascending)
\def\forestmathtype@textdesc{T} % text (descending)
\def\forestmathtype@none{} % internal (for requests - means whatever)
\def\forestmathresult{}
\let\forestmathresulttype\forestmathtype@generic
```

`\forest@tryprocess` takes four “arguments”. The first is a true/false switch telling whether to return the full result array in case we have a `.process` expression. The second is a forestmath expression, delimited by `\forest@spacegen`: if it starts with a `>`, we take it to be a `.process` expression, evaluate it using `\forest@process`, and execute the third argument; it it doesn’t, we execute the fourth argument.

```
\def\forest@tryprocess#1{%
  \def\forest@tryprocess@returnarray{#1}%
  \expandafter\forest@tryprocess@a\romannumeral-`0}
\def\forest@tryprocess@a{\futurelet\forest@temp@token\forest@tryprocess@b}
\def\forest@tryprocess@b{%
  \ifx>\forest@temp@token
    \expandafter\forest@tryprocess@yes
  \else
    \expandafter\forest@tryprocess@no
  \fi
}
\def\forest@spacegen{ \forest@spacegen}
\def\forest@tryprocess@yes#1#2\forest@spacegen{%
  \expandafter\forest@process\expandafter{\forest@tryprocess@returnarray}#2\forest@eov
  \@firstoftwo
}
\def\forest@tryprocess@no#1\forest@spacegen{\@secondoftwo}
```

Forestmath versions of pgfmath macros. They accept process and pgfmath expressions, as described above. In the case of a pgfmath expression, they use `\forest@isnum` and `\forest@isdim` for to see if they can avoid pgfmath evaluation. (These checks are generally faster than pgfmath’s fast track.)

```
\def\forestmathsetcount#1#2{%
  \forest@tryprocess{false}#2\forest@spacegen{%
    #1=\forest@process@result\relax
  }{%
    \forestmathsetcount@#1{#2}%
  }%
}
\def\forestmathsetcount@#1#2{%
  \forest@isnum{#2}{%
    #1=\forest@isnum@count
  }{%
    \pgfmathsetcount#1{#2}%
  }%
}
\def\forestmathsetlength#1#2{%
  \forest@tryprocess{false}#2\forest@spacegen{%
    #1=\forest@process@result\relax
  }{%
    \forestmathsetlength@#1{#2}%
  }%
}
\def\forestmathsetlength@#1#2{%
  \forest@isdim{#2}{%
    #1=\forest@isdim@dimen
  }{%
    \pgfmathsetlength#1{#2}%
  }%
}
\def\forestmathtruncatemacro#1#2{%
  \forest@tryprocess{false}#2\forest@spacegen{%
    \forest@temp@count=\forest@process@result\relax
    \edef#1{\the\forest@temp@count}%
  }{%
    \forestmathtruncatemacro@#1{#2}%
  }%
}
\def\forestmathtruncatemacro@#1#2{%
  \forest@isnum{#2}{%
    \edef#1{\the\forest@isnum@count}%
  }{%
    \pgfmathtruncatemacro#1{#2}%
  }%
}
\def\forestmathsetlengthmacro#1#2{%
  \forest@tryprocess{false}#2\forest@spacegen{%
    \forest@temp@dimen=\forest@process@result\relax
    \edef#1{\the\forest@temp@dimen}%
  }{%
    \forestmathsetlengthmacro@#1{#2}%
  }%
}
\def\forestmathsetlengthmacro@#1#2{%
  \forest@isdim{#2}{%
    \edef#1{\the\forest@isdim@dimen}%
  }{%
    \pgfmathsetlengthmacro#1{#2}%
  }%
}
\def\forestmathsetmacro#1#2{%
  \forest@tryprocess{false}#2\forest@spacegen{%
    \let#1\forest@process@result
    \let\forestmathresulttype\forest@process@result@type
  }{%
    \forestmathsetmacro@#1{#2}%
    \let\forestmathresulttype\forestmathtype@unitless
  }%
}
\def\forestmathsetmacro@#1#2{%
  \forest@isdim{#2}{%
    \edef#1{\expandafter\Pgf@geT\the\forest@isdim@dimen}%
  }{%
    \pgfmathsetmacro#1{#2}%
  }%
}
\def\forestmathparse#1{%
  \forest@tryprocess{false}#1\forest@spacegen{%
    \let\forestmathresult\forest@process@result
    \let\forestmathresulttype\forest@process@result@type
  }{%
    \forestmathparse@{#1}%
    \let\forestmathresulttype\forestmathtype@unitless
  }%
}
\def\forestmathparse@#1{%
  \forest@isdim{#1}{%
    \edef\forestmathresult{\expandafter\Pgf@geT\the\forest@isdim@dimen}%
  }{%
    \pgfmathsetmacro\forestmathresult{#1}%
  }%
}
```

The following macro, which is the only place that sets `\forest@tryprocess`'s `#1` to `true`, is actually not used anywhere. It was meant for an argument processor instruction accepting ⟨*forestmath*⟩, but that got separated into P and p. Not much harm is done by keeping it, however, so we do, just in case.

```
  %\def\forestmathparse@returnarray#1{% same as above, but returns the result as an array (used only internal
  %  \forest@tryprocess{true}#1\forest@spacegen{}{%
```

```
%     \forestmathparse@{#1}%
%     \let\forest@process@result@type\forestmathtype@unitless
%     \forest@process@result@clear
%     \forest@process@result@letappend\forestmathresult
%   }%
%}
```

Evaluates `#1` to a boolean: if true execute `#2`, otherwise `#3`. `#2` and `#3` are TEX code. Includes a shortcut for some common values.

```
\csdef{forest@bh@0}{0}
\csdef{forest@bh@false}{0}
\csdef{forest@bh@1}{1}
\csdef{forest@bh@true}{1}
\def\forestmath@if#1{%
  \ifcsdef{forest@bh@\detokenize{#1}}{%
    \let\forest@next\forestmath@if@fast
  }{%
    \let\forest@next\forestmath@if@slow
  }%
  \forest@next{#1}%
  \ifnum\forest@temp=0
    \expandafter\@secondoftwo
  \else
    \expandafter\@firstoftwo
  \fi
}
\def\forestmath@if@fast#1{\letcs\forest@temp{forest@bh@\detokenize{#1}}}
\def\forestmath@if@slow#1{\forestmathtruncatemacro\forest@temp{#1}}
```

These macros expandably convert a num(n)/dim(d)/unitless dim(P) to a num(n)/dim(d)/unitless dim(P).

```
\def\forestmath@convert@fromto#1#2#3{%
  \edef\forestmathresult{\csname forestmath@convert@from@#1@to@#2\endcsname{#3}}}
\def\forestmath@convert@from#1{\forestmath@convert@fromto{#1}{\forestmathresulttype}}
\def\forestmath@convert@to{\forestmath@convert@fromto{\forestmathresulttype}}
\def\forestmath@convert@from@n@to@n#1{#1}
\def\forestmath@convert@from@n@to@d#1{#1\pgfmath@pt}
\def\forestmath@convert@from@n@to@P#1{#1}
\def\forestmath@convert@from@d@to@n#1{%
    \expandafter\forestmath@convert@uptodot\Pgf@geT#1.\forest@eov}
\def\forestmath@convert@from@d@to@d#1{#1}
\def\forestmath@convert@from@d@to@P#1{\Pgf@geT#1}
\def\forestmath@convert@from@P@to@n#1{%
    \forestmath@convert@uptodot#1.\forest@eov}
\def\forestmath@convert@from@P@to@d#1{#1\pgfmath@pt}
\def\forestmath@convert@from@P@to@P#1{#1}
\def\forestmath@convert@uptodot#1.#2\forest@eov{#1}
\def\forestmathzero{\forestmath@convert@from\forestmathtype@count{0}}
```

These defer conversion (see aggregates).

```
\csdef{forestmath@convert@from@n@to@_}#1{\unexpanded{#1}}
\csdef{forestmath@convert@from@d@to@_}#1{\unexpanded{#1}}
\csdef{forestmath@convert@from@P@to@_}#1{\unexpanded{#1}}
```

Sets `\pgfmathresulttype` to the type of `#1`.

```
\def\forestmathsettypefrom#1{%
  \forest@isnumdim{%
    \let\forestmathresulttype\forestmathtype@count
  }{%
    \let\forestmathresulttype\forestmathtype@dimen
  }{%
    \let\forestmathresulttype\forestmathtype@unitless
  }%
}
```

The following functions expect numbers or (bare or specified) dimensions as their parameters. The version ending in `@` should get the argument type as its first argument; the version without `@` uses `\forestmathresulttype`. The result type doesn't need to be changed, obviously.

```
\def\forestmathadd#1#2{\edef\forestmathresult{%
    \csname forestmathadd@\forestmathresulttype\endcsname{#1}{#2}}}
\def\forestmathadd@#1#2#3{\edef\forestmathresult{%
    \csname forestmathadd@#1\endcsname{#2}{#3}}}
\def\forestmathadd@n#1#2{\the\numexpr#1+#2\relax}
\def\forestmathadd@d#1#2{\the\dimexpr#1+#2\relax}
\def\forestmathadd@P#1#2{\expandafter\Pgf@geT\the\dimexpr#1pt+#2pt\relax}
\def\forestmathmultiply#1#2{%
  \csname forestmathmultiply@\forestmathresulttype\endcsname{#1}{#2}}
\def\forestmathmultiply@#1#2#3{%
  \csname forestmathmultiply@#1\endcsname{#2}{#3}}
\def\forestmathmultiply@n#1#2{\edef\forestmathresult{%
    \the\numexpr#1*#2\relax}}
\def\forestmathmultiply@d#1#2{%
  \edef\forestmath@marshal{\forestmathmultiply@d@{#1}{#2}}\forestmath@marshal
}
\def\forestmathmultiply@d@#1#2{%
  \edef\forestmath@marshal{%
    \noexpand\pgfmathmultiply@{\Pgf@geT#1}{\Pgf@geT#2}%
  }\forestmath@marshal
  \edef\forestmathresult{\pgfmathresult\pgfmath@pt}%
}
\def\forestmathmultiply@P#1#2{%
  \pgfmathmultiply@{#1}{#2}%
  \let\forestmathresult\pgfmathresult
}
```

The return type of `forestmathdivide` is the type of the dividend. So, `n` and `d` type can only be divided by integers; as `\numexpr` and `\dimexpr` are used, the result is rounded.

```
\def\forestmathdivide#1#2{%
  \csname forestmathdivide@\forestmathresulttype\endcsname{#1}{#2}}
\def\forestmathdivide@#1#2#3{%
  \csname forestmathdivide@#1\endcsname{#2}{#3}}
\def\forestmathdivide@n#1#2{\edef\forestmathresult{%
    \the\numexpr#1/#2\relax}}
\def\forestmathdivide@d#1#2{\edef\forestmathresult{%
    \the\dimexpr#1/#2\relax}}
\def\forestmathdivide@P#1#2{%
  \edef\forest@marshal{%
    \noexpand\pgfmathdivide{+#1}{+#2}%
  }\forest@marshal
  \let\forestmathresult\pgfmathresult
}
```

Booleans.

```
\def\forestmathtrue{%
  \def\forestmathresult{1}%
  \let\forestmathresulttype\forestmathtype@count}
\def\forestmathfalse{%
  \def\forestmathresult{0}%
  \let\forestmathresulttype\forestmathtype@count}
```

Comparisons. `\pdfstrcmp` is used to compare text (types `t` and `T`); note that it expands its arguments. `<` and `>` comparison of generic type obviously makes no sense; `=` comparison is done using `\ifx`: this is also the reason why these macros are not fully expandable, as we need to `\def` the arguments to `\ifx`.

Low level `<`.

```
\def\forestmath@if@lt@n#1#2{\ifnum#1<#2\relax
  \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
\def\forestmath@if@lt@d#1#2{\ifdim#1<#2\relax
  \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
\def\forestmath@if@lt@P#1#2{\ifdim#1pt<#2pt
  \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
\def\forestmath@if@lt@t#1#2{\ifnum\pdfstrcmp{#1}{#2}<0
  \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
\def\forestmath@if@lt@T#1#2{\ifnum\pdfstrcmp{#1}{#2}>0
  \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
\def\forest@cmp@error#1#2{\PackageError{forest}{Comparison
    ("<" or ">") of generic type arguments "#1" and "#2"
    makes no sense}{Use one of argument processor instructions
    "n", "d", "P" or "t" to change the type. Use package option
    "debug=process" to see what's happening here.}}
\cslet{forestmath@if@lt@_}\forest@cmp@error
```

Low level =.

```
\def\forestmath@if@eq@n#1#2{\ifnum#1=#2\relax
  \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
\def\forestmath@if@eq@d#1#2{\ifdim#1=#2\relax
  \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
\def\forestmath@if@eq@P#1#2{\ifdim#1pt=#2pt
  \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
\def\forestmath@if@eq@t#1#2{\ifnum\pdfstrcmp{#1}{#2}=0
  \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
\let\forestmath@if@eq@T\forestmath@if@eq@t
\csdef{forestmath@if@eq@_}#1#2{%
  \def\forestmath@tempa{#1}%
  \def\forestmath@tempb{#2}%
  \ifx\forestmath@tempa\forestmath@tempb
    \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
```

High level <, > and =.

```
\def\forestmathlt#1#2{%
  \csname forestmath@if@lt@\forestmathresulttype\endcsname{#1}{#2}%
    \forestmathtrue
    \forestmathfalse}
\def\forestmathlt@#1#2#3{%
  \csname forestmath@if@lt@#1\endcsname{#2}{#3}%
    \forestmathtrue
    \forestmathfalse}
\def\forestmathgt#1#2{%
  \csname forestmath@if@lt@\forestmathresulttype\endcsname{#2}{#1}%
    \forestmathtrue
    \forestmathfalse}
\def\forestmathgt@#1#2#3{%
  \csname forestmath@if@lt@#1\endcsname{#3}{#2}%
    \forestmathtrue
    \forestmathfalse}
\def\forestmatheq#1#2{%
  \csname forestmath@if@eq@\forestmathresulttype\endcsname{#1}{#2}%
    \forestmathtrue
    \forestmathfalse}
\def\forestmatheq@#1#2#3{%
  \csname forestmath@if@eq@#1\endcsname{#2}{#3}%
    \forestmathtrue
    \forestmathfalse}
```

Min and max. The complication here is that for numeric/dimension types, we want the empty value to signal "no argument", i.e. the other argument should be the result; this is used in aggregates. (For text types, the empty value is obviously the lesser one.) The arguments are expanded.

```
\def\forestmathmin{\forestmath@minmax{min}{\forestmathresulttype}}
\def\forestmathmax{\forestmath@minmax{max}{\forestmathresulttype}}
\def\forestmathmin@{\forestmath@minmax{min}}
\def\forestmathmax@{\forestmath@minmax{max}}
\def\forestmath@minmax#1#2#3#4{% #1=min/max, #2=type, #3,#4=args
  \edef\forestmath@tempa{#3}%
  \edef\forestmath@tempb{#4}%
  \if\relax\detokenize\expandafter{\forestmath@tempa}\relax
    \forestmath@minmax@one{#1}{#2}\forestmath@tempb
  \else
    \if\relax\detokenize\expandafter{\forestmath@tempb}\relax
      \forestmath@minmax@one{#1}{#2}\forestmath@tempa
    \else
      \csname forestmath@#1\endcsname{#2}%
    \fi
  \fi
}
\def\forestmath@minmax@one#1#2#3{% #1=min/max, #2=type, #3 = the (possibly) non-empty arg
  \ifcsname forestmath@#1@one@#2\endcsname
    \csname forestmath@#1@one@#2\endcsname#3%
  \else
    \let\forestmathresult#3%
  \fi
}
\def\forestmath@min@one@t#1{\let\forestmathresult\forest@empty}
\def\forestmath@max@one@t#1{\let\forestmathresult#1}
\def\forestmath@min@one@T#1{\let\forestmathresult#1}
\def\forestmath@max@one@T#1{\let\forestmathresult\forest@empty}

\def\forestmath@min#1{% #1 = type
  \csname forestmath@if@lt@#1\endcsname\forestmath@tempa\forestmath@tempb
    {\let\forestmathresult\forestmath@tempa}%
    {\let\forestmathresult\forestmath@tempb}%
}
\def\forestmath@max#1{% #1 = type
  \csname forestmath@if@lt@#1\endcsname\forestmath@tempa\forestmath@tempb
    {\let\forestmathresult\forestmath@tempb}%
    {\let\forestmathresult\forestmath@tempa}%
}
```

4.4 Sorting

Macro `\forest@sort` is the user interface to sorting.

The user should prepare the data in an arbitrarily encoded array,[1] and provide the sorting macro (given in `#1`) and the array let macro (given in `#2`): these are the only ways in which sorting algorithms access the data. Both user-given macros should take two parameters, which expand to array indices. The comparison macro should compare the given array items and call `\forest@sort@cmp@gt`, `\forest@sort@cmp@lt` or `\forest@sort@cmp@eq` to signal that the first item is greater than, less than, or equal to the second item. The let macro should "copy" the contents of the second item onto the first item.

The sorting direction is be given in `#3`: it can one of `\forest@sort@ascending` and `\forest@sort@descending`. `#4` and `#5` must expand to the lower and upper (both inclusive) indices of the array to be sorted.

`\forest@sort` is just a wrapper for the central sorting macro `\forest@@sort`, storing the comparison macro, the array let macro and the direction. The central sorting macro and the algorithm-specific macros take only two arguments: the array bounds.

```
\def\forest@sort#1#2#3#4#5{%
```

---

[1] In forest, arrays are encoded as families of macros. An array-macro name consists of the (optional, but recommended) prefix, the index, and the (optional) suffix (e.g. `\forest@42x`). Prefix establishes the "namespace", while using more than one suffix simulates an array of named tuples. The length of the array is stored in macro `\<prefix>n`.

```
  \let\forest@sort@cmp#1\relax
  \let\forest@sort@let#2\relax
  \let\forest@sort@direction#3\relax
  \forest@@sort{#4}{#5}%
}
```

The central sorting macro. Here it is decided which sorting algorithm will be used: for arrays at least `\forest@quicksort@minarraylength` long, quicksort is used; otherwise, insertion sort.

```
\def\forest@quicksort@minarraylength{10000}
\def\forest@@sort#1#2{%
  \ifnum#1<#2\relax\@escapeif{%
    \forest@sort@m=#2
    \advance\forest@sort@m -#1
    \ifnum\forest@sort@m>\forest@quicksort@minarraylength\relax\@escapeif{%
      \forest@quicksort{#1}{#2}%
    }\else\@escapeif{%
      \forest@insertionsort{#1}{#2}%
    }\fi
  }\fi
}
```

Various counters and macros needed by the sorting algorithms.

```
\newcount\forest@sort@m\newcount\forest@sort@k\newcount\forest@sort@p
\def\forest@sort@ascending{>}
\def\forest@sort@descending{<}
\def\forest@sort@cmp{%
  \PackageError{sort}{You must define forest@sort@cmp function before calling
    sort}{The macro must take two arguments, indices of the array
    elements to be compared, and return '=' if the elements are equal
    and '>'/'<' if the first is greater /less than the secong element.}%
}
\def\forest@sort@cmp@gt{\def\forest@sort@cmp@result{>}}
\def\forest@sort@cmp@lt{\def\forest@sort@cmp@result{<}}
\def\forest@sort@cmp@eq{\def\forest@sort@cmp@result{=}}
\def\forest@sort@let{%
  \PackageError{sort}{You must define forest@sort@let function before calling
    sort}{The macro must take two arguments, indices of the array:
    element 2 must be copied onto element 1.}%
}
```

Quick sort macro (adapted from laansort).

```
\newloop\forest@sort@loop
\newloop\forest@sort@loopA
\def\forest@quicksort#1#2{%
```

Compute the index of the middle element (`\forest@sort@m`).

```
  \forest@sort@m=#2
  \advance\forest@sort@m -#1
  \ifodd\forest@sort@m\relax\advance\forest@sort@m1 \fi
  \divide\forest@sort@m 2
  \advance\forest@sort@m #1
```

The pivot element is the median of the first, the middle and the last element.

```
  \forest@sort@cmp{#1}{#2}%
  \if\forest@sort@cmp@result=%
    \forest@sort@p=#1
  \else
    \if\forest@sort@cmp@result>%
      \forest@sort@p=#1\relax
    \else
      \forest@sort@p=#2\relax
    \fi
    \forest@sort@cmp{\the\forest@sort@p}{\the\forest@sort@m}%
    \if\forest@sort@cmp@result<%
    \else
      \forest@sort@p=\the\forest@sort@m
    \fi
  \fi
```

Exchange the pivot and the first element.

```
  \forest@sort@xch{#1}{\the\forest@sort@p}%
```

Counter `\forest@sort@m` will hold the final location of the pivot element.

```
  \forest@sort@m=#1\relax
```

Loop through the list.

```
  \forest@sort@k=#1\relax
  \forest@sort@loop
  \ifnum\forest@sort@k<#2\relax
    \advance\forest@sort@k 1
```

Compare the pivot and the current element.

```
    \forest@sort@cmp{#1}{\the\forest@sort@k}%
```

If the current element is smaller (ascending) or greater (descending) than the pivot element, move it into the first part of the list, and adjust the final location of the pivot.

```
    \ifx\forest@sort@direction\forest@sort@cmp@result
      \advance\forest@sort@m 1
      \forest@sort@xch{\the\forest@sort@m}{\the\forest@sort@k}
    \fi
  \forest@sort@repeat
```

Move the pivot element into its final position.

```
  \forest@sort@xch{#1}{\the\forest@sort@m}%
```

Recursively call sort on the two parts of the list: elements before the pivot are smaller (ascending order) / greater (descending order) than the pivot; elements after the pivot are greater (ascending order) / smaller (descending order) than the pivot.

```
  \forest@sort@k=\forest@sort@m
  \advance\forest@sort@k -1
  \advance\forest@sort@m 1
  \edef\forest@sort@marshal{%
    \noexpand\forest@@sort{#1}{\the\forest@sort@k}%
    \noexpand\forest@@sort{\the\forest@sort@m}{#2}%
  }%
  \forest@sort@marshal
}
% We defines the item-exchange macro in terms of the (user-provided)
% array let macro.
%    \begin{macrocode}
\def\forest@sort@aux{aux}
\def\forest@sort@xch#1#2{%
  \forest@sort@let{\forest@sort@aux}{#1}%
  \forest@sort@let{#1}{#2}%
  \forest@sort@let{#2}{\forest@sort@aux}%
}
```

Insertion sort.

```
\def\forest@insertionsort#1#2{%
  \forest@sort@m=#1
  \edef\forest@insertionsort@low{#1}%
  \forest@sort@loopA
  \ifnum\forest@sort@m<#2
    \advance\forest@sort@m 1
    \forest@insertionsort@Qbody
  \forest@sort@repeatA
}
```

```
\newif\ifforest@insertionsort@loop
\def\forest@insertionsort@Qbody{%
  \forest@sort@let{\forest@sort@aux}{\the\forest@sort@m}%
  \forest@sort@k\forest@sort@m
  \advance\forest@sort@k -1
  \forest@insertionsort@looptrue
  \forest@sort@loop
  \ifforest@insertionsort@loop
    \forest@insertionsort@qbody
  \forest@sort@repeat
  \advance\forest@sort@k 1
  \forest@sort@let{\the\forest@sort@k}{\forest@sort@aux}%
}
\def\forest@insertionsort@qbody{%
  \forest@sort@cmp{\the\forest@sort@k}{\forest@sort@aux}%
  \ifx\forest@sort@direction\forest@sort@cmp@result\relax
    \forest@sort@p=\forest@sort@k
    \advance\forest@sort@p 1
    \forest@sort@let{\the\forest@sort@p}{\the\forest@sort@k}%
    \advance\forest@sort@k -1
    \ifnum\forest@sort@k<\forest@insertionsort@low\relax
      \forest@insertionsort@loopfalse
    \fi
  \else
    \forest@insertionsort@loopfalse
  \fi
}
```

Below, several helpers for writing comparison macros are provided. They take take two (pairs of) control sequence names and compare their contents.

Compare numbers.

```
\def\forest@sort@cmpnumcs#1#2{%
  \ifnum\csname#1\endcsname>\csname#2\endcsname\relax
   \forest@sort@cmp@gt
 \else
   \ifnum\csname#1\endcsname<\csname#2\endcsname\relax
     \forest@sort@cmp@lt
   \else
     \forest@sort@cmp@eq
   \fi
 \fi
}
```

Compare dimensions.

```
\def\forest@sort@cmpdimcs#1#2{%
  \ifdim\csname#1\endcsname>\csname#2\endcsname\relax
   \forest@sort@cmp@gt
 \else
   \ifdim\csname#1\endcsname<\csname#2\endcsname\relax
     \forest@sort@cmp@lt
   \else
     \forest@sort@cmp@eq
   \fi
 \fi
}
```

Compare points (pairs of dimension) `(#1,#2)` and `(#3,#4)`.

```
\def\forest@sort@cmptwodimcs#1#2#3#4{%
  \ifdim\csname#1\endcsname>\csname#3\endcsname\relax
   \forest@sort@cmp@gt
 \else
   \ifdim\csname#1\endcsname<\csname#3\endcsname\relax
```

```
      \forest@sort@cmp@lt
    \else
      \ifdim\csname#2\endcsname>\csname#4\endcsname\relax
        \forest@sort@cmp@gt
      \else
        \ifdim\csname#2\endcsname<\csname#4\endcsname\relax
          \forest@sort@cmp@lt
        \else
          \forest@sort@cmp@eq
        \fi
      \fi
    \fi
  \fi
}
```

The following macro reverses an array. The arguments: `#1` is the array let macro; `#2` is the start index (inclusive), and `#3` is the end index (exclusive).

```
\def\forest@reversearray#1#2#3{%
  \let\forest@sort@let#1%
  \c@pgf@countc=#2
  \c@pgf@countd=#3
  \advance\c@pgf@countd -1
  \safeloop
  \ifnum\c@pgf@countc<\c@pgf@countd\relax
    \forest@sort@xch{\the\c@pgf@countc}{\the\c@pgf@countd}%
    \advance\c@pgf@countc 1
    \advance\c@pgf@countd -1
  \saferepeat
}
```

5 The bracket representation parser

5.1 The user interface macros

Settings.

```
\def\bracketset#1{\pgfqkeys{/bracket}{#1}}%
\bracketset{%
  /bracket/.is family,
  /handlers/.let/.style={\pgfkeyscurrentpath/.code={\let#1##1}},
  opening bracket/.let=\bracket@openingBracket,
  closing bracket/.let=\bracket@closingBracket,
  action character/.let=\bracket@actionCharacter,
  opening bracket=[,
  closing bracket=],
  action character,
  new node/.code n args={3}{% #1=preamble, #2=node spec, #3=cs receiving the id
    \forest@node@new#3%
    \forestOeset{#3}{given options}{\forest@contentto=\unexpanded{#2}}%
    \ifblank{#1}{}{%
      \forestrset{preamble}{#1}%
    }%
  },
  set afterthought/.code 2 args={% #1=node id, #2=afterthought
    \ifblank{#2}{}{\forestOappto{#1}{given options}{,afterthought={#2}}}%
  }
}
```

`\bracketParse` is the macro that should be called to parse a balanced bracket representation. It takes two parameters: `#1` is the code that will be run after parsing the bracket; `#2` is a control sequence that will receive the id of the root of the created tree structure. (The bracket representation should follow (after optional spaces), but is is not a formal parameter of the macro.)

```
\newtoks\bracket@content
\newtoks\bracket@afterthought
\def\bracketParse#1#2={%
  \def\bracketEndParsingHook{#1}%
  \def\bracket@saveRootNodeTo{#2}%
```

Content and afterthought will be appended to these macros. (The `\bracket@afterthought` toks register is abused for storing the preamble as well — that's ok, the preamble comes before any afterhoughts.)

```
  \bracket@content={}%
  \bracket@afterthought={}%
```

The parser can be in three states: in content (0), in afterthought (1), or starting (2). While in the content/afterthought state, the parser appends all non-control tokens to the content/afterthought macro.

```
  \let\bracket@state\bracket@state@starting
  \bracket@ignorespacestrue
```

By default, don't expand anything.

```
  \bracket@expandtokensfalse
```

We initialize several control sequences that are used to store some nodes while parsing.

```
  \def\bracket@parentNode{0}%
  \def\bracket@rootNode{0}%
  \def\bracket@newNode{0}%
  \def\bracket@afterthoughtNode{0}%
```

Finally, we start the parser.

```
  \bracket@Parse
}
```

The other macro that an end user (actually a power user) can use, is actually just a synonym for `\bracket@Parse`. It should be used to resume parsing when the action code has finished its work.

```
\def\bracketResume{\bracket@Parse}%
```

5.2 Parsing

We first check if the next token is a space. Spaces need special treatment because they are eaten by both the `\romannumeral` trick and TEXs (undelimited) argument parsing algorithm. If a space is found, remember that, eat it up, and restart the parsing.

```
\def\bracket@Parse{%
  \futurelet\bracket@next@token\bracket@Parse@checkForSpace
}
\def\bracket@Parse@checkForSpace{%
  \expandafter\ifx\space\bracket@next@token\@escapeif{%
    \ifbracket@ignorespaces\else
      \bracket@haveSpacetrue
    \fi
    \expandafter\bracket@Parse\romannumeral-`0%
  }\else\@escapeif{%
    \bracket@Parse@maybeexpand
  }\fi
}
```

We either fully expand the next token (using a popular TEXnical trick ...) or don't expand it at all, depending on the state of `\ifbracket@expandtokens`.

```
\newif\ifbracket@expandtokens
\def\bracket@Parse@maybeexpand{%
  \ifbracket@expandtokens\@escapeif{%
    \expandafter\bracket@Parse@peekAhead\romannumeral-`0%
  }\else\@escapeif{%
    \bracket@Parse@peekAhead
  }\fi
}
```

We then look ahead to see what's coming.

```
\def\bracket@Parse@peekAhead{%
  \futurelet\bracket@next@token\bracket@Parse@checkForTeXGroup
}
```

If the next token is a begin-group token, we append the whole group to the content or afterthought macro, depending on the state.

```
\def\bracket@Parse@checkForTeXGroup{%
  \ifx\bracket@next@token\bgroup%
    \@escapeif{\bracket@Parse@appendGroup}%
  \else
    \@escapeif{\bracket@Parse@token}%
  \fi
}
```

This is easy: if a control token is found, run the appropriate macro; otherwise, append the token to the content or afterthought macro, depending on the state.

```
\long\def\bracket@Parse@token#1{%
  \ifx#1\bracket@openingBracket
    \@escapeif{\bracket@Parse@openingBracketFound}%
  \else
    \@escapeif{%
      \ifx#1\bracket@closingBracket
        \@escapeif{\bracket@Parse@closingBracketFound}%
      \else
        \@escapeif{%
          \ifx#1\bracket@actionCharacter
            \@escapeif{\futurelet\bracket@next@token\bracket@Parse@actionCharacterFound}%
          \else
            \@escapeif{\bracket@Parse@appendToken#1}%
          \fi
        }%
      \fi
    }%
  \fi
}
```

Append the token or group to the content or afterthought macro. If a space was found previously, append it as well.

```
\newif\ifbracket@haveSpace
\newif\ifbracket@ignorespaces
\def\bracket@Parse@appendSpace{%
  \ifbracket@haveSpace
    \ifcase\bracket@state\relax
      \eapptotoks\bracket@content\space
    \or
      \eapptotoks\bracket@afterthought\space
    \or
      \eapptotoks\bracket@afterthought\space
    \fi
    \bracket@haveSpacefalse
  \fi
}
\long\def\bracket@Parse@appendToken#1{%
  \bracket@Parse@appendSpace
  \ifcase\bracket@state\relax
    \lapptotoks\bracket@content{#1}%
  \or
    \lapptotoks\bracket@afterthought{#1}%
  \or
    \lapptotoks\bracket@afterthought{#1}%
  \fi
  \bracket@ignorespacesfalse
  \bracket@Parse
}
\def\bracket@Parse@appendGroup#1{%
  \bracket@Parse@appendSpace
  \ifcase\bracket@state\relax
    \apptotoks\bracket@content{{#1}}%
  \or
    \apptotoks\bracket@afterthought{{#1}}%
  \or
    \apptotoks\bracket@afterthought{{#1}}%
  \fi
  \bracket@ignorespacesfalse
  \bracket@Parse
}
```

Declare states.

```
\def\bracket@state@inContent{0}
\def\bracket@state@inAfterthought{1}
\def\bracket@state@starting{2}
```

Welcome to the jungle. In the following two macros, new nodes are created, content and afterthought are sent to them, parents and states are changed… Altogether, we distinguish six cases, as shown below: in the schemas, we have just crossed the symbol after the dots. (In all cases, we reset the `\if` for spaces.)

```
\def\bracket@Parse@openingBracketFound{%
  \bracket@haveSpacefalse
  \ifcase\bracket@state\relax% in content [ ... [
```

`[...[`: we have just finished gathering the content and are about to begin gathering the content of another node. We create a new node (and put the content (…) into it). Then, if there is a parent node, we append the new node to the list of its children. Next, since we have just crossed an opening bracket, we declare the newly created node to be the parent of the coming node. The state does not change. Finally, we continue parsing.

```
    \@escapeif{%
      \bracket@createNode
      \ifnum\bracket@parentNode=0 \else
        \forest@node@Append{\bracket@parentNode}{\bracket@newNode}%
      \fi
      \let\bracket@parentNode\bracket@newNode
      \bracket@Parse
    }%
  \or % in afterthought   ] ... [
```

`]...[`: we have just finished gathering the afterthought and are about to begin gathering the content of another node. We add the afterthought (…) to the “afterthought node” and change into the content state. The parent does not change. Finally, we continue parsing.

```
    \@escapeif{%
      \bracket@addAfterthought
      \let\bracket@state\bracket@state@inContent
      \bracket@Parse
    }%
  \else % starting
```

`{start}...[`: we have just started. Nothing to do yet (we couldn’t have collected any content yet), just get into the content state and continue parsing.

```
    \@escapeif{%
      \let\bracket@state\bracket@state@inContent
      \bracket@Parse
    }%
```

```
  \fi
}
\def\bracket@Parse@closingBracketFound{%
  \bracket@haveSpacefalse
  \ifcase\bracket@state\relax % in content [ ... ]
```

`[...]`: we have just finished gathering the content of a node and are about to begin gathering its afterthought. We create a new node (and put the content (. . .) into it). If there is no parent node, we're done with parsing. Otherwise, we set the newly created node to be the "afterthought node", i.e. the node that will receive the next afterthought, change into the afterthought mode, and continue parsing.

```
    \@escapeif{%
      \bracket@createNode
      \ifnum\bracket@parentNode=0
        \@escapeif\bracketEndParsingHook
      \else
        \@escapeif{%
          \let\bracket@afterthoughtNode\bracket@newNode
          \let\bracket@state\bracket@state@inAfterthought
          \forest@node@Append{\bracket@parentNode}{\bracket@newNode}%
          \bracket@Parse
        }%
      \fi
    }%
  \or % in afterthought ] ... ]
```

`]...]`: we have finished gathering an afterthought of some node and will begin gathering the afterthought of its parent. We first add the afterthought to the afterthought node and set the current parent to be the next afterthought node. We change the parent to the current parent's parent and check if that node is null. If it is, we're done with parsing (ignore the trailing spaces), otherwise we continue.

```
    \@escapeif{%
      \bracket@addAfterthought
      \let\bracket@afterthoughtNode\bracket@parentNode
      \edef\bracket@parentNode{\forestOve{\bracket@parentNode}{@parent}}%
      \ifnum\bracket@parentNode=0
        \expandafter\bracketEndParsingHook
      \else
        \expandafter\bracket@Parse
      \fi
    }%
  \else % starting
```

`{start}...]`: something's obviously wrong with the input here. . .

```
    \PackageError{forest}{You're attempting to start a bracket representation
      with a closing bracket}{}%
  \fi
}
```

The action character code. What happens is determined by the next token.

```
\def\bracket@Parse@actionCharacterFound{%
```

If a braced expression follows, its contents will be fully expanded.

```
  \ifx\bracket@next@token\bgroup\@escapeif{%
    \bracket@Parse@action@expandgroup
  }\else\@escapeif{%
    \bracket@Parse@action@notagroup
  }\fi
}
\def\bracket@Parse@action@expandgroup#1{%
  \edef\bracket@Parse@action@expandgroup@macro{#1}%
  \expandafter\bracket@Parse\bracket@Parse@action@expandgroup@macro
}
\let\bracket@action@fullyexpandCharacter+
\let\bracket@action@dontexpandCharacter-
\let\bracket@action@executeCharacter!
\def\bracket@Parse@action@notagroup#1{%
```

If + follows, tokens will be fully expanded from this point on.

```
  \ifx#1\bracket@action@fullyexpandCharacter\@escapeif{%
    \bracket@expandtokenstrue\bracket@Parse
  }\else\@escapeif{%
```

If - follows, tokens will not be expanded from this point on. (This is the default behaviour.)

```
    \ifx#1\bracket@action@dontexpandCharacter\@escapeif{%
      \bracket@expandtokensfalse\bracket@Parse
    }\else\@escapeif{%
```

Inhibit expansion of the next token.

```
      \ifx#10\@escapeif{%
        \bracket@Parse@appendToken
      }\else\@escapeif{%
```

If another action characted follows, we yield the control. The user is expected to resume the parser manually, using `\bracketResume`.

```
        \ifx#1\bracket@actionCharacter
        \else\@escapeif{%
```

Anything else will be expanded once.

```
          \expandafter\bracket@Parse#1%
        }\fi
      }\fi
    }\fi
  }\fi
}
```

5.3 The tree-structure interface

This macro creates a new node and sets its content (and preamble, if it's a root node). Bracket user must define a 3-arg key `/bracket/new node=`⟨*preamble*⟩⟨*node specification*⟩⟨*node cs*⟩. User's key must define ⟨*node cs*⟩ to be a macro holding the node's id.

```
\def\bracket@createNode{%
  \ifnum\bracket@rootNode=0
    % root node
    \bracketset{new node/.expanded=%
      {\the\bracket@afterthought}%
      {\the\bracket@content}%
      \noexpand\bracket@newNode
    }%
    \bracket@afterthought={}%
    \let\bracket@rootNode\bracket@newNode
    \expandafter\let\bracket@saveRootNodeTo\bracket@newNode
  \else
    % other nodes
    \bracketset{new node/.expanded=%
      {}%
      {\the\bracket@content}%
      \noexpand\bracket@newNode
    }%
  \fi
  \bracket@content={}%
}
```

This macro sets the afterthought. Bracket user must define a 2-arg key `/bracket/set_afterthought=`⟨*node id*⟩⟨*afterthought*⟩.

```
\def\bracket@addAfterthought{%
```

```
  \bracketset{%
    set afterthought/.expanded={\bracket@afterthoughtNode}{\the\bracket@afterthought}%
  }%
  \bracket@afterthought={}%
}
```

6 Nodes

Nodes have numeric ids. The node option values of node n are saved in the `\pgfkeys` tree in path `/forest/@node/`n.

6.1 Option setting and retrieval

Macros for retrieving/setting node options of the current node.

```
 % full expansion expands precisely to the value
\def\forestov#1{\expandafter\expandonce\csname fRsT\forest@cn/#1\endcsname}
 % full expansion expands all the way
\def\forestove#1{\csname fRsT\forest@cn/#1\endcsname}
 % full expansion expands to the cs holding the value
\def\forestom#1{\expandonce{\csname fRsT\forest@cn/#1\endcsname}}
\def\forestoget#1#2{\expandafter\let\expandafter#2\csname fRsT\forest@cn/#1\endcsname}
\def\forestolet#1#2{\expandafter\let\csname fRsT\forest@cn/#1\endcsname#2}
% \def\forestocslet#1#2{%
%   \edef\forest@marshal{%
%     \noexpand\pgfkeyslet{/forest/@node/\forest@cn/#1}{\expandonce{\csname#2\endcsname}}%
%   }\forest@marshal
% }
\def\forestoset#1#2{\expandafter\edef\csname fRsT\forest@cn/#1\endcsname{\unexpanded{#2}}}
\def\forestoeset#1%#2
  {\expandafter\edef\csname fRsT\forest@cn/#1\endcsname
    %{#2}
  }
\def\forestoappto#1#2{%
  \forestoeset{#1}{\forestov{#1}\unexpanded{#2}}%
}
\def\forestoifdefined#1%#2#3
{%
  \ifcsdef{fRsT\forest@cn/#1}%{#2}{#3}%
}
```

User macros for retrieving node options of the current node.

```
\let\forestoption\forestov
\let\foresteoption\forestove
```

Macros for retrieving node options of a node given by its id.

```
\def\forestOv#1#2{\expandafter\expandonce\csname fRsT#1/#2\endcsname}
\def\forestOve#1#2{\csname fRsT#1/#2\endcsname}
 % full expansion expands to the cs holding the value
\def\forestOm#1#2{\expandonce{\csname fRsT#1/#2\endcsname}}
\def\forestOget#1#2#3{\expandafter\let\expandafter#3\csname fRsT#1/#2\endcsname}
\def\forestOlet#1#2#3{\expandafter\let\csname fRsT#1/#2\endcsname#3}
% \def\forestOcslet#1#2#3{%
%   \edef\forest@marshal{%
%     \noexpand\pgfkeyslet{/forest/@node/#1/#2}{\expandonce{\csname#3\endcsname}}%
%   }\forest@marshal
% }
\def\forestOset#1#2#3{\expandafter\edef\csname fRsT#1/#2\endcsname{\unexpanded{#3}}}
\def\forestOeset#1#2%#3
{\expandafter\edef\csname fRsT#1/#2\endcsname
  % {#3}
}
\def\forestOappto#1#2#3{%
  \forestOeset{#1}{#2}{\forestOv{#1}{#2}\unexpanded{#3}}%
}
\def\forestOeappto#1#2#3{%
  \forestOeset{#1}{#2}{\forestOv{#1}{#2}#3}%
}
\def\forestOpreto#1#2#3{%
  \forestOeset{#1}{#2}{\unexpanded{#3}\forestOv{#1}{#2}}%
}
\def\forestOepreto#1#2#3{%
  \forestOeset{#1}{#2}{#3\forestOv{#1}{#2}}%
}
\def\forestOifdefined#1#2%#3#4
{%
  \ifcsdef{fRsT#1/#2}%{#3}{#4}%
}
\def\forestOletO#1#2#3#4{% option #2 of node #1 <-- option #4 of node #3
  \forestOget{#3}{#4}\forestoption@temp
  \forestOlet{#1}{#2}\forestoption@temp}
\def\forestOleto#1#2#3{%
  \forestoget{#3}\forestoption@temp
  \forestOlet{#1}{#2}\forestoption@temp}
\def\forestoletO#1#2#3{%
  \forestOget{#2}{#3}\forestoption@temp
  \forestolet{#1}\forestoption@temp}
\def\forestoleto#1#2{%
  \forestoget{#2}\forestoption@temp
  \forestolet{#1}\forestoption@temp}
```

Macros for retrieving node options given by ⟨*relative node name*⟩.⟨*option*⟩.

```
\def\forestRNOget#1#2{% #1=rn!option, #2 = receiving cs
  \pgfutil@in@{.}{#1}%
  \ifpgfutil@in@
    \forestRNOget@rn#2#1\forest@END
  \else
    \forestoget{#1}#2%
  \fi
}
\def\forestRNOget@rn#1#2.#3\forest@END{%
  \forest@forthis{%
    \forest@nameandgo{#2}%
    \forestoget{#3}#1%
  }%
}
\def\forestRNO@getvalueandtype#1#2#3{% #1=rn.option, #2,#3 = receiving css
  \pgfutil@in@{.}{#1}%
  \ifpgfutil@in@
    \forestRNO@getvalueandtype@rn#2#3#1\forest@END
  \else
    \forestoget{#1}#2%
    \pgfkeysgetvalue{/forest/#1/@type}#3%
  \fi
}
\def\forestRNO@getvalueandtype@rn#1#2#3.#4\forest@END{%
  % #1,#2=receiving css, #3=relative node name, #4=option name
  \forest@forthis{%
    \forest@nameandgo{#3}%
    \forestoget{#4}#1%
  }%
  \pgfkeysgetvalue{/forest/#4/@type}#2%
}
```

Macros for retrieving/setting registers.

```
 % full expansion expands precisely to the value
\def\forestrv#1{\expandafter\expandonce\csname fRsT/#1\endcsname}
 % full expansion expands all the way
\def\forestrve#1{\csname fRsT/#1\endcsname}
 % full expansion expands to the cs holding the value
\def\forestrm#1{\expandonce{\csname fRsT/#1\endcsname}}
\def\forestrget#1#2{\expandafter\let\expandafter#2\csname fRsT/#1\endcsname}
\def\forestrlet#1#2{\expandafter\let\csname fRsT/#1\endcsname#2}
% \def\forestrcslet#1#2{%
%   \edef\forest@marshal{%
%     \noexpand\pgfkeyslet{/forest/@node/register/#1}{\expandonce{\csname#2\endcsname}}%
%   }\forest@marshal
% }
\def\forestrset#1#2{\expandafter\edef\csname fRsT/#1\endcsname{\unexpanded{#2}}}
\def\forestreset#1%#2
  {\expandafter\edef\csname fRsT/#1\endcsname}%{#2}
\def\forestrappto#1#2{%
  \forestreset{#1}{\forestrv{#1}\unexpanded{#2}}%
}
\def\forestrpreto#1#2{%
  \forestreset{#1}{\unexpanded{#2}\forestrv{#1}}%
}
\def\forestrifdefined#1%#2#3
{%
  \ifcsdef{fRsT/#1}%{#2}{#3}%
}
```

User macros for retrieving node options of the current node.

```
\def\forestregister#1{\forestrv{#1}}
\def\foresteregister#1{\forestrve{#1}}
```

Node initialization. Node option declarations append to `\forest@node@init`.

```
\def\forest@node@init{%
  \forestoset{@parent}{0}%
  \forestoset{@previous}{0}% previous sibling
  \forestoset{@next}{0}%     next sibling
  \forestoset{@first}{0}%  primary child
  \forestoset{@last}{0}%   last child
}
\def\forestoinit#1{%
  \pgfkeysgetvalue{/forest/#1}\forestoinit@temp
  \forestolet{#1}\forestoinit@temp
}
\newcount\forest@node@maxid
\def\forest@node@new#1{% #1 = cs receiving the new node id
  \advance\forest@node@maxid1
  \forest@fornode{\the\forest@node@maxid}{%
    \forest@node@init
    \forestoeset{id}{\forest@cn}%
    \forest@node@setname{node@\forest@cn}%
    \forest@initializefromstandardnode
    \edef#1{\forest@cn}%
  }%
}
\let\forestoinit@orig\forestoinit
\def\forest@node@copy#1#2{% #1=from node id, cs receiving the new node id
  \advance\forest@node@maxid1
  \def\forestoinit##1{\ifstrequal{##1}{name}{\forestoset{name}{node@\forest@cn}}{\forestolet0{##1}{#1}{##1}}}
  \forest@fornode{\the\forest@node@maxid}{%
```

```
    \forest@node@init
    \forestoeset{id}{\forest@cn}%
    \forest@node@setname{\forest@copy@name@template{\forestOve{#1}{name}}}%
    \edef#2{\forest@cn}%
  }%
  \let\forestoinit\forestoinit@orig
}
\forestset{
  copy name template/.code={\def\forest@copy@name@template##1{#1}},
  copy name template/.default={node@\the\forest@node@maxid},
  copy name template
}
\def\forest@tree@copy#1#2{% #1=from node id, #2=cs receiving the new node id
  \forest@node@copy{#1}\forest@node@copy@temp@id
  \forest@fornode{\forest@node@copy@temp@id}{%
    \expandafter\forest@tree@copy@\expandafter{\forest@node@copy@temp@id}{#1}%
    \edef#2{\forest@cn}%
  }%
}
\def\forest@tree@copy@#1#2{%
  \forest@node@Foreachchild{#2}{%
    \expandafter\forest@tree@copy\expandafter{\forest@cn}\forest@node@copy@temp@childid
    \forest@node@Append{#1}{\forest@node@copy@temp@childid}%
  }%
}
```

Macro `\forest@cn` holds the current node id (a number). Node 0 is a special “null” node which is used to signal the absence of a node.

```
\def\forest@cn{0}
\forest@node@init
```

6.2 Tree structure

Node insertion/removal.

For the lowercase variants, `\forest@cn` is the parent/removed node. For the uppercase variants, `#1` is the parent/removed node. For efficiency, the public macros all expand the arguments before calling the internal macros.

```
\def\forest@node@append#1{\expandtwonumberargs\forest@node@Append{\forest@cn}{#1}}
\def\forest@node@prepend#1{\expandtwonumberargs\forest@node@Insertafter{\forest@cn}{#1}{0}}
\def\forest@node@insertafter#1#2{%
  \expandthreenumberargs\forest@node@Insertafter{\forest@cn}{#1}{#2}}
\def\forest@node@insertbefore#1#2{%
  \expandthreenumberargs\forest@node@Insertafter{\forest@cn}{#1}{\forestOve{#2}{@previous}}%
}
\def\forest@node@remove{\expandnumberarg\forest@node@Remove{\forest@cn}}
\def\forest@node@Append#1#2{\expandtwonumberargs\forest@node@Append@{#1}{#2}}
\def\forest@node@Prepend#1#2{\expandtwonumberargs\forest@node@Insertafter{#1}{#2}{0}}
\def\forest@node@Insertafter#1#2#3{% #2 is inserted after #3
  \expandthreenumberargs\forest@node@Insertafter@{#1}{#2}{#3}%
}
\def\forest@node@Insertbefore#1#2#3{% #2 is inserted before #3
  \expandthreenumberargs\forest@node@Insertafter{#1}{#2}{\forestOve{#3}{@previous}}%
}
\def\forest@node@Remove#1{\expandnumberarg\forest@node@Remove@{#1}}
\def\forest@node@Insertafter@#1#2#3{%
  \ifnum\forestOve{#2}{@parent}=0
  \else
    \PackageError{forest}{Insertafter(#1,#2,#3):
      node #2 already has a parent (\forestOve{#2}{@parent})}{}%
  \fi
  \ifnum#3=0
  \else
    \ifnum#1=\forestOve{#3}{@parent}
    \else
      \PackageError{forest}{Insertafter(#1,#2,#3): node #1 is not the parent of the
            intended sibling #3 (with parent \forestOve{#3}{@parent})}{}%
    \fi
  \fi
  \forestOeset{#2}{@parent}{#1}%
  \forestOeset{#2}{@previous}{#3}%
  \ifnum#3=0
    \forestOget{#1}{@first}\forest@node@temp
    \forestOeset{#1}{@first}{#2}%
  \else
    \forestOget{#3}{@next}\forest@node@temp
    \forestOeset{#3}{@next}{#2}%
  \fi
  \forestOeset{#2}{@next}{\forest@node@temp}%
  \ifnum\forest@node@temp=0
    \forestOeset{#1}{@last}{#2}%
  \else
    \forestOeset{\forest@node@temp}{@previous}{#2}%
  \fi
}
\def\forest@node@Append@#1#2{%
  \ifnum\forestOve{#2}{@parent}=0
  \else
    \PackageError{forest}{Append(#1,#2):
      node #2 already has a parent (\forestOve{#2}{@parent})}{}%
  \fi
  \forestOeset{#2}{@parent}{#1}%
  \forestOget{#1}{@last}\forest@node@temp
  \forestOeset{#1}{@last}{#2}%
  \forestOeset{#2}{@previous}{\forest@node@temp}%
  \ifnum\forest@node@temp=0
    \forestOeset{#1}{@first}{#2}%
  \else
    \forestOeset{\forest@node@temp}{@next}{#2}%
  \fi
}
\def\forest@node@Remove@#1{%
  \forestOget{#1}{@parent}\forest@node@temp@parent
  \ifnum\forest@node@temp@parent=0
  \else
    \forestOget{#1}{@previous}\forest@node@temp@previous
    \forestOget{#1}{@next}\forest@node@temp@next
    \ifnum\forest@node@temp@previous=0
      \forestOeset{\forest@node@temp@parent}{@first}{\forest@node@temp@next}%
    \else
      \forestOeset{\forest@node@temp@previous}{@next}{\forest@node@temp@next}%
    \fi
    \ifnum\forest@node@temp@next=0
      \forestOeset{\forest@node@temp@parent}{@last}{\forest@node@temp@previous}%
    \else
      \forestOeset{\forest@node@temp@next}{@previous}{\forest@node@temp@previous}%
    \fi
    \forestOset{#1}{@parent}{0}%
    \forestOset{#1}{@previous}{0}%
    \forestOset{#1}{@next}{0}%
  \fi
}
```

Do some stuff and return to the current node.

```
\def\forest@forthis#1{%
  \edef\forest@node@marshal{\unexpanded{#1}\def\noexpand\forest@cn}%
  \expandafter\forest@node@marshal\expandafter{\forest@cn}%
}
\def\forest@fornode#1#2{%
  \edef\forest@node@marshal{\edef\noexpand\forest@cn{#1}\unexpanded{#2}\def\noexpand\forest@cn}%
  \expandafter\forest@node@marshal\expandafter{\forest@cn}%
}
```

Looping methods: children.

```
\def\forest@node@foreachchild#1{\forest@node@Foreachchild{\forest@cn}{#1}}
\def\forest@node@Foreachchild#1#2{%
  \forest@fornode{\forestOve{#1}{@first}}{\forest@node@@forselfandfollowingsiblings{#2}}%
}
\def\forest@node@@forselfandfollowingsiblings#1{%
  \ifnum\forest@cn=0
  \else
    \forest@forthis{#1}%
    \@escapeif{%
      \edef\forest@cn{\forestove{@next}}%
      \forest@node@@forselfandfollowingsiblings{#1}%
    }%
  \fi
}
\def\forest@node@@forselfandfollowingsiblings@reversed#1{%
  \ifnum\forest@cn=0
  \else
    \@escapeif{%
      \edef\forest@marshal{%
        \noexpand\def\noexpand\forest@cn{\forestove{@next}}%
        \noexpand\forest@node@@forselfandfollowingsiblings@reversed{\unexpanded{#1}}%
        \noexpand\forest@fornode{\forest@cn}{\unexpanded{#1}}%
      }\forest@marshal
    }%
  \fi
}
\def\forest@node@foreachchild@reversed#1{\forest@node@Foreachchild@reversed{\forest@cn}{#1}}
\def\forest@node@Foreachchild@reversed#1#2{%
  \forest@fornode{\forestOve{#1}{@last}}{\forest@node@@forselfandprecedingsiblings@reversed{#2}}%
}
\def\forest@node@@forselfandprecedingsiblings@reversed#1{%
  \ifnum\forest@cn=0
  \else
    \forest@forthis{#1}%
    \@escapeif{%
      \edef\forest@cn{\forestove{@previous}}%
      \forest@node@@forselfandprecedingsiblings@reversed{#1}%
    }%
  \fi
}
\def\forest@node@@forselfandprecedingsiblings#1{%
  \ifnum\forest@cn=0
  \else
    \@escapeif{%
      \edef\forest@marshal{%
        \noexpand\def\noexpand\forest@cn{\forestove{@previous}}%
        \noexpand\forest@node@@forselfandprecedingsiblings{\unexpanded{#1}}%
        \noexpand\forest@fornode{\forest@cn}{\unexpanded{#1}}%
      }\forest@marshal
    }%
  \fi
}
```

Looping methods: (sub)tree and descendants.

```
\def\forest@node@@foreach#1#2#3#4{%
  % #1 = do what
  % #2 = do that -1=before,1=after processing children
  % #3 & #4: normal or reversed order of children?
  %   #3 = @first/@last
  %   #4 = \forest@node@@forselfandfollowingsiblings / \forest@node@@forselfandprecedingsiblings@reversed
  \ifnum#2<0 \forest@forthis{#1}\fi
  \ifnum\forestove{#3}=0
  \else\@escapeif{%
    \forest@forthis{%
      \edef\forest@cn{\forestove{#3}}%
      #4{\forest@node@@foreach{#1}{#2}{#3}{#4}}%
    }%
  }\fi
  \ifnum#2>0 \forest@forthis{#1}\fi
}
\def\forest@node@foreach#1{%
  \forest@node@@foreach{#1}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}
\def\forest@node@Foreach#1#2{%
  \forest@fornode{#1}{\forest@node@@foreach{#2}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}}
\def\forest@node@foreach@reversed#1{%
  \forest@node@@foreach{#1}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
\def\forest@node@Foreach@reversed#1#2{%
  \forest@fornode{#1}{\forest@node@@foreach{#2}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed
\def\forest@node@foreach@childrenfirst#1{%
  \forest@node@@foreach{#1}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}
\def\forest@node@Foreach@childrenfirst#1#2{%
  \forest@fornode{#1}{\forest@node@@foreach{#2}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}}
\def\forest@node@foreach@childrenfirst@reversed#1{%
  \forest@node@@foreach{#1}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
\def\forest@node@Foreach@childrenfirst@reversed#1#2{%
  \forest@fornode{#1}{\forest@node@@foreach{#2}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}
\def\forest@node@foreachdescendant#1{%
  \forest@node@foreachchild{\forest@node@@foreach{#1}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}
\def\forest@node@Foreachdescendant#1#2{%
  \forest@node@Foreachchild{#1}{\forest@node@@foreach{#2}{-1}{@first}{\forest@node@@forselfandfollowingsiblin
\def\forest@node@foreachdescendant@reversed#1{%
  \forest@node@foreachchild@reversed{\forest@node@@foreach{#1}{-1}{@last}{\forest@node@@forselfandprecedingsi
\def\forest@node@Foreachdescendant@reversed#1#2{%
  \forest@node@Foreachchild@reversed{#1}{\forest@node@@foreach{#2}{-1}{@last}{\forest@node@@forselfandprecedi
\def\forest@node@foreachdescendant@childrenfirst#1{%
  \forest@node@foreachchild{\forest@node@@foreach{#1}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}}
\def\forest@node@Foreachdescendant@childrenfirst#1#2{%
  \forest@node@Foreachchild{#1}{\forest@node@@foreach{#2}{1}{@first}{\forest@node@@forselfandfollowingsibling
\def\forest@node@foreachdescendant@childrenfirst@reversed#1{%
  \forest@node@foreachchild@reversed{\forest@node@@foreach{#1}{1}{@last}{\forest@node@@forselfandprecedingsib
\def\forest@node@Foreachdescendant@childrenfirst@reversed#1#2{%
  \forest@node@Foreachchild@reversed{#1}{\forest@node@@foreach{#2}{1}{@last}{\forest@node@@forselfandprecedin
```

Looping methods: breadth-first.

```
\def\forest@node@foreach@breadthfirst#1#2{% #1 = max level, #2 = code
  \forest@node@Foreach@breadthfirst@{\forest@cn}{@first}{@next}{#1}{#2}}
\def\forest@node@foreach@breadthfirst@reversed#1#2{% #1 = max level, #2 = code
  \forest@node@Foreach@breadthfirst@{\forest@cn}{@last}{@previous}{#1}{#2}}
\def\forest@node@Foreach@breadthfirst#1#2#3{% #1 = node id, #2 = max level, #3 = code
  \forest@node@Foreach@breadthfirst@{#1}{@first}{@next}{#2}{#3}}
\def\forest@node@Foreach@breadthfirst@reversed#1#2#3{% #1 = node id, #2 = max level, #3 = code
  \forest@node@Foreach@breadthfirst@{#1}{@last}{@previous}{#2}{#3}}
```

```
\def\forest@node@Foreach@breadthfirst@#1#2#3#4#5{%
  % #1 = root node,
  % #2 = @first/@last, #3 = @next/@previous (must be in sync with #2),
  % #4 = max level (< 0 means infinite)
  % #5 = code to execute at each node
  \forest@node@Foreach@breadthfirst@processqueue{#1,}{#2}{#3}{#4}{#5}%
}
\def\forest@node@Foreach@breadthfirst@processqueue#1#2#3#4#5{%
  % #1 = queue,
  % #2 = @first/@last, #3 = @next/@previous (must be in sync with #2),
  % #4 = max level (< 0 means infinite)
  % #5 = code to execute at each node
  \ifstrempty{#1}{}{%
    \forest@node@Foreach@breadthfirst@processqueue@#1\forest@node@Foreach@breadthfirst@processqueue@
      {#2}{#3}{#4}{#5}%
  }%
}
\def\forest@node@Foreach@breadthfirst@processqueue@#1,#2\forest@node@Foreach@breadthfirst@processqueue@#3#4#5
  % #1 = first,
  % #2 = rest,
  % #3 = @first/@last, #4 = next/previous (must be in sync with #2),
  % #5 = max level (< 0 means infinite)
  % #6 = code to execute at each node
  \forest@fornode{#1}{%
    #6%
    \ifnum#5<0
      \forest@node@getlistofchildren\forest@temp{#3}{#4}%
    \else
      \ifnum\forestove{level}>#5\relax
        \def\forest@temp{}%
      \else
        \forest@node@getlistofchildren\forest@temp{#3}{#4}%
      \fi
    \fi
    \edef\forest@marshal{%
      \noexpand\forest@node@Foreach@breadthfirst@processqueue{\unexpanded{#2}\forest@temp}%
        {#3}{#4}{#5}{\unexpanded{#6}}%
    }\forest@marshal
  }%
}
\def\forest@node@getlistofchildren#1#2#3{% #1 = list cs, #2 = @first/@last, #3 = @next/@previous
  \forest@node@Getlistofchildren{\forest@cn}{#1}{#2}{#3}%
}
\def\forest@node@Getlistofchildren#1#2#3#4{% #1 = node, #2 = list cs, #3 = @first/@last, #4 = @next/@previous
  \def#2{}%
  \ifnum\forestove{#3}=0
  \else
    \eappto#2{\forestOve{#1}{#3},}%
    \@escapeif{%
      \edef\forest@marshal{%
        \noexpand\forest@node@Getlistofchildren@{\forestOve{#1}{#3}}\noexpand#2{#4}%
      }\forest@marshal
    }%
  \fi
}
\def\forest@node@Getlistofchildren@#1#2#3{% #1 = node, #2 = list cs, #3 = @next/@previous
  \ifnum\forestOve{#1}{#3}=0
  \else
    \eappto#2{\forestOve{#1}{#3},}%
    \@escapeif{%
      \edef\forest@marshal{%
```

```
        \noexpand\forest@node@Getlistofchildren@{\forestOve{#1}{#3}}\noexpand#2{#3}%
      }\forest@marshal
    }%
  \fi
}
```

Compute `n`, `n'`, `n children` and `level`.

```
\def\forest@node@Compute@numeric@ts@info@#1{%
  \forest@node@Foreach{#1}{\forest@node@@compute@numeric@ts@info}%
  \ifnum\forestOve{#1}{@parent}=0
  \else
    \forest@fornode{#1}{\forest@node@@compute@numeric@ts@info@nbar}%
    % hack: the parent of the node we called the update for gets +1 for n_children
    \edef\forest@node@temp{\forestOve{#1}{@parent}}%
    \forestOeset{\forest@node@temp}{n children}{%
      \number\numexpr\forestOve{\forest@node@temp}{n children}-1%
    }%
  \fi
  \forest@node@Foreachdescendant{#1}{\forest@node@@compute@numeric@ts@info@nbar}%
}
\def\forest@node@@compute@numeric@ts@info{%
  \forestoset{n children}{0}%
  %
  \edef\forest@node@temp{\forestove{@previous}}%
  \ifnum\forest@node@temp=0
    \forestoset{n}{1}%
  \else
    \forestoeset{n}{\number\numexpr\forestOve{\forest@node@temp}{n}+1}%
  \fi
  %
  \edef\forest@node@temp{\forestove{@parent}}%
  \ifnum\forest@node@temp=0
    \forestoset{n}{0}%
    \forestoset{n'}{0}%
    \forestoset{level}{0}%
  \else
    \forestOeset{\forest@node@temp}{n children}{%
      \number\numexpr\forestOve{\forest@node@temp}{n children}+1%
    }%
    \forestoeset{level}{%
      \number\numexpr\forestOve{\forest@node@temp}{level}+1%
    }%
  \fi
}
\def\forest@node@@compute@numeric@ts@info@nbar{%
  \forestoeset{n'}{\number\numexpr\forestOve{\forestove{@parent}}{n children}-\forestove{n}+1}%
}
\def\forest@node@compute@numeric@ts@info#1{%
  \expandnumberarg\forest@node@Compute@numeric@ts@info@{\forest@cn}%
}
\def\forest@node@Compute@numeric@ts@info#1{%
  \expandnumberarg\forest@node@Compute@numeric@ts@info@{#1}%
}
```

Tree structure queries.

```
\def\forest@node@rootid{%
  \expandnumberarg\forest@node@Rootid{\forest@cn}%
}
\def\forest@node@Rootid#1{% #1=node
  \ifnum\forestOve{#1}{@parent}=0
    #1%
  \else
```

```
    \@escapeif{\expandnumberarg\forest@node@Rootid{\forestOve{#1}{@parent}}}%
  \fi
}
\def\forest@node@nthchildid#1{% #1=n
  \ifnum#1<1
    0%
  \else
    \expandnumberarg\forest@node@nthchildid@{\number\forestove{@first}}{#1}%
  \fi
}
\def\forest@node@nthchildid@#1#2{%
  \ifnum#1=0
    0%
  \else
    \ifnum#2>1
      \@escapeifif{\expandtwonumberargs
        \forest@node@nthchildid@{\forestOve{#1}{@next}}{\numexpr#2-1}}%
    \else
      #1%
    \fi
  \fi
}
\def\forest@node@nbarthchildid#1{% #1=n
  \expandnumberarg\forest@node@nbarthchildid@{\number\forestove{@last}}{#1}%
}
\def\forest@node@nbarthchildid@#1#2{%
  \ifnum#1=0
    0%
  \else
    \ifnum#2>1
      \@escapeifif{\expandtwonumberargs
        \forest@node@nbarthchildid@{\forestOve{#1}{@previous}}{\numexpr#2-1}}%
    \else
      #1%
    \fi
  \fi
}
\def\forest@node@nornbarthchildid#1{%
  \ifnum#1>0
    \forest@node@nthchildid{#1}%
  \else
    \ifnum#1<0
      \forest@node@nbarthchildid{-#1}%
    \else
      \forest@node@nornbarthchildid@error
    \fi
  \fi
}
\def\forest@node@nornbarthchildid@error{%
  \PackageError{forest}{In \string\forest@node@nornbarthchildid, n should !=0}{}%
}
\def\forest@node@previousleafid{%
  \expandnumberarg\forest@node@Previousleafid{\forest@cn}%
}
\def\forest@node@Previousleafid#1{%
  \ifnum\forestOve{#1}{@previous}=0
    \@escapeif{\expandnumberarg\forest@node@previousleafid@Goup{#1}}%
  \else
    \expandnumberarg\forest@node@previousleafid@Godown{\forestOve{#1}{@previous}}%
  \fi
}
```

```
\def\forest@node@previousleafid@Goup#1{%
  \ifnum\forestOve{#1}{@parent}=0
    \PackageError{forest}{get previous leaf: this is the first leaf}{}%
  \else
    \@escapeif{\expandnumberarg\forest@node@Previousleafid{\forestOve{#1}{@parent}}}%
  \fi
}
\def\forest@node@previousleafid@Godown#1{%
  \ifnum\forestOve{#1}{@last}=0
    #1%
  \else
    \@escapeif{\expandnumberarg\forest@node@previousleafid@Godown{\forestOve{#1}{@last}}}%
  \fi
}
\def\forest@node@nextleafid{%
  \expandnumberarg\forest@node@Nextleafid{\forest@cn}%
}
\def\forest@node@Nextleafid#1{%
  \ifnum\forestOve{#1}{@next}=0
    \@escapeif{\expandnumberarg\forest@node@nextleafid@Goup{#1}}%
  \else
    \expandnumberarg\forest@node@nextleafid@Godown{\forestOve{#1}{@next}}%
  \fi
}
\def\forest@node@nextleafid@Goup#1{%
  \ifnum\forestOve{#1}{@parent}=0
    \PackageError{forest}{get next leaf: this is the last leaf}{}%
  \else
    \@escapeif{\expandnumberarg\forest@node@Nextleafid{\forestOve{#1}{@parent}}}%
  \fi
}
\def\forest@node@nextleafid@Godown#1{%
  \ifnum\forestOve{#1}{@first}=0
    #1%
  \else
    \@escapeif{\expandnumberarg\forest@node@nextleafid@Godown{\forestOve{#1}{@first}}}%
  \fi
}



\def\forest@node@linearnextid{%
  \ifnum\forestove{@first}=0
    \expandafter\forest@node@linearnextnotdescendantid
  \else
    \forestove{@first}%
  \fi
}
\def\forest@node@linearnextnotdescendantid{%
  \expandnumberarg\forest@node@Linearnextnotdescendantid{\forest@cn}%
}
\def\forest@node@Linearnextnotdescendantid#1{%
  \ifnum\forestOve{#1}{@next}=0
    \ifnum\forestOve{#1}{@parent}=0
      0%
      \else
    \@escapeifif{\expandnumberarg\forest@node@Linearnextnotdescendantid{\forestOve{#1}{@parent}}}%
    \fi
  \else
    \forestOve{#1}{@next}%
  \fi
}
\def\forest@node@linearpreviousid{%
  \ifnum\forestove{@previous}=0
    \forestove{@parent}%
  \else
    \forest@node@previousleafid
  \fi
}
```

Test if the current node is an ancestor the node given by its id in the first argument. The code graciously deals with circular trees. The second and third argument (not formally present) are the true and the false case code.

```

\def\forest@ifancestorof#1{% is the current node an ancestor of #1? Yes: #2, no: #3
  \begingroup
  \expandnumberarg\forest@ifancestorof@{\forestOve{#1}{@parent}}%
}
\def\forest@ifancestorof@#1{%
  \ifnum#1=0
    \def\forest@ifancestorof@next{\expandafter\endgroup\@secondoftwo}%
  \else
    \ifnum\forest@cn=#1
      \def\forest@ifancestorof@next{\expandafter\endgroup\@firstoftwo}%
    \else
      \ifcsdef{forest@circularity@used#1}{%
```

We have just detected circularity: the potential descendant is in fact an ancestor of itself. Our answer is “false”: the current node is not an ancestor of the potential descendant.

```
        \def\forest@ifancestorof@next{\expandafter\endgroup\@secondoftwo}%
      }{%
        \csdef{forest@circularity@used#1}{}%
        \def\forest@ifancestorof@next{\expandnumberarg\forest@ifancestorof@{\forestOve{#1}{@parent}}}%
      }%
    \fi
  \fi
  \forest@ifancestorof@next
}
```

A debug tool which prints out the hierarchy of all nodes.

```
\NewDocumentCommand\forestdebugtypeouttrees{o}{%
  \forestdebug@typeouttrees\forest@temp
  \typeout{%
    \forestdebugtypeouttreesprefix
    \IfValueTF{#1}{#1: }{}%
    \detokenize\expandafter{\forest@temp}%
    \forestdebugtypeouttreessuffix
  }%
}
\def\forestdebug@typeouttrees#1{% #1 = cs to store the result
  \begingroup
  \edef\forest@temp@message{}%
  \def\forestdebug@typeouttrees@n{0}%
```

Loop through all known ids. When finding a node that has not been visited yet (probably as a part of a previous tree), find its root and typeout the root’s tree.

```
  \loop
  \ifnum\forestdebug@typeouttrees@n<\forest@node@maxid
    \edef\forestdebug@typeouttrees@n{\number\numexpr\forestdebug@typeouttrees@n+1}%
    \ifcsdef{forestdebug@typeouttree@used@\forestdebug@typeouttrees@n}{}{%
      \forest@fornode{\forestdebug@typeouttrees@n}{%
```

After finding the root, we need to restore our notes about visited nodes.

```
        \begingroup
        \forestdebug@typeouttrees@findroot
        \expandafter\endgroup
        \expandafter\edef\expandafter\forest@cn\expandafter{\forest@cn}%
        \forestdebug@typeouttree@build
        \appto\forest@temp@message{ }%
      }%
    }%
  \repeat
  \expandafter\endgroup
  \expandafter\def\expandafter#1\expandafter{\forest@temp@message}%
}
\def\forestdebug@typeouttrees@findroot{%
  \let\forestdebug@typeouttrees@next\relax
  \edef\forestdebug@typeouttrees@parent{\forestOve{\forest@cn}{@parent}}%
  \ifnum\forestdebug@typeouttrees@parent=0
  \else
    \ifcsdef{forestdebug@typeouttree@used@\forest@cn}{}{%
      \csdef{forestdebug@typeouttree@used@\forest@cn}{}%
      \edef\forest@cn{\forestdebug@typeouttrees@parent}%
      \let\forestdebug@typeouttrees@next\forestdebug@typeouttrees@findroot
    }%
  \fi
  \forestdebug@typeouttrees@next
}
\def\forestdebug@typeouttree#1#2{% #1=root id, #2=cs to receive result
  \begingroup
  \edef\forest@temp@message{}%
  \forest@fornode{#1}{\forestdebug@typeouttree@build}%
  \expandafter\endgroup
  \expandafter\edef\expandafter#2\expandafter{\forest@temp@message}%
}
\NewDocumentCommand\forestdebugtypeouttree{d() O{\forest@cn}}{%
  \forestdebug@typeouttree{#2}\forest@temp
  \typeout{\IfValueTF{#1}{#1: }{}\forest@temp}%
}
```

Recurse through the tree. If a circularity is detected, mark it with * and stop recursion.

```
\def\forestdebug@typeouttree@build{%
  \eappto\forest@temp@message{[\forestdebugtypeouttreenodeinfo%]
    \ifcsdef{forestdebug@typeouttree@used@\forest@cn}{*}{}%
  }%
  \ifcsdef{forestdebug@typeouttree@used@\forest@cn}{}{%
    \csdef{forestdebug@typeouttree@used@\forest@cn}{}%
    \forest@node@foreachchild{\forestdebug@typeouttree@build}%
  }%
  \eappto\forest@temp@message{%[
    ]}%
}
\def\forestdebugtypeouttreenodeinfo{\forest@cn}
\def\forestdebugtypeouttreesprefix{}
\def\forestdebugtypeouttreessuffix{}
```

6.3 Node options

6.3.1 Option-declaration mechanism

Common code for declaring options.

```
\def\forest@declarehandler#1#2#3{%#1=handler for specific type,#2=option name,#3=default value
  \pgfkeyssetvalue{/forest/#2}{#3}%
  \appto\forest@node@init{\forestoinit{#2}}%
  \pgfkeyssetvalue{/forest/#2/node@or@reg}{\forest@cn}%
  \forest@convert@others@to@underscores{#2}\forest@pgfmathoptionname
  \edef\forest@marshal{%
    \noexpand#1{/forest/#2}{/forest}{#2}{\forest@pgfmathoptionname}%
  }\forest@marshal
}
\def\forest@def@with@pgfeov#1#2{% \pgfeov mustn't occur in the arg of the .code handler!!!
  \long\def#1##1\pgfeov{#2}%
}
```

Option-declaration handlers.

```
\def\forest@declaretoks@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
  \forest@declaretoks@handler@A{#1}{#2}{#3}{#4}{}%
}
\def\forest@declarekeylist@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
  \forest@declaretoks@handler@A{#1}{#2}{#3}{#4}{,}%
  \forest@copycommandkey{#1}{#1'}%
  \pgfkeyssetvalue{#1'/option@name}{#3}%
  \forest@copycommandkey{#1+}{#1}%
  \pgfkeysalso{#1-/.code={%
      \forest@fornode{\forest@setter@node}{%
        \forest@node@removekeysfromkeylist{##1}{#3}%
      }}}%
  \pgfkeyssetvalue{#1-/option@name}{#3}%
}
\def\forest@declaretoks@handler@A#1#2#3#4#5{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
  \pgfkeysalso{%
    #1/.code={\forestOset{\forest@setter@node}{#3}{##1}},
    #2/if #3/.code n args={3}{%
      \forestoget{#3}\forest@temp@option@value
      \edef\forest@temp@compared@value{\unexpanded{##1}}%
      \ifx\forest@temp@option@value\forest@temp@compared@value
        \pgfkeysalso{##2}%
      \else
        \pgfkeysalso{##3}%
      \fi
    },
    #2/if in #3/.code n args={3}{%
      \forestoget{#3}\forest@temp@option@value
      \edef\forest@temp@compared@value{\unexpanded{##1}}%
      \expandafter\expandafter\expandafter\pgfutil@in@\expandafter\expandafter\expandafter{\expandafter\fores
      \ifpgfutil@in@
        \pgfkeysalso{##2}%
      \else
        \pgfkeysalso{##3}%
      \fi
    },
    #2/where #3/.style n args={3}{for tree={#2/if #3={##1}{##2}{##3}}},
    #2/where in #3/.style n args={3}{for tree={#2/if in #3={##1}{##2}{##3}}}
  }%
  \ifstrempty{#5}{%
    \pgfkeysalso{%
      #1+/.code={\forestOappto{\forest@setter@node}{#3}{#5##1}},
      #2/+#3/.code={\forestOpreto{\forest@setter@node}{#3}{##1#5}},
    }%
  }{%
    \pgfkeysalso{%
      #1+/.code={%
        \forestOget{\forest@setter@node}{#3}\forest@temp
        \ifdefempty{\forest@temp}{%
          \forestOset{\forest@setter@node}{#3}{##1}%
        }{%
          \forestOappto{\forest@setter@node}{#3}{#5##1}%
        }%
      },
      #2/+#3/.code={%
        \forestOget{\forest@setter@node}{#3}\forest@temp
        \ifdefempty{\forest@temp}{%
          \forestOset{\forest@setter@node}{#3}{##1}%
        }{%
          \forestOpreto{\forest@setter@node}{#3}{##1#5}%
        }%
      }%
    }%
  }%
  \pgfkeyssetvalue{#1/option@name}{#3}%
  \pgfkeyssetvalue{#1+/option@name}{#3}%
  \pgfkeyssetvalue{#2/+#3/option@name}{#3}%
  \pgfkeyslet{#1/@type}\forestmathtype@generic % for .process & co
  \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@toks{##1}{#3}}%
}
\def\forest@declareautowrappedtoks@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
  \forest@declaretoks@handler{#1}{#2}{#3}{#4}%
  \forest@copycommandkey{#1}{#1’}%
  \pgfkeysalso{#1/.style={#1’/.wrap value={##1}}}%
  \pgfkeyssetvalue{#1’/option@name}{#3}%
  \forest@copycommandkey{#1+}{#1+’}%
  \pgfkeysalso{#1+/.style={#1+’/.wrap value={##1}}}%
  \pgfkeyssetvalue{#1+’/option@name}{#3}%
  \forest@copycommandkey{#2/+#3}{#2/+#3’}%
  \pgfkeysalso{#2/+#3/.style={#2/+#3’/.wrap value={##1}}}%
  \pgfkeyssetvalue{#2/+#3’/option@name}{#3}%
}
\def\forest@declarereadonlydimen@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
  % this is to have ‘pt‘ with the correct category code
  \pgfutil@tempdima=\pgfkeysvalueof{/forest/#3}\relax
  \edef\forest@marshal{%
    \noexpand\pgfkeyssetvalue{/forest/#3}{\the\pgfutil@tempdima}%
  }\forest@marshal
  \pgfkeysalso{%
    #2/if #3/.code n args={3}{%
      \forestoget{#3}\forest@temp@option@value
      \ifdim\forest@temp@option@value=##1\relax
        \pgfkeysalso{##2}%
      \else
        \pgfkeysalso{##3}%
      \fi
    },
    #2/if #3</.code n args={3}{%
      \forestoget{#3}\forest@temp@option@value
      \ifdim\forest@temp@option@value>##1\relax
        \pgfkeysalso{##3}%
      \else
        \pgfkeysalso{##2}%
      \fi
    },
    #2/if #3>/.code n args={3}{%
      \forestoget{#3}\forest@temp@option@value
      \ifdim\forest@temp@option@value<##1\relax
        \pgfkeysalso{##3}%
      \else
        \pgfkeysalso{##2}%
      \fi
    },
    #2/where #3/.style n args={3}{for tree={#2/if #3={##1}{##2}{##3}}},
    #2/where #3</.style n args={3}{for tree={#2/if #3<={##1}{##2}{##3}}},
    #2/where #3>/.style n args={3}{for tree={#2/if #3>={##1}{##2}{##3}}},
  }%
  \pgfkeyslet{#1/@type}\forestmathtype@dimen  % for .process & co
  \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@dimen{##1}{#3}}%
}
\def\forest@declaredimen@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
  \forest@declarereadonlydimen@handler{#1}{#2}{#3}{#4}%
  \pgfkeysalso{%
    #1/.code={%
      \forestmathsetlengthmacro\forest@temp{##1}%
      \forestOlet{\forest@setter@node}{#3}\forest@temp
    },
    #1+/.code={%
      \forestmathsetlengthmacro\forest@temp{##1}%
      \pgfutil@tempdima=\forestove{#3}
      \advance\pgfutil@tempdima\forest@temp\relax
      \forestOeset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
    },
    #1-/.code={%
      \forestmathsetlengthmacro\forest@temp{##1}%
      \pgfutil@tempdima=\forestove{#3}
      \advance\pgfutil@tempdima-\forest@temp\relax
      \forestOeset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
    },
    #1*/.style={%
      #1={#4()*(##1)}%
    },
    #1:/.style={%
      #1={#4()/(##1)}%
    },
    #1'/.code={%
      \pgfutil@tempdima=##1\relax
      \forestOeset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
    },
    #1'+/.code={%
      \pgfutil@tempdima=\forestove{#3}\relax
      \advance\pgfutil@tempdima##1\relax
      \forestOeset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
    },
    #1'-/.code={%
      \pgfutil@tempdima=\forestove{#3}\relax
      \advance\pgfutil@tempdima-##1\relax
      \forestOeset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
    },
    #1'*/.style={%
      \pgfutil@tempdima=\forestove{#3}\relax
      \multiply\pgfutil@tempdima##1\relax
      \forestOeset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
    },
    #1':/.style={%
      \pgfutil@tempdima=\forestove{#3}\relax
      \divide\pgfutil@tempdima##1\relax
      \forestOeset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
    },
  }%
  \pgfkeyssetvalue{#1/option@name}{#3}%
  \pgfkeyssetvalue{#1+/option@name}{#3}%
  \pgfkeyssetvalue{#1-/option@name}{#3}%
  \pgfkeyssetvalue{#1*/option@name}{#3}%
  \pgfkeyssetvalue{#1:/option@name}{#3}%
  \pgfkeyssetvalue{#1'/option@name}{#3}%
  \pgfkeyssetvalue{#1'+/option@name}{#3}%
  \pgfkeyssetvalue{#1'-/option@name}{#3}%
  \pgfkeyssetvalue{#1'*/option@name}{#3}%
  \pgfkeyssetvalue{#1':/option@name}{#3}%
}
\def\forest@declarereadonlycount@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
  \pgfkeysalso{
    #2/if #3/.code n args={3}{%
      \forestoget{#3}\forest@temp@option@value
      \ifnum\forest@temp@option@value=##1\relax
        \pgfkeysalso{##2}%
      \else
        \pgfkeysalso{##3}%
      \fi
    },
    #2/if #3</.code n args={3}{%
      \forestoget{#3}\forest@temp@option@value
      \ifnum\forest@temp@option@value>##1\relax
        \pgfkeysalso{##3}%
      \else
        \pgfkeysalso{##2}%
      \fi
    },
    #2/if #3>/.code n args={3}{%
      \forestoget{#3}\forest@temp@option@value
      \ifnum\forest@temp@option@value<##1\relax
        \pgfkeysalso{##3}%
      \else
        \pgfkeysalso{##2}%
      \fi
    },
    #2/where #3/.style n args={3}{for tree={#2/if #3={##1}{##2}{##3}}},
    #2/where #3</.style n args={3}{for tree={#2/if #3<={##1}{##2}{##3}}},
    #2/where #3>/.style n args={3}{for tree={#2/if #3>={##1}{##2}{##3}}},
  }%
  \pgfkeyslet{#1/@type}\forestmathtype@count  % for .process & co
  \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@count{##1}{#3}}%
}
\def\forest@declarecount@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
  \forest@declarereadonlycount@handler{#1}{#2}{#3}{#4}%
  \pgfkeysalso{
    #1/.code={%
      \forestmathtruncatemacro\forest@temp{##1}%
      \forestOlet{\forest@setter@node}{#3}\forest@temp
    },
    #1+/.code={%
      \forestmathtruncatemacro\forest@temp{##1}%
      \c@pgf@counta=\forestove{#3}\relax
      \advance\c@pgf@counta\forest@temp\relax
      \forestOeset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
    },
    #1-/.code={%
      \forestmathtruncatemacro\forest@temp{##1}%
      \c@pgf@counta=\forestove{#3}\relax
      \advance\c@pgf@counta-\forest@temp\relax
      \forestOeset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
    },
    #1*/.code={%
      \forestmathtruncatemacro\forest@temp{##1}%
      \c@pgf@counta=\forestove{#3}\relax
      \multiply\c@pgf@counta\forest@temp\relax
      \forestOeset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
    },
    #1:/.code={%
      \forestmathtruncatemacro\forest@temp{##1}%
      \c@pgf@counta=\forestove{#3}\relax
      \divide\c@pgf@counta\forest@temp\relax
      \forestOeset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
    },
    #1'/.code={%
      \c@pgf@counta=##1\relax
      \forestOeset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
    },
    #1'+/.code={%
      \c@pgf@counta=\forestove{#3}\relax
      \advance\c@pgf@counta##1\relax
      \forestOeset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
    },
    #1'-/.code={%
      \c@pgf@counta=\forestove{#3}\relax
      \advance\c@pgf@counta-##1\relax
      \forestOeset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
    },
    #1'*/.style={%
      \c@pgf@counta=\forestove{#3}\relax
      \multiply\c@pgf@counta##1\relax
      \forestOeset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
    },
    #1':/.style={%
      \c@pgf@counta=\forestove{#3}\relax
      \divide\c@pgf@counta##1\relax
      \forestOeset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
    },
  }%
  \pgfkeyssetvalue{#1/option@name}{#3}%
  \pgfkeyssetvalue{#1+/option@name}{#3}%
  \pgfkeyssetvalue{#1-/option@name}{#3}%
  \pgfkeyssetvalue{#1*/option@name}{#3}%
  \pgfkeyssetvalue{#1:/option@name}{#3}%
  \pgfkeyssetvalue{#1'/option@name}{#3}%
  \pgfkeyssetvalue{#1'+/option@name}{#3}%
  \pgfkeyssetvalue{#1'-/option@name}{#3}%
  \pgfkeyssetvalue{#1'*/option@name}{#3}%
  \pgfkeyssetvalue{#1':/option@name}{#3}%
}
```

Nothing else should be defined in this namespace.

```
\def\forest@declareboolean@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
  \pgfkeysalso{%
    #1/.code={%
      \forestmath@if{##1}{%
        \def\forest@temp{1}%
      }{%
        \def\forest@temp{0}%
      }%
      \forestOlet{\forest@setter@node}{#3}\forest@temp
    },
    #1/.default=1,
    #2/not #3/.code={\forestOset{\forest@setter@node}{#3}{0}},
    #2/if #3/.code 2 args={%
      \forestoget{#3}\forest@temp@option@value
      \ifnum\forest@temp@option@value=0
        \pgfkeysalso{##2}%
      \else
        \pgfkeysalso{##1}%
      \fi
    },
    #2/where #3/.style 2 args={for tree={#2/if #3={##1}{##2}}}
  }%
  \pgfkeyssetvalue{#1/option@name}{#3}%
  \pgfkeyslet{#1/@type}\forestmathtype@count  % for .process & co
  \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@count{##1}{#3}}%
}
\forestset{
  declare toks/.code 2 args={%
    \forest@declarehandler\forest@declaretoks@handler{#1}{#2}%
  },
  declare autowrapped toks/.code 2 args={%
    \forest@declarehandler\forest@declareautowrappedtoks@handler{#1}{#2}%
  },
  declare keylist/.code 2 args={%
    \forest@declarehandler\forest@declarekeylist@handler{#1}{#2}%
  },
  declare readonly dimen/.code 2 args={%
    \forestmathsetlengthmacro\forest@temp{#2}%
    \edef\forest@marshal{%
      \unexpanded{\forest@declarehandler\forest@declarereadonlydimen@handler{#1}}{\forest@temp}%
    }\forest@marshal
  },
  declare dimen/.code 2 args={%
    \forestmathsetlengthmacro\forest@temp{#2}%
    \edef\forest@marshal{%
      \unexpanded{\forest@declarehandler\forest@declaredimen@handler{#1}}{\forest@temp}%
    }\forest@marshal
  },
  declare readonly count/.code 2 args={%
    \forestmathtruncatemacro\forest@temp{#2}%
    \edef\forest@marshal{%
      \unexpanded{\forest@declarehandler\forest@declarereadonlycount@handler{#1}}{\forest@temp}%
    }\forest@marshal
  },
  declare count/.code 2 args={%
    \forestmathtruncatemacro\forest@temp{#2}%
    \edef\forest@marshal{%
      \unexpanded{\forest@declarehandler\forest@declarecount@handler{#1}}{\forest@temp}%
    }\forest@marshal
  },
  declare boolean/.code 2 args={%
    \forestmath@if{#2}{%
      \def\forest@temp{1}%
    }{%
      \def\forest@temp{0}%
    }%
    \edef\forest@marshal{%
      \unexpanded{\forest@declarehandler\forest@declareboolean@handler{#1}}{\forest@temp}%
    }\forest@marshal
  },
```

7 Handlers

```
  /handlers/.restore default value/.code={%
    \edef\forest@handlers@currentpath{\pgfkeyscurrentpath}%
    \pgfkeysgetvalue{\pgfkeyscurrentpath/option@name}\forest@currentoptionname
    \pgfkeysgetvalue{/forest/\forest@currentoptionname}\forest@temp
    \expandafter\pgfkeysalso\expandafter{\forest@handlers@currentpath/.expand once=\forest@temp}%
  },
  /handlers/.pgfmath/.code={%
    \pgfmathparse{#1}%
    \pgfkeysalso{\pgfkeyscurrentpath/.expand once=\pgfmathresult}%
  },
  /handlers/.wrap value/.code={%
    \edef\forest@handlers@wrap@currentpath{\pgfkeyscurrentpath}%
    \pgfkeysgetvalue{\forest@handlers@wrap@currentpath/option@name}\forest@currentoptionname
    \forestOget{\pgfkeysvalueof{/forest/\forest@currentoptionname/node@or@reg}}{\forest@currentoptionname}\fo
    \forest@def@with@pgfeov\forest@wrap@code{#1}%
    \expandafter\edef\expandafter\forest@wrapped@value\expandafter{\expandafter\expandonce\expandafter{\expan
    \pgfkeysalso{\forest@handlers@wrap@currentpath/.expand once=\forest@wrapped@value}%
  },
  /handlers/.option/.code={%
    \edef\forest@temp{\pgfkeyscurrentpath}%
    \expandafter\forest@handlers@option\expandafter{\forest@temp}{#1}%
  },
}
\def\forest@handlers@option#1#2{%#1=pgfkeyscurrentpath,#2=relative node name
  \forestRNOget{#2}\forest@temp
  \pgfkeysalso{#1/.expand once={\forest@temp}}%
}%
\forestset{
  /handlers/.register/.code={%
    \edef\forest@marshal{%
      \noexpand\pgfkeysalso{\pgfkeyscurrentpath={\forestregister{#1}}}%
    }\forest@marshal
  },
  /handlers/.wrap pgfmath arg/.code 2 args={%
    \forestmathparse{#2}\let\forest@wrap@arg@i\forestmathresult
    \edef\forest@wrap@args{{\expandonce\forest@wrap@arg@i}}%
    \def\forest@wrap@code##1{#1}%
    % here we don't call \forest@wrap@pgfmath@args@@@wrapandpasson, as compat-2.0.2-wrappgfmathargs changes t
    \expandafter\expandafter\expandafter\forest@temp@toks\expandafter\expandafter\expandafter{\expandafter\fo
    \expandafter\pgfkeysalso\expandafter{\expandafter\pgfkeyscurrentpath\expandafter=\expandafter{\the\forest
  },
  /handlers/.wrap 2 pgfmath args/.code n args={3}{%
    \forestmathparse{#2}\let\forest@wrap@arg@i\forestmathresult
    \forestmathparse{#3}\let\forest@wrap@arg@ii\forestmathresult
    \edef\forest@wrap@args{{\expandonce\forest@wrap@arg@i}{\expandonce\forest@wrap@arg@ii}}%
    \def\forest@wrap@code##1##2{#1}%
    \forest@wrap@pgfmath@args@@@wrapandpasson
  },
  /handlers/.wrap 3 pgfmath args/.code n args={4}{%
    \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{}{}{}{}{}{3}%
    \forest@wrap@n@pgfmath@do{#1}{3}},
  /handlers/.wrap 4 pgfmath args/.code n args={5}{%
    \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{}{}{}{}{4}%
    \forest@wrap@n@pgfmath@do{#1}{4}},
  /handlers/.wrap 5 pgfmath args/.code n args={6}{%
    \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{}{}{}{5}%
    \forest@wrap@n@pgfmath@do{#1}{5}},
  /handlers/.wrap 6 pgfmath args/.code n args={7}{%
    \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{#7}{}{}{6}%
    \forest@wrap@n@pgfmath@do{#1}{6}},
  /handlers/.wrap 7 pgfmath args/.code n args={8}{%
    \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{#7}{#8}{}{7}%
    \forest@wrap@n@pgfmath@do{#1}{7}},
  /handlers/.wrap 8 pgfmath args/.code n args={9}{%
    \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}{8}%
    \forest@wrap@n@pgfmath@do{#1}{8}},
}
\def\forest@wrap@n@pgfmath@args#1#2#3#4#5#6#7#8#9{%
  \forestmathparse{#1}\let\forest@wrap@arg@i\forestmathresult
  \ifnum#9>1 \forestmathparse{#2}\let\forest@wrap@arg@ii\forestmathresult\fi
  \ifnum#9>2 \forestmathparse{#3}\let\forest@wrap@arg@iii\forestmathresult\fi
  \ifnum#9>3 \forestmathparse{#4}\let\forest@wrap@arg@iv\forestmathresult\fi
  \ifnum#9>4 \forestmathparse{#5}\let\forest@wrap@arg@v\forestmathresult\fi
  \ifnum#9>5 \forestmathparse{#6}\let\forest@wrap@arg@vi\forestmathresult\fi
  \ifnum#9>6 \forestmathparse{#7}\let\forest@wrap@arg@vii\forestmathresult\fi
  \ifnum#9>7 \forestmathparse{#8}\let\forest@wrap@arg@viii\forestmathresult\fi
  \edef\forest@wrap@args{%
    {\expandonce\forest@wrap@arg@i}
    \ifnum#9>1 {\expandonce\forest@wrap@arg@ii}\fi
    \ifnum#9>2 {\expandonce\forest@wrap@arg@iii}\fi
    \ifnum#9>3 {\expandonce\forest@wrap@arg@iv}\fi
    \ifnum#9>4 {\expandonce\forest@wrap@arg@v}\fi
    \ifnum#9>5 {\expandonce\forest@wrap@arg@vi}\fi
    \ifnum#9>6 {\expandonce\forest@wrap@arg@vii}\fi
    \ifnum#9>7 {\expandonce\forest@wrap@arg@viii}\fi
  }%
}
\def\forest@wrap@n@pgfmath@do#1#2{%
  \ifcase#2\relax
  \or\def\forest@wrap@code##1{#1}%
  \or\def\forest@wrap@code##1##2{#1}%
  \or\def\forest@wrap@code##1##2##3{#1}%
  \or\def\forest@wrap@code##1##2##3##4{#1}%
  \or\def\forest@wrap@code##1##2##3##4##5{#1}%
  \or\def\forest@wrap@code##1##2##3##4##5##6{#1}%
  \or\def\forest@wrap@code##1##2##3##4##5##6##7{#1}%
  \or\def\forest@wrap@code##1##2##3##4##5##6##7##8{#1}%
  \fi
  \forest@wrap@pgfmath@args@@@wrapandpasson
}
```

The following macro is redefined by compat key `2.0.2-wrappgfmathargs`.

```
\def\forest@wrap@pgfmath@args@@@wrapandpasson{%
  \expandafter\expandafter\expandafter\forest@temp@toks
      \expandafter\expandafter\expandafter{%
        \expandafter\forest@wrap@code\forest@wrap@args}%
  \expandafter\pgfkeysalso\expandafter{%
    \expandafter\pgfkeyscurrentpath\expandafter=\expandafter{%
        \the\forest@temp@toks}}%
}
```

7.1 .process

```
\def\forest@process@catregime{} % filled by processor defs
\forest@newarray\forest@process@left@ % processed args
\forest@newarray\forest@process@right@ % unprocessed args
\forest@newarray\forest@process@saved@ % used by instructions |S| and |U|
\let\forest@process@savedtype\forestmathtype@none
\forest@newglobalarray\forest@process@result@
\newif\ifforest@process@returnarray@
```

Processing instruction need not (but may) be enclosed in braces.

```
\def\forest@process#1#2#{%  #1 = true/false (should we return an array?)
                          %  #2 = processing instructions (if non-empty),
                          %  (initial) args follow
  \ifblank{#2}{\forest@process@a{#1}}{\forest@process@a{#1}{#2}}%
}
\Inline\def\forest@process@a#1#2{%
  \begingroup
  \forest@process@left@clear
  \forest@process@right@clear
  \forest@process@saved@clear
  \let\forest@process@savedtype\forestmathtype@generic
  \csname forest@process@returnarray@#1\endcsname
  \def\forest@topextend@next{%
    \ExpandIfT{forestdebug}{%
      \edef\forest@process@debug@args{\unexpanded{#2}}%
      \forest@processor@debuginfo@template{Start "\unexpanded{#2}}%
    }%
    \forest@process@catregime
    \endlinechar=-1
    \scantokens{#2}%
    \forest@process@finish
  }%
  \forest@process@right@topextend
}
\pgfkeys{%
  /handlers/.process/.code={%
    \forest@process{true}#1\forest@eov
    \edef\forest@marshal{%
      \noexpand\pgfkeysalso{\noexpand\pgfkeyscurrentpath=\forest@process@result@values}%
    }\forest@marshal
  },
  /forest/copy command key={/handlers/.process}{/handlers/.process args},
}
\def\forest@process@finish{%
  \ifforest@process@returnarray@
    \forest@process@finish@array
  \else
    \forest@process@finish@single
  \fi
  \global\let\forest@process@result@type\forestmathresulttype
  \ifforestdebugprocess\forest@process@debug@end\fi
  \endgroup
}
\def\forest@process@finish@single{%
  \edef\forest@temp{forest@process@finish@single@%
    \the\numexpr\forest@process@left@N-\forest@process@left@M\relax
    \the\numexpr\forest@process@right@N-\forest@process@right@M\relax
  }%
  \ifcsname\forest@temp\endcsname
    \csname\forest@temp\endcsname
    \global\let\forest@process@result\forest@temp
  \else
    \forest@process@lengtherror
  \fi
}
\csdef{forest@process@finish@single@10}{\forest@process@left@toppop\forest@temp}
\csdef{forest@process@finish@single@01}{\forest@process@right@toppop\forest@temp}
\def\forest@process@finish@array{%
  \forest@process@result@clear
  \forest@temp@count\forest@process@left@M\relax
  \forest@loop
  \ifnum\forest@temp@count<\forest@process@left@N\relax
    \forest@process@left@get@{\the\forest@temp@count}\forest@temp
    \forest@process@result@letappend\forest@temp
    \advance\forest@temp@count1
  \forest@repeat
  \forest@temp@count\forest@process@right@M\relax
  \forest@loop
  \ifnum\forest@temp@count<\forest@process@right@N\relax
    \forest@process@right@get@{\the\forest@temp@count}\forest@temp
    \forest@process@result@letappend\forest@temp
    \advance\forest@temp@count1
  \forest@repeat
}
```

Debugging and error messages.

```
\ifforestdebug
  \let\forest@process@d\forest@process@b
  \def\forest@process@b#1\forest@eov{% save and print initial arguments
    \edef\forest@process@debug@args{\unexpanded{#1}}%
    \typeout{[forest .process] Start "\unexpanded{#1}"}%
    \forest@process@d#1\forest@eov
  }
\fi
\def\forest@process@debug@end{%
  \typeout{[forest .process] End "\expandonce{\forest@process@debug@args}" -> "\forest@process@left@values\fo
}
\def\forest@process@lengtherror{%
  \PackageError{forest}{%
    The ".process" expression was expected to evaluate to a single argument,
    but the result is \the\forest@process@result@N
    \space items long.}{}%
}
```

Define the definer of processors. First, deal with the catcode of the instruction char.

```
\def\forest@def@processor#1{%
  {%
    \def\forest@dp@double##1{%
      \gdef\forest@global@temp{\forest@def@processor@{#1}{##1}}%
    }%
    \let\\\forest@dp@double
    \catcode‘#1=13
    \scantokens{\\#1}%
  }%
  \forest@global@temp
}
\def\forest@def@processor@#1#2{%
  % #1 = instruction char (normal catcode), #2 = instruction char (active)
  % #3 = default n (optional numeric arg, which precedes any other args;
  %                 if the default is empty, this means no optional n)
  % #4 = args spec,
  % #5 = code
  \eappto\forest@process@catregime{%
    \unexpanded{\let#2}\expandonce{\csname forest@processor@#1\endcsname}%
    \unexpanded{\catcode‘#1=13 }%
  }%
  \def\forest@def@processor@inschar{#1}%
  \forest@def@processor@@
}
```

If `#1` is non-empty, the processor accepts the optional numeric argument: `#1` is the default.

```
\def\forest@def@processor@@#1{%
  \ifstrempty{#1}{%
    \forest@def@processor@@non
  }{%
    \def\forest@def@processor@@default@n{#1}%
    \forest@def@processor@@n
  }%
}
```

We need `\relax` below because the next instruction character might get expanded when assigning the optional numerical argument which is not there.

No optional n:

```
\def\forest@def@processor@@non#1#2{% #1=args spec, #2=code
  \csedef{forest@processor@\forest@def@processor@inschar}#1{%
    \relax %% we need this (see above)
    \unexpanded{#2}%
    \expandafter\forest@def@processor@debuginfo\expandafter{%
      \expandafter"\forest@def@processor@inschar"\ifstrempty{#1}{}{(#1)}}%
    \ignorespaces
  }%
}
```

Optional n: `*` after the given default means that the operation should be repeated n times.

```
\def\forest@def@processor@@n{%
  \@ifnextchar*%
    {\forest@temptrue\forest@def@processor@@n@}%
    {\forest@tempfalse\forest@def@processor@@n@@}%
}
\def\forest@def@processor@@n@*{\forest@def@processor@@n@@}
\def\forest@def@processor@@n@@#1#2{% #1=args spec, #2=code
  \csedef{forest@processor@\forest@def@processor@inschar}{%
    \relax  %% we need this (see above)
    \noexpand\forestprocess@get@n
      {\forest@def@processor@@default@n}%
      \expandonce{\csname forest@processor@\forest@def@processor@inschar @\endcsname}%
  }%
  \ifforest@temp
    \csedef{forest@processor@\forest@def@processor@inschar @}{%
      \noexpand\forest@repeat@n@times{\forest@process@n}{%
        \expandonce{\csname forest@processor@\forest@def@processor@inschar @rep\endcsname}%
      }%
    }%
  \fi
  \edef\forest@temp{%
    \forest@def@processor@inschar
    \ifforest@temp\else\noexpand\the\forest@process@n\fi
    "}%
  \csedef{forest@processor@\forest@def@processor@inschar @\ifforest@temp rep\fi}#1{%
    \unexpanded{#2}%
    \expandafter\forest@def@processor@debuginfo\expandafter{%
      \forest@temp
      \ifstrempty{#1}{}{(#1)}}%
  }%
}
\def\forest@def@processor@debuginfo#1{% #1 = instruction call
  \ifforestdebug
    \expandonce{\forest@processor@debuginfo@template{\space\space After #1}}%
  \fi
}
\def\forest@processor@debuginfo@template#1{%
  \ifforestdebugprocess
    \edef\forest@temp@left{\forest@process@left@values}%
    \edef\forest@temp@right{\forest@process@right@values}%
    \edef\forest@temp@saved{\forest@process@saved@values}%
    \typeout{[forest .process] #1: left="\expandonce{\forest@temp@left}", right="\expandonce{\forest@temp@rig
  \fi
}
```

A helper macro which puts the optional numeric argument into count `\forest@process@n` (default being `#1`) and then executes control sequence `#2`.

```
\newcount\forest@process@n
\def\forestprocess@get@n#1#2{%
  \def\forestprocess@default@n{#1}%
  \let\forestprocess@after@get@n@#2%
  \afterassignment\forestprocess@get@n@\forest@process@n=0%
}
\def\forestprocess@get@n@{%
  \ifnum\forest@process@n=0
    \forest@process@n\forestprocess@default@n\relax
  \fi
  \forestprocess@after@get@n@
}
```

Definitions of processing instructions. Processors should be defined using `\forest@def@processor`. If they take arguments: yes, they follow, but they were scanned in `\forest@process@catregime`. Processors should manipulate arrays `\forest@process@left@` and `\forest@process@right`. They should set `\def\forestmathresulttype` to `_` not defined, `n` number, `d` dimension, `P` pgfmath or `t` text.

```
\forest@def@processor{_}{1}*{}{% no processing, no type
  \forest@process@right@bottompop\forest@temp
  \forest@process@left@letappend\forest@temp
}
\forest@def@processor{n}{1}*{}{% numexpr
  \forest@process@right@bottompop\forest@temp
  \forest@process@left@esetappend{\number\numexpr\forest@temp}%
  \let\forestmathresulttype\forestmathtype@count
}
\forest@def@processor{d}{1}*{}{% dimexpr
  \forest@process@right@bottompop\forest@temp
  \forest@process@left@esetappend{\the\dimexpr\forest@temp}%
  \let\forestmathresulttype\forestmathtype@dimen
}
\forest@def@processor{P}{1}*{}{% pgfmath expression
  \forest@process@right@bottompop\forest@temp
  \pgfmathparse{\forest@temp}%
  \forest@process@left@letappend\pgfmathresult
  \let\forestmathresulttype\forestmathtype@unitless
}
\forest@def@processor{p}{1}*{}{% process expression
  \forest@process@right@bottompop\forest@temp@a
  \def\forest@temp{\forest@process{true}}%
  \expandafter\forest@temp\forest@temp@a\forest@eov
  \let\forest@topextend@next\relax
  \edef\forest@temp{\forest@process@result@values}%
  \expandafter\forest@process@left@topextend\forest@temp\forest@eov
  \let\forestmathresulttype\forest@process@result@type
}
\forest@def@processor{t}{1}*{}{% text
  \forest@process@right@bottompop\forest@temp
  \forest@process@left@letappend\forest@temp
  \let\forestmathresulttype\forestmathtype@textasc
}
\forest@def@processor{-}{}{}{% toggle ascending/descending
  \forest@process@left@toppop\forestmathresult
  \csname forest@processor@-@\forestmathresulttype\endcsname
  \forest@process@left@letappend\forestmathresult
}
\cslet{forest@processor@-@\forestmathtype@generic}\relax
```

```
\csdef{forest@processor@-@\forestmathtype@count}{%
  \forestmathadd{\forestmathzero}{-\forestmathresult}}
\csletcs{forest@processor@-@\forestmathtype@dimen}
        {forest@processor@-@\forestmathtype@count}
\csletcs{forest@processor@-@\forestmathtype@unitless}
        {forest@processor@-@\forestmathtype@count}
\csdef{forest@processor@-@\forestmathtype@textasc}{%
  \let\forestmathresulttype\forestmathtype@textdesc}
\csdef{forest@processor@-@\forestmathtype@textdesc}{%
  \let\forestmathresulttype\forestmathtype@textasc}

\forest@def@processor{c}{}{}{% to lowercase
  \forest@process@right@bottompop\forest@temp
  \expandafter\lowercase\expandafter{\expandafter\def\expandafter\forest@temp\expandafter{\forest@temp}}%
  \forest@process@left@letappend\forest@temp
}
\forest@def@processor{C}{}{}{% to uppercase
  \forest@process@right@bottompop\forest@temp
  \expandafter\uppercase\expandafter{\expandafter\def\expandafter\forest@temp\expandafter{\forest@temp}}%
  \forest@process@left@letappend\forest@temp
}
```

Expansions:

```
\forest@def@processor{x}{}{}{% expand
  \forest@process@right@bottompop\forest@temp
  \forest@process@left@esetappend{\forest@temp}%
  \let\forestmathresulttype\forestmathtype@generic
}
\forest@def@processor{o}{1}{}{% expand once (actually, \forest@count@n times)
  \forest@process@right@bottompop\forest@temp
  \forest@repeat@n@times{\forest@process@n}{%
    \expandafter\expandafter\expandafter\def
      \expandafter\expandafter\expandafter\forest@temp
      \expandafter\expandafter\expandafter{\forest@temp}%
  }%
  \expandafter\forest@process@left@setappend\expandafter{\forest@temp}%
  \let\forestmathresulttype\forestmathtype@generic
}
```

Access to Forest data.

```
\forest@def@processor{O}{1}*{}{% option
  \forest@process@right@bottompop\forest@temp
  \expandafter\forestRNO@getvalueandtype\expandafter{\forest@temp}\forest@tempvalue\forest@temp@type
  \let\forestmathresulttype\forest@temp@type
  \forest@process@left@letappend\forest@tempvalue
}
\forest@def@processor{R}{1}*{}{% register
  \forest@process@right@bottompop\forest@temp
  \forestrget{\forest@temp}\forest@tempvalue
  \forest@process@left@letappend\forest@tempvalue
  \pgfkeysgetvalue{/forest/\forest@temp/@type}\forest@temp@type
  \let\forestmathresulttype\forest@temp@type
}
```

The following processors muck about with the argument / result list.

```
\forest@def@processor{+}{1}*{}{% join processors = pop one from result
  \forest@process@left@toppop\forest@temp
  \forest@process@right@letprepend\forest@temp
}
\forest@def@processor{u}{}{}{% ungroup: remove braces and leave in the argument list
  \forest@process@right@bottompop\forest@temp
  \forest@temparray@clear
  \let\forestmathresulttype\forestmathtype@generic
  \let\forest@topextend@next\forest@processor@u@
  \expandafter\forest@temparray@topextend\forest@temp\forest@eov
}
\def\forest@processor@u@{%
  \forest@loop
  \ifnum\forest@temparray@N>0
    \forest@temparray@toppop\forest@temp
    \expandafter\forest@process@right@setprepend\expandafter{\forest@temp}%
  \forest@repeat
}
\def\forest@process@check@mn#1#2#3#4{%
  % #1 = processor, #2 = given n, #3/#4 = lower/upper bound (inclusive)
  \ifnum#3>#2\relax
    \forest@process@check@n@error{#1}{#2}{#3<=}{<=#4}%
  \else
    \ifnum#4<#2\relax
      \forest@process@check@n@error{#1}{#2}{#3<=}{<=#4}%
    \fi
  \fi
}
\def\forest@process@check@m#1#2#3{%
  % #1 = processor, #2 = given n, #3 = lower bound (inclusive)
  \ifnum#2<#3\relax
    \forest@process@check@n@error{#1}{#2}{#3<=}{}%
  \fi
}
\def\forest@process@check@n@error#1#2#3#4{%
  \PackageError{forest}{'.process' instruction '#1' requires a numeric modifier #3n#4, but n="#2" was given.}
}
\newif\ifforest@process@W
\forest@def@processor{w}{1}{}{% consuming wrap: first test 1<=#1<=9
  \forest@process@Wtrue
  \forest@process@check@mn{w}{0\the\forest@process@n}{1}{9}%
  \expandafter\forest@processor@wW@\expandafter{\the\forest@process@n}%
}
\forest@def@processor{W}{1}{}{% nonconsuming wrap: first test 1<=#1<=9
  \forest@process@Wfalse
  \forest@process@check@mn{W}{0\the\forest@process@n}{1}{9}%
  \expandafter\forest@processor@wW@\expandafter{\the\forest@process@n}%
}
\def\forest@processor@wW@#1{%
  \forest@process@left@checkindex{\forest@process@left@N-#1}%
  \edef\forest@marshal{%
    \edef\noexpand\forest@temp@args{%
      \noexpand\forest@process@left@valuesfromrange
        {\number\numexpr\forest@process@left@N-#1}%
        {\the\forest@process@left@N}%
    }%
  }\forest@marshal
  \ifforest@process@W
    \advance\forest@process@left@N-#1\relax
  \fi
  \forest@process@right@bottompop\forest@temp@macrobody
  \expandafter\forest@def@n\expandafter\forest@process@temp@macro\expandafter{\expandafter#1\expandafter}\exp
  \expandafter\expandafter\expandafter\forest@process@left@setappend\expandafter\expandafter\expandafter{\exp
  \let\forestmathresulttype\forestmathtype@generic
}
\def\forest@def@n#1#2{\csname forest@def@n@#2\endcsname#1}
\csdef{forest@def@n@1}#1{\def#1##1}
\csdef{forest@def@n@2}#1{\def#1##1##2}
\csdef{forest@def@n@3}#1{\def#1##1##2##3}
```

```
\csdef{forest@def@n@4}#1{\def#1##1##2##3##4}
\csdef{forest@def@n@5}#1{\def#1##1##2##3##4##5}
\csdef{forest@def@n@6}#1{\def#1##1##2##3##4##5##6}
\csdef{forest@def@n@7}#1{\def#1##1##2##3##4##5##6##7}
\csdef{forest@def@n@8}#1{\def#1##1##2##3##4##5##6##7##8}
\csdef{forest@def@n@9}#1{\def#1##1##2##3##4##5##6##7##8##9}
```

Save last `n` arguments from the left side into a special place. `s` deletes them from the left side, `S` keeps them there as well.

```
\forest@def@processor{s}{1}{}{%
  \forest@temptrue   % delete the originals
  \expandafter\forest@processor@save\expandafter{%
      \the\numexpr\forest@process@left@N-\forest@process@n}}
\forest@def@processor{S}{1}{}{%
  \forest@tempfalse  % keep the originals
  \expandafter\forest@processor@save\expandafter{%
      \the\numexpr\forest@process@left@N-\forest@process@n}}
\def\forest@processor@save#1{%
  \forest@process@left@checkindex{#1}%
  \forest@temp@count#1
  \forest@loop
  \ifnum\forest@temp@count<\forest@process@left@N\relax
    \forest@process@left@get@{\the\forest@temp@count}\forest@temp
    \forest@process@saved@letappend\forest@temp
    \advance\forest@temp@count+1
  \forest@repeat
  \let\forest@process@savedtype\forestmathresulttype
  \ifforest@temp
    \forest@process@left@N=#1
  \fi
}
```

Load `n` arguments from the end of the special place to the left side. If $n = 0$, load the entire special place. `l` deletes the args from the special place, `L` keeps them there as well.

```
\forest@def@processor{l}{0}{}{%
  \forest@temptrue
  \forest@processor@U@@
}
\forest@def@processor{L}{0}{}{%
  \forest@tempfalse
  \forest@processor@U@@
}

\def\forest@processor@U@@{%
  \ifnum\forest@process@n=0
    \forest@process@n\forest@process@saved@N\relax
  \fi
  \expandafter\forest@processor@U@@@\expandafter{%
     \the\numexpr\forest@process@saved@N-\forest@process@n}%
}
\def\forest@processor@U@@@#1{%
  \forest@temp@count#1
  \forest@loop
  \ifnum\forest@temp@count<\forest@process@saved@N\relax
    \forest@process@saved@get@{\the\forest@temp@count}\forest@temp
    \forest@process@left@letappend\forest@temp
    \advance\forest@temp@count1
  \forest@repeat
  \let\forestmathresulttype\forest@process@savedtype
  \ifforest@temp
    \let\forest@process@savedtype\forestmathtype@none
    \forest@process@saved@N#1
  \fi
}
```

Boolean operations:

```
\forest@def@processor{&}{2}{}{% and
  \def\forest@tempa{1}%
  \forest@repeat@n@times{\forest@process@n}{%
    \forest@process@left@toppop\forest@tempb
    \edef\forest@tempa{\ifnum10<\forest@tempa\forest@tempb\space 1\else0\fi}%
  }%
  \forest@process@left@esetappend{\forest@tempa}%
  \let\forestmathresulttype\forestmathtype@count
}
\forest@def@processor{|}{2}{}{% or
  \def\forest@tempa{0}%
  \forest@repeat@n@times{\forest@process@n}{%
    \forest@process@left@toppop\forest@tempb
    \edef\forest@tempa{\ifnum0=\forest@tempa\forest@tempb\space 0\else1\fi}%
  }%
  \forest@process@left@esetappend{\forest@tempa}%
  \let\forestmathresulttype\forestmathtype@count
}
\forest@def@processor{!}{}{}{% not
  \forest@process@left@toppop\forest@temp
  \forest@process@left@esetappend{\ifnum0=\forest@temp\space 1\else0\fi}%
  \let\forestmathresulttype\forestmathtype@count
}
\forest@def@processor{?}{}{}{%
  \forest@process@left@toppop\forest@temp
  \forest@process@right@bottompop\forest@tempa
  \forest@process@right@bottompop\forest@tempb
  \ifnum\forest@temp=0
    \forest@process@right@letprepend\forest@tempb
  \else
    \forest@process@right@letprepend\forest@tempa
  \fi
  \let\forestmathresulttype\forestmathtype@generic
}
```

Comparisons. They automatically determine the type (number, dimen, other) of the arguments, by checking what the last processing instruction was.

```
\forest@def@processor{=}{}{}{%
  \forest@process@left@toppop\forest@tempa
  \forest@process@left@toppop\forest@tempb
  \forest@process@left@esetappend{\ifx\forest@tempa\forest@tempb 1\else0\fi}%
  \let\forestmathresulttype\forestmathtype@count
}
\forest@def@processor{<}{}{}{%
  \forest@process@left@toppop\forest@tempb
  \forest@process@left@toppop\forest@tempa
  \ifx\forestmathresulttype\forestmathtype@generic
    \forest@cmp@error\forest@tempa\forest@tempb
  \else
    \forestmathlt{\forest@tempa}{\forest@tempb}%
    \forest@process@left@esetappend{\forestmathresult}%
  \fi
}
\forest@def@processor{>}{}{}{%
  \forest@process@left@toppop\forest@tempb
  \forest@process@left@toppop\forest@tempa
  \ifx\forestmathresulttype\forestmathtype@generic
    \forest@cmp@error\forest@tempa\forest@tempb
  \else
    \forestmathgt{\forest@tempa}{\forest@tempb}%
    \forest@process@left@esetappend{\forestmathresult}%
  \fi
}
```

Various.

```
\forest@def@processor{r}{}{}{% reverse keylist
  \forest@process@right@bottompop\forest@temp
  \expandafter\forest@processor@r@\expandafter{\forest@temp}%
}
\def\forest@processor@r@#1{%
  \forest@process@left@esetappend{}%
  \def\forest@tempcomma{}%
  \pgfqkeys{/forest}{split={#1}{,}{process@rk}}%
  \let\forestmathresulttype\forestmathtype@generic
}
\forestset{%
  process@rk/.code={%
    \forest@process@left@toppop\forest@temp
    \forest@temp@toks{#1}%
    \forest@process@left@esetappend{\the\forest@temp@toks\forest@tempcomma\expandonce{\forest@temp}}%
    \def\forest@tempcomma{,}%
  }%
}
```

7.1.1 Registers

Register declaration mechanism is an adjusted copy-paste of the option declaration mechanism.

```
\def\forest@pgfmathhelper@register@toks#1#2{% #1 is discarded: it is present only for analogy with options
  \forestrget{#2}\pgfmathresult
}
\def\forest@pgfmathhelper@register@dimen#1#2{%
  \forestrget{#2}\forest@temp
  \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
}
\def\forest@pgfmathhelper@register@count#1#2{%
  \forestrget{#2}\pgfmathresult
}
\def\forest@declareregisterhandler#1#2{%#1=handler for specific type,#2=option name
  \pgfkeyssetvalue{/forest/#2/node@or@reg}{}% empty = register (node id=node)
  \forest@convert@others@to@underscores{#2}\forest@pgfmathoptionname
  \edef\forest@marshal{%
    \noexpand#1{/forest/#2}{/forest}{#2}{\forest@pgfmathoptionname}%
  }\forest@marshal
}
\def\forest@declaretoksregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
  \forest@declaretoksregister@handler@A{#1}{#2}{#3}{#4}{}%
}
\def\forest@declarekeylistregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
  \forest@declaretoksregister@handler@A{#1}{#2}{#3}{#4}{,}%
  \forest@copycommandkey{#1}{#1'}%
  \pgfkeyssetvalue{#1'/option@name}{#3}%
  \forest@copycommandkey{#1+}{#1}%
  \pgfkeysalso{#1-/.code={%
      \forest@fornode{}{%
        \forest@node@removekeysfromkeylist{##1}{#3}%
      }}}%
  \pgfkeyssetvalue{#1-/option@name}{#3}%
}
\def\forest@declaretoksregister@handler@A#1#2#3#4#5{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
  \pgfkeysalso{%
```

```
    #1/.code={\forestrset{#3}{##1}},
    #2/if #3/.code n args={3}{%
      \forestrget{#3}\forest@temp@option@value
      \edef\forest@temp@compared@value{\unexpanded{##1}}%
      \ifx\forest@temp@option@value\forest@temp@compared@value
        \pgfkeysalso{##2}%
      \else
        \pgfkeysalso{##3}%
      \fi
    },
    #2/if in #3/.code n args={3}{%
      \forestrget{#3}\forest@temp@option@value
      \edef\forest@temp@compared@value{\unexpanded{##1}}%
      \expandafter\expandafter\expandafter\pgfutil@in@\expandafter\expandafter\expandafter{\expandafter\fores
      \ifpgfutil@in@
        \pgfkeysalso{##2}%
      \else
        \pgfkeysalso{##3}%
      \fi
    },
  }%
  \ifstrempty{#5}{%
    \pgfkeysalso{%
      #1+/.code={\forestrappto{#3}{#5##1}},
      #2/+#3/.code={\forestrpreto{#3}{##1#5}},
    }%
  }{%
    \pgfkeysalso{%
      #1+/.code={%
        \forestrget{#3}\forest@temp
        \ifdefempty{\forest@temp}{%
          \forestrset{#3}{##1}%
        }{%
          \forestrappto{#3}{#5##1}%
        }%
      },
      #2/+#3/.code={%
        \forestrget{#3}\forest@temp
        \ifdefempty{\forest@temp}{%
          \forestrset{#3}{##1}%
        }{%
          \forestrpreto{#3}{##1#5}%
        }%
      }%
    }%
  }%
  \pgfkeyssetvalue{#1/option@name}{#3}%
  \pgfkeyssetvalue{#1+/option@name}{#3}%
  \pgfkeyssetvalue{#2/+#3/option@name}{#3}%
  \pgfkeyslet{#1/@type}\forestmathtype@generic  % for .process & co
  \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@toks{##1}{#3}}%
}
\def\forest@declareautowrappedtoksregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
  \forest@declaretoksregister@handler{#1}{#2}{#3}{#4}%
  \forest@copycommandkey{#1}{#1'}%
  \pgfkeysalso{#1/.style={#1'/.wrap value={##1}}}%
  \pgfkeyssetvalue{#1'/option@name}{#3}%
  \forest@copycommandkey{#1+}{#1+'}%
  \pgfkeysalso{#1+/.style={#1+'/.wrap value={##1}}}%
  \pgfkeyssetvalue{#1+'/option@name}{#3}%
  \forest@copycommandkey{#2/+#3}{#2/+#3'}%
  \pgfkeysalso{#2/+#3/.style={#2/+#3’/.wrap value={##1}}}%
  \pgfkeyssetvalue{#2/+#3’/option@name}{#3}%
}
\def\forest@declarereadonlydimenregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
  \pgfkeysalso{%
    #2/if #3/.code n args={3}{%
      \forestrget{#3}\forest@temp@option@value
      \ifdim\forest@temp@option@value=##1\relax
        \pgfkeysalso{##2}%
      \else
        \pgfkeysalso{##3}%
      \fi
    },
    #2/if #3</.code n args={3}{%
      \forestrget{#3}\forest@temp@option@value
      \ifdim\forest@temp@option@value>##1\relax
        \pgfkeysalso{##3}%
      \else
        \pgfkeysalso{##2}%
      \fi
    },
    #2/if #3>/.code n args={3}{%
      \forestrget{#3}\forest@temp@option@value
      \ifdim\forest@temp@option@value<##1\relax
        \pgfkeysalso{##3}%
      \else
        \pgfkeysalso{##2}%
      \fi
    },
  }%
  \pgfkeyslet{#1/@type}\forestmathtype@dimen   % for .process & co
  \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@dimen{##1}{#3}}%
}
\def\forest@declaredimenregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
  \forest@declarereadonlydimenregister@handler{#1}{#2}{#3}{#4}%
  \pgfkeysalso{%
    #1/.code={%
      \forestmathsetlengthmacro\forest@temp{##1}%
      \forestrlet{#3}\forest@temp
    },
    #1+/.code={%
      \forestmathsetlengthmacro\forest@temp{##1}%
      \pgfutil@tempdima=\forestrve{#3}
      \advance\pgfutil@tempdima\forest@temp\relax
      \forestreset{#3}{\the\pgfutil@tempdima}%
    },
    #1-/.code={%
      \forestmathsetlengthmacro\forest@temp{##1}%
      \pgfutil@tempdima=\forestrve{#3}
      \advance\pgfutil@tempdima-\forest@temp\relax
      \forestreset{#3}{\the\pgfutil@tempdima}%
    },
    #1*/.style={%
      #1={#4()*(##1)}%
    },
    #1:/.style={%
      #1={#4()/(##1)}%
    },
    #1’/.code={%
      \pgfutil@tempdima=##1\relax
      \forestreset{#3}{\the\pgfutil@tempdima}%
    },
    #1’+/.code={%
      \pgfutil@tempdima=\forestrve{#3}\relax
      \advance\pgfutil@tempdima##1\relax
      \forestreset{#3}{\the\pgfutil@tempdima}%
    },
    #1’-/.code={%
      \pgfutil@tempdima=\forestrve{#3}\relax
      \advance\pgfutil@tempdima-##1\relax
      \forestreset{#3}{\the\pgfutil@tempdima}%
    },
    #1’*/.style={%
      \pgfutil@tempdima=\forestrve{#3}\relax
      \multiply\pgfutil@tempdima##1\relax
      \forestreset{#3}{\the\pgfutil@tempdima}%
    },
    #1’:/.style={%
      \pgfutil@tempdima=\forestrve{#3}\relax
      \divide\pgfutil@tempdima##1\relax
      \forestreset{#3}{\the\pgfutil@tempdima}%
    },
  }%
  \pgfkeyssetvalue{#1/option@name}{#3}%
  \pgfkeyssetvalue{#1+/option@name}{#3}%
  \pgfkeyssetvalue{#1-/option@name}{#3}%
  \pgfkeyssetvalue{#1*/option@name}{#3}%
  \pgfkeyssetvalue{#1:/option@name}{#3}%
  \pgfkeyssetvalue{#1’/option@name}{#3}%
  \pgfkeyssetvalue{#1’+/option@name}{#3}%
  \pgfkeyssetvalue{#1’-/option@name}{#3}%
  \pgfkeyssetvalue{#1’*/option@name}{#3}%
  \pgfkeyssetvalue{#1’:/option@name}{#3}%
}
\def\forest@declarereadonlycountregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
  \pgfkeysalso{
    #2/if #3/.code n args={3}{%
      \forestrget{#3}\forest@temp@option@value
      \ifnum\forest@temp@option@value=##1\relax
        \pgfkeysalso{##2}%
      \else
        \pgfkeysalso{##3}%
      \fi
    },
    #2/if #3</.code n args={3}{%
      \forestrget{#3}\forest@temp@option@value
      \ifnum\forest@temp@option@value>##1\relax
        \pgfkeysalso{##3}%
      \else
        \pgfkeysalso{##2}%
      \fi
    },
    #2/if #3>/.code n args={3}{%
      \forestrget{#3}\forest@temp@option@value
      \ifnum\forest@temp@option@value<##1\relax
        \pgfkeysalso{##3}%
      \else
        \pgfkeysalso{##2}%
      \fi
    },
  }%
  \pgfkeyslet{#1/@type}\forestmathtype@count   % for .process & co
  \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@count{##1}{#3}}%
}
\def\forest@declarecountregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
  \forest@declarereadonlycountregister@handler{#1}{#2}{#3}{#4}%
  \pgfkeysalso{
    #1/.code={%
      \forestmathtruncatemacro\forest@temp{##1}%
      \forestrlet{#3}\forest@temp
    },
    #1+/.code={%
      \forestmathtruncatemacro\forest@temp{##1}%
      \c@pgf@counta=\forestrve{#3}\relax
      \advance\c@pgf@counta\forest@temp\relax
      \forestreset{#3}{\the\c@pgf@counta}%
    },
    #1-/.code={%
      \forestmathtruncatemacro\forest@temp{##1}%
      \c@pgf@counta=\forestrve{#3}\relax
      \advance\c@pgf@counta-\forest@temp\relax
      \forestreset{#3}{\the\c@pgf@counta}%
    },
    #1*/.code={%
      \forestmathtruncatemacro\forest@temp{##1}%
      \c@pgf@counta=\forestrve{#3}\relax
      \multiply\c@pgf@counta\forest@temp\relax
      \forestreset{#3}{\the\c@pgf@counta}%
    },
    #1:/.code={%
      \forestmathtruncatemacro\forest@temp{##1}%
      \c@pgf@counta=\forestrve{#3}\relax
      \divide\c@pgf@counta\forest@temp\relax
      \forestreset{#3}{\the\c@pgf@counta}%
    },
    #1'/.code={%
      \c@pgf@counta=##1\relax
      \forestreset{#3}{\the\c@pgf@counta}%
    },
    #1'+/.code={%
      \c@pgf@counta=\forestrve{#3}\relax
      \advance\c@pgf@counta##1\relax
      \forestreset{#3}{\the\c@pgf@counta}%
    },
    #1'-/.code={%
      \c@pgf@counta=\forestrve{#3}\relax
      \advance\c@pgf@counta-##1\relax
      \forestreset{#3}{\the\c@pgf@counta}%
    },
    #1'*/.style={%
      \c@pgf@counta=\forestrve{#3}\relax
      \multiply\c@pgf@counta##1\relax
      \forestreset{#3}{\the\c@pgf@counta}%
    },
    #1':/.style={%
      \c@pgf@counta=\forestrve{#3}\relax
      \divide\c@pgf@counta##1\relax
      \forestreset{#3}{\the\c@pgf@counta}%
    },
  }%
  \pgfkeyssetvalue{#1/option@name}{#3}%
  \pgfkeyssetvalue{#1+/option@name}{#3}%
  \pgfkeyssetvalue{#1-/option@name}{#3}%
  \pgfkeyssetvalue{#1*/option@name}{#3}%
  \pgfkeyssetvalue{#1:/option@name}{#3}%
  \pgfkeyssetvalue{#1'/option@name}{#3}%
  \pgfkeyssetvalue{#1'+/option@name}{#3}%
  \pgfkeyssetvalue{#1'-/option@name}{#3}%
  \pgfkeyssetvalue{#1'*/option@name}{#3}%
  \pgfkeyssetvalue{#1':/option@name}{#3}%
}
\def\forest@declarebooleanregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
  \pgfkeysalso{%
    #1/.code={%
      \ifcsdef{forest@bh@\detokenize{##1}}{%
        \letcs\forest@temp{forest@bh@\detokenize{##1}}%
      }{%
        \forestmathtruncatemacro\forest@temp{##1}%
        \ifx\forest@temp0\else\def\forest@temp{1}\fi
      }%
      \forestrlet{#3}\forest@temp
    },
    #1/.default=1,
    #2/not #3/.code={\forestrset{#3}{0}},
    #2/if #3/.code 2 args={%
      \forestrget{#3}\forest@temp@option@value
      \ifnum\forest@temp@option@value=1
        \pgfkeysalso{##1}%
      \else
        \pgfkeysalso{##2}%
      \fi
    },
  }%
  \pgfkeyssetvalue{#1/option@name}{#3}%
  \pgfkeyslet{#1/@type}\forestmathtype@count   % for .process & co
  \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@count{##1}{#3}}%
}
\forestset{
  declare toks register/.code={%
    \forest@declareregisterhandler\forest@declaretoksregister@handler{#1}%
    \forestset{#1={}}%
  },
  declare autowrapped toks register/.code={%
    \forest@declareregisterhandler\forest@declareautowrappedtoksregister@handler{#1}%
    \forestset{#1={}}%
  },
  declare keylist register/.code={%
    \forest@declareregisterhandler\forest@declarekeylistregister@handler{#1}%
    \forestset{#1'={}}%
  },
  declare dimen register/.code={%
    \forest@declareregisterhandler\forest@declaredimenregister@handler{#1}%
    \forestset{#1'=0pt}%
  },
  declare count register/.code={%
    \forest@declareregisterhandler\forest@declarecountregister@handler{#1}%
    \forestset{#1'=0}%
  },
  declare boolean register/.code={%
    \forest@declareregisterhandler\forest@declarebooleanregister@handler{#1}%
    \forestset{#1=0}%
  },
}
```

Declare some temporary registers.

```
\forestset{
  declare toks register=temptoksa,temptoksa={},
  declare toks register=temptoksb,temptoksb={},
  declare toks register=temptoksc,temptoksc={},
  declare toks register=temptoksd,temptoksd={},
  declare keylist register=tempkeylista,tempkeylista’={},
  declare keylist register=tempkeylistb,tempkeylistb’={},
  declare keylist register=tempkeylistc,tempkeylistc’={},
  declare keylist register=tempkeylistd,tempkeylistd’={},
  declare dimen register=tempdima,tempdima’={0pt},
  declare dimen register=tempdimb,tempdimb’={0pt},
  declare dimen register=tempdimc,tempdimc’={0pt},
  declare dimen register=tempdimd,tempdimd’={0pt},
  declare dimen register=tempdimx,tempdimx’={0pt},
  declare dimen register=tempdimxa,tempdimxa’={0pt},
  declare dimen register=tempdimxb,tempdimxb’={0pt},
  declare dimen register=tempdimy,tempdimy’={0pt},
  declare dimen register=tempdimya,tempdimya’={0pt},
  declare dimen register=tempdimyb,tempdimyb’={0pt},
  declare dimen register=tempdiml,tempdiml’={0pt},
  declare dimen register=tempdimla,tempdimla’={0pt},
  declare dimen register=tempdimlb,tempdimlb’={0pt},
  declare dimen register=tempdims,tempdims’={0pt},
  declare dimen register=tempdimsa,tempdimsa’={0pt},
  declare dimen register=tempdimsb,tempdimsb’={0pt},
  declare count register=tempcounta,tempcounta’={0},
  declare count register=tempcountb,tempcountb’={0},
  declare count register=tempcountc,tempcountc’={0},
  declare count register=tempcountd,tempcountd’={0},
  declare boolean register=tempboola,tempboola={0},
  declare boolean register=tempboolb,tempboolb={0},
  declare boolean register=tempboolc,tempboolc={0},
  declare boolean register=tempboold,tempboold={0},
}
```

7.1.2 Declaring options

```
\def\forest@node@Nametoid#1{% #1 = name
  \csname forest@id@of@#1\endcsname
}
\def\forest@node@Ifnamedefined#1#2#3{% #1 = name, #2=true,#3=false
  \ifcsvoid{forest@id@of@#1}{#3}{#2}%
}
\def\forest@node@setname#1{%
  \def\forest@temp@setname{y}%
  \def\forest@temp@silent{n}%
  \def\forest@temp@propagating{n}%
  \forest@node@setnameoralias{#1}%
}
\def\forest@node@setname@silent#1{%
  \def\forest@temp@setname{y}%
  \def\forest@temp@silent{y}%
  \def\forest@temp@propagating{n}%
  \forest@node@setnameoralias{#1}%
}
\def\forest@node@setalias#1{%
  \def\forest@temp@setname{n}%
  \def\forest@temp@silent{n}%
  \def\forest@temp@propagating{n}%
  \forest@node@setnameoralias{#1}%
}
\def\forest@node@setalias@silent#1{%
  \def\forest@temp@setname{n}%
  \def\forest@temp@silent{y}%
  \def\forest@temp@propagating{n}%
  \forest@node@setnameoralias{#1}%
}
\def\forest@node@setnameoralias#1{%
  \ifstrempty{#1}{%
    \forest@node@setnameoralias{node@\forest@cn}%
  }{%
    \forest@node@Ifnamedefined{#1}{%
      \if y\forest@temp@propagating
        % this will find a unique name, eventually:
        \@escapeif{\forest@node@setnameoralias{#1@\forest@cn}}%
      \else\@escapeif{%
        \if y\forest@temp@setname
          \edef\forest@marshal{%
            \ifstrequal{\forestove{name}}{#1}%
          }\forest@marshal{%
            % same name, no problem
          }{%
            \@escapeif{\forest@node@setnameoralias@nameclash{#1}}%
          }%
        \else\@escapeif{% setting an alias: clashing with alias is not a problem
          \forestOget{\forest@node@Nametoid{#1}}{name}\forest@temp
          \expandafter\ifstrequal\expandafter{\forest@temp}{#1}{%
            \forest@node@setnameoralias@nameclash{#1}%
          }{%
            \forest@node@setnameoralias@do{#1}%
          }%
        }\fi
      }\fi
    }{%
      \forest@node@setnameoralias@do{#1}%
    }%
  }%
}
\def\forest@node@setnameoralias@nameclash#1{%
  \if y\forest@temp@silent
    \forest@fornode{\forest@node@Nametoid{#1}}{%
      \def\forest@temp@propagating{y}%
      \forest@node@setnameoralias{}%
    }%
    \forest@node@setnameoralias@do{#1}%
  \else
    \PackageError{forest}{Node name "#1" is already used}{}%
  \fi
}
\def\forest@node@setnameoralias@do#1{%
  \if y\forest@temp@setname
    \csdef{forest@id@of@\forestove{name}}{}%
    \forestoeset{name}{#1}%
  \fi
  \csedef{forest@id@of@#1}{\forest@cn}%
}
\forestset{
  TeX/.code={#1},
  TeX'/.code={\appto\forest@externalize@loadimages{#1}#1},
  TeX''/.code={\appto\forest@externalize@loadimages{#1}},
  options/.code={\forestset{#1}},
  also/.code={\pgfkeysalso{#1}},
  typeout/.style={TeX={\typeout{#1}}},
  declare toks={name}{},
  name/.code={% override the default setter
    \forest@fornode{\forest@setter@node}{\forest@node@setname{#1}}%
  },
  name/.default={},
  name'/.code={% override the default setter
    \forest@fornode{\forest@setter@node}{\forest@node@setname@silent{#1}}%
  },
  name'/.default={},
  alias/.code={\forest@fornode{\forest@setter@node}{\forest@node@setalias{#1}}},
  alias'/.code={\forest@fornode{\forest@setter@node}{\forest@node@setalias@silent{#1}}},
  begin draw/.code={\begin{tikzpicture}},
  end draw/.code={\end{tikzpicture}},
  declare keylist register=default preamble,
  default preamble'={},
  declare keylist register=preamble,
  preamble'={},
  declare autowrapped toks={content}{},
  % #1 = which option to split, #2 = separator (one char!), #3 = receiving options
  split option/.code n args=3{%
    \forestRNOget{#1}\forest@temp
    \edef\forest@marshal{%
      \noexpand\pgfkeysalso{split={\expandonce{\forest@temp}}\unexpanded{{#2}{#3}}}%
    }\forest@marshal
  },
  split register/.code n args=3{% #1 = which register to split, #2 = separator (one char!), #3 = receiving op
    \forestrget{#1}\forest@temp
    \edef\forest@marshal{%
      \noexpand\pgfkeysalso{split={\expandonce{\forest@temp}}\unexpanded{{#2}{#3}}}%
    }\forest@marshal
  },
  TeX={%
    \def\forest@split@sourcevalues{}%
    \def\forest@split@sourcevalue{}%
    \def\forest@split@receivingoptions{}%
    \def\forest@split@receivingoption{}%
  },
  split/.code n args=3{% #1 = string to split, #2 = separator (one char!), #3 = receiving options
    \forest@saveandrestoremacro\forest@split@sourcevalues{%
      \forest@saveandrestoremacro\forest@split@sourcevalue{%
        \forest@saveandrestoremacro\forest@split@receivingoptions{%
          \forest@saveandrestoremacro\forest@split@receivingoption{%
            \def\forest@split@sourcevalues{#1#2}%
            \edef\forest@split@receivingoptions{#3,}%
            \def\forest@split@receivingoption{}%
            \safeloop
              \expandafter\forest@split\expandafter{\forest@split@sourcevalues}{#2}\forest@split@sourcevalue\
              \ifdefempty\forest@split@receivingoptions{}{%
                \expandafter\forest@split\expandafter{\forest@split@receivingoptions}{,}\forest@temp\forest@s
              \ifdefempty\forest@temp{}{\let\forest@split@receivingoption\forest@temp\def\forest@temp{}}%
              }%
              \edef\forest@marshal{%
                \noexpand\pgfkeysalso{\forest@split@receivingoption={\expandonce{\forest@split@sourcevalue}}}
              }\forest@marshal
              \ifdefempty\forest@split@sourcevalues{\forest@tempfalse}{\forest@temptrue}%
            \ifforest@temp
            \saferepeat
          }}}}%
```

```
},
declare count={grow}{270},
TeX={% a hack for grow-reversed connection, and compass-based grow specification
  \forest@copycommandkey{/forest/grow}{/forest/grow@@}%
  %\pgfkeysgetvalue{/forest/grow/.@cmd}\forest@temp
  %\pgfkeyslet{/forest/grow@@/.@cmd}\forest@temp
},
grow/.style={grow@={#1},reversed=0},
grow'/.style={grow@={#1},reversed=1},
grow''/.style={grow@={#1}},
grow@/.is choice,
grow@/east/.style={/forest/grow@@=0},
grow@/north east/.style={/forest/grow@@=45},
grow@/north/.style={/forest/grow@@=90},
grow@/north west/.style={/forest/grow@@=135},
grow@/west/.style={/forest/grow@@=180},
grow@/south west/.style={/forest/grow@@=225},
grow@/south/.style={/forest/grow@@=270},
grow@/south east/.style={/forest/grow@@=315},
grow@/.unknown/.code={\let\forest@temp@grow\pgfkeyscurrentname
  \pgfkeysalso{/forest/grow@@/.expand once=\forest@temp@grow}},
declare boolean={reversed}{0},
declare toks={parent anchor}{},
declare toks={child anchor}{},
declare toks={anchor}{base},
Autoforward={anchor}{
  node options-=anchor,
  node options+={anchor={##1}}
},
anchor'/.style={anchor@no@compass=true,anchor=#1},
anchor+'/.style={anchor@no@compass=true,anchor+=#1},
anchor-'/.style={anchor@no@compass=true,anchor-=#1},
anchor*'/.style={anchor@no@compass=true,anchor*=#1},
anchor:'/.style={anchor@no@compass=true,anchor:=#1},
anchor'+'/.style={anchor@no@compass=true,anchor'+=#1},
anchor'-'/.style={anchor@no@compass=true,anchor'-=#1},
anchor'*'/.style={anchor@no@compass=true,anchor'*=#1},
anchor':'/.style={anchor@no@compass=true,anchor':=#1},
% /tikz/forest anchor/.style={
%   /forest/TeX={\forestanchortotikzanchor{#1}\forest@temp@anchor},
%   anchor/.expand once=\forest@temp@anchor
% },
declare toks={calign}{midpoint},
TeX={%
  \forest@copycommandkey{/forest/calign}{/forest/calign'}%
},
calign/.is choice,
calign/child/.style={calign'=child},
calign/first/.style={calign'=child,calign primary child=1},
calign/last/.style={calign'=child,calign primary child=-1},
calign with current/.style={for parent/.wrap pgfmath arg={calign=child,calign primary child=##1}{n}},
calign with current edge/.style={for parent/.wrap pgfmath arg={calign=child edge,calign primary child=##1}{
calign/child edge/.style={calign'=child edge},
calign/midpoint/.style={calign'=midpoint},
calign/center/.style={calign'=midpoint,calign primary child=1,calign secondary child=-1},
calign/edge midpoint/.style={calign'=edge midpoint},
calign/fixed angles/.style={calign'=fixed angles},
calign/fixed edge angles/.style={calign'=fixed edge angles},
calign/.unknown/.code={\PackageError{forest}{unknown calign '\pgfkeyscurrentname'}{}},
declare count={calign primary child}{1},
declare count={calign secondary child}{-1},
declare count={calign primary angle}{-35},
declare count={calign secondary angle}{35},
calign child/.style={calign primary child={#1}},
calign angle/.style={calign primary angle={-#1},calign secondary angle={#1}},
declare toks={tier}{},
declare toks={fit}{tight},
declare boolean={ignore}{0},
declare boolean={ignore edge}{0},
no edge/.style={edge'={},ignore edge},
declare keylist={edge}{draw},
declare toks={edge path}{%
  \noexpand\path[\forestoption{edge}]%
  (\forestOve{\forestove{@parent}}{name}.parent anchor)--(\forestove{name}.child anchor)
  % =
  % (!u.parent anchor)--(.child anchor)\forestoption{edge label};
  \forestoption{edge label};%
},
edge path'/.style={
  edge path={%
    \noexpand\path[\forestoption{edge}]%
    #1%
    \forestoption{edge label};
  }
},
declare toks={edge label}{},
declare boolean={phantom}{0},
baseline/.style={alias={forest@baseline@node}},
declare readonly count={id}{0},
declare readonly count={n}{0},
declare readonly count={n'}{0},
declare readonly count={n children}{-1},
declare readonly count={level}{-1},
declare dimen=x{0pt},
declare dimen=y{0pt},
declare dimen={s}{0pt},
declare dimen={l}{6ex}, % just in case: should be set by the calibration
declare dimen={s sep}{0.6666em},
declare dimen={l sep}{1ex},  % just in case: calibration!
declare keylist={node options}{anchor=base},
declare toks={tikz}{},
afterthought/.style={tikz+={#1}},
label/.style={tikz+={\path[late options={%
      name=\forestoption{name},label={#1}}];}},
pin/.style={tikz+={\path[late options={%
      name=\forestoption{name},pin={#1}}];}},
declare toks={content format}{\forestoption{content}},
plain content/.style={content format={\forestoption{content}}},
math content/.style={content format={\noexpand\ensuremath{\forestoption{content}}}},
declare toks={node format}{%
  \noexpand\node
  (\forestoption{name})%
  [\forestoption{node options}]%
  {\foresteoption{content format}};%
},
node format'/.style={
  node format={\noexpand\node(\forestoption{name})#1;}
},
tabular@environment/.style={content format={%
  \noexpand\begin{tabular}[\forestoption{base}]{\forestoption{align}}%
     \forestoption{content}%
   \noexpand\end{tabular}%
```

```
}},
declare toks={align}{},
TeX={%
  \forest@copycommandkey{/forest/align}{/forest/align'}%
  %\pgfkeysgetvalue{/forest/align/.@cmd}\forest@temp
  %\pgfkeyslet{/forest/align'/.@cmd}\forest@temp
},
align/.is choice,
align/.unknown/.code={%
  \edef\forest@marshal{%
    \noexpand\pgfkeysalso{%
      align'={\pgfkeyscurrentname},%
      tabular@environment
    }%
  }\forest@marshal
},
align/center/.style={align'={@{}c@{}},tabular@environment},
align/left/.style={align'={@{}l@{}},tabular@environment},
align/right/.style={align'={@{}r@{}},tabular@environment},
declare toks={base}{t},
TeX={%
  \forest@copycommandkey{/forest/base}{/forest/base'}%
  %\pgfkeysgetvalue{/forest/base/.@cmd}\forest@temp
  %\pgfkeyslet{/forest/base'/.@cmd}\forest@temp
},
base/.is choice,
base/top/.style={base'=t},
base/bottom/.style={base'=b},
base/.unknown/.style={base'/.expand once=\pgfkeyscurrentname},
unknown to/.store in=\forest@unknownto,
unknown to=node options,
unknown key error/.code={\PackageError{forest}{Unknown keyval: \detokenize{#1}}{}},
content to/.store in=\forest@contentto,
content to=content,
.unknown/.code={%
  \expandafter\pgfutil@in@\expandafter.\expandafter{\pgfkeyscurrentname}%
  \ifpgfutil@in@
    \expandafter\forest@relatednode@option@setter\pgfkeyscurrentname=#1\forest@END
  \else
    \edef\forest@marshal{%
      \noexpand\pgfkeysalso{\forest@unknownto={\pgfkeyscurrentname=\unexpanded{#1}}}%
    }\forest@marshal
  \fi
},
get node boundary/.code={%
  \forestoget{@boundary}\forest@node@boundary
  \def#1{}%
  \forest@extendpath#1\forest@node@boundary{\pgfqpoint{\forestove{x}}{\forestove{y}}}%
},
% get min l tree boundary/.code={%
%   \forest@get@tree@boundary{negative}{\the\numexpr\forestove{grow}-90\relax}#1},
% get max l tree boundary/.code={%
%   \forest@get@tree@boundary{positive}{\the\numexpr\forestove{grow}-90\relax}#1},
get min s tree boundary/.code={%
  \forest@get@tree@boundary{negative}{\forestove{grow}}#1},
get max s tree boundary/.code={%
  \forest@get@tree@boundary{positive}{\forestove{grow}}#1},
use as bounding box/.style={%
  before drawing tree={
    tikz+/.expanded={%
      \noexpand\pgfresetboundingbox
```

```
        \noexpand\useasboundingbox
        ($(.anchor)+(\forestoption{min x},\forestoption{min y})$)
        rectangle
        ($(.anchor)+(\forestoption{max x},\forestoption{max y})$)
        ;
      }
    }
  },
  use as bounding box'/.style={%
    before drawing tree={
      tikz+/.expanded={%
        \noexpand\pgfresetboundingbox
        \noexpand\useasboundingbox
        ($(.anchor)+(\forestoption{min x}+\pgfkeysvalueof{/pgf/outer xsep}/2+\pgfkeysvalueof{/pgf/inner xsep}
        rectangle
        ($(.anchor)+(\forestoption{max x}-\pgfkeysvalueof{/pgf/outer xsep}/2-\pgfkeysvalueof{/pgf/inner xsep}
        ;
      }
    }
  },
}%
\def\forest@iftikzkey#1#2#3{% #1 = key name, #2 = true code, #3 = false code
  \forest@temptrue
  \pgfkeysifdefined{/tikz/\pgfkeyscurrentname}{}{%
    \pgfkeysifdefined{/tikz/\pgfkeyscurrentname/.@cmd}{}{%
      \pgfkeysifdefined{/pgf/\pgfkeyscurrentname}{}{%
        \pgfkeysifdefined{/pgf/\pgfkeyscurrentname/.@cmd}{}{%
          \forest@tempfalse
        }}}}%
  \ifforest@temp\@escapeif{#2}\else\@escapeif{#3}\fi
}
\def\forest@ifoptionortikzkey#1#2#3{% #1 = key name, #2 = true code, #3 = false code
  \forest@temptrue
  \pgfkeysifdefined{/forest/\pgfkeyscurrentname}{}{%
    \pgfkeysifdefined{/forest/\pgfkeyscurrentname/.@cmd}{}{%
      \forest@iftikzkey{#1}{}{}%
    }}%
  \ifforest@temp\@escapeif{#2}\else\@escapeif{#3}\fi
}
\def\forest@get@tree@boundary#1#2#3{%#1=pos/neg,#2=grow,#3=receiving cs
  \def#3{}%
  \forest@node@getedge{#1}{#2}\forest@temp@boundary
  \forest@extendpath#3\forest@temp@boundary{\pgfqpoint{\forestove{x}}{\forestove{y}}}%
}
\def\forest@setter@node{\forest@cn}%
\def\forest@relatednode@option@compat@ignoreinvalidsteps#1{#1}
\def\forest@relatednode@option@setter#1.#2=#3\forest@END{%
  \forest@forthis{%
    \forest@relatednode@option@compat@ignoreinvalidsteps{%
      \forest@nameandgo{#1}%
      \let\forest@setter@node\forest@cn
    }%
  }%
  \ifnum\forest@setter@node=0
  \else
    \forestset{#2={#3}}%
  \fi
  \def\forest@setter@node{\forest@cn}%
}%
\def\forest@split#1#2#3#4{% #1=list (assuming that the list is nonempty and finishes with the separator), #2
  \def\forest@split@@##1#2##2\forest@split@@##3##4{\def##3{##1}\def##4{##2}}%
  \forest@split@@#1\forest@split@@{#3}{#4}}
```

7.1.3 Option propagation

The propagators targeting single nodes are automatically defined by nodewalk steps definitions.

```
\forestset{
  for tree'/.style 2 args={#1,for children={for tree'={#1}{#2}},#2},
  if/.code n args={3}{%
    \forestmathtruncatemacro\forest@temp{#1}%
    \ifnum\forest@temp=0
      \@escapeif{\pgfkeysalso{#3}}%
    \else
      \@escapeif{\pgfkeysalso{#2}}%
    \fi
  },
  %LaTeX if/.code n args={3}{#1{\pgfkeysalso{#2}}{\pgfkeysalso{#3}}},
  if nodewalk valid/.code n args={3}{%
    \forest@forthis{%
      \forest@configured@nodewalk{independent}{inherited}{fake}{%
        #1,
        TeX={\global\let\forest@global@temp\forest@cn}
      }{}%
    }%
    \ifnum\forest@global@temp=0
      \@escapeif{\pgfkeysalso{#3}}%
    \else
      \@escapeif{\pgfkeysalso{#2}}%
    \fi
  },
  if nodewalk empty/.code n args={3}{%
    \forest@forthis{%
      \forest@configured@nodewalk{independent}{independent}{fake}{%
        #1,
        TeX={\global\let\forest@global@temp\forest@nodewalk@n},
      }{}%
    }%
    \ifnum\forest@global@temp=0
      \@escapeif{\pgfkeysalso{#2}}%
    \else
      \@escapeif{\pgfkeysalso{#3}}%
    \fi
  },
  if current nodewalk empty/.code 2 args={%
    \ifnum\forest@nodewalk@n=0
      \@escapeif{\pgfkeysalso{#1}}%
    \else
      \@escapeif{\pgfkeysalso{#2}}%
    \fi
  },
  where/.style n args={3}{for tree={if={#1}{#2}{#3}}},
  where nodewalk valid/.style n args={3}{for tree={if nodewalk valid={#1}{#2}{#3}}},
  where nodewalk empty/.style n args={3}{for tree={if nodewalk empty={#1}{#2}{#3}}},
  repeat/.code 2 args={%
    \forestmathtruncatemacro\forest@temp{#1}%
    \expandafter\forest@repeatkey\expandafter{\forest@temp}{#2}%
  },
  until/.code 2 args={%
    \ifstrempty{#1}{%
      \forest@untilkey{\ifnum\forest@cn=0\else\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
    }{%
      \forest@untilkey{\forestmath@if{#1}{\forestloopbreak}{}}{#2}%
    }%
  },
  while/.code 2 args={%
    \ifstrempty{#1}{%
      \forest@untilkey{\ifnum\forest@cn=0\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
    }{%
      \forest@untilkey{\forestmath@if{#1}{}{\forestloopbreak}}{#2}%
    }%
  },
  do until/.code 2 args={%
    \ifstrempty{#1}{%
      \forest@dountilkey{\ifnum\forest@cn=0\else\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
    }{%
      \forest@dountilkey{\forestmath@if{#1}{\forestloopbreak}{}}{#2}%
    }%
  },
  do while/.code 2 args={%
    \ifstrempty{#1}{%
      \forest@dountilkey{\ifnum\forest@cn=0\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
    }{%
      \forest@dountilkey{\forestmath@if{#1}{}{\forestloopbreak}}{#2}%
    }%
  },
  until nodewalk valid/.code 2 args={%
    \forest@untilkey{\forest@forthis{%
        \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\else\forestloopbreak\fi}}{}}}{#2
  },
  while nodewalk valid/.code 2 args={%
    \forest@untilkey{\forest@forthis{%
        \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\forestloopbreak\fi}}{}}}{#2}%
  },
  do until nodewalk valid/.code 2 args={%
    \forest@dountilkey{\forest@forthis{%
        \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\else\forestloopbreak\fi}}{}}}{#2
  },
  do while nodewalk valid/.code 2 args={%
    \forest@dountilkey{\forest@forthis{%
        \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\forestloopbreak\fi}}{}}}{#2}%
  },
  until nodewalk empty/.code 2 args={%
    \forest@untilkey{\forest@forthis{%
        \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\forestloopbreak\fi}}{}}}
  },
  while nodewalk empty/.code 2 args={%
    \forest@untilkey{\forest@forthis{%
        \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\else\forestloopbreak\fi}
  },
  do until nodewalk empty/.code 2 args={%
    \forest@dountilkey{\forest@forthis{%
        \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\forestloopbreak\fi}}{}}}
  },
  do while nodewalk empty/.code 2 args={%
    \forest@dountilkey{\forest@forthis{%
        \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\else\forestloopbreak\fi}
  },
  break/.code={\forestloopBreak{#1}},
  break/.default=0,
}
\def\forest@repeatkey#1#2{%
  \safeRKloop
  \ifnum\safeRKloopn>#1\relax
  \csuse{safeRKbreak@\the\safeRKloop@depth true}%
  \fi
  \expandafter\unless\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
    \pgfkeysalso{#2}%
  \safeRKrepeat
}
\def\forest@untilkey#1#2{% #1 = condition, #2 = keys
  \safeRKloop
  #1%
  \expandafter\unless\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
    \pgfkeysalso{#2}%
  \safeRKrepeat
}
\def\forest@dountilkey#1#2{% #1 = condition, #2 = keys
  \safeRKloop
  \pgfkeysalso{#2}%
  #1%
  \expandafter\unless\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
  \safeRKrepeat
}
\def\forestloopbreak{%
  \csname safeRKbreak@\the\safeRKloop@depth true\endcsname
}
\def\forestloopBreak#1{%
  \csname safeRKbreak@\number\numexpr\the\safeRKloop@depth-#1\relax true\endcsname
}
\def\forestloopcount{%
  \csname safeRKloopn@\number\numexpr\the\safeRKloop@depth\endcsname
}
\def\forestloopCount#1{%
  \csname safeRKloopn@\number\numexpr\the\safeRKloop@depth-#1\endcsname
}
\pgfmathdeclarefunction{forestloopcount}{1}{%
  \edef\pgfmathresult{\forestloopCount{\ifstrempty{#1}{0}{#1}}}%
}
\forest@copycommandkey{/forest/repeat}{/forest/nodewalk/repeat}
\forest@copycommandkey{/forest/while}{/forest/nodewalk/while}
\forest@copycommandkey{/forest/do while}{/forest/nodewalk/do while}
\forest@copycommandkey{/forest/until}{/forest/nodewalk/until}
\forest@copycommandkey{/forest/do until}{/forest/nodewalk/do until}
\forest@copycommandkey{/forest/if}{/forest/nodewalk/if}
\forest@copycommandkey{/forest/if nodewalk valid}{/forest/nodewalk/if nodewalk valid}
%
```

7.2 Aggregate functions

```
\forestset{
  aggregate postparse/.is choice,
  aggregate postparse/int/.code={%
    \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@toint},
  aggregate postparse/none/.code={%
    \let\forest@aggregate@pgfmathpostparse\relax},
  aggregate postparse/print/.code={%
    \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@print},
  aggregate postparse/macro/.code={%
    \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@usemacro},
  aggregate postparse macro/.store in=\forest@aggregate@pgfmathpostparse@macro,
}
\def\forest@aggregate@pgfmathpostparse@print{%
  \pgfmathprintnumberto{\pgfmathresult}{\pgfmathresult}%
```

```
}
\def\forest@aggregate@pgfmathpostparse@toint{%
  \expandafter\forest@split\expandafter{\pgfmathresult.}{.}\pgfmathresult\forest@temp
}
\def\forest@aggregate@pgfmathpostparse@usemacro{%
  \forest@aggregate@pgfmathpostparse@macro
}
\let\forest@aggregate@pgfmathpostparse\relax
\forestset{
  /handlers/.aggregate/.code n args=4{%
    % #1 = start value (forestmath)
    % #2 = forestmath expression that calculates "aggregate result" at each step
    % #3 = forestmath expression that calculates "aggregate result" at the end of the nodewalk
    % #4 = nodewalk
    \forest@aggregate@handler{\forest@aggregate@generic{#1}{#2}{#3}{#4}}%
  },
  /handlers/.sum/.code 2 args={% #1=forestmath, #2=nodewalk
    \forest@aggregate@handler{\forest@aggregate@sum{#1}{#2}}%
  },
  /handlers/.count/.code={% #1=nodewalk
    \forest@aggregate@handler{\forest@aggregate@count{#1}}%
  },
  /handlers/.average/.code 2 args={% #1=forestmath, #2=nodewalk
    \forest@aggregate@handler{\forest@aggregate@average{#1}{#2}}%
  },
  /handlers/.product/.code 2 args={% #1=forestmath, #2=nodewalk
    \forest@aggregate@handler{\forest@aggregate@product{#1}{#2}}%
  },
  /handlers/.min/.code 2 args={% #1=forestmath, #2=nodewalk
    \forest@aggregate@handler{\forest@aggregate@min{#1}{#2}}%
  },
  /handlers/.max/.code 2 args={% #1=forestmath, #2=nodewalk
    \forest@aggregate@handler{\forest@aggregate@max{#1}{#2}}%
  },
  declare count register={aggregate n},
  declare toks register={aggregate value},
  declare toks register={aggregate result},
  aggregate result={},
}
\def\forest@aggregate@handler#1{%
  \edef\forest@marshal{%
    \unexpanded{%
      #1%
    }{%
      \noexpand\pgfkeysalso{\pgfkeyscurrentpath/.register=aggregate result}%
    }%
  }\forest@marshal
}
\def\forest@aggregate@pgfmathfunction@finish{%
  \forestrget{aggregate result}\pgfmathresult
}
\pgfmathdeclarefunction{aggregate}{4}{%
  \forest@aggregate@generic{#1}{#2}{#3}{#4}%
  \forest@aggregate@pgfmathfunction@finish
}
\pgfmathdeclarefunction{aggregate_count}{1}{%
  \forest@aggregate@sum{#1}%
  \forest@aggregate@pgfmathfunction@finish
}
\pgfmathdeclarefunction{aggregate_sum}{2}{%
  \forest@aggregate@sum{#1}{#2}%
  \forest@aggregate@pgfmathfunction@finish
}
\pgfmathdeclarefunction{aggregate_product}{2}{%
  \forest@aggregate@product{#1}{#2}%
  \forest@aggregate@pgfmathfunction@finish
}
\pgfmathdeclarefunction{aggregate_average}{2}{%
  \forest@aggregate@average{#1}{#2}%
  \forest@aggregate@pgfmathfunction@finish
}
\pgfmathdeclarefunction{aggregate_min}{2}{%
  \forest@aggregate@min{#1}{#2}%
  \forest@aggregate@pgfmathfunction@finish
}
\pgfmathdeclarefunction{aggregate_max}{2}{%
  \forest@aggregate@max{#1}{#2}%
  \forest@aggregate@pgfmathfunction@finish
}
```

Define particular aggregate functions.

```
\def\forest@aggregate#1#2#3#4#5#6{% #1...#5=real args,
                                % #6=what to do with |aggregate result| register
    % #1 = start value (forestmath)
    % #2 = forestmath expression that calculates "aggregate current" at each step
    % #3 = forestmath expression that calculates "aggregate result" at each step
    % #4 = forestmath expression that calculates "aggregate result" at the end of the nodewalk
    % #5 = nodewalk
  \forest@saveandrestoreregister{aggregate result}{%
    \forest@saveandrestoreregister{aggregate n}{%
      \forest@aggregate@{#1}{#2}{#3}{#4}{#5}%
      #6%
    }%
  }%
}
\def\forest@aggregate@generic#1#2#3#4{\forest@aggregate
  {\forestmathparse{#1}}%
  {}%
  {\forestmathparse{#2}}%
  {\forestmathparse{#3}}%
  {#4}%
}
\def\forest@aggregate@sum#1#2{\forest@aggregate
  {\forestmath@convert@fromto\forestmathtype@count\forestmathtype@generic{0}}%
  {\forestmathparse{#1}}%
  {\forestmathadd{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
  {\forestrget{aggregate result}\forestmathresult}%
  {#2}%
}
\def\forest@aggregate@count#1{\forest@aggregate
  {\def\forestmathresult{0}\let\forestmathresulttype\forestmathtype@count}%
  {\def\forestmathresult{1}\let\forestmathresulttype\forestmathtype@count}%
  {\edef\forestmathresult{\the\numexpr\forestregister{aggregate result}+1}\let\forestmathresulttype\forestmat
  {\forestrget{aggregate result}\forestmathresult\let\forestmathresulttype\forestmathtype@count}%
  {#1}%
}
\def\forest@aggregate@average#1#2{\forest@aggregate
  {\forestmath@convert@fromto\forestmathtype@count\forestmathtype@generic{0}}%
  {\forestmathparse{#1}}%
  {\forestmathadd{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
  {\forestmathdivide@P{\forestregister{aggregate result}}{\forestregister{aggregate n}}}%
  {#2}%
}
\def\forest@aggregate@product#1#2{\forest@aggregate
  {\forestmath@convert@fromto\forestmathtype@count\forestmathtype@generic{1}}%
  {\forestmathparse{#1}}%
  {\forestmathmultiply{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
  {\forestrget{aggregate result}\forestmathresult}%
  {#2}%
}
\def\forest@aggregate@min#1#2{\forest@aggregate
  {\def\forestmathresult{}}%
  {\forestmathparse{#1}}%
  {\forestmathmin{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
  {\forestrget{aggregate result}\forestmathresult}%
  {#2}%
}
\def\forest@aggregate@max#1#2{\forest@aggregate
  {\def\forestmathresult{}}%
  {\forestmathparse{#1}}%
  {\forestmathmax{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
  {\forestrget{aggregate result}\forestmathresult}%
  {#2}%
}
```

Actual computation.

```
\def\forest@aggregate@#1#2#3#4#5{%
    % #1 = start value (forestmath)
    % #2 = forestmath expression that calculates "aggregate current" at each step
    % #3 = forestmath expression that calculates "aggregate result" at each step
    % #4 = forestmath expression that calculates "aggregate result" at the end of the nodewalk
    % #5 = nodewalk
  #1%
  \forestrlet{aggregate result}\forestmathresult
  \forestrset{aggregate value}{}%
  \forestrset{aggregate n}{0}%
  \forest@forthis{%
    \forest@nodewalk{#5}{%
      TeX={%
        \forestreset{aggregate n}{\number\numexpr\forestrv{aggregate n}+1}%
        #2%
        \forestrlet{aggregate value}\forestmathresult
        #3%
        \forestrlet{aggregate result}\forestmathresult
      }%
    }{}%
  }%
  #4%
  \let\forest@temp@pgfmathpostparse\pgfmathpostparse
  \let\pgfmathpostparse\forest@aggregate@pgfmathpostparse
  \forestmath@convert@to\forestmathtype@dimen{\forestmathresult}%
  \pgfmathqparse{\forestmathresult}%
  \let\pgfmathpostparse\forest@temp@pgfmathpostparse
  \forestrlet{aggregate result}\pgfmathresult
}
```

7.2.1 pgfmath extensions

```
\pgfmathdeclarefunction{strequal}{2}{%
  \ifstrequal{#1}{#2}{\def\pgfmathresult{1}}{\def\pgfmathresult{0}}%
}
\pgfmathdeclarefunction{instr}{2}{%
  \pgfutil@in@{#1}{#2}%
  \ifpgfutil@in@\def\pgfmathresult{1}\else\def\pgfmathresult{0}\fi
```

```
}
\pgfmathdeclarefunction{strcat}{...}{%
  \edef\pgfmathresult{\forest@strip@braces{#1}}%
}
\pgfmathdeclarefunction{min_s}{2}{% #1 = node, #2 = context node (for growth rotation)
  \forest@forthis{%
    \forest@nameandgo{#1}%
    \forest@compute@minmax@ls{#2}%
    \edef\forest@temp{\forestove{min@s}}%
    \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
  }%
}
\pgfmathdeclarefunction{min_l}{2}{% #1 = node, #2 = context node (for growth rotation)
  \forest@forthis{%
    \forest@nameandgo{#1}%
    \forest@compute@minmax@ls{#2}%
    \edef\forest@temp{\forestove{min@l}}%
    \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
  }%
}
\pgfmathdeclarefunction{max_s}{2}{% #1 = node, #2 = context node (for growth rotation)
  \forest@forthis{%
    \forest@nameandgo{#1}%
    \forest@compute@minmax@ls{#2}%
    \edef\forest@temp{\forestove{max@s}}%
    \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
  }%
}
\pgfmathdeclarefunction{max_l}{2}{% #1 = node, #2 = context node (for growth rotation)
  \forest@forthis{%
    \forest@nameandgo{#1}%
    \forest@compute@minmax@ls{#2}%
    \edef\forest@temp{\forestove{max@l}}%
    \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
  }%
}
\def\forest@compute@minmax@ls#1{% #1 = nodewalk; in the context of which node?
  {%
    \pgftransformreset
    \forest@forthis{%
      \forest@nameandgo{#1}%
      \forest@pgfqtransformrotate{-\forestove{grow}}%
    }%
    \forestoget{min x}\forest@temp@minx
    \forestoget{min y}\forest@temp@miny
    \forestoget{max x}\forest@temp@maxx
    \forestoget{max y}\forest@temp@maxy
    \pgfpointtransformed{\pgfqpoint{\forest@temp@minx}{\forest@temp@miny}}%
    \forestoeset{min@l}{\the\pgf@x}%
    \forestoeset{min@s}{\the\pgf@y}%
    \forestoeset{max@l}{\the\pgf@x}%
    \forestoeset{max@s}{\the\pgf@y}%
    \pgfpointtransformed{\pgfqpoint{\forest@temp@minx}{\forest@temp@maxy}}%
    \ifdim\pgf@x<\forestove{min@l}\relax\forestoeset{min@l}{\the\pgf@x}\fi
    \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
    \ifdim\pgf@x>\forestove{max@l}\relax\forestoeset{max@l}{\the\pgf@x}\fi
    \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
    \pgfpointtransformed{\pgfqpoint{\forest@temp@maxx}{\forest@temp@miny}}%
    \ifdim\pgf@x<\forestove{min@l}\relax\forestoeset{min@l}{\the\pgf@x}\fi
    \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
    \ifdim\pgf@x>\forestove{max@l}\relax\forestoeset{max@l}{\the\pgf@x}\fi
    \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
    \pgfpointtransformed{\pgfqpoint{\forest@temp@maxx}{\forest@temp@maxy}}%
    \ifdim\pgf@x<\forestove{min@l}\relax\forestoeset{min@l}{\the\pgf@x}\fi
    \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
    \ifdim\pgf@x>\forestove{max@l}\relax\forestoeset{max@l}{\the\pgf@x}\fi
    \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
    % smuggle out
    \edef\forest@marshal{%
      \noexpand\forestoeset{min@l}{\forestove{min@l}}%
      \noexpand\forestoeset{min@s}{\forestove{min@s}}%
      \noexpand\forestoeset{max@l}{\forestove{max@l}}%
      \noexpand\forestoeset{max@s}{\forestove{max@s}}%
    }\expandafter
  }\forest@marshal
}
\def\forest@pgfmathhelper@attribute@toks#1#2{%
  \forest@forthis{%
    \forest@nameandgo{#1}%
    \ifnum\forest@cn=0
      \def\pgfmathresult{}%
    \else
      \forestoget{#2}\pgfmathresult
    \fi
  }%
}
\def\forest@pgfmathhelper@attribute@dimen#1#2{%
  \forest@forthis{%
    \forest@nameandgo{#1}%
    \ifnum\forest@cn=0
      \def\pgfmathresult{0}%
    \else
      \forestoget{#2}\forest@temp
      \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
    \fi
  }%
}
\def\forest@pgfmathhelper@attribute@count#1#2{%
  \forest@forthis{%
    \forest@nameandgo{#1}%
    \ifnum\forest@cn=0
      \def\pgfmathresult{0}%
    \else
      \forestoget{#2}\pgfmathresult
    \fi
  }%
}
\pgfmathdeclarefunction*{id}{1}{%
  \forest@forthis{%
    \forest@nameandgo{#1}%
    \let\pgfmathresult\forest@cn
  }%
}
```

7.3 Nodewalk

Setup machinery.

```
\def\forest@nodewalk@n{0}
\def\forest@nodewalk@historyback{0,}
\def\forest@nodewalk@historyforward{0,}
\def\forest@nodewalk@origin{0}
\def\forest@nodewalk@config@everystep@independent@before#1{% #1 = every step keylist
```

```
  \forestrset{every step}{#1}%
}
\def\forest@nodewalk@config@everystep@independent@after{%
  \noexpand\forestrset{every step}{\forestrv{every step}}%
}
\def\forest@nodewalk@config@history@independent@before{%
  \def\forest@nodewalk@n{0}%
  \edef\forest@nodewalk@origin{\forest@cn}%
  \def\forest@nodewalk@historyback{0,}%
  \def\forest@nodewalk@historyforward{0,}%
}
\def\forest@nodewalk@config@history@independent@after{%
  \edef\noexpand\forest@nodewalk@n{\expandonce{\forest@nodewalk@n}}%
  \edef\noexpand\forest@nodewalk@origin{\expandonce{\forest@nodewalk@origin}}%
  \edef\noexpand\forest@nodewalk@historyback{\expandonce{\forest@nodewalk@historyback}}%
  \edef\noexpand\forest@nodewalk@historyforward{\expandonce{\forest@nodewalk@historyforward}}%
}
\def\forest@nodewalk@config@everystep@shared@before#1{}% #1 = every step keylist
\def\forest@nodewalk@config@everystep@shared@after{}
\def\forest@nodewalk@config@history@shared@before{}
\def\forest@nodewalk@config@history@shared@after{}
\def\forest@nodewalk@config@everystep@inherited@before#1{}% #1 = every step keylist
\let\forest@nodewalk@config@everystep@inherited@after\forest@nodewalk@config@everystep@independent@after
\def\forest@nodewalk@config@history@inherited@before{}
\let\forest@nodewalk@config@history@inherited@after\forest@nodewalk@config@history@independent@after
\def\forest@nodewalk#1#2{% #1 = nodewalk, #2 = every step keylist
  \forest@configured@nodewalk{independent}{independent}{inherited}{#1}{#2}%
}
\def\forest@configured@nodewalk#1#2#3#4#5{%
  % #1 = every step method, #2 = history method, #3 = on invalid
  % #4 = nodewalk, #5 = every step keylist
  \def\forest@nodewalk@config@everystep@method{#1}%
  \def\forest@nodewalk@config@history@method{#2}%
  \def\forest@nodewalk@config@oninvalid{#3}%
  \forest@Nodewalk{#4}{#5}%
}
\def\forest@nodewalk@oninvalid@inherited@text{inherited}
\def\forest@Nodewalk#1#2{% #1 = nodewalk, #2 = every step keylist
  \ifx\forest@nodewalk@config@oninvalid\forest@nodewalk@oninvalid@inherited@text
    \edef\forest@nodewalk@config@oninvalid{\forest@nodewalk@oninvalid}%
  \fi
  \edef\forest@nw@marshal{%
    \noexpand\pgfqkeys{/forest/nodewalk}{\unexpanded{#1}}%
    \csname forest@nodewalk@config@everystep@\forest@nodewalk@config@everystep@method @after\endcsname
    \csname forest@nodewalk@config@history@\forest@nodewalk@config@history@method @after\endcsname
    \edef\noexpand\forest@nodewalk@oninvalid{\forest@nodewalk@oninvalid}%
  }%
  \csname forest@nodewalk@config@everystep@\forest@nodewalk@config@everystep@method @before\endcsname{#2}%
  \csname forest@nodewalk@config@history@\forest@nodewalk@config@history@method @before\endcsname
  \edef\forest@nodewalk@oninvalid{\forest@nodewalk@config@oninvalid}%
  \forest@saveandrestoreifcs{forest@nodewalk@fake}{%
    \forest@nodewalk@fakefalse
    \forest@nw@marshal
  }%
}
\pgfmathdeclarefunction{valid}{1}{%
  \forest@forthis{%
    \forest@nameandgo{#1}%
    \edef\pgfmathresult{\ifnum\forest@cn=0 0\else 1\fi}%
  }%
}
```

```
\pgfmathdeclarefunction{invalid}{1}{%
  \forest@forthis{%
    \forest@nameandgo{#1}%
    \edef\pgfmathresult{\ifnum\forest@cn=0 1\else 0\fi}%
  }%
}
\newif\ifforest@nodewalk@fake
\def\forest@nodewalk@oninvalid{error}
\def\forest@nodewalk@makestep{%
  \ifnum\forest@cn=0
    \csname forest@nodewalk@makestep@oninvalid@\forest@nodewalk@oninvalid\endcsname
  \else
    \forest@nodewalk@makestep@
  \fi
}
\csdef{forest@nodewalk@makestep@oninvalid@error if real}{\ifforest@nodewalk@fake\expandafter\forest@nodewalk@
\csdef{forest@nodewalk@makestep@oninvalid@last valid}{%
  \forest@nodewalk@tolastvalid
  \ifforestdebugnodewalks\forest@nodewalk@makestep@invalidtolastvalid@debug\fi}%
\def\forest@nodewalk@makestep@oninvalid@error{\PackageError{forest}{nodewalk stepped to the invalid node\Mess
\let\forest@nodewalk@makestep@oninvalid@fake\relax
\def\forest@nodewalk@makestep@oninvalid@compatfake{%
  \forest@deprecated{last step in stack "\forest@nodewalk@currentstepname", which stepped on an invalid node;
}%
\def\forest@nodewalk@makestep@{%
  \ifforestdebugnodewalks\forest@nodewalk@makestep@debug\fi
  \ifforest@nodewalk@fake
  \else
    \edef\forest@nodewalk@n{\number\numexpr\forest@nodewalk@n+1}%
    \epreto\forest@nodewalk@historyback{\forest@cn,}%
    \def\forest@nodewalk@historyforward{0,}%
    \forest@process@keylist@register{every step}%
  \fi
}
\def\forest@nodewalk@makestep@debug{%
  \edef\forest@marshal{%
    \noexpand\typeout{\ifforest@nodewalk@fake fake \fi "\forest@nodewalk@currentstepname" step to node id=\fo
  }\forest@marshal
}%
\def\forest@nodewalk@makestep@invalidtolastvalid@debug{%
  \edef\forest@marshal{%
    \noexpand\typeout{\ifforest@nodewalk@fake fake \fi "\forest@nodewalk@currentstepname" step to invalid nod
  }\forest@marshal
}%
\def\forest@handlers@savecurrentpath{%
  \edef\pgfkeyscurrentkey{\pgfkeyscurrentpath}%
  \let\forest@currentkey\pgfkeyscurrentkey
  \pgfkeys@split@path
  \edef\forest@currentpath{\pgfkeyscurrentpath}%
  \let\forest@currentname\pgfkeyscurrentname
}
\pgfkeys{/handlers/save current path/.code={\forest@handlers@savecurrentpath}}
\newif\ifforest@nodewalkstephandler@style
\newif\ifforest@nodewalkstephandler@autostep
\newif\ifforest@nodewalkstephandler@stripfakesteps
\newif\ifforest@nodewalkstephandler@muststartatvalidnode
\newif\ifforest@nodewalkstephandler@makefor
\let\forest@nodewalkstephandler@styletrueorfalse\forest@nodewalkstephandler@stylefalse
\def\forest@nodewalk@currentstepname{}
\forestset{
  /forest/define@step/style/.is if=forest@nodewalkstephandler@style,
  /forest/define@step/autostep/.is if=forest@nodewalkstephandler@autostep,
  % the following is useful because some macros use grouping (by \forest@forthis or similar) and therefore, a
  /forest/define@step/strip fake steps/.is if=forest@nodewalkstephandler@stripfakesteps,
  % this can never happen with autosteps ...
  /forest/define@step/autostep/.append code={%
    \ifforest@nodewalkstephandler@autostep
      \forest@nodewalkstephandler@stripfakestepsfalse
    \fi
  },
  /forest/define@step/must start at valid node/.is if=forest@nodewalkstephandler@muststartatvalidnode,
  /forest/define@step/n args/.store in=\forest@nodewalkstephandler@nargs,
  /forest/define@step/make for/.is if=forest@nodewalkstephandler@makefor,
  /forest/define@step/@bare/.style={strip fake steps=false,must start at valid node=false,make for=false},
  define long step/.code n args=3{%
    \forest@nodewalkstephandler@styletrueorfalse % true for end users; but in the package, most of steps are
    \forest@nodewalkstephandler@autostepfalse
    \forest@nodewalkstephandler@stripfakestepstrue
    \forest@nodewalkstephandler@muststartatvalidnodetrue % most steps can only start at a valid node
    \forest@nodewalkstephandler@makefortrue % make for prefix?
    \def\forest@nodewalkstephandler@nargs{0}%
    \pgfqkeys{/forest/define@step}{#2}%
    \forest@temp@toks{#3}% handler code
    \ifforest@nodewalkstephandler@style
      \expandafter\forest@temp@toks\expandafter{%
        \expandafter\pgfkeysalso\expandafter{\the\forest@temp@toks}%
      }%
    \fi
    \ifforest@nodewalkstephandler@autostep
      \apptotoks\forest@temp@toks{\forest@nodewalk@makestep}%
    \fi
    \ifforest@nodewalkstephandler@stripfakesteps
      \expandafter\forest@temp@toks\expandafter{\expandafter\forest@nodewalk@stripfakesteps\expandafter{\the\
    \fi
    \ifforest@nodewalkstephandler@muststartatvalidnode
      \edef\forest@marshal{%
        \noexpand\forest@temp@toks{%
          \unexpanded{%
            \ifnum\forest@cn=0
              \csname forest@nodewalk@start@oninvalid@\forest@nodewalk@oninvalid\endcsname{#1}%
            \else
          }%
              \noexpand\@escapeif{\the\forest@temp@toks}%
            \noexpand\fi
          }%
        }\forest@marshal
    \fi
    \pretotoks\forest@temp@toks{\appto\forest@nodewalk@currentstepname{,#1}}%
    \expandafter\forest@temp@toks\expandafter{\expandafter\forest@saveandrestoremacro\expandafter\forest@node
    \ifforestdebugnodewalks
      \epretotoks\forest@temp@toks{\noexpand\typeout{Starting step "#1" from id=\noexpand\forest@cn
          \ifnum\forest@nodewalkstephandler@nargs>0 \space with args \noexpand\unexpanded{####1}\fi
          \ifnum\forest@nodewalkstephandler@nargs>1 ,\noexpand\unexpanded{####2}\fi
          \ifnum\forest@nodewalkstephandler@nargs>2 ,\noexpand\unexpanded{####3}\fi
          \ifnum\forest@nodewalkstephandler@nargs>3 ,\noexpand\unexpanded{####4}\fi
          \ifnum\forest@nodewalkstephandler@nargs>4 ,\noexpand\unexpanded{####5}\fi
          \ifnum\forest@nodewalkstephandler@nargs>5 ,\noexpand\unexpanded{####6}\fi
          \ifnum\forest@nodewalkstephandler@nargs>6 ,\noexpand\unexpanded{####7}\fi
          \ifnum\forest@nodewalkstephandler@nargs>7 ,\noexpand\unexpanded{####8}\fi
          \ifnum\forest@nodewalkstephandler@nargs>8 ,\noexpand\unexpanded{####9}\fi
        }}%
    \fi
```

```
    \def\forest@temp{/forest/nodewalk/#1/.code}%
    \ifnum\forest@nodewalkstephandler@nargs<2
      \eappto\forest@temp{=}%
    \else\ifnum\forest@nodewalkstephandler@nargs=2
      \eappto\forest@temp{ 2 args=}%
    \else
      \eappto\forest@temp{ n args={\forest@nodewalkstephandler@nargs}}%
    \fi\fi
    \eappto\forest@temp{{\the\forest@temp@toks}}%
    \expandafter\pgfkeysalso\expandafter{\forest@temp}%
    \ifforest@nodewalkstephandler@makefor
      \ifnum\forest@nodewalkstephandler@nargs=0
        \forestset{%
          for #1/.code={\forest@forstepwrapper{#1}{##1}},
        }%
      \else\ifnum\forest@nodewalkstephandler@nargs=1
        \forestset{%
          for #1/.code 2 args={\forest@forstepwrapper{#1={##1}}{##2}},
        }%
      \else
        \forestset{%
          for #1/.code n args/.expanded=%
            {\number\numexpr\forest@nodewalkstephandler@nargs+1}%
            {\noexpand\forest@forstepwrapper{#1\ifnum\forest@nodewalkstephandler@nargs>0=\fi\forest@util@narg
        }%
      \fi\fi
    \fi
  },
}
{\csname forest@@doc@@hook@bigbadforlist\endcsname}%
\pgfqkeys{/handlers}{
  .nodewalk style/.code={\forest@handlers@savecurrentpath\pgfkeysalso{%
      \forest@currentpath/nodewalk/\forest@currentname/.style={#1}%
    }},
}
```

`\forest@forstepwrapper` is defined so that it can be changed by `compat` to create unfailable spatial propagators from v1.0.

```
\def\forest@forstepwrapper#1#2{\forest@forthis{\forest@nodewalk{#1}{#2}}}
\def\forest@util@nargs#1#2#3{% #1 = prefix (#, ##, ...), #2 = n args, #3=start; returns {#start+1}...{#start+
  \ifnum#2>0 {#1\number\numexpr#3+1}\fi
  \ifnum#2>1 {#1\number\numexpr#3+2}\fi
  \ifnum#2>2 {#1\number\numexpr#3+3}\fi
  \ifnum#2>3 {#1\number\numexpr#3+4}\fi
  \ifnum#2>4 {#1\number\numexpr#3+5}\fi
  \ifnum#2>5 {#1\number\numexpr#3+6}\fi
  \ifnum#2>6 {#1\number\numexpr#3+7}\fi
  \ifnum#2>7 {#1\number\numexpr#3+8}\fi
  \ifnum#2>8 {#1\number\numexpr#3+9}\fi
}
\def\forest@nodewalk@start@oninvalid@fake#1{}
\def\forest@nodewalk@start@oninvalid@compatfake#1{%
  \forest@deprecated{last step in stack "\forest@nodewalk@currentstepname", which started from an invalid nod
}%
\let\forest@nodewalk@start@oninvalid@errorifreal\forest@nodewalk@start@oninvalid@fake % the step will be to a
\let\forest@nodewalk@start@oninvalid@lastvalid\forest@nodewalk@start@oninvalid@fake
\def\forest@nodewalk@start@oninvalid@error#1{\PackageError{forest}{nodewalk step "#1" cannot start at the inv
```

Define long-form single-step walks.

```
\forestset{
  define long step={current}{autostep}{},
  define long step={next}{autostep}{\edef\forest@cn{\forestove{@next}}},
  define long step={previous}{autostep}{\edef\forest@cn{\forestove{@previous}}},
  define long step={parent}{autostep}{\edef\forest@cn{\forestove{@parent}}},
  define long step={first}{autostep}{\edef\forest@cn{\forestove{@first}}},
  define long step={last}{autostep}{\edef\forest@cn{\forestove{@last}}},
  define long step={sibling}{autostep}{%
    \edef\forest@cn{%
      \ifnum\forestove{@previous}=0
        \forestove{@next}%
      \else
        \forestove{@previous}%
      \fi
    }%
  },
  define long step={next node}{autostep}{\edef\forest@cn{\forest@node@linearnextid}},
  define long step={previous node}{autostep}{\edef\forest@cn{\forest@node@linearpreviousid}},
  define long step={first leaf}{autostep}{%
    \safeloop
      \edef\forest@cn{\forestove{@first}}%
    \unless\ifnum\forestove{@first}=0
    \saferepeat
  },
  define long step={first leaf'}{autostep}{%
    \safeloop
    \unless\ifnum\forestove{@first}=0
      \edef\forest@cn{\forestove{@first}}%
    \saferepeat
  },
  define long step={last leaf}{autostep}{%
    \safeloop
      \edef\forest@cn{\forestove{@last}}%
    \unless\ifnum\forestove{@last}=0
    \saferepeat
  },
  define long step={last leaf'}{autostep}{%
    \safeloop
    \unless\ifnum\forestove{@last}=0
      \edef\forest@cn{\forestove{@last}}%
    \saferepeat
  },
  define long step={next leaf}{style,strip fake steps=false}{group={do until={n_children()==0}{next node}}},
  define long step={previous leaf}{style,strip fake steps=false}{group={do until={n_children()==0}{previous n
  define long step={next on tier}{autostep,n args=1}{%
    \def\forest@temp{#1}%
    \ifx\forest@temp\pgfkeysnovalue@text
      \forestoget{tier}\forest@nodewalk@giventier
    \else
      \def\forest@nodewalk@giventier{#1}%
    \fi
    \edef\forest@cn{\forest@node@linearnextid}%
    \safeloop
      \forest@nodewalk@gettier
    \ifforest@temp
      \edef\forest@cn{\forest@node@linearnextid}%
    \saferepeat
  },
  define long step={previous on tier}{autostep,n args=1}{%
    \def\forest@temp{#1}%
    \ifx\forest@temp\pgfkeysnovalue@text
      \forestoget{tier}\forest@nodewalk@giventier
    \else
```

```
    \def\forest@nodewalk@giventier{#1}%
  \fi
  \safeloop
    \edef\forest@cn{\forest@node@linearpreviousid}%
    \forest@nodewalk@gettier
  \ifforest@temp
  \saferepeat
},
TeX={%
  \def\forest@nodewalk@gettier{%
    \ifnum\forest@cn=0
      \forest@tempfalse
    \else
      \forestoget{tier}\forest@temp
      \ifx\forest@temp\forest@nodewalk@giventier
        \forest@tempfalse
      \else
        \forest@temptrue
      \fi
    \fi
  }%
},
%
define long step={root}{autostep,must start at valid node=false}{%
  \edef\forest@cn{\forest@node@rootid}},
define long step={root'}{autostep,must start at valid node=false}{%
  \forestOifdefined{\forest@root}{@parent}{\edef\forest@cn{\forest@root}}{\edef\forest@cn{0}}%
},
define long step={origin}{autostep,must start at valid node=false}{\edef\forest@cn{\forest@nodewalk@origin}
%
define long step={n}{autostep,n args=1}{%
  \forestmathtruncatemacro\forest@temp@n{#1}%
  \edef\forest@cn{\forest@node@nthchildid{\forest@temp@n}}%
},
define long step={n}{autostep,make for=false,n args=1}{%
  % Yes, twice. ;-)
  % n=1 and n(ext)
  \def\forest@nodewalk@temp{#1}%
  \ifx\forest@nodewalk@temp\pgfkeysnovalue@text
    \edef\forest@cn{\forestove{@next}}%
  \else
    \forestmathtruncatemacro\forest@temp@n{#1}%
    \edef\forest@cn{\forest@node@nthchildid{\forest@temp@n}}%
  \fi
},
define long step={n'}{autostep,n args=1}{%
  \forestmathtruncatemacro\forest@temp@n{#1}%
  \edef\forest@cn{\forest@node@nbarthchildid{\forest@temp@n}}%
},
define long step={to tier}{autostep,n args=1}{%
  \def\forest@nodewalk@giventier{#1}%
  \safeloop
    \forest@nodewalk@gettier
  \ifforest@temp
    \forestoget{@parent}\forest@cn
  \saferepeat
},
%
define long step={name}{autostep,n args=1,must start at valid node=false}{%
  \edef\forest@cn{%
    \forest@node@Ifnamedefined{#1}{\forest@node@Nametoid{#1}}{0}%
  }%
},
define long step={id}{autostep,n args=1,must start at valid node=false}{%
  \forestOifdefined{#1}{@parent}{\edef\forest@cn{#1}}{\edef\forest@cn{0}}%
},
define long step={Nodewalk}{n args=3,@bare}{% #1 = config, #2 = nodewalk
  \def\forest@nodewalk@config@everystep@method{independent}%
  \def\forest@nodewalk@config@history@method{shared}%
  \def\forest@nodewalk@config@oninvalid{inherited}%
  \pgfqkeys{/forest/nodewalk@config}{#1}%
  \forest@Nodewalk{#2}{#3}%
},
define long step={nodewalk}{n args=2,@bare}{% #1 = nodewalk, #2 = every step
  \forest@nodewalk{#1}{#2}%
},
define long step={nodewalk’}{n args=1,@bare}{% #1 = nodewalk
  \forest@configured@nodewalk{inherited}{independent}{inherited}{#1}{}%
},
% these "for ..." keys must be defined explicitely
% (and copied into node keyspace manually),
% as prefix "for" normally introduces the every-step keylist
define long step={for nodewalk}{n args=2,@bare}{% #1 = nodewalk, #2 = every step
  \forest@forthis{\forest@nodewalk{#1}{#2}}},
define long step={for nodewalk’}{n args=1,@bare}{% #1 = nodewalk
  \forest@forthis{%
    \forest@configured@nodewalk{inherited}{independent}{inherited}{#1}{}%
  }%
},
define long step={for Nodewalk}{n args=3,@bare}{% #1 = config, #2 = nodewalk, #3 = every-step
  \def\forest@nodewalk@config@everystep@method{independent}%
  \def\forest@nodewalk@config@history@method{shared}%
  \def\forest@nodewalk@config@oninvalid{inherited}%
  \pgfqkeys{/forest/nodewalk@config}{#1}%
  \forest@forthis{\forest@Nodewalk{#2}{#3}}%
},
copy command key={/forest/nodewalk/Nodewalk}{/forest/Nodewalk},
copy command key={/forest/nodewalk/for nodewalk}{/forest/for nodewalk},
copy command key={/forest/nodewalk/for Nodewalk}{/forest/for Nodewalk},
declare keylist register=every step,
every step’={},
%%% begin nodewalk config
nodewalk@config/.cd,
every@step/.is choice,
every@step/independent/.code={},
every@step/inherited/.code={},
every@step/shared/.code={},
every step/.store in=\forest@nodewalk@config@everystep@method,
every step/.prefix style={every@step=#1},
@history/.is choice,
@history/independent/.code={},
@history/inherited/.code={},
@history/shared/.code={},
history/.store in=\forest@nodewalk@config@history@method,
history/.prefix style={@history=#1},
on@invalid/.is choice,
on@invalid/error/.code={},
on@invalid/fake/.code={},
on@invalid/error if real/.code={},
on@invalid/last valid/.code={},
on@invalid/inherited/.code={},
on invalid/.store in=\forest@nodewalk@config@oninvalid,
  on invalid/.prefix style={on@invalid=#1},
  %%% end nodewalk config
}
\newtoks\forest@nodewalk@branch@toks
\forestset{
  declare toks register=branch@temp@toks,
  branch@temp@toks={},
  declare keylist register=branched@nodewalk,
  branched@nodewalk={},
  define long step={branch}{n args=1,@bare,make for,style}{@branch={#1}{branch@build@realstep,branch@build@fa
  define long step={branch'}{n args=1,@bare,make for,style}{@branch={#1}{branch@build@realstep}},
  @branch/.style 2 args={%
    save and restore register={branched@nodewalk}{
      branch@temp@toks={},
      split/.process={r}{#1}{,}{#2},
      also/.register=branch@temp@toks,
      also/.register=branched@nodewalk,
    }
  },
  nodewalk/branch@build@realstep/.style={% #1 = nodewalk for this branch
    branch@temp@toks/.expanded={for nodewalk={\unexpanded{#1}}{
        branched@nodewalk+/.expanded={id=\noexpand\forestoption{id}},
        \forestregister{branch@temp@toks}}},
  },
  nodewalk/branch@build@fakestep/.style={% #1 = nodewalk for this branch
    branch@temp@toks/.expanded={for nodewalk={\unexpanded{#1}}{
        \forestregister{branch@temp@toks}}},
  },
  define long step={group}{autostep,n args=1}{\forest@go{#1}},
  nodewalk/fake/.code={%
    \forest@saveandrestoreifcs{forest@nodewalk@fake}{%
      \forest@nodewalk@faketrue
      \pgfkeysalso{#1}%
    }%
  },
  nodewalk/real/.code={%
    \forest@saveandrestoreifcs{forest@nodewalk@fake}{%
      \forest@nodewalk@fakefalse
      \pgfkeysalso{#1}%
    }%
  },
  declare keylist register=filtered@nodewalk,
  filtered@nodewalk={},
  define long step={filter}{n args=2,@bare,make for,style}{% #1 = nodewalk, #2 = condition
    save and restore register={filtered@nodewalk}{
      filtered@nodewalk'={},
      Nodewalk=%
        {history=inherited}%
        {#1}%
        {if={#2}{filtered@nodewalk+/.expanded={id=\forestoption{id}}}{}},
      filtered@nodewalk@style/.style/.register=filtered@nodewalk,
      filtered@nodewalk@style
    },
  },
  on@invalid/.is choice,
  on@invalid/error/.code={},
  on@invalid/fake/.code={},
  on@invalid/error if real/.code={},
  on@invalid/last valid/.code={},
  on invalid/.code 2 args={%
    \pgfkeysalso{/forest/on@invalid={#1}}%
    \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
      \def\forest@nodewalk@oninvalid{#1}%
      \pgfkeysalso{#2}%
    }%
  },
  define long step={strip fake steps}{n args=1,@bare}{%
    \forest@nodewalk@stripfakesteps{\pgfkeysalso{#1}}},
  define long step={unique}{n args=1}{%
    \begingroup
    \def\forest@nodewalk@unique@temp{}%
    \forest@nodewalk{#1}{%
      TeX={%
        \forestoget{unique@visited}\forest@temp
        \ifx\forest@temp\relax
          \forestoset{unique@visited}{1}%
          \eappto\forest@nodewalk@unique@temp{,id=\forest@cn}%
        \fi
      }%
    }%
    \global\let\forest@global@temp\forest@nodewalk@unique@temp
    \endgroup
    \pgfkeysalsofrom{\forest@global@temp}%
  },
  define long step={walk back}{n args=1,@bare}{%
    \forestmathtruncatemacro\forest@temp@n{#1}%
    \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\ifnum\forest@cn
    \forest@nodewalk@back@updatehistory
  },
  nodewalk/walk back/.default=1,
  define long step={jump back}{n args=1,@bare}{%
    \forestmathtruncatemacro\forest@temp@n{(#1)+\ifnum\forest@cn=0 0\else1\fi}%
    \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\forest@temp@n-1
    \forest@nodewalk@back@updatehistory
  },
  nodewalk/jump back/.default=1,
  define long step={back}{n args=1,@bare}{%
    \forestmathtruncatemacro\forest@temp@n{#1}%
    \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\ifnum\forest@cn
    \forest@nodewalk@back@updatehistory
  },
  nodewalk/back/.default=1,
  define long step={walk forward}{n args=1,@bare}{%
    \forestmathtruncatemacro\forest@temp@n{#1}%
    \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{0}{\forest@temp@
    \forest@nodewalk@forward@updatehistory
  },
  nodewalk/walk forward/.default=1,
  define long step={jump forward}{n args=1,@bare}{%
    \forestmathtruncatemacro\forest@temp@n{#1}%
    \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{\forest@temp@n-1
    \forest@nodewalk@forward@updatehistory
  },
  nodewalk/jump forward/.default=1,
  define long step={forward}{n args=1,@bare}{%
    \forestmathtruncatemacro\forest@temp@n{#1}%
    \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{0}{\forest@temp@
    \forest@nodewalk@forward@updatehistory
  },
  nodewalk/forward/.default=1,
  define long step={last valid'}{@bare}{%
    \ifnum\forest@cn=0
```

```
    \forest@nodewalk@tolastvalid
    \forest@nodewalk@makestep
  \fi
},
define long step={last valid}{@bare}{%
  \forest@nodewalk@tolastvalid
},
define long step={reverse}{n args=1,@bare,make for}{%
  \forest@nodewalk{#1,TeX={%
      \global\let\forest@global@temp\forest@nodewalk@historyback
      \global\let\forest@global@tempn\forest@nodewalk@n
    }}{}%
  \forest@nodewalk@walklist{}{\forest@global@temp}{0}{\forest@global@tempn}{\let\forest@cn\forest@nodewalk@
},
define long step={walk and reverse}{n args=1,@bare,make for}{%
  \edef\forest@marshal{%
    \noexpand\pgfkeysalso{\unexpanded{#1}}%
    \noexpand\forest@nodewalk@walklist{}{\noexpand\forest@nodewalk@historyback}{0}{\noexpand\forest@nodewal
  }\forest@marshal
},
define long step={sort}{n args=1,@bare,make for}{%
  \forest@nodewalk{#1,TeX={%
      \global\let\forest@global@temp\forest@nodewalk@historyback
      \global\let\forest@global@tempn\forest@nodewalk@n
    }}{}%
  \forest@nodewalk@sortlist{\forest@global@temp}{\forest@global@tempn}\forest@sort@ascending
},
define long step={sort'}{n args=1,@bare,make for}{%
  \forest@nodewalk{#1,TeX={%
      \global\let\forest@global@temp\forest@nodewalk@historyback
      \global\let\forest@global@tempn\forest@nodewalk@n
    }}{}%
  \forest@nodewalk@sortlist{\forest@global@temp}{\forest@global@tempn}\forest@sort@descending
},
define long step={walk and sort}{n args=1,@bare,make for}{% walk as given, then walk sorted
  \edef\forest@marshal{%
    \noexpand\pgfkeysalso{\unexpanded{#1}}%
    \noexpand\forest@nodewalk@sortlist{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\
  }\forest@marshal
},
define long step={walk and sort'}{n args=1,@bare,make for}{%
  \edef\forest@marshal{%
    \noexpand\pgfkeysalso{\unexpanded{#1}}%
    \noexpand\forest@nodewalk@sortlist{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\
  }\forest@marshal
},
declare keylist register=sort by,
copy command key={/forest/sort by'}{/forest/sort by},
sort by={},
define long step={save}{n args=2,@bare,make for}{% #1 = name, #2 = nodewalk
  \forest@forthis{%
    \forest@nodewalk{#2,TeX={%
        \global\let\forest@global@temp\forest@nodewalk@historyback
        \global\let\forest@global@tempn\forest@nodewalk@n
      }}{}%
  }%
  \forest@nodewalk@walklist{}{\forest@global@temp}{0}{\forest@global@tempn}\relax
  \csedef{forest@nodewalk@saved@#1}{\forest@nodewalk@walklist@walked}%
},
define long step={walk and save}{n args=2,@bare,make for}{% #1 = name, #2 = nodewalk
  \edef\forest@marshal{%
```

```
   \noexpand\pgfkeysalso{\unexpanded{#2}}%
   \noexpand\forest@nodewalk@walklist{}{\noexpand\forest@nodewalk@historyback}{0}{\noexpand\forest@nodewal
  }\forest@marshal
  \csedef{forest@nodewalk@saved@#1}{\forest@nodewalk@walklist@walked}%
 },
 define long step={save append}{style,n args=2,@bare,make for}{% #1 = nodewalk name, #2 = nodewalk
  save@append@prepend={#1}{#2}{save}{\cseappto}},
 define long step={save prepend}{style,n args=2,@bare,make for}{% #1 = nodewalk name, #2 = nodewalk
  save@append@prepend={#1}{#2}{save}{\csepreto}},
 define long step={walk and save append}{style,n args=2,@bare,make for}{% #1 = nodewalk name, #2 = nodewalk
  save@append@prepend={#1}{#2}{walk and save}{\cseappto}},
 define long step={walk and save prepend}{style,n args=2,@bare,make for}{% #1 = nodewalk name, #2 = nodewalk
  save@append@prepend={#1}{#2}{walk and save}{\csepreto}},
 nodewalk/save@append@prepend/.code n args=4{%
  % #1 = nodewalk name, #2 = nodewalk
  % #3 = "(walk and) save" #4 = \cseappto/\csepreto
  \pgfkeysalso{#3={@temp}{#2}}%
  \letcs\forest@temp{forest@nodewalk@saved@@temp}%
  #4{forest@nodewalk@saved@#1}{\expandonce{\forest@temp}}%
 },
 nodewalk/save history/.code 2 args={% #1 = back, forward
  \csedef{forest@nodewalk@saved@#1}{\forest@nodewalk@historyback}%
  \csedef{forest@nodewalk@saved@#2}{\forest@nodewalk@historyforward}%
 },
 define long step={load}{n args=1,@bare,make for}{%
  \forest@nodewalk@walklist{}{\csuse{forest@nodewalk@saved@#1}0,}{0}{-1}{\ifnum\forest@nodewalk@cn=0 \else\
 },
 if in saved nodewalk/.code n args=4{% is node #1 in nodewalk #2; yes: #3, no: #4
  \forest@forthis{%
   \forest@go{#1}%
   \edef\forest@marshal{%
    \noexpand\pgfutil@in@{,\forest@cn,}{,\csuse{forest@nodewalk@saved@#2},}%
   }\forest@marshal
  }%
  \ifpgfutil@in@
   \@escapeif{\pgfkeysalso{#3}}%
  \else
   \@escapeif{\pgfkeysalso{#4}}%
  \fi
 },
 where in saved nodewalk/.style n args=4{
  for tree={if in saved nodewalk={#1}{#2}{#3}{#4}}
 },
 nodewalk/options/.code={\forestset{#1}},
 nodewalk/TeX/.code={#1},
 nodewalk/TeX'/.code={\appto\forest@externalize@loadimages{#1}#1},
 nodewalk/TeX''/.code={\appto\forest@externalize@loadimages{#1}},
 nodewalk/typeout/.style={TeX={\typeout{#1}}},
 % repeat is taken later from /forest/repeat
}
\def\forest@nodewalk@walklist#1#2#3#4#5{%
 % #1 = list of preceding, #2 = list to walk
 % #3 = from, #4 = to
 % #5 = every step code
 \let\forest@nodewalk@cn\forest@cn
 \edef\forest@marshal{%
  \noexpand\forest@nodewalk@walklist@{#1}{#2}{\number\numexpr#3}{\number\numexpr#4}{1}{0}{\unexpanded{#5}}%
 }\forest@marshal
}
\def\forest@nodewalk@walklist@#1#2#3#4#5#6#7{%
 % #1 = list of walked, #2 = list to walk
```

```
  % #3 = from, #4 = to
  % #5 = current step n, #6 = steps made
  % #7 = every step code
  \def\forest@nodewalk@walklist@walked{#1}%
  \def\forest@nodewalk@walklist@rest{#2}%
  \edef\forest@nodewalk@walklist@stepsmade{#6}%
  \ifnum#4<0
    \forest@temptrue
  \else
    \ifnum#5>#4\relax
      \forest@tempfalse
    \else
      \forest@temptrue
    \fi
  \fi
  \ifforest@temp
    \edef\forest@nodewalk@cn{\forest@csvlist@getfirst@{#2}}%
    \ifnum\forest@nodewalk@cn=0
      #7%
    \else
      \ifnum#5>#3\relax
        #7%
        \edef\forest@nodewalk@walklist@stepsmade{\number\numexpr#6+1}%
      \fi
      \forest@csvlist@getfirstrest@{#2}\forest@nodewalk@cn\forest@nodewalk@walklist@rest
      \@escapeifif{%
        \edef\forest@marshal{%
          \noexpand\forest@nodewalk@walklist@
            {\forest@nodewalk@cn,#1}{\forest@nodewalk@walklist@rest}{#3}{#4}{\number\numexpr#5+1}{\forest@nod
        }\forest@marshal
      }%
    \fi
  \fi
}

\def\forest@nodewalk@back@updatehistory{%
  \ifnum\forest@cn=0
    \let\forest@nodewalk@historyback\forest@nodewalk@walklist@rest
    \let\forest@nodewalk@historyforward\forest@nodewalk@walklist@walked
  \else
    \expandafter\forest@csvlist@getfirstrest@\expandafter{\forest@nodewalk@walklist@walked}\forest@temp\fores
    \edef\forest@nodewalk@historyback{\forest@temp,\forest@nodewalk@walklist@rest}%
  \fi
}
\def\forest@nodewalk@forward@updatehistory{%
  \let\forest@nodewalk@historyforward\forest@nodewalk@walklist@rest
  \let\forest@nodewalk@historyback\forest@nodewalk@walklist@walked
}
\def\forest@go#1{%
  \forest@configured@nodewalk{independent}{inherited}{inherited}{#1}{}%
}
\def\forest@csvlist@getfirst@#1{% assuming that the list is nonempty and finishes with a comma
  \forest@csvlist@getfirst@@#1\forest@csvlist@getfirst@@}
\def\forest@csvlist@getfirst@@#1,#2\forest@csvlist@getfirst@@{#1}
\def\forest@csvlist@getrest@#1{% assuming that the list is nonempty and finishes with a comma
  \forest@csvlist@getrest@@#1\forest@csvlist@getrest@@}
\def\forest@csvlist@getrest@@#1,#2\forest@csvlist@getrest@@{#2}
\def\forest@csvlist@getfirstrest@#1#2#3{% assuming that the list is nonempty and finishes with a comma
  % #1 = list, #2 = cs receiving first, #3 = cs receiving rest
  \forest@csvlist@getfirstrest@@#1\forest@csvlist@getfirstrest@@{#2}{#3}}
\def\forest@csvlist@getfirstrest@@#1,#2\forest@csvlist@getfirstrest@@#3#4{%
```

```
  \def#3{#1}%
  \def#4{#2}%
}
\def\forest@nodewalk@stripfakesteps#1{%
  % go to the last valid node if the walk contained any nodes, otherwise restore the current node
  \edef\forest@marshal{%
    \unexpanded{#1}%
    \noexpand\ifnum\noexpand\forest@nodewalk@n=\forest@nodewalk@n\relax
      \def\noexpand\forest@cn{\forest@cn}%
    \noexpand\else
      \unexpanded{%
        \edef\forest@cn{%
          \expandafter\forest@csvlist@getfirst@\expandafter{\forest@nodewalk@historyback}%
        }%
      }%
    \noexpand\fi
  }\forest@marshal
}
\def\forest@nodewalk@tolastvalid{%
  \ifnum\forest@cn=0
    \edef\forest@cn{\expandafter\forest@csvlist@getfirst@\expandafter{\forest@nodewalk@historyback}}%
    \ifnum\forest@cn=0
      \let\forest@cn\forest@nodewalk@origin
    \fi
  \fi
}
\def\forest@nodewalk@sortlist#1#2#3{%#1=list,#2=to,#3=asc/desc
  \edef\forest@nodewalksort@list{#1}%
  \expandafter\forest@nodewalk@sortlist@\expandafter{\number\numexpr#2}{#3}%
}
\def\forest@nodewalk@sortlist@#1#2{%#1=to,#2=asc/desc
  \safeloop
    \unless\ifnum\safeloopn>#1\relax
    \expandafter\forest@csvlist@getfirstrest@\expandafter{\forest@nodewalksort@list}\forest@nodewalksort@cn\f
    \csedef{forest@nodesort@\safeloopn}{\forest@nodewalksort@cn}%
  \saferepeat
  \forestrget{sort by}\forest@nodesort@sortkey
  \forest@sort\forest@nodesort@cmpnodes\forest@nodesort@let#2{1}{#1}%
  \def\forest@nodewalksort@sorted{}%
  \safeloop
    \unless\ifnum\safeloopn>#1\relax
    \edef\forest@cn{\csname forest@nodesort@\safeloopn\endcsname}%
    \forest@nodewalk@makestep
  \saferepeat
}
```

Find minimal/maximal node in a walk.

```
\forestset{
  define long step={min}{n args=1,@bare,make for}{% the first min in the argument nodewalk
    \forest@nodewalk{#1,TeX={%
        \global\let\forest@global@temp\forest@nodewalk@historyback
      }}{}%
    \forest@nodewalk@minmax{\forest@global@temp}{-1}{<}{\forest@nodewalk@minmax@node,}%
  },
  define long step={mins}{n args=1,@bare,make for}{% all mins in the argument nodewalk
    \forest@nodewalk{#1,TeX={%
        \global\let\forest@global@temp\forest@nodewalk@historyback
      }}{}%
    \forest@nodewalk@minmax{\forest@global@temp}{-1}{<}{\forest@nodewalk@minmax@nodes}%
  },
  define long step={walk and min}{n args=1,@bare}{%
```

```
    \edef\forest@marshal{%
      \noexpand\pgfkeysalso{\unexpanded{#1}}%
      \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
    }\forest@marshal
  },
  define long step={walk and mins}{n args=1,@bare}{%
    \edef\forest@marshal{%
      \noexpand\pgfkeysalso{\unexpanded{#1}}%
      \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
    }\forest@marshal
  },
  define long step={min in nodewalk}{@bare}{% find the first min in the preceding nodewalk, step to it
    \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{\forest@nodewalk@minmax@node,}%
  },
  define long step={mins in nodewalk}{@bare}{% find mins in the preceding nodewalk, step to mins
    \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{\forest@nodewalk@minmax@nodes}%
  },
  define long step={min in nodewalk'}{@bare}{% find the first min in the preceding nodewalk, step to min in h
    \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{}%
  },
  %
  define long step={max}{n args=1,@bare,make for}{% the first max in the argument nodewalk
    \forest@nodewalk{#1,TeX={%
      \global\let\forest@global@temp\forest@nodewalk@historyback
    }}{}%
    \forest@nodewalk@minmax{\forest@global@temp}{-1}{>}{\forest@nodewalk@minmax@node,}%
  },
  define long step={maxs}{n args=1,@bare,make for}{% all maxs in the argument nodewalk
    \forest@nodewalk{#1,TeX={%
      \global\let\forest@global@temp\forest@nodewalk@historyback
    }}{}%
    \forest@nodewalk@minmax{\forest@global@temp}{-1}{>}{\forest@nodewalk@minmax@nodes}%
  },
  define long step={walk and max}{n args=1,@bare}{%
    \edef\forest@marshal{%
      \noexpand\pgfkeysalso{\unexpanded{#1}}%
      \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
    }\forest@marshal
  },
  define long step={walk and maxs}{n args=1,@bare}{%
    \edef\forest@marshal{%
      \noexpand\pgfkeysalso{\unexpanded{#1}}%
      \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
    }\forest@marshal
  },
  define long step={max in nodewalk}{@bare}{% find the first max in the preceding nodewalk, step to it
    \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{\forest@nodewalk@minmax@node,}%
  },
  define long step={maxs in nodewalk}{@bare}{% find maxs in the preceding nodewalk, step to maxs
    \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{\forest@nodewalk@minmax@nodes}%
  },
  define long step={max in nodewalk'}{@bare}{% find the first max in the preceding nodewalk, step to max in h
    \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{}%
  },
}

\def\forest@nodewalk@minmax#1#2#3#4{%
  % #1 = list of nodes
  % #2 = max index in list (start with 1)
  % #3 = min/max = ascending/descending = </>
  % #4 = how many results? 1 = {\forest@nodewalk@minmax@node,}, all={\forest@nodewalk@minmax@nodes}, walk in
  \forestrget{sort by}\forest@nodesort@sortkey
  \edef\forest@nodewalk@minmax@N{\number\numexpr#2}%
  \edef\forest@nodewalk@minmax@n{}%
  \edef\forest@nodewalk@minmax@list{#1}%
  \def\forest@nodewalk@minmax@nodes{}%
  \def\forest@nodewalk@minmax@node{}%
  \ifdefempty{\forest@nodewalk@minmax@list}{%
  }{%
    \safeloop
      \expandafter\forest@csvlist@getfirstrest@\expandafter{\forest@nodewalk@minmax@list}\forest@nodewalk@min
      \ifnum\forest@nodewalk@minmax@cn=0 \else
        \ifdefempty{\forest@nodewalk@minmax@node}{%
          \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
          \edef\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,}%
          \edef\forest@nodewalk@minmax@n{\safeloopn}%
        }{%
          \csedef{forest@nodesort@1}{\forest@nodewalk@minmax@node}%
          \csedef{forest@nodesort@2}{\forest@nodewalk@minmax@cn}%
          \forest@nodesort@cmpnodes{2}{1}%
          \if=\forest@sort@cmp@result
            \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
            \epreto\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,}%
            \edef\forest@nodewalk@minmax@n{\safeloopn}%
          \else
            \if#3\forest@sort@cmp@result
              \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
              \edef\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,}%
              \edef\forest@nodewalk@minmax@n{\safeloopn}%
            \fi
          \fi
        }%
      \fi
      \ifdefempty{\forest@nodewalk@minmax@list}{\forest@tempfalse}{\forest@temptrue}%
      \ifnum\safeloopn=\forest@nodewalk@minmax@N\relax\forest@temptrue\fi
    \ifforest@temp
    \saferepeat
    \edef\forest@nodewalk@minmax@list{#4}%
    \ifdefempty\forest@nodewalk@minmax@list{%
      \forestset{nodewalk/jump back=\forest@nodewalk@minmax@n-1}% CHECK
    }{%
      \safeloop
        \expandafter\forest@csvlist@getfirstrest@\expandafter{\forest@nodewalk@minmax@list}\forest@cn\forest@
        \forest@nodewalk@makestep
        \ifdefempty{\forest@nodewalk@minmax@list}{\forest@tempfalse}{\forest@temptrue}%
      \ifforest@temp
      \saferepeat
    }%
  }%
}
```

The short-form step mechanism. The complication is that we want to be able to collect tikz and pgf options here, and it is impossible(?) to know in advance what keys are valid there. So we rather check whether the given keyname is a sequence of short steps; if not, we pass the key on.

```
\newtoks\forest@nodewalk@shortsteps@resolution
\newif\ifforest@nodewalk@areshortsteps
\pgfqkeys{/forest/nodewalk}{
  .unknown/.code={%
    \forest@nodewalk@areshortstepsfalse
    \ifx\pgfkeyscurrentvalue\pgfkeysnovalue@text % no value, so possibly short steps
      \forest@nodewalk@shortsteps@resolution{}%
      \forest@nodewalk@areshortstepstrue
      \expandafter\forest@nodewalk@shortsteps\pgfkeyscurrentname==========,% "=" and "," cannot be short step
    \fi
    \ifforest@nodewalk@areshortsteps
      \@escapeif{\expandafter\pgfkeysalso\expandafter{\the\forest@nodewalk@shortsteps@resolution}}%
    \else
      \@escapeif{\pgfkeysalso{/forest/\pgfkeyscurrentname={#1}}}%
    \fi
  },
}
\def\forest@nodewalk@shortsteps{%
  \futurelet\forest@nodewalk@nexttoken\forest@nodewalk@shortsteps@
}
\def\forest@nodewalk@shortsteps@{%
  \ifx\forest@nodewalk@nexttoken=%
    \let\forest@nodewalk@nextop\forest@nodewalk@shortsteps@end
  \else
    \ifx\forest@nodewalk@nexttoken\bgroup
      \letcs\forest@nodewalk@nextop{forest@shortstep@group}%
    \else
      \let\forest@nodewalk@nextop\forest@nodewalk@shortsteps@@
    \fi
  \fi
  \forest@nodewalk@nextop
}
\def\forest@nodewalk@shortsteps@@#1{%
  \ifcsdef{forest@shortstep@#1}{%
    \csname forest@shortstep@#1\endcsname
  }{%
    \forest@nodewalk@areshortstepsfalse
    \forest@nodewalk@shortsteps@end
  }%
}
% in the following definitions:
% #1 = short step
% #2 = (long) step, or a style in /forest/nodewalk (taking n args)
\csdef{forest@nodewalk@defshortstep@0@args}#1#2{%
  \csdef{forest@shortstep@#1}{%
    \apptotoks\forest@nodewalk@shortsteps@resolution{,#2}%
    \forest@nodewalk@shortsteps}}
\csdef{forest@nodewalk@defshortstep@1@args}#1#2{%
  \csdef{forest@shortstep@#1}##1{%
    \edef\forest@marshal####1{#2}%
    \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}}%
    \forest@nodewalk@shortsteps}}
\csdef{forest@nodewalk@defshortstep@2@args}#1#2{%
  \csdef{forest@shortstep@#1}##1##2{%
    \edef\forest@marshal####1####2{#2}%
    \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}}%
    \forest@nodewalk@shortsteps}}
\csdef{forest@nodewalk@defshortstep@3@args}#1#2{%
  \csdef{forest@shortstep@#1}##1##2##3{%
    \edef\forest@marshal####1####2####3{#2}%
    \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}}%
    \forest@nodewalk@shortsteps}}
\csdef{forest@nodewalk@defshortstep@4@args}#1#2{%
  \csdef{forest@shortstep@#1}##1##2##3##4{%
    \edef\forest@marshal####1####2####3####4{#2}%
    \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}}%
    \forest@nodewalk@shortsteps}}
\csdef{forest@nodewalk@defshortstep@5@args}#1#2{%
  \csdef{forest@shortstep@#1}##1##2##3##4##5{%
```

```
    \edef\forest@marshal####1####2####3####4####5{#2}%
    \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}}%
    \forest@nodewalk@shortsteps}}
\csdef{forest@nodewalk@defshortstep@6@args}#1#2{%
  \csdef{forest@shortstep@#1}##1##2##3##4##5##6{%
    \edef\forest@marshal####1####2####3####4####5####6{#2}%
    \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}}%
    \forest@nodewalk@shortsteps}}
\csdef{forest@nodewalk@defshortstep@7@args}#1#2{%
  \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7{%
    \edef\forest@marshal####1####2####3####4####5####6####7{#2}%
    \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}}%
    \forest@nodewalk@shortsteps}}
\csdef{forest@nodewalk@defshortstep@8@args}#1#2{%
  \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7##8{%
    \edef\forest@marshal####1####2####3####4####5####6####7####8{#2}%
    \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}{##8
    \forest@nodewalk@shortsteps}}
\csdef{forest@nodewalk@defshortstep@9@args}#1#2{%
  \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7##8##9{%
    \edef\forest@marshal####1####2####3####4####5####6####7####8####9{#2}%
    \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}{##8
    \forest@nodewalk@shortsteps}}
\forestset{
  define short step/.code n args=3{% #1 = short step, #2 = n args, #3 = long step
    \csname forest@nodewalk@defshortstep@#2@args\endcsname{#1}{#3}%
  },
}
\def\forest@nodewalk@shortsteps@end#1,{}
```

Define short-form steps.

```
\forestset{
  define short step={group}{1}{group={#1}}, % {braces} are special
  define short step={p}{0}{previous},
  define short step={n}{0}{next},
  define short step={u}{0}{parent},
  define short step={s}{0}{sibling},
  define short step={c}{0}{current},
  define short step={o}{0}{origin},
  define short step={r}{0}{root},
  define short step={R}{0}{root'},
  define short step={P}{0}{previous leaf},
  define short step={N}{0}{next leaf},
  define short step={F}{0}{first leaf},
  define short step={L}{0}{last leaf},
  define short step={>}{0}{next on tier},
  define short step={<}{0}{previous on tier},
  define short step={1}{0}{n=1},
  define short step={2}{0}{n=2},
  define short step={3}{0}{n=3},
  define short step={4}{0}{n=4},
  define short step={5}{0}{n=5},
  define short step={6}{0}{n=6},
  define short step={7}{0}{n=7},
  define short step={8}{0}{n=8},
  define short step={9}{0}{n=9},
  define short step={l}{0}{last},
  define short step={b}{0}{back},
  define short step={f}{0}{forward},
  define short step={v}{0}{last valid},
  define short step={*}{2}{repeat={#1}{#2}},
  for 1/.style={for nodewalk={n=1}{#1}},
  for 2/.style={for nodewalk={n=2}{#1}},
  for 3/.style={for nodewalk={n=3}{#1}},
  for 4/.style={for nodewalk={n=4}{#1}},
  for 5/.style={for nodewalk={n=5}{#1}},
  for 6/.style={for nodewalk={n=6}{#1}},
  for 7/.style={for nodewalk={n=7}{#1}},
  for 8/.style={for nodewalk={n=8}{#1}},
  for 9/.style={for nodewalk={n=9}{#1}},
  for -1/.style={for nodewalk={n’=1}{#1}},
  for -2/.style={for nodewalk={n’=2}{#1}},
  for -3/.style={for nodewalk={n’=3}{#1}},
  for -4/.style={for nodewalk={n’=4}{#1}},
  for -5/.style={for nodewalk={n’=5}{#1}},
  for -6/.style={for nodewalk={n’=6}{#1}},
  for -7/.style={for nodewalk={n’=7}{#1}},
  for -8/.style={for nodewalk={n’=8}{#1}},
  for -9/.style={for nodewalk={n’=9}{#1}},
}
```

Define multiple-step walks.

```
\forestset{
  define long step={tree}{}{\forest@node@foreach{\forest@nodewalk@makestep}},
  define long step={tree reversed}{}{\forest@node@foreach@reversed{\forest@nodewalk@makestep}},
  define long step={tree children-first}{}{\forest@node@foreach@childrenfirst{\forest@nodewalk@makestep}},
  define long step={tree children-first reversed}{}{\forest@node@foreach@childrenfirst@reversed{\forest@nodew
  define long step={tree breadth-first}{}{\forest@node@foreach@breadthfirst{-1}{\forest@nodewalk@makestep}},
  define long step={tree breadth-first reversed}{}{\forest@node@foreach@breadthfirst@reversed{-1}{\forest@nod
  define long step={descendants}{}{\forest@node@foreachdescendant{\forest@nodewalk@makestep}},
  define long step={descendants reversed}{}{\forest@node@foreachdescendant@reversed{\forest@nodewalk@makestep
  define long step={descendants children-first}{}{\forest@node@foreachdescendant@childrenfirst{\forest@nodewa
  define long step={descendants children-first reversed}{}{\forest@node@foreachdescendant@childrenfirst@rever
  define long step={descendants breadth-first}{}{\forest@node@foreach@breadthfirst{0}{\forest@nodewalk@makest
  define long step={descendants breadth-first reversed}{}{\forest@node@foreach@breadthfirst@reversed{0}{\fore
  define long step={level}{n args=1}{%
    \forestmathtruncatemacro\forest@temp{#1}%
    \edef\forest@marshal{%
      \noexpand\forest@node@foreach@breadthfirst
        {\forest@temp}%
        {\noexpand\ifnum\noexpand\forestove{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexp
    }\forest@marshal
  },
  define long step={level>}{n args=1}{%
    \forestmathtruncatemacro\forest@temp{#1}%
    \edef\forest@marshal{%
      \noexpand\forest@node@foreach@breadthfirst
        {-1}%
        {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
    }\forest@marshal
  },
  define long step={level<}{n args=1}{%
    \forestmathtruncatemacro\forest@temp{(#1)-1}%
    \ifnum\forest@temp=-1
      % special case, as \forest@node@foreach@breadthfirst uses level<0 as a signal for unlimited max level
      \ifnum\forestove{level}=0
        \forest@nodewalk@makestep
      \fi
    \else
      \edef\forest@marshal{%
        \noexpand\forest@node@foreach@breadthfirst
          {\forest@temp}%
          {\noexpand\forest@nodewalk@makestep}%
      }\forest@marshal
    \fi
  },
  define long step={level reversed}{n args=1}{%
    \forestmathtruncatemacro\forest@temp{#1}%
    \edef\forest@marshal{%
      \noexpand\forest@node@foreach@breadthfirst@reversed
        {\forest@temp}%
        {\noexpand\ifnum\noexpand\forestove{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexp
    }\forest@marshal
  },
  define long step={level reversed>}{n args=1}{%
    \forestmathtruncatemacro\forest@temp{#1}%
    \edef\forest@marshal{%
      \noexpand\forest@node@foreach@breadthfirst@reversed
        {-1}%
        {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
    }\forest@marshal
  },
  define long step={level reversed<}{n args=1}{%
    \forestmathtruncatemacro\forest@temp{(#1)-1}%
    \edef\forest@marshal{%
      \noexpand\forest@node@foreach@breadthfirst@reversed
        {\forest@temp}%
        {\noexpand\forest@nodewalk@makestep}%
    }\forest@marshal
  },
  %
  define long step={relative level}{n args=1}{%
    \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
    \edef\forest@marshal{%
      \noexpand\forest@node@foreach@breadthfirst
        {\forest@temp}%
        {\noexpand\ifnum\noexpand\forestove{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexp
    }\forest@marshal
  },
  define long step={relative level>}{n args=1}{%
    \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
    \edef\forest@marshal{%
      \noexpand\forest@node@foreach@breadthfirst
        {-1}%
        {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
    }\forest@marshal
  },
  define long step={relative level<}{n args=1}{%
    \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}-1}%
    \edef\forest@marshal{%
      \noexpand\forest@node@foreach@breadthfirst
        {\forest@temp}%
        {\noexpand\forest@nodewalk@makestep}%
    }\forest@marshal
  },
  define long step={relative level reversed}{n args=1}{%
    \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
    \edef\forest@marshal{%
      \noexpand\forest@node@foreach@breadthfirst@reversed
        {\forest@temp}%
        {\noexpand\ifnum\noexpand\forestove{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexp
    }\forest@marshal
  },
  define long step={relative level reversed>}{n args=1}{%
    \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
    \edef\forest@marshal{%
      \noexpand\forest@node@foreach@breadthfirst@reversed
        {-1}%
        {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
    }\forest@marshal
  },
  define long step={relative level reversed<}{n args=1}{%
    \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}-1}%
    \edef\forest@marshal{%
      \noexpand\forest@node@foreach@breadthfirst@reversed
        {\forest@temp}%
        {\noexpand\forest@nodewalk@makestep}%
    }\forest@marshal
  },
  define long step={leaves}{}{%
    \forest@node@foreach{%
      \ifnum\forestove{n children}=0
        \forest@nodewalk@makestep
      \fi
    }%
  },
  define long step={-level}{n args=1,style}{%
    unique={branch={leaves,{group={repeat={#1}{parent}}}}}
  },
  define long step={-level’}{n args=1,style}{%
    unique={on invalid={fake}{branch={leaves,{group={repeat={#1}{parent}}}}}}
  },
  define long step={children}{}{\forest@node@foreachchild{\forest@nodewalk@makestep}},
  define long step={children reversed}{}{\forest@node@foreachchild@reversed{\forest@nodewalk@makestep}},
  define long step={current and following siblings}{}{\forest@node@@forselfandfollowingsiblings{\forest@nodew
  define long step={following siblings}{style}{if nodewalk valid={next}{fake=next,current and following sibli
  define long step={current and preceding siblings}{}{\forest@node@@forselfandprecedingsiblings{\forest@nodew
  define long step={preceding siblings}{style}{if nodewalk valid={previous}{fake=previous,current and precedi
  define long step={current and following siblings reversed}{}{\forest@node@@forselfandfollowingsiblings@reve
  define long step={following siblings reversed}{style}{fake=next,current and following siblings reversed},
  define long step={current and preceding siblings reversed}{}{\forest@node@@forselfandprecedingsiblings@reve
  define long step={preceding siblings reversed}{style}{fake=previous,current and preceding siblings reversed
  define long step={siblings}{style}{for nodewalk’={preceding siblings},following siblings},
  define long step={siblings reversed}{style}{for nodewalk’={following siblings reversed},preceding siblings
  define long step={current and siblings}{style}{for nodewalk’={preceding siblings},current and following sib
  define long step={current and siblings reversed}{style}{for nodewalk’={current and following siblings rever
  define long step={ancestors}{style}{while={}{parent},last valid},
  define long step={current and ancestors}{style}{current,ancestors},
  define long step={following nodes}{style}{while={}{next node},last valid},
  define long step={preceding nodes}{style}{while={}{previous node},last valid},
  define long step={current and following nodes}{style}{current,following nodes},
  define long step={current and preceding nodes}{style}{current,preceding nodes},
}
\let\forest@nodewalkstephandler@styletrueorfalse\forest@nodewalkstephandler@styletrue
```

7.4 Dynamic tree

```
\def\forest@last@node{0}
\csdef{forest@nodewalk@saved@dynamic nodes}{}
\def\forest@nodehandleby@name@nodewalk@or@bracket#1{%
  \ifx\pgfkeysnovalue#1%
    \edef\forest@last@node{\forest@node@Nametoid{forest@last@node}}%
  \else
```

```
    \forest@nodehandleby@nnb@checkfirst#1\forest@END
  \fi
}
\def\forest@nodehandleby@nnb@checkfirst#1#2\forest@END{%
  \ifx[#1%]
    \forest@create@node{#1#2}%
    \cseappto{forest@nodewalk@saved@dynamic nodes}{\forest@last@node,}%
  \else
    \forest@forthis{%
      \forest@nameandgo{#1#2}%
      \ifnum\forest@cn=0
        \PackageError{forest}{Cannot use a dynamic key on the invalid node}{}%
      \fi
      \let\forest@last@node\forest@cn
    }%
  \fi
}
\def\forest@create@node#1{% #1=bracket representation
  \bracketParse{\forest@create@collectafterthought}%
                \forest@last@node=#1\forest@end@create@node
}
\def\forest@create@collectafterthought#1\forest@end@create@node{%
  \forest@node@Foreach{\forest@last@node}{%
    \forestoleto{delay}{given options}%
    \forestoset{given options}{}%
  }%
  \forestOeappto{\forest@last@node}{delay}{,\unexpanded{#1}}%
  \forestOset{\forest@last@node}{given options}{delay={}}%
}
\def\forest@create@node@and@process@given@options#1{% #1=bracket representation
  \bracketParse{\forest@createandprocess@collectafterthought}%
                \forest@last@node=#1\forest@end@create@node
}
\def\forest@createandprocess@collectafterthought#1\forest@end@create@node{%
  \forest@node@Compute@numeric@ts@info{\forest@last@node}%
  \forest@saveandrestoremacro\forest@root{%
    \let\forest@root\forest@last@node
    \forestset{process keylist=given options}%
  }%
}
\def\forest@saveandrestoremacro#1#2{% #1 = the (zero-arg) macro to save before and restore after processing c
  \edef\forest@marshal{%
    \unexpanded{#2}%
    \noexpand\def\noexpand#1{\expandonce{#1}}%
  }\forest@marshal
}
\def\forest@saveandrestoreifcs#1#2{% #1 = the if cs to save before and restore after processing code in #2
  \edef\forest@marshal{%
    \unexpanded{#2}%
    \ifbool{#1}{\noexpand\setbool{#1}{true}}{\noexpand\setbool{#1}{false}}%
  }\forest@marshal
}
\def\forest@globalsaveandrestoreifcs#1#2{% #1 = the if cs to save before and restore after processing code in
  \edef\forest@marshal{%
    \unexpanded{#2}%
    \ifbool{#1}{\global\noexpand\setbool{#1}{true}}{\global\noexpand\setbool{#1}{false}}%
  }\forest@marshal
}
\def\forest@saveandrestoretoks#1#2{% #1 = the toks to save before and restore after processing code in #2
  \edef\forest@marshal{%
    \unexpanded{#2}%
    \noexpand#1{\the#1}%
  }\forest@marshal
}
\def\forest@saveandrestoreregister#1#2{% #1 = the register to save before and restore after processing code i
  \edef\forest@marshal{%
    \unexpanded{#2}%
    \noexpand\forestrset{#1}{\forestregister{#1}}%
  }\forest@marshal
}
\forestset{
  save and restore register/.code 2 args={%
    \forest@saveandrestoreregister{#1}{%
      \pgfkeysalso{#2}%
    }%
  },
}
\def\forest@remove@node#1{%
  \ifforestdebugdynamics\forestdebug@dynamics{before removing #1}\fi
  \forest@node@Remove{#1}%
}
\def\forest@append@node#1#2{%
  \ifforestdebugdynamics\forestdebug@dynamics{before appending #2 to #1}\fi
  \forest@dynamic@circularitytest{#2}{#1}{append}%
  \forest@node@Remove{#2}%
  \forest@node@Append{#1}{#2}%
}
\def\forest@prepend@node#1#2{%
  \ifforestdebugdynamics\forestdebug@dynamics{before prepending #2 to #1}\fi
  \forest@dynamic@circularitytest{#2}{#1}{prepend}%
  \forest@node@Remove{#2}%
  \forest@node@Prepend{#1}{#2}%
}
\def\forest@insertafter@node#1#2{%
  \ifforestdebugdynamics\forestdebug@dynamics{before inserting #2 after #1}\fi
  \forest@node@Remove{#2}%
  \forest@node@Insertafter{\forestOve{#1}{@parent}}{#2}{#1}%
}
\def\forest@insertbefore@node#1#2{%
  \ifforestdebugdynamics\forestdebug@dynamics{before inserting #2 before #1}\fi
  \forest@node@Remove{#2}%
  \forest@node@Insertbefore{\forestOve{#1}{@parent}}{#2}{#1}%
}
\def\forest@set@root#1#2{%
  \ifforestdebugdynamics\forestdebug@dynamics{before setting #1 as root}\fi
  \def\forest@root{#2}%
}
\def\forest@dynamic@circularitytest#1#2#3{%
  % #1=potenitial ancestor,#2=potential descendant, #3=message prefix
  \ifnum#1=#2
    \forest@circularityerror{#1}{#2}{#3}%
  \else
    \forest@fornode{#1}{%
      \forest@ifancestorof{#2}{\forest@circularityerror{#1}{#2}{#3}}{}%
    }%
  \fi
}
\def\forest@circularityerror#1#2#3{%
  \forestdebug@typeouttrees{\forest@temp}%
  \PackageError{forest}{#3ing node id=#1 to id=#2 would result in a circular tree\MessageBreak forest of ids:
}%
\def\forestdebug@dynamics#1{%
```

```
  \forestdebug@typeouttrees\forest@temp
  \typeout{#1: \forest@temp}%
}
\def\forest@appto@do@dynamics#1#2{%
   \forest@nodehandleby@name@nodewalk@or@bracket{#2}%
   \ifcase\forest@dynamics@copyhow\relax\or
     \forest@tree@copy{\forest@last@node}\forest@last@node
   \or
     \forest@node@copy{\forest@last@node}\forest@last@node
   \fi
   \forest@node@Ifnamedefined{forest@last@node}{%
     \forestOepreto{\forest@last@node}{delay}
       {for id={\forest@node@Nametoid{forest@last@node}}{alias=forest@last@node},}%
     }{}%
   \edef\forest@marshal{%
     \noexpand\apptotoks\noexpand\forest@do@dynamics{%
       \noexpand#1{\forest@cn}{\forest@last@node}}%
   }\forest@marshal
}
\forestset{%
  create/.code={%
    \forest@create@node{#1}%
    \forest@fornode{\forest@last@node}{%
      \forest@node@setalias{forest@last@node}%
      \cseappto{forest@nodewalk@saved@dynamic nodes}{\forest@last@node,}%
    }%
  },
  create'/.code={%
    \forest@create@node@and@process@given@options{#1}%
    \forest@fornode{\forest@last@node}{%
      \forest@node@setalias{forest@last@node}%
      \cseappto{forest@nodewalk@saved@dynamic nodes}{\forest@last@node,}%
    }%
  },
  append/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@append@node{#1}},
  prepend/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@prepend@node{#1}},
  insert after/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@insertafter@node{#1}},
  insert before/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@insertbefore@node{#1}}
  append'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@append@node{#1}},
  prepend'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@prepend@node{#1}},
  insert after'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@insertafter@node{#1}},
  insert before'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@insertbefore@node{#1}
  append''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@append@node{#1}},
  prepend''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@prepend@node{#1}},
  insert after''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@insertafter@node{#1}}
  insert before''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@insertbefore@node{#1
  remove/.code={%
    \pgfkeysalso{alias=forest@last@node}%
    \cseappto{forest@nodewalk@saved@dynamic nodes}{\forest@cn,}%
    \expandafter\apptotoks\expandafter\forest@do@dynamics\expandafter{%
      \expandafter\forest@remove@node\expandafter{\forest@cn}}%
  },
  set root/.code={%
    \def\forest@dynamics@copyhow{0}%
    \forest@appto@do@dynamics\forest@set@root{#1}%
  },
  replace by/.code={\forest@replaceby@code{#1}{insert after}},
  replace by'/.code={\forest@replaceby@code{#1}{insert after'}},
  replace by''/.code={\forest@replaceby@code{#1}{insert after''}},
  sort/.code={%
    \eapptotoks\forest@do@dynamics{%
```

```
      \def\noexpand\forest@nodesort@sortkey{\forestrv{sort by}}%
      \noexpand\forest@nodesort\noexpand\forest@sort@ascending{\forest@cn}
    }%
  },
  sort'/.code={%
    \eapptotoks\forest@do@dynamics{%
      \def\noexpand\forest@nodesort@sortkey{\forestrv{sort by}}%
      \noexpand\forest@nodesort\noexpand\forest@sort@descending{\forest@cn}
    }%
  },
}
\def\forest@replaceby@code#1#2{%#1=node spec,#2=insert after['][']
  \ifnum\forestove{@parent}=0
    \cseappto{forest@nodewalk@saved@dynamic nodes}{\forest@cn,}%
    \pgfkeysalso{alias=forest@last@node,set root={#1}}%
  \else
    \cseappto{forest@nodewalk@saved@dynamic nodes}{\forest@cn,}%
    \pgfkeysalso{alias=forest@last@node,#2={#1}}%
    \eapptotoks\forest@do@dynamics{%
      \noexpand\ifnum\noexpand\forestOve{\forest@cn}{@parent}=\forestove{@parent}
        \noexpand\forest@remove@node{\forest@cn}%
      \noexpand\fi
    }%
  \fi
}
\def\forest@nodesort#1#2{% #1 = direction, #2 = parent node
  \ifforestdebugdynamics\forestdebug@dynamics{before sorting children of #2}\fi
    \forest@fornode{#2}{\forest@nodesort@#1}%
  \ifforestdebugdynamics\forestdebug@dynamics{after sorting children of #2}\fi
}
\def\forest@nodesort@#1{%
  % prepare the array of child ids
  \c@pgf@counta=0
  \forestoget{@first}\forest@nodesort@id
  \forest@loop
  \ifnum\forest@nodesort@id>0
    \advance\c@pgf@counta 1
    \csedef{forest@nodesort@\the\c@pgf@counta}{\forest@nodesort@id}%
    \forestOget{\forest@nodesort@id}{@next}\forest@nodesort@id
  \forest@repeat
  % sort
  \forestoget{n children}\forest@nodesort@n
  \forest@sort\forest@nodesort@cmpnodes\forest@nodesort@let#1{1}{\forest@nodesort@n}%
  % remove all children
  \forestoget{@first}\forest@nodesort@id
  \forest@loop
  \ifnum\forest@nodesort@id>0
    \forest@node@Remove{\forest@nodesort@id}%
    \forestoget{@first}\forest@nodesort@id
  \forest@repeat
  % insert the children in new order
  \c@pgf@counta=0
  \forest@loop
  \ifnum\c@pgf@counta<\forest@nodesort@n\relax
    \advance\c@pgf@counta 1
    \forest@node@append{\csname forest@nodesort@\the\c@pgf@counta\endcsname}%
  \forest@repeat
}
\def\forest@nodesort@cmpnodes#1#2{%
  \expandafter\forest@nodesort@cmpnodes@\forest@nodesort@sortkey,\forest@END{#1}{#2}%
}
```

```
\def\forest@nodesort@cmpnodes@#1,#2\forest@END#3#4{%
  % #1=process ins+arg for this dimension, #2=for next dimensions
  % #3, #4 = node ids
  {%
    \forest@fornode{\csname forest@nodesort@#3\endcsname}{%
      \forestmathsetmacro\forest@nodesort@resulta{#1}%
    }%
    \forest@fornode{\csname forest@nodesort@#4\endcsname}{%
      \forestmathsetmacro\forest@nodesort@resultb{#1}%
    }%
    \ifx\forestmathresulttype\forestmathtype@generic
      \forest@cmp@error{\forest@nodesort@resulta}{\forest@nodesort@resultb}%
    \fi
    \edef\forest@temp{%
      \noexpand\forest@nodesort@cmp
      {\expandonce{\forest@nodesort@resulta}}%
      {\expandonce{\forest@nodesort@resultb}}%
    }%
    \xdef\forest@global@temp{\forest@temp}%
  }%
  \if=\forest@global@temp
    \let\forest@next\forest@nodesort@cmpnodes@
  \else
    \let\forest@next\forest@nodesort@cmpnodes@finish
  \fi
  \ifstrempty{#2}{\let\forest@next\forest@nodesort@cmpnodes@finish}{}%
  \forest@next#2\forest@END{#3}{#4}%
}
\def\forest@nodesort@cmpnodes@finish#1\forest@END#2#3{%
  \let\forest@sort@cmp@result\forest@global@temp
}
```

Usage: `\forest@nodesort@cmp`⟨*first*⟩⟨*second*⟩. Fully expandable. Return `<`, `=` or `>`, as required by `\forest@sort`.

```
\def\forest@nodesort@cmp{\csname fRsT@nsc@\forestmathresulttype\endcsname}
\def\fRsT@nsc@#1{\csname fRsT@nsc@#1\endcsname}
\def\fRsT@nsc@n#1#2{\ifnum#1<#2 <\else\ifnum#1=#2 =\else>\fi\fi}
\def\fRsT@nsc@d#1#2{\ifdim#1<#2 <\else\ifdim#1=#2 =\else>\fi\fi}
\def\fRsT@nsc@P#1#2{\ifdim#1pt<#2pt <\else\ifdim#1pt=#2pt =\else>\fi\fi}
\def\fRsT@nsc@t#1#2{\csname fRsT@nsc@\pdfstrcmp{#1}{#2}\endcsname}
\def\fRsT@nsc@T#1#2{\csname fRsT@nsc@\pdfstrcmp{#2}{#1}\endcsname}
\csdef{fRsT@nsc@-1}{<}
\csdef{fRsT@nsc@0}{=}
\csdef{fRsT@nsc@1}{>}
\def\forest@nodesort@let#1#2{%
  \csletcs{forest@nodesort@#1}{forest@nodesort@#2}%
}
\forestset{
  define long step={last dynamic node}{style,must start at valid node=false}{%
    name=forest@last@node
  }
}
```

8 Stages

```
\def\forest@root{0}
  %%% begin listing region: stages
\forestset{
  stages/.style={
    for root'={
      process keylist register=default preamble,
      process keylist register=preamble
    },
    process keylist=given options,
    process keylist=before typesetting nodes,
    typeset nodes stage,
    process keylist=before packing,
    pack stage,
    process keylist=before computing xy,
    compute xy stage,
    process keylist=before drawing tree,
    draw tree stage
  },
  typeset nodes stage/.style={for root'=typeset nodes},
  pack stage/.style={for root'=pack},
  compute xy stage/.style={for root'=compute xy},
  draw tree stage/.style={for root'=draw tree},
}
  %%% end listing region: stages
\forestset{
  process keylist/.code={%
    \forest@process@hook@keylist{#1}{#1 processing order/.try,processing order/.lastretry}},
  process keylist'/.code 2 args={\forest@process@hook@keylist@nodynamics{#1}{#2}},
  process keylist''/.code 2 args={\forest@process@hook@keylist@{#1}{#2}},
  process keylist register/.code={\forest@process@keylist@register{#1}},
  process delayed/.code={%
    \forest@havedelayedoptions{@delay}{#1}%
    \ifforest@havedelayedoptions
      \forest@process@hook@keylist@nodynamics{@delay}{#1}%
    \fi
  },
  do dynamics/.code={%
    \the\forest@do@dynamics
    \forest@do@dynamics{}%
    \forest@node@Compute@numeric@ts@info{\forest@root}%
  },
  declare keylist={given options}{},
  declare keylist={before typesetting nodes}{},
  declare keylist={before packing}{},
  declare keylist={before packing node}{},
  declare keylist={after packing node}{},
  declare keylist={before computing xy}{},
  declare keylist={before drawing tree}{},
  declare keylist={delay}{},
  delay n/.code 2 args={%
    \forestmathsetcount\forest@temp@count{#1}%
    \pgfkeysalso{delay n'={\forest@temp@count}{#2}}%
  },
  delay n'/.code 2 args={
    \ifnum#1=0
      \pgfkeysalso{#2}%
    \else
      \pgfkeysalso{delay={delay n'/.expand once=\expandafter{\number\numexpr#1-1\relax}{#2}}}%
    \fi
  },
  if have delayed/.style 2 args={if have delayed'={processing order}{#1}{#2}},
  if have delayed'/.code n args=3{%
    \forest@havedelayedoptionsfalse
    \forest@forthis{%
      \forest@nodewalk{#1}{%
        TeX={%
          \forestoget{delay}\forest@temp@delayed
          \ifdefempty\forest@temp@delayed{}{\forest@havedelayedoptionstrue}%
        }%
      }%
    }%
    \ifforest@havedelayedoptions\pgfkeysalso{#2}\else\pgfkeysalso{#3}\fi
  },
  typeset nodes/.code={%
    \forest@drawtree@preservenodeboxes@false
    \forest@nodewalk
      {typeset nodes processing order/.try,processing order/.lastretry}%
      {TeX={\forest@node@typeset}}%
    },
  typeset nodes'/.code={%
    \forest@drawtree@preservenodeboxes@true
    \forest@nodewalk
      {typeset nodes processing order/.try,processing order/.lastretry}%
      {TeX={\forest@node@typeset}}%
    },
  typeset node/.code={%
    \forest@drawtree@preservenodeboxes@false
    \forest@node@typeset
  },
  pack/.code={\forest@pack},
  pack'/.code={\forest@pack@onlythisnode},
  compute xy/.code={\forest@node@computeabsolutepositions},
  draw tree box/.store in=\forest@drawtreebox,
  draw tree box,
  draw tree/.code={%
    \forest@drawtree@preservenodeboxes@false
    \forest@node@drawtree
  },
  draw tree'/.code={%
    \forest@drawtree@preservenodeboxes@true
    \forest@node@drawtree
  },
  %%% begin listing region: draw_tree_method
  draw tree method/.style={
    for nodewalk={
      draw tree nodes processing order/.try,
      draw tree processing order/.retry,
      processing order/.lastretry
    }{draw tree node},
    for nodewalk={
      draw tree edges processing order/.try,
      draw tree processing order/.retry,
      processing order/.lastretry
    }{draw tree edge},
    for nodewalk={
      draw tree tikz processing order/.try,
      draw tree processing order/.retry,
      processing order/.lastretry
    }{draw tree tikz}
  },
  %%% end listing region: draw_tree_method
  draw tree node/.code={\forest@draw@node},
  draw tree node'/.code={\forest@draw@node@},
  if node drawn/.code n args={3}{%
    \forest@forthis{%
      \forest@configured@nodewalk{independent}{inherited}{fake}{#1}{}%
      \ifnum\forest@cn=0
        \forest@tempfalse
      \else
        \ifcsdef{forest@drawn@\forest@cn}{\forest@temptrue}{\forest@tempfalse}%
      \fi
    }%
    \ifforest@temp\pgfkeysalso{#2}\else\pgfkeysalso{#3}\fi
  },
  draw tree edge/.code={\forest@draw@edge},
  draw tree edge'/.code={\forest@draw@edge@},
  draw tree tikz/.code={\forest@draw@tikz@}, % always!
  draw tree tikz'/.code={\forest@draw@tikz@},
  processing order/.nodewalk style={tree},
  %given options processing order/.style={processing order},
  %before typesetting nodes processing order/.style={processing order},
  %before packing processing order/.style={processing order},
  %before computing xy processing order/.style={processing order},
  %before drawing tree processing order/.style={processing order},
}
\newtoks\forest@do@dynamics
\newif\ifforest@havedelayedoptions
\def\forest@process@hook@keylist#1#2{%,#1=keylist,#2=processing order nodewalk
  \safeloop
    \forest@fornode{\forest@root}{\forest@process@hook@keylist@{#1}{#2}}%
    \expandafter\ifstrempty\expandafter{\the\forest@do@dynamics}{}{%
      \the\forest@do@dynamics
      \forest@do@dynamics={}%
      \forest@node@Compute@numeric@ts@info{\forest@root}%
    }%
  \forest@fornode{\forest@root}{\forest@havedelayedoptions{#1}{#2}}%
  \ifforest@havedelayedoptions
  \saferepeat
}
\def\forest@process@hook@keylist@nodynamics#1#2{%#1=keylist,#2=processing order nodewalk
  % note: this macro works on (nodewalk starting at) the current node
  \safeloop
    \forest@forthis{\forest@process@hook@keylist@{#1}{#2}}%
  \forest@havedelayedoptions{#1}{#2}%
  \ifforest@havedelayedoptions
  \saferepeat
}
\def\forest@process@hook@keylist@#1#2{%#1=keylist,#2=processing order nodewalk
  \forest@nodewalk{#2}{%
    TeX={%
      \forestoget{#1}\forest@temp@keys
      \ifdefvoid\forest@temp@keys{}{%
        \forestoset{#1}{}%
        \expandafter\forestset\expandafter{\forest@temp@keys}%
      }%
    }%
  }%
}
\def\forest@process@keylist@register#1{%
  \edef\forest@marshal{%
    \noexpand\forestset{\forestregister{#1}}%
  }\forest@marshal
}
```

Clear the keylist, transfer delayed into it, and set `\ifforest@havedelayedoptions`.

```
\def\forest@havedelayedoptions#1#2{%#1 = keylist, #2=nodewalk
  \forest@havedelayedoptionsfalse
  \forest@forthis{%
    \forest@nodewalk{#2}{%
```

```
      TeX={%
        \forestoget{delay}\forest@temp@delayed
        \ifdefempty\forest@temp@delayed{}{\forest@havedelayedoptionstrue}%
        \forestolet{#1}\forest@temp@delayed
        \forestoset{delay}{}%
      }%
    }%
  }%
}
```

8.1 Typesetting nodes

```
\def\forest@node@typeset{%
  \let\forest@next\forest@node@typeset@
  \forestoifdefined{@box}{%
    \forestoget{@box}\forest@temp
    \ifdefempty\forest@temp{%
      \locbox\forest@temp@box
      \forestolet{@box}\forest@temp@box
    }{%
      \ifforest@drawtree@preservenodeboxes@
        \let\forest@next\relax
      \fi
    }%
  }{%
    \locbox\forest@temp@box
    \forestolet{@box}\forest@temp@box
  }%
  \def\forest@node@typeset@restore{}%
  \ifdefined\ifsa@tikz\forest@standalone@hack\fi
  \forest@next
  \forest@node@typeset@restore
}
\def\forest@standalone@hack{%
  \ifsa@tikz
    \let\forest@standalone@tikzpicture\tikzpicture
    \let\forest@standalone@endtikzpicture\endtikzpicture
    \let\tikzpicture\sa@orig@tikzpicture
    \let\endtikzpicture\sa@orig@endtikzpicture
    \def\forest@node@typeset@restore{%
      \let\tikzpicture\forest@standalone@tikzpicture
      \let\endtikzpicture\forest@standalone@endtikzpicture
    }%
  \fi
}
\newbox\forest@box
\def\forest@pgf@notyetpositioned{not yet positionedPGFINTERNAL}
\def\forest@node@typeset@{%
  \forestanchortotikzanchor{anchor}\forest@temp
  \edef\forest@marshal{%
    \noexpand\forestolet{anchor}\noexpand\forest@temp
    \noexpand\forest@node@typeset@@
    \noexpand\forestoset{anchor}{\forestov{anchor}}%
  }\forest@marshal
}
\def\forest@node@typeset@@{%
  \forestoget{name}\forest@nodename
  \edef\forest@temp@nodeformat{\forestove{node format}}%
  \gdef\forest@smuggle{}%
  \setbox0=\hbox{%
    \begin{tikzpicture}[%
      /forest/copy command key={/tikz/anchor}{/tikz/forest@orig@anchor},
      anchor/.style={%
        /forest/TeX={\forestanchortotikzanchor{##1}\forest@temp@anchor},
        forest@orig@anchor/.expand once=\forest@temp@anchor
      }]
      \pgfpositionnodelater{\forest@positionnodelater@save}%
      \forest@temp@nodeformat
      \pgfinterruptpath
      \pgfpointanchor{\forest@pgf@notyetpositioned\forest@nodename}{forestcomputenodeboundary}%
      \endpgfinterruptpath
    \end{tikzpicture}%
  }%
  \setbox\forestove{@box}=\box\forest@box % smuggle the box
  \forestolet{@boundary}\forest@global@boundary
  \forest@smuggle % ... and the rest
}


\forestset{
  declare readonly dimen={min x}{0pt},
  declare readonly dimen={min y}{0pt},
  declare readonly dimen={max x}{0pt},
  declare readonly dimen={max y}{0pt},
}
\def\forest@patch@enormouscoordinateboxbounds@plus#1{%
  \expandafter\ifstrequal\expandafter{#1}{16000.0pt}{\edef#1{0.0\pgfmath@pt}}{}%
}
\def\forest@patch@enormouscoordinateboxbounds@minus#1{%
  \expandafter\ifstrequal\expandafter{#1}{-16000.0pt}{\edef#1{0.0\pgfmath@pt}}{}%
}
\def\forest@positionnodelater@save{%
  \global\setbox\forest@box=\box\pgfpositionnodelaterbox
  \xappto\forest@smuggle{\noexpand\forestoset{later@name}{\pgfpositionnodelatername}}%
  % a bug in pgf? ---well, here's a patch
  \forest@patch@enormouscoordinateboxbounds@plus\pgfpositionnodelaterminx
  \forest@patch@enormouscoordinateboxbounds@plus\pgfpositionnodelaterminy
  \forest@patch@enormouscoordinateboxbounds@minus\pgfpositionnodelatermaxx
  \forest@patch@enormouscoordinateboxbounds@minus\pgfpositionnodelatermaxy
  % end of patch
  % when shape=coordinate, pgf returns \pgfpositionnodelater... with wrong catcode of pt
  \forest@pnn@addtosmuggle{min x}\pgfpositionnodelaterminx
  \forest@pnn@addtosmuggle{max x}\pgfpositionnodelatermaxx
  \forest@pnn@addtosmuggle{min y}\pgfpositionnodelaterminy
  \forest@pnn@addtosmuggle{max y}\pgfpositionnodelatermaxy
}
\def\forest@pnn@addtosmuggle#1#2{%
  {%
    \pgfutil@tempdima=#2\relax
    \xappto\forest@smuggle{\noexpand\forestoset{#1}{\the\pgfutil@tempdima}}%
  }%
}
\def\forest@node@forest@positionnodelater@restore{%
  \ifforest@drawtree@preservenodeboxes@
    \let\forest@boxorcopy\copy
  \else
    \let\forest@boxorcopy\box
  \fi
  \forestoget{@box}\forest@temp
  \setbox\pgfpositionnodelaterbox=\forest@boxorcopy\forest@temp
  \edef\pgfpositionnodelatername{\forestove{later@name}}%
  \edef\pgfpositionnodelaterminx{\forestove{min x}}%
  \edef\pgfpositionnodelaterminy{\forestove{min y}}%
  \edef\pgfpositionnodelatermaxx{\forestove{max x}}%
  \edef\pgfpositionnodelatermaxy{\forestove{max y}}%
  \ifforest@drawtree@preservenodeboxes@
  \else
    \forestoset{@box}{}%
  \fi
}
```

8.2 Packing

Method `pack` should be called to calculate the positions of descendant nodes; the positions are stored in attributes `l` and `s` of these nodes, in a level/sibling coordinate system with origin at the parent's anchor.

```
\def\forest@pack{%
  \pgfsyssoftpath@getcurrentpath\forest@pack@original@path
  \forest@pack@computetiers
  \forest@pack@computegrowthuniformity
  \forest@@pack
  \pgfsyssoftpath@setcurrentpath\forest@pack@original@path
}
\def\forest@@pack{%
  \ifnum\forestove{uniform growth}>0
    \ifnum\forestove{n children}>0
      \forest@pack@level@uniform
      \forest@pack@aligntiers@ofsubtree
      \forest@pack@sibling@uniform@recursive
    \fi
  \else
    \forest@node@foreachchild{\forest@@pack}%
    \forest@process@hook@keylist@nodynamics{before packing node}{current}%
    \ifnum\forestove{n children}>0
      \forest@pack@level@nonuniform
      \forest@pack@aligntiers
      \forest@pack@sibling@uniform@applyreversed
    \fi
    \forestoget{after packing node}\forest@temp@keys
    \forest@process@hook@keylist@nodynamics{after packing node}{current}%
  \fi
}
% \forestset{recalculate tree boundary/.code={\forest@node@recalculate@edges}}
% \def\forest@node@recalculate@edges{%
%   \edef\forest@marshal{%
%     \noexpand\forest@forthis{\noexpand\forest@node@getedges{\forestove{grow}}}%
%   }\forest@marshal
% }
\def\forest@pack@onlythisnode{%
  \ifnum\forestove{n children}>0
    \forest@pack@computetiers
    \forest@pack@level@nonuniform
    \forest@pack@aligntiers
    \forest@node@foreachchild{\forestoset{s}{0\pgfmath@pt}}%
    \forest@pack@sibling@uniform@applyreversed
  \fi
}
```

Compute growth uniformity for the subtree. A tree grows uniformly is all its branching nodes have the same `grow`.

```
\def\forest@pack@computegrowthuniformity{%
  \forest@node@foreachchild{\forest@pack@computegrowthuniformity}%
  \edef\forest@pack@cgu@uniformity{%
    \ifnum\forestove{n children}=0
    2\else 1\fi
  }%
  \forestoget{grow}\forest@pack@cgu@parentgrow
  \forest@node@foreachchild{%
    \ifnum\forestove{uniform growth}=0
      \def\forest@pack@cgu@uniformity{0}%
    \else
      \ifnum\forestove{uniform growth}=1
        \ifnum\forestove{grow}=\forest@pack@cgu@parentgrow\relax\else
          \def\forest@pack@cgu@uniformity{0}%
        \fi
      \fi
    \fi
  }%
  \forestoget{before packing node}\forest@temp@a
  \forestoget{after packing node}\forest@temp@b
  \expandafter\expandafter\expandafter\ifstrempty\expandafter\expandafter\expandafter{\expandafter\forest@tem
    \forestolet{uniform growth}\forest@pack@cgu@uniformity
  }{%
    \forestoset{uniform growth}{0}%
  }%
}
```

Pack children in the level dimension in a uniform tree.

```
\def\forest@pack@level@uniform{%
  \let\forest@plu@minchildl\relax
  \forestoget{grow}\forest@plu@grow
  \forest@node@foreachchild{%
    \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
    \advance\pgf@xa\forestove{l}\relax
    \ifx\forest@plu@minchildl\relax
      \edef\forest@plu@minchildl{\the\pgf@xa}%
    \else
      \ifdim\pgf@xa<\forest@plu@minchildl\relax
        \edef\forest@plu@minchildl{\the\pgf@xa}%
      \fi
    \fi
  }%
  \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
  \pgfutil@tempdima=\pgf@xb\relax
  \advance\pgfutil@tempdima -\forest@plu@minchildl\relax
  \advance\pgfutil@tempdima \forestove{l sep}\relax
  \ifdim\pgfutil@tempdima>0pt
    \forest@node@foreachchild{%
      \forestoeset{l}{\the\dimexpr\forestove{l}+\the\pgfutil@tempdima}%
    }%
  \fi
  \forest@node@foreachchild{%
    \ifnum\forestove{n children}>0
      \forest@pack@level@uniform
    \fi
  }%
}
```

Pack children in the level dimension in a non-uniform tree. (Expects the children to be fully packed.)

```
\def\forest@pack@level@nonuniform{%
  \let\forest@plu@minchildl\relax
  \forestoget{grow}\forest@plu@grow
  \forest@node@foreachchild{%
    \forest@node@getedge{negative}{\forest@plu@grow}{\forest@plnu@negativechildedge}%
    \forest@node@getedge{positive}{\forest@plu@grow}{\forest@plnu@positivechildedge}%
    \def\forest@plnu@childedge{\forest@plnu@negativechildedge\forest@plnu@positivechildedge}%
    \forest@path@getboundingrectangle@ls\forest@plnu@childedge{\forest@plu@grow}%
    \advance\pgf@xa\forestove{l}\relax
    \ifx\forest@plu@minchildl\relax
      \edef\forest@plu@minchildl{\the\pgf@xa}%
    \else
      \ifdim\pgf@xa<\forest@plu@minchildl\relax
        \edef\forest@plu@minchildl{\the\pgf@xa}%
      \fi
    \fi
  }%
  \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
  \pgfutil@tempdima=\pgf@xb\relax
  \advance\pgfutil@tempdima -\forest@plu@minchildl\relax
  \advance\pgfutil@tempdima \forestove{l sep}\relax
  \ifdim\pgfutil@tempdima>0pt
    \forest@node@foreachchild{%
      \forestoeset{l}{\the\dimexpr\the\pgfutil@tempdima+\forestove{l}}%
    }%
  \fi
}
```

Align tiers.

```
\def\forest@pack@aligntiers{%
  \forestoget{grow}\forest@temp@parentgrow
  \forestoget{@tiers}\forest@temp@tiers
  \forlistloop\forest@pack@aligntier@\forest@temp@tiers
}
\def\forest@pack@aligntiers@ofsubtree{%
  \forest@node@foreach{\forest@pack@aligntiers}%
}
\def\forest@pack@aligntiers@computeabsl{%
  \forestoleto{abs@l}{l}%
  \forest@node@foreachdescendant{\forest@pack@aligntiers@computeabsl@}%
}
\def\forest@pack@aligntiers@computeabsl@{%
  \forestoeset{abs@l}{\the\dimexpr\forestove{l}+\forestOve{\forestove{@parent}}{abs@l}}%
}
\def\forest@pack@aligntier@#1{%
  \forest@pack@aligntiers@computeabsl
  \pgfutil@tempdima=-\maxdimen\relax
  \def\forest@temp@currenttier{#1}%
  \forest@node@foreach{%
    \forestoget{tier}\forest@temp@tier
    \ifx\forest@temp@currenttier\forest@temp@tier
      \ifdim\pgfutil@tempdima<\forestove{abs@l}\relax
        \pgfutil@tempdima=\forestove{abs@l}\relax
      \fi
    \fi
  }%
  \ifdim\pgfutil@tempdima=-\maxdimen\relax\else
    \forest@node@foreach{%
      \forestoget{tier}\forest@temp@tier
      \ifx\forest@temp@currenttier\forest@temp@tier
        \forestoeset{l}{\the\dimexpr\pgfutil@tempdima-\forestove{abs@l}+\forestove{l}}%
      \fi
    }%
  \fi
}
```

Pack children in the sibling dimension in a uniform tree: recursion.

```
\def\forest@pack@sibling@uniform@recursive{%
  \forest@node@foreachchild{\forest@pack@sibling@uniform@recursive}%
  \forest@pack@sibling@uniform@applyreversed
}
```

Pack children in the sibling dimension in a uniform tree: applyreversed.

```
\def\forest@pack@sibling@uniform@applyreversed{%
  \ifnum\forestove{n children}>1
    \ifnum\forestove{reversed}=0
      \forest@pack@sibling@uniform@main{first}{last}{next}{previous}%
    \else
      \forest@pack@sibling@uniform@main{last}{first}{previous}{next}%
    \fi
  \else
    \ifnum\forestove{n children}=1
```

No need to run packing, but we still need to align the children.

```
      \csname forest@calign@\forestove{calign}\endcsname
    \fi
  \fi
}
```

Pack children in the sibling dimension in a uniform tree: the main routine.

```
\def\forest@pack@sibling@uniform@main#1#2#3#4{%
```

Loop through the children. At each iteration, we compute the distance between the negative edge of the current child and the positive edge of the block of the previous children, and then set the `s` attribute of the current child accordingly.

We start the loop with the second (to last) child, having initialized the positive edge of the previous children to the positive edge of the first child.

```
  \forestoget{@#1}\forest@child
  \edef\forest@marshal{%
    \noexpand\forest@fornode{\forestove{@#1}}{%
      \noexpand\forest@node@getedge
        {positive}%
        {\forestove{grow}}%
        \noexpand\forest@temp@edge
    }%
  }\forest@marshal
  \forest@pack@pgfpoint@childsposition\forest@child
  \let\forest@previous@positive@edge\pgfutil@empty
  \forest@extendpath\forest@previous@positive@edge\forest@temp@edge{}%
  \forestOget{\forest@child}{@#3}\forest@child
```

Loop until the current child is the null node.

```
  \edef\forest@previous@child@s{0\pgfmath@pt}%
  \safeloop
  \unless\ifnum\forest@child=0
```

Get the negative edge of the child.

```
    \edef\forest@temp{%
      \noexpand\forest@fornode{\forest@child}{%
        \noexpand\forest@node@getedge
          {negative}%
          {\forestove{grow}}%
          \noexpand\forest@temp@edge
        }%
    }\forest@temp
```

Set `\pgf@x` and `\pgf@y` to the position of the child (in the coordinate system of this node).

```
    \forest@pack@pgfpoint@childsposition\forest@child
```

Translate the edge of the child by the child's position.

```
    \let\forest@child@negative@edge\pgfutil@empty
    \forest@extendpath\forest@child@negative@edge\forest@temp@edge{}%
```

Setup the grow line: the angle is given by this node's `grow` attribute.

```
    \forest@setupgrowline{\forestove{grow}}%
```

Get the distance (wrt the grow line) between the positive edge of the previous children and the negative edge of the current child. (The distance can be negative!)

```
    \forest@distance@between@edge@paths\forest@previous@positive@edge\forest@child@negative@edge\forest@csdis
```

If the distance is `\relax`, the projections of the edges onto the grow line don't overlap: do nothing. Otherwise, shift the current child so that its distance to the block of previous children is `s_sep`.

```
    \ifx\forest@csdistance\relax
      %\forestOeset{\forest@child}{s}{\forest@previous@child@s}%
    \else
      \advance\pgfutil@tempdimb-\forest@csdistance\relax
      \advance\pgfutil@tempdimb\forestove{s sep}\relax
      \forestOeset{\forest@child}{s}{\the\dimexpr\forestOve{\forest@child}{s}-\forest@csdistance+\forestove{s
    \fi
```

Retain monotonicity (is this ok?). (This problem arises when the adjacent children's `l` are too far apart.)

```
    \ifdim\forestOve{\forest@child}{s}<\forest@previous@child@s\relax
      \forestOeset{\forest@child}{s}{\forest@previous@child@s}%
    \fi
```

Prepare for the next iteration: add the current child's positive edge to the positive edge of the previous children, and set up the next current child.

```
    \forestOget{\forest@child}{s}\forest@child@s
    \edef\forest@previous@child@s{\forest@child@s}%
    \edef\forest@temp{%
      \noexpand\forest@fornode{\forest@child}{%
        \noexpand\forest@node@getedge
          {positive}%
          {\forestove{grow}}%
          \noexpand\forest@temp@edge
      }%
    }\forest@temp
    \forest@pack@pgfpoint@childsposition\forest@child
    \forest@extendpath\forest@previous@positive@edge\forest@temp@edge{}%
    \forest@getpositivetightedgeofpath\forest@previous@positive@edge\forest@previous@positive@edge
    \forestOget{\forest@child}{@#3}\forest@child
  \saferepeat
```

Shift the position of all children to achieve the desired alignment of the parent and its children.

```
  \csname forest@calign@\forestove{calign}\endcsname
}
```

Get the position of child `#1` in the current node, in node's l-s coordinate system.

```
\def\forest@pack@pgfpoint@childsposition#1{%
  {%
    \pgftransformreset
    \forest@pgfqtransformrotate{\forestove{grow}}%
    \forest@fornode{#1}{%
      \pgfpointtransformed{\pgfqpoint{\forestove{l}}{\forestove{s}}}%
    }%
  }%
}
```

Get the position of the node in the grow (`#1`)-rotated coordinate system.

```
\def\forest@pack@pgfpoint@positioningrow#1{%
  {%
    \pgftransformreset
    \forest@pgfqtransformrotate{#1}%
    \pgfpointtransformed{\pgfqpoint{\forestove{l}}{\forestove{s}}}%
  }%
}
```

Child alignment.

```
\def\forest@calign@s@shift#1{%
  \pgfutil@tempdima=#1\relax
  \forest@node@foreachchild{%
    \forestoeset{s}{\the\dimexpr\forestove{s}+\pgfutil@tempdima}%
  }%
}
\def\forest@calign@child{%
  \forest@calign@s@shift{-\forestOve{\forest@node@nornbarthchildid{\forestove{calign primary child}}}{s}}%
}
\csdef{forest@calign@child edge}{%
  {%
    \edef\forest@temp@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
    \pgftransformreset
    \forest@pgfqtransformrotate{\forestove{grow}}%
    \pgfpointtransformed{\pgfqpoint{\forestOve{\forest@temp@child}{l}}{\forestOve{\forest@temp@child}{s}}}%
    \pgf@xa=\pgf@x\relax\pgf@ya=\pgf@y\relax
    \forest@Pointanchor{\forest@temp@child}{child anchor}%
    \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
    \forest@pointanchor{parent anchor}%
    \advance\pgf@xa-\pgf@x\relax\advance\pgf@ya-\pgf@y\relax
    \edef\forest@marshal{%
      \noexpand\pgftransformreset
      \noexpand\forest@pgfqtransformrotate{-\forestove{grow}}%
      \noexpand\pgfpointtransformed{\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}%
    }\forest@marshal
  }%
  \forest@calign@s@shift{\the\dimexpr-\the\pgf@y}%
}
\csdef{forest@calign@midpoint}{%
  \forest@calign@s@shift{\the\dimexpr 0pt -%
    (\forestOve{\forest@node@nornbarthchildid{\forestove{calign primary child}}}{s}%
    +\forestOve{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}{s}%
    )/2\relax
  }%
}
\csdef{forest@calign@edge midpoint}{%
  {%
    \edef\forest@temp@firstchild{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
    \edef\forest@temp@secondchild{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
    \pgftransformreset
    \forest@pgfqtransformrotate{\forestove{grow}}%
    \pgfpointtransformed{\pgfqpoint{\forestOve{\forest@temp@firstchild}{l}}{\forestOve{\forest@temp@firstchil
    \pgf@xa=\pgf@x\relax\pgf@ya=\pgf@y\relax
    \forest@Pointanchor{\forest@temp@firstchild}{child anchor}%
    \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
    \edef\forest@marshal{%
      \noexpand\pgfpointtransformed{\noexpand\pgfqpoint{\forestOve{\forest@temp@secondchild}{l}}{\forestOve{\
    }\forest@marshal
    \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
    \forest@Pointanchor{\forest@temp@secondchild}{child anchor}%
    \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
    \divide\pgf@xa2 \divide\pgf@ya2
    \forest@pointanchor{parent anchor}%
    \advance\pgf@xa-\pgf@x\relax\advance\pgf@ya-\pgf@y\relax
    \edef\forest@marshal{%
      \noexpand\pgftransformreset
      \noexpand\forest@pgfqtransformrotate{-\forestove{grow}}%
      \noexpand\pgfpointtransformed{\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}%
    }\forest@marshal
  }%
  \forest@calign@s@shift{\the\dimexpr-\the\pgf@y}%
}
```

Aligns the children to the center of the angles given by the options `calign_first_angle` and `calign_second_angle` and spreads them additionally if needed to fill the whole space determined by the option. The version `fixed_angles` calculates the angles between node anchors; the version `fixes_edge_angles` calculates the angles between the node edges.

```
\def\forest@edef@strippt#1#2{%
  \edef#1{#2}%
  \edef#1{\expandafter\Pgf@geT#1}%
}
\csdef{forest@calign@fixed angles}{%
  \ifnum\forestove{n children}>1
    \edef\forest@ca@first@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
    \edef\forest@ca@second@child{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
    \ifnum\forestove{reversed}=1
      \let\forest@temp\forest@ca@first@child
      \let\forest@ca@first@child\forest@ca@second@child
      \let\forest@ca@second@child\forest@temp
    \fi
    \forestOget{\forest@ca@first@child}{l}\forest@ca@first@l
    \edef\forest@ca@first@l{\expandafter\Pgf@geT\forest@ca@first@l}%
    \forestOget{\forest@ca@second@child}{l}\forest@ca@second@l
    \edef\forest@ca@second@l{\expandafter\Pgf@geT\forest@ca@second@l}%
    \pgfmathtan@{\forestove{calign secondary angle}}%
    \pgfmathmultiply@{\pgfmathresult}{\forest@ca@second@l}%
    \let\forest@calign@temp\pgfmathresult
    \pgfmathtan@{\forestove{calign primary angle}}%
    \pgfmathmultiply@{\pgfmathresult}{\forest@ca@first@l}%
    \edef\forest@ca@desired@s@distance{\the\dimexpr
      \forest@calign@temp pt-\pgfmathresult pt}%
    % \pgfmathsetlengthmacro\forest@ca@desired@s@distance{%
    %   tan(\forestove{calign secondary angle})*\forest@ca@second@l
    %   -tan(\forestove{calign primary angle})*\forest@ca@first@l
    % }%
    \forestOget{\forest@ca@first@child}{s}\forest@ca@first@s
    \forestOget{\forest@ca@second@child}{s}\forest@ca@second@s
    \edef\forest@ca@actual@s@distance{\the\dimexpr
      \forest@ca@second@s-\forest@ca@first@s}%
    %\pgfmathsetlengthmacro\forest@ca@actual@s@distance{%
    %  \forest@ca@second@s-\forest@ca@first@s}%
    \ifdim\forest@ca@desired@s@distance>\forest@ca@actual@s@distance\relax
      \ifdim\forest@ca@actual@s@distance=0pt
        \pgfmathtan@{\forestove{calign primary angle}}%
        \pgfmathmultiply@{\pgfmathresult}{\forest@ca@second@l}%
        \pgfutil@tempdima=\pgfmathresult pt
        % \pgfmathsetlength\pgfutil@tempdima{tan(\forestove{calign primary angle})*\forest@ca@second@l}%
        \pgfutil@tempdimb=\dimexpr
          \forest@ca@desired@s@distance/(\forestove{n children}-1)\relax%
        %\pgfmathsetlength\pgfutil@tempdimb{\forest@ca@desired@s@distance/(\forestove{n children}-1)}%
        \forest@node@foreachchild{%
          \forestoeset{s}{\the\pgfutil@tempdima}%
          \advance\pgfutil@tempdima\pgfutil@tempdimb
        }%
        \def\forest@calign@anchor{0pt}%
      \else
        \edef\forest@marshal{\noexpand\pgfmathdivide@
          {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
          {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
        }\forest@marshal
        \let\forest@ca@ratio\pgfmathresult
        %\pgfmathsetmacro\forest@ca@ratio{%
        %  \forest@ca@desired@s@distance/\forest@ca@actual@s@distance}%
        \forest@node@foreachchild{%
          \forest@edef@strippt\forest@temp{\forestove{s}}%
          \pgfmathmultiply@{\forest@ca@ratio}{\forest@temp}%
          \forestoeset{s}{\the\dimexpr\pgfmathresult pt}%
          %\pgfmathsetlengthmacro\forest@temp{\forest@ca@ratio*\forestove{s}}%
          %\forestolet{s}\forest@temp
        }%
        \pgfmathtan@{\forestove{calign primary angle}}%
        \pgfmathmultiply@{\pgfmathresult}{\forest@ca@first@l}%
        \edef\forest@calign@anchor{\the\dimexpr-\pgfmathresult pt}%
        %\pgfmathsetlengthmacro\forest@calign@anchor{%
        %  -tan(\forestove{calign primary angle})*\forest@ca@first@l}%
      \fi
    \else
      \ifdim\forest@ca@desired@s@distance<\forest@ca@actual@s@distance\relax
        \edef\forest@marshal{\noexpand\pgfmathdivide@
          {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
          {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
        }\forest@marshal
        \let\forest@ca@ratio\pgfmathresult
        %\pgfmathsetlengthmacro\forest@ca@ratio{%
        %  \forest@ca@actual@s@distance/\forest@ca@desired@s@distance}%
        \forest@node@foreachchild{%
          \forest@edef@strippt\forest@temp{\forestove{l}}%
          \pgfmathmultiply@{\forest@ca@ratio}{\forest@temp}%
          \forestoeset{l}{\the\dimexpr\pgfmathresult pt}%
          %\pgfmathsetlengthmacro\forest@temp{\forest@ca@ratio*\forestove{l}}%
          %\forestolet{l}\forest@temp
        }%
        \forestOget{\forest@ca@first@child}{l}\forest@ca@first@l
        \edef\forest@ca@first@l{\expandafter\Pgf@geT\forest@ca@first@l}%
        \pgfmathtan@{\forestove{calign primary angle}}%
        \pgfmathmultiply@{\pgfmathresult}{\forest@ca@first@l}%
        \edef\forest@calign@anchor{\the\dimexpr-\pgfmathresult pt}%
        %\pgfmathsetlengthmacro\forest@calign@anchor{%
        %  -tan(\forestove{calign primary angle})*\forest@ca@first@l}%
      \fi
    \fi
    \forest@calign@s@shift{-\forest@calign@anchor}%
  \fi
}
\csdef{forest@calign@fixed edge angles}{%
  \ifnum\forestove{n children}>1
    \edef\forest@ca@first@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
    \edef\forest@ca@second@child{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
    \ifnum\forestove{reversed}=1
      \let\forest@temp\forest@ca@first@child
      \let\forest@ca@first@child\forest@ca@second@child
      \let\forest@ca@second@child\forest@temp
    \fi
    \forestOget{\forest@ca@first@child}{l}\forest@ca@first@l
    \forestOget{\forest@ca@second@child}{l}\forest@ca@second@l
    \forest@pointanchor{parent anchor}%
    \edef\forest@ca@parent@anchor@s{\the\pgf@x}%
    \edef\forest@ca@parent@anchor@l{\the\pgf@y}%
    \forest@Pointanchor{\forest@ca@first@child}{child anchor}%
    \edef\forest@ca@first@child@anchor@s{\the\pgf@x}%
    \edef\forest@ca@first@child@anchor@l{\the\pgf@y}%
    \forest@Pointanchor{\forest@ca@second@child}{child anchor}%
    \edef\forest@ca@second@child@anchor@s{\the\pgf@x}%
    \edef\forest@ca@second@child@anchor@l{\the\pgf@y}%
    \pgfmathtan@{\forestove{calign secondary angle}}%
    \edef\forest@temp{\the\dimexpr
      \forest@ca@second@l-\forest@ca@second@child@anchor@l+\forest@ca@parent@anchor@l}%
    \pgfmathmultiply@{\pgfmathresult}{\expandafter\Pgf@geT\forest@temp}%
    \edef\forest@ca@desired@second@edge@s{\the\dimexpr\pgfmathresult pt}%
    %\pgfmathsetlengthmacro\forest@ca@desired@second@edge@s{%
    %  tan(\forestove{calign secondary angle})*%
    %  (\forest@ca@second@l-\forest@ca@second@child@anchor@l+\forest@ca@parent@anchor@l)}%
    \pgfmathtan@{\forestove{calign primary angle}}%
    \edef\forest@temp{\the\dimexpr
      \forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@l}%
    \pgfmathmultiply@{\pgfmathresult}{\expandafter\Pgf@geT\forest@temp}%
    \edef\forest@ca@desired@first@edge@s{\the\dimexpr\pgfmathresult pt}%
    %\pgfmathsetlengthmacro\forest@ca@desired@first@edge@s{%
    %  tan(\forestove{calign primary angle})*%
    %  (\forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@l)}%
    \edef\forest@ca@desired@s@distance{\the\dimexpr
      \forest@ca@desired@second@edge@s-\forest@ca@desired@first@edge@s}%
    %\pgfmathsetlengthmacro\forest@ca@desired@s@distance{\forest@ca@desired@second@edge@s-\forest@ca@desired@
    \forestOget{\forest@ca@first@child}{s}\forest@ca@first@s
    \forestOget{\forest@ca@second@child}{s}\forest@ca@second@s
    \edef\forest@ca@actual@s@distance{\the\dimexpr
      \forest@ca@second@s+\forest@ca@second@child@anchor@s
      -\forest@ca@first@s-\forest@ca@first@child@anchor@s}%
    %\pgfmathsetlengthmacro\forest@ca@actual@s@distance{%
    %  \forest@ca@second@s+\forest@ca@second@child@anchor@s
    %  -\forest@ca@first@s-\forest@ca@first@child@anchor@s}%
    \ifdim\forest@ca@desired@s@distance>\forest@ca@actual@s@distance\relax
      \ifdim\forest@ca@actual@s@distance=0pt
        \forestoget{n children}\forest@temp@n@children
        \forest@node@foreachchild{%
          \forest@pointanchor{child anchor}%
          \edef\forest@temp@child@anchor@s{\the\pgf@x}%
          \forestoeset{s}{\the\dimexpr
            \forest@ca@desired@first@edge@s+\forest@ca@desired@s@distance*(\forestove{n}-1)/(\forest@temp@n@c
          %\pgfmathsetlengthmacro\forest@temp{%
          %  \forest@ca@desired@first@edge@s+(\forestove{n}-1)*\forest@ca@desired@s@distance/(\forest@temp@n@
          %\forestolet{s}\forest@temp
        }%
        \def\forest@calign@anchor{0pt}%
      \else
        \edef\forest@marshal{\noexpand\pgfmathdivide@
          {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
          {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
        }\forest@marshal
        \let\forest@ca@ratio\pgfmathresult
        %\pgfmathsetmacro\forest@ca@ratio{%
        %  \forest@ca@desired@s@distance/\forest@ca@actual@s@distance}%
        \forest@node@foreachchild{%
          \forest@pointanchor{child anchor}%
          \edef\forest@temp@child@anchor@s{\the\pgf@x}%
          \edef\forest@marshal{\noexpand\pgfmathmultiply@
            {\forest@ca@ratio}%
            {\expandafter\Pgf@geT\the\dimexpr
              \forestove{s}-\forest@ca@first@s+%
              \forest@temp@child@anchor@s-\forest@ca@first@child@anchor@s}%
          }\forest@marshal
          \forestoeset{s}{\the\dimexpr\pgfmathresult pt+\forest@ca@first@s
            +\forest@ca@first@child@anchor@s-\forest@temp@child@anchor@s}%
          % \pgfmathsetlengthmacro\forest@temp{%
          %  \forest@ca@ratio*(%
          %   \forestove{s}-\forest@ca@first@s
          %   +\forest@temp@child@anchor@s-\forest@ca@first@child@anchor@s)%
          %  +\forest@ca@first@s
          %  +\forest@ca@first@child@anchor@s-\forest@temp@child@anchor@s}%
          % \forestolet{s}\forest@temp
         }%
        \pgfmathtan@{\forestove{calign primary angle}}%
        \edef\forest@marshal{\noexpand\pgfmathmultiply@
          {\pgfmathresult}%
          {\expandafter\Pgf@geT\the\dimexpr
            \forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@l}%
        }\forest@marshal
        \edef\forest@calign@anchor{\the\dimexpr
          -\pgfmathresult pt+\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s}%
        % \pgfmathsetlengthmacro\forest@calign@anchor{%
        %  -tan(\forestove{calign primary angle})*(\forest@ca@first@l-\forest@ca@first@child@anchor@l+\fores
        %  +\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s
        % }%
       \fi
    \else
      \ifdim\forest@ca@desired@s@distance<\forest@ca@actual@s@distance\relax
        \edef\forest@marshal{\noexpand\pgfmathdivide@
          {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
          {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
        }\forest@marshal
        \let\forest@ca@ratio\pgfmathresult
        %\pgfmathsetlengthmacro\forest@ca@ratio{%
        %  \forest@ca@actual@s@distance/\forest@ca@desired@s@distance}%
        \forest@node@foreachchild{%
          \forest@pointanchor{child anchor}%
          \edef\forest@temp@child@anchor@l{\the\pgf@y}%
          \edef\forest@marshal{\noexpand\pgfmathmultiply@
            {\forest@ca@ratio}%
            {\expandafter\Pgf@geT\the\dimexpr\forestove{l}+\forest@ca@parent@anchor@l-\forest@temp@child@anch
          }\forest@marshal
          \forestoeset{l}{\the\dimexpr
            \pgfmathresult pt-\forest@ca@parent@anchor@l+\forest@temp@child@anchor@l}%
          % \pgfmathsetlengthmacro\forest@temp{%
          %  \forest@ca@ratio*(%
          %   \forestove{l}+\forest@ca@parent@anchor@l-\forest@temp@child@anchor@l)
          %   -\forest@ca@parent@anchor@l+\forest@temp@child@anchor@l}%
          % \forestolet{l}\forest@temp
        }%
        \forestOget{\forest@ca@first@child}{l}\forest@ca@first@l
        \pgfmathtan@{\forestove{calign primary angle}}%
        \edef\forest@marshal{\noexpand\pgfmathmultiply@
          {\pgfmathresult}%
          {\expandafter\Pgf@geT\the\dimexpr
            \forest@ca@first@l+\forest@ca@parent@anchor@l-\forest@temp@child@anchor@l}%
        }\forest@marshal
        \edef\forest@calign@anchor{\the\dimexpr
          -\pgfmathresult pt+\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s}%
        % \pgfmathsetlengthmacro\forest@calign@anchor{%
        %  -tan(\forestove{calign primary angle})*(\forest@ca@first@l+\forest@ca@parent@anchor@l-\forest@tem
        %  +\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s
        % }%
      \fi
    \fi
```

```
    \forest@calign@s@shift{-\forest@calign@anchor}%
  \fi
}
```

Get edge: `#1` = `positive/negative`, `#2` = grow (in degrees), `#3` = the control sequence receiving the resulting path. The edge is taken from the cache (attribute `#1@edge@#2`) if possible; otherwise, both positive and negative edge are computed and stored in the cache.

```
\def\forest@node@getedge#1#2#3{%
  \forestoget{#1@edge@#2}#3%
  \ifx#3\relax
    \forest@node@foreachchild{%
      \forest@node@getedge{#1}{#2}{\forest@temp@edge}%
    }%
    \forest@forthis{\forest@node@getedges{#2}}%
    \forestoget{#1@edge@#2}#3%
  \fi
}
```

Get edges. `#1` = grow (in degrees). The result is stored in attributes `negative@edge@#1` and `positive@edge@#1`. This method expects that the children's edges are already cached.

```
\def\forest@node@getedges#1{%
```

Run the computation in a TeX group.

```
  %{%
```

Setup the grow line.

```
    \forest@setupgrowline{#1}%
```

Get the edge of the node itself.

```
    \ifnum\forestove{ignore}=0
      \forestoget{@boundary}\forest@node@boundary
    \else
      \def\forest@node@boundary{}%
    \fi
    \csname forest@getboth\forestove{fit}edgesofpath\endcsname
        \forest@node@boundary\forest@negative@node@edge\forest@positive@node@edge
    \forestolet{negative@edge@#1}\forest@negative@node@edge
    \forestolet{positive@edge@#1}\forest@positive@node@edge
```

Add the edges of the children.

```
    \forest@get@edges@merge{negative}{#1}%
    \forest@get@edges@merge{positive}{#1}%
  %}%
}
```

Merge the `#1` (=`negative` or `positive`) edge of the node with `#1` edges of the children. `#2` = grow angle.

```
\def\forest@get@edges@merge#1#2{%
  \ifnum\forestove{n children}>0
    \forestoget{#1@edge@#2}\forest@node@edge
```

Remember the node's `parent anchor` and add it to the path (for breaking).

```
    \forest@pointanchor{parent anchor}%
    \edef\forest@getedge@pa@l{\the\pgf@x}%
    \edef\forest@getedge@pa@s{\the\pgf@y}%
    \eappto\forest@node@edge{\noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@pa@l}{\forest@getedge@pa@s}
```

Switch to this node's `(l,s)` coordinate system (origin at the node's anchor).

```
    \pgfgettransform\forest@temp@transform
    \pgftransformreset
    \forest@pgfqtransformrotate{\forestove{grow}}%
```

Get the child's (cached) edge, translate it by the child's position, and add it to the path holding all edges. Also add the edge from parent to the child to the path. This gets complicated when the child and/or parent anchor is empty, i.e. automatic border: we can get self-intersecting paths. So we store all the parent–child edges to a safe place first, compute all the possible breaking points (i.e. all the points in node@edge path), and break the parent–child edges on these points.

```
    \def\forest@all@edges{}%
    \forest@node@foreachchild{%
      \forestoget{#1@edge@#2}\forest@temp@edge
      \pgfpointtransformed{\pgfqpoint{\forestove{l}}{\forestove{s}}}%
      \forest@extendpath\forest@node@edge\forest@temp@edge{}%
      \ifnum\forestove{ignore edge}=0
        \pgfpointadd
          {\pgfpointtransformed{\pgfqpoint{\forestove{l}}{\forestove{s}}}}%
          {\forest@pointanchor{child anchor}}%
        \pgfgetlastxy{\forest@getedge@ca@l}{\forest@getedge@ca@s}%
        \eappto\forest@all@edges{%
          \noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@pa@l}{\forest@getedge@pa@s}%
          \noexpand\pgfsyssoftpath@linetotoken{\forest@getedge@ca@l}{\forest@getedge@ca@s}%
        }%
        % this deals with potential overlap of the edges:
        \eappto\forest@node@edge{\noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@ca@l}{\forest@getedge@c
      \fi
    }%
    \ifdefempty{\forest@all@edges}{}{%
      \pgfintersectionofpaths{\pgfsetpath\forest@all@edges}{\pgfsetpath\forest@node@edge}%
      \def\forest@edgenode@intersections{}%
      \forest@merge@intersectionloop
      \eappto\forest@node@edge{\expandonce{\forest@all@edges}\expandonce{\forest@edgenode@intersections}}%
    }%
    \pgfsettransform\forest@temp@transform
```

Process the path into an edge and store the edge.

```
    \csname forest@get#1\forestove{fit}edgeofpath\endcsname\forest@node@edge\forest@node@edge
    \forestolet{#1@edge@#2}\forest@node@edge
  \fi
}
%\newloop\forest@merge@loop
\def\forest@merge@intersectionloop{%
  \c@pgf@counta=0
  \forest@loop
  \ifnum\c@pgf@counta<\pgfintersectionsolutions\relax
    \advance\c@pgf@counta1
    \pgfpointintersectionsolution{\the\c@pgf@counta}%
    \eappto\forest@edgenode@intersections{\noexpand\pgfsyssoftpath@movetotoken
      {\the\pgf@x}{\the\pgf@y}}%
  \forest@repeat
}
```

Get the bounding rectangle of the node (without descendants). `#1` = grow.

```
\def\forest@node@getboundingrectangle@ls#1{%
  \forestoget{@boundary}\forest@node@boundary
  \forest@path@getboundingrectangle@ls\forest@node@boundary{#1}%
}
```

Applies the current coordinate transformation to the points in the path `#1`. Returns via the current path (so that the coordinate transformation can be set up as local).

```
\def\forest@pgfpathtransformed#1{%
  \forest@save@pgfsyssoftpath@tokendefs
  \let\pgfsyssoftpath@movetotoken\forest@pgfpathtransformed@moveto
  \let\pgfsyssoftpath@linetotoken\forest@pgfpathtransformed@lineto
  \pgfsyssoftpath@setcurrentpath\pgfutil@empty
  #1%
  \forest@restore@pgfsyssoftpath@tokendefs
}
```

```
\def\forest@pgfpathtransformed@moveto#1#2{%
  \forest@pgfpathtransformed@op\pgfsyssoftpath@moveto{#1}{#2}%
}
\def\forest@pgfpathtransformed@lineto#1#2{%
  \forest@pgfpathtransformed@op\pgfsyssoftpath@lineto{#1}{#2}%
}
\def\forest@pgfpathtransformed@op#1#2#3{%
  \pgfpointtransformed{\pgfqpoint{#2}{#3}}%
  \edef\forest@temp{%
    \noexpand#1{\the\pgf@x}{\the\pgf@y}%
  }%
  \forest@temp
}
```

8.2.1 Tiers

Compute tiers to be aligned at a node. The result in saved in attribute `@tiers`.

```
\def\forest@pack@computetiers{%
  {%
    \forest@pack@tiers@getalltiersinsubtree
    \forest@pack@tiers@computetierhierarchy
    \forest@pack@tiers@findcontainers
    \forest@pack@tiers@raisecontainers
    \forest@pack@tiers@computeprocessingorder
    \gdef\forest@smuggle{}%
    \forest@pack@tiers@write
  }%
  \forest@node@foreach{\forestoset{@tiers}{}}%
  \forest@smuggle
}
```

Puts all tiers contained in the subtree into attribute `tiers`.

```
\def\forest@pack@tiers@getalltiersinsubtree{%
  \ifnum\forestove{n children}>0
    \forest@node@foreachchild{\forest@pack@tiers@getalltiersinsubtree}%
  \fi
  \forestoget{tier}\forest@temp@mytier
  \def\forest@temp@mytiers{}%
  \ifdefempty\forest@temp@mytier{}{%
    \listeadd\forest@temp@mytiers\forest@temp@mytier
  }%
  \ifnum\forestove{n children}>0
    \forest@node@foreachchild{%
      \forestoget{tiers}\forest@temp@tiers
      \forlistloop\forest@pack@tiers@forhandlerA\forest@temp@tiers
    }%
  \fi
  \forestolet{tiers}\forest@temp@mytiers
}
\def\forest@pack@tiers@forhandlerA#1{%
  \ifinlist{#1}\forest@temp@mytiers{}{%
    \listeadd\forest@temp@mytiers{#1}%
  }%
}
```

Compute a set of higher and lower tiers for each tier. Tier A is higher than tier B iff a node on tier A is an ancestor of a node on tier B.

```
\def\forest@pack@tiers@computetierhierarchy{%
  \def\forest@tiers@ancestors{}%
  \forestoget{tiers}\forest@temp@mytiers
  \forlistloop\forest@pack@tiers@cth@init\forest@temp@mytiers
  \forest@pack@tiers@computetierhierarchy@
}
\def\forest@pack@tiers@cth@init#1{%
  \csdef{forest@tiers@higher@#1}{}%
  \csdef{forest@tiers@lower@#1}{}%
}
\def\forest@pack@tiers@computetierhierarchy@{%
  \forestoget{tier}\forest@temp@mytier
  \ifdefempty\forest@temp@mytier{}{%
    \forlistloop\forest@pack@tiers@forhandlerB\forest@tiers@ancestors
    \listeadd\forest@tiers@ancestors\forest@temp@mytier
  }%
  \forest@node@foreachchild{%
    \forest@pack@tiers@computetierhierarchy@
  }%
  \forestoget{tier}\forest@temp@mytier
  \ifdefempty\forest@temp@mytier{}{%
    \forest@listedel\forest@tiers@ancestors\forest@temp@mytier
  }%
}
\def\forest@pack@tiers@forhandlerB#1{%
  \def\forest@temp@tier{#1}%
  \ifx\forest@temp@tier\forest@temp@mytier
    \PackageError{forest}{Circular tier hierarchy (tier \forest@temp@mytier)}{}%
  \fi
  \ifinlistcs{#1}{forest@tiers@higher@\forest@temp@mytier}{}{%
    \listcsadd{forest@tiers@higher@\forest@temp@mytier}{#1}}%
  \xifinlistcs\forest@temp@mytier{forest@tiers@lower@#1}{}{%
    \listcseadd{forest@tiers@lower@#1}{\forest@temp@mytier}}%
}
\def\forest@pack@tiers@findcontainers{%
  \forestoget{tiers}\forest@temp@tiers
  \forlistloop\forest@pack@tiers@findcontainer\forest@temp@tiers
}
\def\forest@pack@tiers@findcontainer#1{%
  \def\forest@temp@tier{#1}%
  \forestoget{tier}\forest@temp@mytier
  \ifx\forest@temp@tier\forest@temp@mytier
    \csedef{forest@tiers@container@#1}{\forest@cn}%
  \else\@escapeif{%
    \forest@pack@tiers@findcontainerA{#1}%
  }\fi%
}
\def\forest@pack@tiers@findcontainerA#1{%
  \c@pgf@counta=0
  \forest@node@foreachchild{%
    \forestoget{tiers}\forest@temp@tiers
    \ifinlist{#1}\forest@temp@tiers{%
      \advance\c@pgf@counta 1
      \let\forest@temp@child\forest@cn
    }{}%
  }%
  \ifnum\c@pgf@counta>1
    \csedef{forest@tiers@container@#1}{\forest@cn}%
  \else\@escapeif{% surely =1
    \forest@fornode{\forest@temp@child}{%
      \forest@pack@tiers@findcontainer{#1}%
    }%
  }\fi
}
\def\forest@pack@tiers@raisecontainers{%
```

```
  \forestoget{tiers}\forest@temp@mytiers
  \forlistloop\forest@pack@tiers@rc@forhandlerA\forest@temp@mytiers
}
\def\forest@pack@tiers@rc@forhandlerA#1{%
  \edef\forest@tiers@temptier{#1}%
  \letcs\forest@tiers@containernodeoftier{forest@tiers@container@#1}%
  \letcs\forest@temp@lowertiers{forest@tiers@lower@#1}%
  \forlistloop\forest@pack@tiers@rc@forhandlerB\forest@temp@lowertiers
}
\def\forest@pack@tiers@rc@forhandlerB#1{%
  \letcs\forest@tiers@containernodeoflowertier{forest@tiers@container@#1}%
  \forestOget{\forest@tiers@containernodeoflowertier}{content}\lowercontent
  \forestOget{\forest@tiers@containernodeoftier}{content}\uppercontent
  \forest@fornode{\forest@tiers@containernodeoflowertier}{%
    \forest@ifancestorof
      {\forest@tiers@containernodeoftier}
      {\csletcs{forest@tiers@container@\forest@tiers@temptier}{forest@tiers@container@#1}}%
      {}%
  }%
}
\def\forest@pack@tiers@computeprocessingorder{%
  \def\forest@tiers@processingorder{}%
  \forestoget{tiers}\forest@tiers@cpo@tierstodo
  \safeloop
    \ifdefempty\forest@tiers@cpo@tierstodo{\forest@tempfalse}{\forest@temptrue}%
  \ifforest@temp
    \def\forest@tiers@cpo@tiersremaining{}%
    \def\forest@tiers@cpo@tiersindependent{}%
    \forlistloop\forest@pack@tiers@cpo@forhandlerA\forest@tiers@cpo@tierstodo
    \ifdefempty\forest@tiers@cpo@tiersindependent{%
      \PackageError{forest}{Circular tiers!}{}}{}%
    \forlistloop\forest@pack@tiers@cpo@forhandlerB\forest@tiers@cpo@tiersremaining
    \let\forest@tiers@cpo@tierstodo\forest@tiers@cpo@tiersremaining
  \saferepeat
}
\def\forest@pack@tiers@cpo@forhandlerA#1{%
  \ifcsempty{forest@tiers@higher@#1}{%
    \listadd\forest@tiers@cpo@tiersindependent{#1}%
    \listadd\forest@tiers@processingorder{#1}%
  }{%
    \listadd\forest@tiers@cpo@tiersremaining{#1}%
  }%
}
\def\forest@pack@tiers@cpo@forhandlerB#1{%
  \def\forest@pack@tiers@cpo@aremainingtier{#1}%
  \forlistloop\forest@pack@tiers@cpo@forhandlerC\forest@tiers@cpo@tiersindependent
}
\def\forest@pack@tiers@cpo@forhandlerC#1{%
  \ifinlistcs{#1}{forest@tiers@higher@\forest@pack@tiers@cpo@aremainingtier}{%
    \forest@listcsdel{forest@tiers@higher@\forest@pack@tiers@cpo@aremainingtier}{#1}%
  }{}%
}
\def\forest@pack@tiers@write{%
  \forlistloop\forest@pack@tiers@write@forhandler\forest@tiers@processingorder
}
\def\forest@pack@tiers@write@forhandler#1{%
  \forest@fornode{\csname forest@tiers@container@#1\endcsname}{%
    \forest@pack@tiers@check{#1}%
  }%
  \xappto\forest@smuggle{%
    \noexpand\listadd
```

```
      \forestOm{\csname forest@tiers@container@#1\endcsname}{@tiers}%
      {#1}%
  }%
}
 % checks if the tier is compatible with growth changes and calign=node/edge angle
\def\forest@pack@tiers@check#1{%
  \def\forest@temp@currenttier{#1}%
  \forest@node@foreachdescendant{%
    \ifnum\forestove{grow}=\forestOve{\forestove{@parent}}{grow}
    \else
      \forest@pack@tiers@check@grow
    \fi
    \ifnum\forestove{n children}>1
      \forestoget{calign}\forest@temp
      \ifx\forest@temp\forest@pack@tiers@check@nodeangle
        \forest@pack@tiers@check@calign
      \fi
      \ifx\forest@temp\forest@pack@tiers@check@edgeangle
        \forest@pack@tiers@check@calign
      \fi
    \fi
  }%
}
\def\forest@pack@tiers@check@nodeangle{node angle}%
\def\forest@pack@tiers@check@edgeangle{edge angle}%
\def\forest@pack@tiers@check@grow{%
  \forestoget{content}\forest@temp@content
  \let\forest@temp@currentnode\forest@cn
  \forest@node@foreachdescendant{%
    \forestoget{tier}\forest@temp
    \ifx\forest@temp@currenttier\forest@temp
      \forest@pack@tiers@check@grow@error
    \fi
  }%
}
\def\forest@pack@tiers@check@grow@error{%
  \PackageError{forest}{Tree growth direction changes in node \forest@temp@currentnode\space
    (content: \forest@temp@content), while tier '\forest@temp' is specified for nodes both
    out- and inside the subtree rooted in node \forest@temp@currentnode.  This will not work.}{}%
}
\def\forest@pack@tiers@check@calign{%
  \forest@node@foreachchild{%
    \forestoget{tier}\forest@temp
    \ifx\forest@temp@currenttier\forest@temp
      \forest@pack@tiers@check@calign@warning
    \fi
  }%
}
\def\forest@pack@tiers@check@calign@warning{%
  \PackageWarning{forest}{Potential option conflict: node \forestove{@parent} (content:
    '\forestOve{\forestove{@parent}}{content}') was given 'calign=\forestove{calign}', while its
    child \forest@cn\space (content: '\forestove{content}') was given 'tier=\forestove{tier}'.
    The parent's 'calign' will only work if the child was the lowest node on its tier before the
    alignment.}%
}
```

8.2.2 Node boundary

Compute the node boundary: it will be put in the pgf's current path. The computation is done within a generic anchor so that the shape's saved anchors and macros are available.

```
\pgfdeclaregenericanchor{forestcomputenodeboundary}{%
  \letcs\forest@temp@boundary@macro{forest@compute@node@boundary@#1}%
  \ifcsname forest@compute@node@boundary@#1\endcsname
    \csname forest@compute@node@boundary@#1\endcsname
  \else
    \forest@compute@node@boundary@rectangle
  \fi
  \pgfsyssoftpath@getcurrentpath\forest@temp
  \global\let\forest@global@boundary\forest@temp
}
\def\forest@mt#1{%
  \expandafter\pgfpointanchor\expandafter{\pgfreferencednodename}{#1}%
  \pgfsyssoftpath@moveto{\the\pgf@x}{\the\pgf@y}%
}%
\def\forest@lt#1{%
  \expandafter\pgfpointanchor\expandafter{\pgfreferencednodename}{#1}%
  \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
}%
\def\forest@compute@node@boundary@coordinate{%
  \forest@mt{center}%
}
\def\forest@compute@node@boundary@circle{%
  \forest@mt{east}%
  \forest@lt{north east}%
  \forest@lt{north}%
  \forest@lt{north west}%
  \forest@lt{west}%
  \forest@lt{south west}%
  \forest@lt{south}%
  \forest@lt{south east}%
  \forest@lt{east}%
}
\def\forest@compute@node@boundary@rectangle{%
  \forest@mt{south west}%
  \forest@lt{south east}%
  \forest@lt{north east}%
  \forest@lt{north west}%
  \forest@lt{south west}%
}
\def\forest@compute@node@boundary@diamond{%
  \forest@mt{east}%
  \forest@lt{north}%
  \forest@lt{west}%
  \forest@lt{south}%
  \forest@lt{east}%
}
\let\forest@compute@node@boundary@ellipse\forest@compute@node@boundary@circle
\def\forest@compute@node@boundary@trapezium{%
  \forest@mt{top right corner}%
  \forest@lt{top left corner}%
  \forest@lt{bottom left corner}%
  \forest@lt{bottom right corner}%
  \forest@lt{top right corner}%
}
\def\forest@compute@node@boundary@semicircle{%
  \forest@mt{arc start}%
  \forest@lt{north}%
  \forest@lt{east}%
  \forest@lt{north east}%
  \forest@lt{apex}%
  \forest@lt{north west}%
  \forest@lt{west}%
  \forest@lt{arc end}%
  \forest@lt{arc start}%
}
%\newloop\forest@computenodeboundary@loop
\csdef{forest@compute@node@boundary@regular polygon}{%
  \forest@mt{corner 1}%
  \c@pgf@counta=\sides\relax
  \forest@loop
  \ifnum\c@pgf@counta>0
    \forest@lt{corner \the\c@pgf@counta}%
    \advance\c@pgf@counta-1
  \forest@repeat
}%
\def\forest@compute@node@boundary@star{%
  \forest@mt{outer point 1}%
  \c@pgf@counta=\totalstarpoints\relax
  \divide\c@pgf@counta2
  \forest@loop
  \ifnum\c@pgf@counta>0
    \forest@lt{inner point \the\c@pgf@counta}%
    \forest@lt{outer point \the\c@pgf@counta}%
    \advance\c@pgf@counta-1
  \forest@repeat
}%
\csdef{forest@compute@node@boundary@isosceles triangle}{%
  \forest@mt{apex}%
  \forest@lt{left corner}%
  \forest@lt{right corner}%
  \forest@lt{apex}%
}
\def\forest@compute@node@boundary@kite{%
  \forest@mt{upper vertex}%
  \forest@lt{left vertex}%
  \forest@lt{lower vertex}%
  \forest@lt{right vertex}%
  \forest@lt{upper vertex}%
}
\def\forest@compute@node@boundary@dart{%
  \forest@mt{tip}%
  \forest@lt{left tail}%
  \forest@lt{tail center}%
  \forest@lt{right tail}%
  \forest@lt{tip}%
}
\csdef{forest@compute@node@boundary@circular sector}{%
  \forest@mt{sector center}%
  \forest@lt{arc start}%
  \forest@lt{arc center}%
  \forest@lt{arc end}%
  \forest@lt{sector center}%
}
\def\forest@compute@node@boundary@cylinder{%
  \forest@mt{top}%
  \forest@lt{after top}%
  \forest@lt{before bottom}%
  \forest@lt{bottom}%
  \forest@lt{after bottom}%
  \forest@lt{before top}%
  \forest@lt{top}%
}
```

```
\cslet{forest@compute@node@boundary@forbidden sign}\forest@compute@node@boundary@circle
\cslet{forest@compute@node@boundary@magnifying glass}\forest@compute@node@boundary@circle
\def\forest@compute@node@boundary@cloud{%
  \getradii
  \forest@mt{puff 1}%
  \c@pgf@counta=\puffs\relax
  \forest@loop
  \ifnum\c@pgf@counta>0
    \forest@lt{puff \the\c@pgf@counta}%
    \advance\c@pgf@counta-1
  \forest@repeat
}
\def\forest@compute@node@boundary@starburst{
  \calculatestarburstpoints
  \forest@mt{outer point 1}%
  \c@pgf@counta=\totalpoints\relax
  \divide\c@pgf@counta2
  \forest@loop
  \ifnum\c@pgf@counta>0
    \forest@lt{inner point \the\c@pgf@counta}%
    \forest@lt{outer point \the\c@pgf@counta}%
    \advance\c@pgf@counta-1
  \forest@repeat
}%
\def\forest@compute@node@boundary@signal{%
  \forest@mt{east}%
  \forest@lt{south east}%
  \forest@lt{south west}%
  \forest@lt{west}%
  \forest@lt{north west}%
  \forest@lt{north east}%
  \forest@lt{east}%
}
\def\forest@compute@node@boundary@tape{%
  \forest@mt{north east}%
  \forest@lt{60}%
  \forest@lt{north}%
  \forest@lt{120}%
  \forest@lt{north west}%
  \forest@lt{south west}%
  \forest@lt{240}%
  \forest@lt{south}%
  \forest@lt{310}%
  \forest@lt{south east}%
  \forest@lt{north east}%
}
\csdef{forest@compute@node@boundary@single arrow}{%
  \forest@mt{tip}%
  \forest@lt{after tip}%
  \forest@lt{after head}%
  \forest@lt{before tail}%
  \forest@lt{after tail}%
  \forest@lt{before head}%
  \forest@lt{before tip}%
  \forest@lt{tip}%
}
\csdef{forest@compute@node@boundary@double arrow}{%
  \forest@mt{tip 1}%
  \forest@lt{after tip 1}%
  \forest@lt{after head 1}%
  \forest@lt{before head 2}%
  \forest@lt{before tip 2}%
  \forest@mt{tip 2}%
  \forest@lt{after tip 2}%
  \forest@lt{after head 2}%
  \forest@lt{before head 1}%
  \forest@lt{before tip 1}%
  \forest@lt{tip 1}%
}
\csdef{forest@compute@node@boundary@arrow box}{%
  \forest@mt{before north arrow}%
  \forest@lt{before north arrow head}%
  \forest@lt{before north arrow tip}%
  \forest@lt{north arrow tip}%
  \forest@lt{after north arrow tip}%
  \forest@lt{after north arrow head}%
  \forest@lt{after north arrow}%
  \forest@lt{north east}%
  \forest@lt{before east arrow}%
  \forest@lt{before east arrow head}%
  \forest@lt{before east arrow tip}%
  \forest@lt{east arrow tip}%
  \forest@lt{after east arrow tip}%
  \forest@lt{after east arrow head}%
  \forest@lt{after east arrow}%
  \forest@lt{south east}%
  \forest@lt{before south arrow}%
  \forest@lt{before south arrow head}%
  \forest@lt{before south arrow tip}%
  \forest@lt{south arrow tip}%
  \forest@lt{after south arrow tip}%
  \forest@lt{after south arrow head}%
  \forest@lt{after south arrow}%
  \forest@lt{south west}%
  \forest@lt{before west arrow}%
  \forest@lt{before west arrow head}%
  \forest@lt{before west arrow tip}%
  \forest@lt{west arrow tip}%
  \forest@lt{after west arrow tip}%
  \forest@lt{after west arrow head}%
  \forest@lt{after west arrow}%
  \forest@lt{north west}%
  \forest@lt{before north arrow}%
}
\cslet{forest@compute@node@boundary@circle split}\forest@compute@node@boundary@circle
\cslet{forest@compute@node@boundary@circle solidus}\forest@compute@node@boundary@circle
\cslet{forest@compute@node@boundary@ellipse split}\forest@compute@node@boundary@ellipse
\cslet{forest@compute@node@boundary@rectangle split}\forest@compute@node@boundary@rectangle
\def\forest@compute@node@boundary@@callout{%
  \beforecalloutpointer
  \pgfsyssoftpath@moveto{\the\pgf@x}{\the\pgf@y}%
  \calloutpointeranchor
  \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
  \aftercalloutpointer
  \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
}
\csdef{forest@compute@node@boundary@rectangle callout}{%
  \forest@compute@node@boundary@rectangle
  \rectanglecalloutpoints
  \forest@compute@node@boundary@@callout
}
\csdef{forest@compute@node@boundary@ellipse callout}{%
```

```
  \forest@compute@node@boundary@ellipse
  \ellipsecalloutpoints
  \forest@compute@node@boundary@@callout
}
\csdef{forest@compute@node@boundary@cloud callout}{%
  \forest@compute@node@boundary@cloud
  % at least a first approx...
  \forest@mt{center}%
  \forest@lt{pointer}%
}%
\csdef{forest@compute@node@boundary@cross out}{%
  \forest@mt{south east}%
  \forest@lt{north west}%
  \forest@mt{south west}%
  \forest@lt{north east}%
}%
\csdef{forest@compute@node@boundary@strike out}{%
  \forest@mt{north east}%
  \forest@lt{south west}%
}%
\csdef{forest@compute@node@boundary@rounded rectangle}{%
  \forest@mt{east}%
  \forest@lt{north east}%
  \forest@lt{north}%
  \forest@lt{north west}%
  \forest@lt{west}%
  \forest@lt{south west}%
  \forest@lt{south}%
  \forest@lt{south east}%
  \forest@lt{east}%
}%
\csdef{forest@compute@node@boundary@chamfered rectangle}{%
  \forest@mt{before south west}%
  \forest@mt{after south west}%
  \forest@lt{before south east}%
  \forest@lt{after south east}%
  \forest@lt{before north east}%
  \forest@lt{after north east}%
  \forest@lt{before north west}%
  \forest@lt{after north west}%
  \forest@lt{before south west}%
}%
```

8.3 Compute absolute positions

Computes absolute positions of descendants relative to this node. Stores the results in attributes `x` and `y`.

```
\def\forest@node@computeabsolutepositions{%
  \edef\forest@marshal{%
    \noexpand\forest@node@foreachchild{%
      \noexpand\forest@node@computeabsolutepositions@{\forestove{x}}{\forestove{y}}{\forestove{grow}}%
    }%
  }\forest@marshal
}
\def\forest@node@computeabsolutepositions@#1#2#3{%
  \pgfpointadd
    {\pgfqpoint{#1}{#2}}%
    {\pgfpointadd
      {\pgfqpointpolar{#3}{\forestove{l}}}%
      {\pgfqpointpolar{\numexpr 90+#3\relax}{\forestove{s}}}%
    }%
  \pgfgetlastxy\forest@temp@x\forest@temp@y
  \forestolet{x}\forest@temp@x
  \forestolet{y}\forest@temp@y
  \edef\forest@marshal{%
    \noexpand\forest@node@foreachchild{%
      \noexpand\forest@node@computeabsolutepositions@{\forest@temp@x}{\forest@temp@y}{\forestove{grow}}%
    }%
  }\forest@marshal
}
```

8.4 Drawing the tree

```
\newif\ifforest@drawtree@preservenodeboxes@
\def\forest@node@drawtree{%
  \expandafter\ifstrequal\expandafter{\forest@drawtreebox}{\pgfkeysnovalue}{%
    \let\forest@drawtree@beginbox\relax
    \let\forest@drawtree@endbox\relax
  }{%
    \edef\forest@drawtree@beginbox{\global\setbox\forest@drawtreebox=\hbox\bgroup}%
    \let\forest@drawtree@endbox\egroup
  }%
  \ifforest@external@
    \ifforest@externalize@tree@
      \forest@temptrue
    \else
      \tikzifexternalizing{%
        \ifforest@was@tikzexternalwasenable
          \forest@temptrue
          \pgfkeys{/tikz/external/optimize=false}%
          \let\forest@drawtree@beginbox\relax
          \let\forest@drawtree@endbox\relax
        \else
          \forest@tempfalse
        \fi
      }{%
        \forest@tempfalse
      }%
    \fi
    \ifforest@temp
      \advance\forest@externalize@inner@n 1
      \edef\forest@externalize@filename{%
        \tikzexternalrealjob-forest-\forest@externalize@outer@n
        \ifnum\forest@externalize@inner@n=0 \else.\the\forest@externalize@inner@n\fi}%
      \expandafter\tikzsetnextfilename\expandafter{\forest@externalize@filename}%
      \tikzexternalenable
      \pgfkeysalso{/tikz/external/remake next,/tikz/external/export next}%
    \fi
    \ifforest@externalize@tree@
      \typeout{forest: Invoking a recursive call to generate the external picture
        ’\forest@externalize@filename’ for the following context+code:
        ’\expandafter\detokenize\expandafter{\forest@externalize@id}’}%
    \fi
  \fi
  %
  \ifforesttikzcshack
    \let\forest@original@tikz@parse@node\tikz@parse@node
    \let\tikz@parse@node\forest@tikz@parse@node
  \fi
  \pgfkeysgetvalue{/forest/begin draw/.@cmd}\forest@temp@begindraw
  \pgfkeysgetvalue{/forest/end draw/.@cmd}\forest@temp@enddraw
  \edef\forest@marshal{%
    \noexpand\forest@drawtree@beginbox
    \expandonce{\forest@temp@begindraw\pgfkeysnovalue\pgfeov}%
    \noexpand\forest@node@drawtree@
    \expandonce{\forest@temp@enddraw\pgfkeysnovalue\pgfeov}%
    \noexpand\forest@drawtree@endbox
  }\forest@marshal
  \ifforesttikzcshack
    \let\tikz@parse@node\forest@original@tikz@parse@node
  \fi
  %
  \ifforest@external@
    \ifforest@externalize@tree@
      \tikzexternaldisable
      \eappto\forest@externalize@checkimages{%
        \noexpand\forest@includeexternal@check{\forest@externalize@filename}%
      }%
      \expandafter\ifstrequal\expandafter{\forest@drawtreebox}{\pgfkeysnovalue}{%
        \eappto\forest@externalize@loadimages{%
          \noexpand\forest@includeexternal{\forest@externalize@filename}%
        }%
      }{%
        \eappto\forest@externalize@loadimages{%
          \noexpand\forest@includeexternal@box\forest@drawtreebox{\forest@externalize@filename}%
        }%
      }%
    \fi
  \fi
}
\def\forest@drawtree@root{0}
\def\forest@node@drawtree@{%
  \def\forest@clear@drawn{}%
  \forest@forthis{%
    \forest@saveandrestoremacro\forest@drawtree@root{%
      \edef\forest@drawtree@root{\forest@cn}%
      \forestset{draw tree method}%
    }%
  }%
  \forest@node@Ifnamedefined{forest@baseline@node}{%
    \edef\forest@baseline@id{\forest@node@Nametoid{forest@baseline@node}}%
    \ifnum\forest@baseline@id=0
    \else
      \ifcsdef{forest@drawn@\forest@baseline@id}{%
        \edef\forest@marshal{%
          \noexpand\pgfsetbaselinepointlater{%
            \noexpand\pgfpointanchor
              {\forestOve{\forest@baseline@id}{name}}%
              {\forestOve{\forest@baseline@id}{anchor}}%
          }%
        }\forest@marshal
      }{%
        \PackageWarning{forest}{Baseline node (id=\forest@cn) was not drawn (most likely it's a phantom node)
      }%
    \fi
  }%
  \forest@clear@drawn
}
\def\forest@draw@node{%
  \ifnum\forestove{phantom}=0
    \forest@draw@node@
  \fi
}
\def\forest@draw@node@{%
  \forest@node@forest@positionnodelater@restore
  \ifforest@drawtree@preservenodeboxes@
    \pgfnodealias{forest@temp}{\forestove{later@name}}%
  \fi
  \pgfpositionnodenow{\pgfqpoint{\forestove{x}}{\forestove{y}}}%
  \ifforest@drawtree@preservenodeboxes@
    \pgfnodealias{\forestove{later@name}}{forest@temp}%
  \fi
  \csdef{forest@drawn@\forest@cn}{}%
  \eappto\forest@clear@drawn{\noexpand\csundef{forest@drawn@\forest@cn}}%
}
\def\forest@draw@edge{%
  \ifcsdef{forest@drawn@\forest@cn}{% was the current node drawn?
    \ifnum\forestove{@parent}=0 % do we have a parent?
    \else
      \ifcsdef{forest@drawn@\forestove{@parent}}{% was the parent drawn?
        \forest@draw@edge@
      }{}%
    \fi
  }{}%
}
\def\forest@draw@edge@{%
  \edef\forest@temp{\forestove{edge path}}\forest@temp
}
\def\forest@draw@tikz{%
  \ifnum\forestove{phantom}=0
    \forest@draw@tikz@
  \fi
}
\def\forest@draw@tikz@{%
  \forestove{tikz}%
}
```

9 Geometry

A α *grow line* is a line through the origin at angle α. The following macro sets up the grow line, which can then be used by other code (the change is local to the TeX group). More precisely, two normalized vectors are set up: one (x_g, y_g) on the grow line, and one (x_s, y_s) orthogonal to it—to get (x_s, y_s), rotate (x_g, y_g) 90° counter-clockwise.

```
\newdimen\forest@xg
\newdimen\forest@yg
\newdimen\forest@xs
\newdimen\forest@ys
\def\forest@setupgrowline#1{%
  \edef\forest@grow{#1}%
  \pgfqpointpolar{\forest@grow}{1pt}%
  \forest@xg=\pgf@x
  \forest@yg=\pgf@y
  \forest@xs=-\pgf@y
  \forest@ys=\pgf@x
}
```

9.1 Projections

The following macro belongs to the `\pgfpoint...` family: it projects point `#1` on the grow line. (The result is returned via `\pgf@x` and `\pgf@y`.) The implementation is based on code from `tikzlibrarycalc`, but optimized for projecting on grow lines, and split to optimize serial usage in `\forest@projectpath`.

```
\def\forest@pgfpointprojectiontogrowline#1{{%
  \pgf@process{#1}%
```

Calculate the scalar product of (x, y) and (x_g, y_g): that's the distance of (x, y) to the grow line.

```
  \pgfutil@tempdima=\pgf@sys@tonumber{\pgf@x}\forest@xg%
  \advance\pgfutil@tempdima by\pgf@sys@tonumber{\pgf@y}\forest@yg%
```

The projection is (x_g, y_g) scaled by the distance.

```
  \global\pgf@x=\pgf@sys@tonumber{\pgfutil@tempdima}\forest@xg%
  \global\pgf@y=\pgf@sys@tonumber{\pgfutil@tempdima}\forest@yg%
}}
```

The following macro calculates the distance of point `#2` to the grow line and stores the result in TeX-dimension `#1`. The distance is the scalar product of the point vector and the normalized vector orthogonal to the grow line.

```
\def\forest@distancetogrowline#1#2{%
  \pgf@process{#2}%
  #1=\pgf@sys@tonumber{\pgf@x}\forest@xs\relax
  \advance#1 by\pgf@sys@tonumber{\pgf@y}\forest@ys\relax
}
```

Note that the distance to the grow line is positive for points on one of its sides and negative for points on the other side. (It is positive on the side which (x_s, y_s) points to.) We thus say that the grow line partitions the plane into a *positive* and a *negative* side.

The following macro projects all segment edges ("points") of a simple[2] path `#1` onto the grow line. The result is an array of tuples (`xo`, `yo`, `xp`, `yp`), where `xo` and `yo` stand for the *o*riginal point, and `xp` and `yp` stand for its *p*rojection. The prefix of the array is given by `#2`. If the array already exists, the new items are appended to it. The array is not sorted: the order of original points in the array is their order in the path. The computation does not destroy the current path. All result-macros have local scope.

The macro is just a wrapper for `\forest@projectpath@process`.

```
\let\forest@pp@n\relax
\def\forest@projectpathtogrowline#1#2{%
  \edef\forest@pp@prefix{#2}%
  \forest@save@pgfsyssoftpath@tokendefs
  \let\pgfsyssoftpath@movetotoken\forest@projectpath@processpoint
  \let\pgfsyssoftpath@linetotoken\forest@projectpath@processpoint
  \c@pgf@counta=0
  #1%
  \csedef{#2n}{\the\c@pgf@counta}%
  \forest@restore@pgfsyssoftpath@tokendefs
}
```

For each point, remember the point and its projection to grow line.

```
\def\forest@projectpath@processpoint#1#2{%
  \pgfqpoint{#1}{#2}%
  \expandafter\edef\csname\forest@pp@prefix\the\c@pgf@counta xo\endcsname{\the\pgf@x}%
  \expandafter\edef\csname\forest@pp@prefix\the\c@pgf@counta yo\endcsname{\the\pgf@y}%
  \forest@pgfpointprojectiontogrowline{}%
  \expandafter\edef\csname\forest@pp@prefix\the\c@pgf@counta xp\endcsname{\the\pgf@x}%
  \expandafter\edef\csname\forest@pp@prefix\the\c@pgf@counta yp\endcsname{\the\pgf@y}%
  \advance\c@pgf@counta 1\relax
}
```

Sort the array (prefix `#1`) produced by `\forest@projectpathtogrowline` by `(xp,yp)`, in the ascending order.

```
\def\forest@sortprojections#1{%
  % todo: optimize in cases when we know that the array is actually a
  % merger of sorted arrays; when does this happen? in
  % distance_between_paths, and when merging the edges of the parent
```

---

[2] A path is *simple* if it consists of only move-to and line-to operations.

```
  % and its children in a uniform growth tree
  \edef\forest@ppi@inputprefix{#1}%
  \c@pgf@counta=\csname#1n\endcsname\relax
  \advance\c@pgf@counta -1
  \forest@sort\forest@ppiraw@cmp\forest@ppiraw@let\forest@sort@ascending{0}{\the\c@pgf@counta}%
}
```

The following macro processes the data gathered by (possibly more than one invocation of) `\forest@projectpathtogrowline` into array with prefix `#1`. The resulting data is the following.

- Array of projections (prefix `#2`)
  - its items are tuples `(x,y)` (the array is sorted by `x` and `y`), and
  - an inner array of original points (prefix `#2N@`, where N is the index of the item in array `#2`. The items of `#2N@` are `x`, `y` and `d`: `x` and `y` are the coordinates of the original point; `d` is its distance to the grow line. The inner array is not sorted.
- A "dictionary" `#3`: keys are the coordinates `(x,y)` of the original points; a value is the index of the original point's projection in array `#2`.[3] In v2.1.4, the "dictionary" was reimplemented using a toks register, to prevent using up the string pool; that's when `#3` was introduced.

```
\def\forest@processprojectioninfo#1#2#3{%
  \edef\forest@ppi@inputprefix{#1}%
```

Loop (counter `\c@pgf@counta`) through the sorted array of raw data.

```
  \c@pgf@counta=0
  \c@pgf@countb=-1
  \safeloop
  \ifnum\c@pgf@counta<\csname#1n\endcsname\relax
```

Check if the projection tuple in the current raw item equals the current projection.

```
    \letcs\forest@xo{#1\the\c@pgf@counta xo}%
    \letcs\forest@yo{#1\the\c@pgf@counta yo}%
    \letcs\forest@xp{#1\the\c@pgf@counta xp}%
    \letcs\forest@yp{#1\the\c@pgf@counta yp}%
    \ifnum\c@pgf@countb<0
      \forest@equaltotolerancefalse
    \else
      \forest@equaltotolerance
        {\pgfqpoint\forest@xp\forest@yp}%
        {\pgfqpoint
          {\csname#2\the\c@pgf@countb x\endcsname}%
          {\csname#2\the\c@pgf@countb y\endcsname}%
        }%
    \fi
    \ifforest@equaltotolerance\else
```

It not, we will append a new item to the outer result array.

```
      \advance\c@pgf@countb 1
      \cslet{#2\the\c@pgf@countb x}\forest@xp
      \cslet{#2\the\c@pgf@countb y}\forest@yp
      \csdef{#2\the\c@pgf@countb @n}{0}%
    \fi
```

If the projection is actually a projection of one point in our path (it will not be when this macro is called from `\forest@distance@between@edge@paths`):

```
    % todo: this is ugly!
    \ifdefined\forest@xo\ifx\forest@xo\relax\else
      \ifdefined\forest@yo\ifx\forest@yo\relax\else
```

---

[3] At first sight, this information could be cached "at the source": by forest@pgfpointprojectiontogrowline. However, due to imprecise intersecting (in breakpath), we cheat and merge very adjacent projection points, expecting that the points to project to the merged projection point. All this depends on the given path, so a generic cache is not feasible.

Append the point of the current raw item to the inner array of points projecting to the current projection.

```
        \forest@append@point@to@inner@array
          \forest@xo\forest@yo
          {#2\the\c@pgf@countb @}%
```

Put a new item in the dictionary: key = the original point, value = the projection index.

```
        \eapptotoks#3{(\forest@xo,\forest@yo){\the\c@pgf@countb}}%
      \fi\fi
    \fi\fi
```

Clean-up the raw array item.

```
  % todo: is this really necessary? (yes: see the "ugly" thing above)
  \cslet{#1\the\c@pgf@counta xo}\relax
  \cslet{#1\the\c@pgf@counta yo}\relax
  \cslet{#1\the\c@pgf@counta xp}\relax
  \cslet{#1\the\c@pgf@counta yp}\relax
  \advance\c@pgf@counta 1
  \saferepeat
```

Clean up the raw array length.

```
  % todo: is this really necessary?
  \cslet{#1n}\relax
```

Store the length of the outer result array.

```
  \advance\c@pgf@countb 1
  \csedef{#2n}{\the\c@pgf@countb}%
}
```

Item-exchange macro for sorting the raw projection data. (`#1` is copied into `#2`.)

```
\def\forest@ppiraw@let#1#2{%
  \csletcs{\forest@ppi@inputprefix#1xo}{\forest@ppi@inputprefix#2xo}%
  \csletcs{\forest@ppi@inputprefix#1yo}{\forest@ppi@inputprefix#2yo}%
  \csletcs{\forest@ppi@inputprefix#1xp}{\forest@ppi@inputprefix#2xp}%
  \csletcs{\forest@ppi@inputprefix#1yp}{\forest@ppi@inputprefix#2yp}%
}
```

Item comparision macro for sorting the raw projection data.

```
\def\forest@ppiraw@cmp#1#2{%
  \forest@sort@cmptwodimcs
    {\forest@ppi@inputprefix#1xp}{\forest@ppi@inputprefix#1yp}%
    {\forest@ppi@inputprefix#2xp}{\forest@ppi@inputprefix#2yp}%
}
```

Append the point `(#1,#2)` to the (inner) array of points (prefix `#3`).

```
\def\forest@append@point@to@inner@array#1#2#3{%
  \c@pgf@countc=\csname#3n\endcsname\relax
  \csedef{#3\the\c@pgf@countc x}{#1}%
  \csedef{#3\the\c@pgf@countc y}{#2}%
  \forest@distancetogrowline\pgfutil@tempdima{\pgfqpoint#1#2}%
  \csedef{#3\the\c@pgf@countc d}{\the\pgfutil@tempdima}%
  \advance\c@pgf@countc 1
  \csedef{#3n}{\the\c@pgf@countc}%
}
```

9.2 Break path

The following macro computes from the given path (`#1`) a “broken” path (`#4`) that contains the same points of the plane, but has potentially more segments, so that, for every point from a given set of points on the grow line, a line through this point perpendicular to the grow line intersects the broken path only at its edge segments (i.e. not between them).

The macro works only for *simple* paths, i.e. paths built using only move-to and line-to operations. Furthermore, `\forest@processprojectioninfo` must be called before calling `\forest@breakpath`: we expect information in an array with prefix `#2` (projections and (an inner array of) their original points) and toks register `#3` (a “dictionary”: for each original points, the index of its projection in `#2`). The macro updates array `#2`. (No need to update `#3`, as it is not used anymore.)

```
\def\forest@breakpath#1#2#3#4{%
```

Store the current path in a macro and empty it, then process the stored path. The processing creates a new current path.

```
  \edef\forest@bp@prefix{#2}%
  \let\forest@breakpath@toks#3%
  \forest@save@pgfsyssoftpath@tokendefs
  \let\pgfsyssoftpath@movetotoken\forest@breakpath@processfirstpoint
  \let\pgfsyssoftpath@linetotoken\forest@breakpath@processfirstpoint
  %\pgfusepath{}% empty the current path. ok?
  #1%
  \forest@restore@pgfsyssoftpath@tokendefs
  \pgfsyssoftpath@getcurrentpath#4%
}
```

The original and the broken path start in the same way. (This code implicitely “repairs” a path that starts illegally, with a line-to operation.)

```
\def\forest@breakpath@processfirstpoint#1#2{%
  \forest@breakpath@processmoveto{#1}{#2}%
  \let\pgfsyssoftpath@movetotoken\forest@breakpath@processmoveto
  \let\pgfsyssoftpath@linetotoken\forest@breakpath@processlineto
}
```

When a move-to operation is encountered, it is simply copied to the broken path, starting a new subpath. Then we remember the last point, its projection’s index (the point dictionary is used here) and the actual projection point.

```
\def\forest@breakpath@processmoveto#1#2{%
  \pgfsyssoftpath@moveto{#1}{#2}%
  \def\forest@previous@x{#1}%
  \def\forest@previous@y{#2}%
  \forest@breakpath@getfromtoks\forest@breakpath@toks\forest@previous@i{#1}{#2}%
  \expandafter\let\expandafter\forest@previous@px
    \csname\forest@bp@prefix\forest@previous@i x\endcsname
  \expandafter\let\expandafter\forest@previous@py
    \csname\forest@bp@prefix\forest@previous@i y\endcsname
}
\def\forest@breakpath@getfromtoks#1#2#3#4{%
  % #1=cache toks register, #2=receiving cs, (#3,#4)=point;
  % we rely on the fact that the point we're looking up should always be present
  \def\forest@breakpath@getfromtoks@##1(#3,#4)##2##3\forest@END{##2}%
  \edef#2{\expandafter\forest@breakpath@getfromtoks@\the#1\forest@END}%
}
```

This is the heart of the path-breaking procedure.

```
\def\forest@breakpath@processlineto#1#2{%
```

Usually, the broken path will continue with a line-to operation (to the current point `(#1,#2)`).

```
  \let\forest@breakpath@op\pgfsyssoftpath@lineto
```

Get the index of the current point’s projection and the projection itself. (The point dictionary is used here.)

```
  \forest@breakpath@getfromtoks\forest@breakpath@toks\forest@i{#1}{#2}%
  \expandafter\let\expandafter\forest@px
    \csname\forest@bp@prefix\forest@i x\endcsname
  \expandafter\let\expandafter\forest@py
    \csname\forest@bp@prefix\forest@i y\endcsname
```

Test whether the projections of the previous and the current point are the same.

```
  \forest@equaltotolerance
    {\pgfqpoint{\forest@previous@px}{\forest@previous@py}}%
    {\pgfqpoint{\forest@px}{\forest@py}}%
  \ifforest@equaltotolerance
```

If so, we are dealing with a segment, perpendicular to the grow line. This segment must be removed, so we change the operation to move-to.

```
    \let\forest@breakpath@op\pgfsyssoftpath@moveto
  \else
```

Figure out the “direction” of the segment: in the order of the array of projections, or in the reversed order? Setup the loop step and the test condition.

```
    \forest@temp@count=\forest@previous@i\relax
    \ifnum\forest@previous@i<\forest@i\relax
      \def\forest@breakpath@step{1}%
      \def\forest@breakpath@test{\forest@temp@count<\forest@i\relax}%
    \else
      \def\forest@breakpath@step{-1}%
      \def\forest@breakpath@test{\forest@temp@count>\forest@i\relax}%
    \fi
```

Loop through all the projections between (in the (possibly reversed) array order) the projections of the previous and the current point (both exclusive).

```
    \safeloop
      \advance\forest@temp@count\forest@breakpath@step\relax
    \expandafter\ifnum\forest@breakpath@test
```

Intersect the current segment with the line through the current (in the loop!) projection perpendicular to the grow line. (There *will* be an intersection.)

```
      \pgfpointintersectionoflines
        {\pgfqpoint
          {\csname\forest@bp@prefix\the\forest@temp@count x\endcsname}%
          {\csname\forest@bp@prefix\the\forest@temp@count y\endcsname}%
        }%
        {\pgfpointadd
          {\pgfqpoint
            {\csname\forest@bp@prefix\the\forest@temp@count x\endcsname}%
            {\csname\forest@bp@prefix\the\forest@temp@count y\endcsname}%
          }%
          {\pgfqpoint{\forest@xs}{\forest@ys}}%
        }%
        {\pgfqpoint{\forest@previous@x}{\forest@previous@y}}%
        {\pgfqpoint{#1}{#2}}%
```

Break the segment at the intersection.

```
      \pgfgetlastxy\forest@last@x\forest@last@y
      \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
```

Append the breaking point to the inner array for the projection.

```
      \forest@append@point@to@inner@array
        \forest@last@x\forest@last@y
        {\forest@bp@prefix\the\forest@temp@count @}%
    \saferepeat
  \fi
```

Add the current point.

```
  \forest@breakpath@op{#1}{#2}%
```

Setup new “previous” info: the segment edge, its projection’s index, and the projection.

```
  \def\forest@previous@x{#1}%
  \def\forest@previous@y{#2}%
  \let\forest@previous@i\forest@i
  \let\forest@previous@px\forest@px
  \let\forest@previous@py\forest@py
}
```

Patch for speed: no need to call `\pgfmathparse` here.

```
\patchcmd{\pgfpointintersectionoflines}{\pgfpoint}{\pgfqpoint}{}{}
```

9.3 Get tight edge of path

This is one of the central algorithms of the package. Given a simple path and a grow line, this method computes its (negative and positive) "tight edge", which we (informally) define as follows.

Imagine an infinitely long light source parallel to the grow line, on the grow line's negative/positive side.[4] Furthermore imagine that the path is opaque. Then the negative/positive tight edge of the path is the part of the path that is illuminated.

This macro takes three arguments: `#1` is the path; `#2` and `#3` are macros which will receive the negative and the positive edge, respectively. The edges are returned in the softpath format. Grow line should be set before calling this macro.

Enclose the computation in a TEX group. This is actually quite crucial: if there was no enclosure, the temporary data (the segment dictionary, to be precise) computed by the prior invocations of the macro could corrupt the computation in the current invocation.

```
\def\forest@getnegativetightedgeofpath#1#2{%
  \forest@get@onetightedgeofpath#1\forest@sort@ascending#2}
\def\forest@getpositivetightedgeofpath#1#2{%
  \forest@get@onetightedgeofpath#1\forest@sort@descending#2}
\def\forest@get@onetightedgeofpath#1#2#3{%
  {%
    \forest@get@one@tightedgeofpath#1#2\forest@gep@edge
    \global\let\forest@gep@global@edge\forest@gep@edge
  }%
  \let#3\forest@gep@global@edge
}
\newtoks\forest@pi@toks
\newtoks\forest@segment@toks
\def\forest@get@one@tightedgeofpath#1#2#3{%
```

Project the path to the grow line and compile some useful information.

```
  \forest@projectpathtogrowline#1{forest@pp@}%
  \forest@sortprojections{forest@pp@}%
  \forest@processprojectioninfo{forest@pp@}{forest@pi@}\forest@pi@toks
```

Break the path.

```
  \forest@breakpath#1{forest@pi@}\forest@pi@toks\forest@brokenpath
```

Compile some more useful information.

```
  \forest@sort@inner@arrays{forest@pi@}#2%
  \forest@pathtodict\forest@brokenpath\forest@segment@toks
```

The auxiliary data is set up: do the work!

```
  \forest@gettightedgeofpath@getedge\forest@edge
```

Where possible, merge line segments of the path into a single line segment. This is an important optimization, since the edges of the subtrees are computed recursively. Not simplifying the edge could result in a wild growth of the length of the edge (in the sense of the number of segments).

```
  \forest@simplifypath\forest@edge#3%
}
```

Get both negative (stored in `#2`) and positive (stored in `#3`) edge of the path `#1`.

```
\def\forest@getbothtightedgesofpath#1#2#3{%
  {%
    \forest@get@one@tightedgeofpath#1\forest@sort@ascending\forest@gep@firstedge
```

Reverse the order of items in the inner arrays.

```
    \c@pgf@counta=0
    \forest@loop
```

---

[4] For the definition of negative/positive side, see `\forest@distancetogrowline` in §9.1

```
    \ifnum\c@pgf@counta<\forest@pi@n\relax
      \forest@ppi@deflet{forest@pi@\the\c@pgf@counta @}%
      \forest@reversearray\forest@ppi@let
        {0}%
        {\csname forest@pi@\the\c@pgf@counta @n\endcsname}%
      \advance\c@pgf@counta 1
    \forest@repeat
```

Calling `\forest@gettightedgeofpath@getedge` now will result in the positive edge.

```
    \forest@gettightedgeofpath@getedge\forest@edge
    \forest@simplifypath\forest@edge\forest@gep@secondedge
```

Smuggle the results out of the enclosing TEX group.

```
    \global\let\forest@gep@global@firstedge\forest@gep@firstedge
    \global\let\forest@gep@global@secondedge\forest@gep@secondedge
  }%
  \let#2\forest@gep@global@firstedge
  \let#3\forest@gep@global@secondedge
}
```

Sort the inner arrays of original points wrt the distance to the grow line. `#2` = `\forest@sort@ascending/\forest@sor`

```
\def\forest@sort@inner@arrays#1#2{%
  \c@pgf@counta=0
  \safeloop
  \ifnum\c@pgf@counta<\csname#1n\endcsname
    \c@pgf@countb=\csname#1\the\c@pgf@counta @n\endcsname\relax
    \ifnum\c@pgf@countb>1
      \advance\c@pgf@countb -1
      \forest@ppi@deflet{#1\the\c@pgf@counta @}%
      \forest@ppi@defcmp{#1\the\c@pgf@counta @}%
      \forest@sort\forest@ppi@cmp\forest@ppi@let#2{0}{\the\c@pgf@countb}%
    \fi
    \advance\c@pgf@counta 1
  \saferepeat
}
```

A macro that will define the item exchange macro for quicksorting the inner arrays of original points. It takes one argument: the prefix of the inner array.

```
\def\forest@ppi@deflet#1{%
  \edef\forest@ppi@let##1##2{%
    \noexpand\csletcs{#1##1x}{#1##2x}%
    \noexpand\csletcs{#1##1y}{#1##2y}%
    \noexpand\csletcs{#1##1d}{#1##2d}%
  }%
}
```

A macro that will define the item-compare macro for quicksorting the embedded arrays of original points. It takes one argument: the prefix of the inner array.

```
\def\forest@ppi@defcmp#1{%
  \edef\forest@ppi@cmp##1##2{%
    \noexpand\forest@sort@cmpdimcs{#1##1d}{#1##2d}%
  }%
}
```

Put path segments into a "segment dictionary": for each segment of the pgf path (given in `#1`) from (x_1, y_1) to (x_2, y_2) we put `(x1,y1)--(x2,y2)` into toks `#2`. (The "dictionary" was reimplemented in v2.1.4. It's based on a toks register now, we search using `\pgfutil@in@`.)

```
\def\forest@pathtodict#1#2{%
  \let\forest@pathtodict@toks#2%
  \forest@save@pgfsyssoftpath@tokendefs
  \let\pgfsyssoftpath@movetotoken\forest@pathtodict@movetoop
  \let\pgfsyssoftpath@linetotoken\forest@pathtodict@linetoop
  \def\forest@pathtodict@subpathstart{}%
  #1%
  \forest@restore@pgfsyssoftpath@tokendefs
}
```

When a move-to operation is encountered:

```
\def\forest@pathtodict@movetoop#1#2{%
  \apptotoks\forest@pathtodict@toks{(#1,#2)}%
}
```

When a line-to operation is encountered:

```
\def\forest@pathtodict@linetoop#1#2{%
  \apptotoks\forest@pathtodict@toks{--(#1,#2)}%
}
```

In this macro, the edge is actually computed.

```
\def\forest@gettightedgeofpath@getedge#1{% cs to store the edge into
```

Clear the path and the last projection.

```
  \pgfsyssoftpath@setcurrentpath\pgfutil@empty
  \let\forest@last@x\relax
  \let\forest@last@y\relax
```

Loop through the (ordered) array of projections. (Since we will be dealing with the current and the next projection in each iteration of the loop, we loop the counter from the first to the second-to-last projection.)

```
  \c@pgf@counta=0
  \forest@temp@count=\forest@pi@n\relax
  \advance\forest@temp@count -1
  \edef\forest@nminusone{\the\forest@temp@count}%
  \safeloop
  \ifnum\c@pgf@counta<\forest@nminusone\relax
    \forest@gettightedgeofpath@getedge@loopa
  \saferepeat
```

A special case: the edge ends with a degenerate subpath (a point).

```
  \ifnum\forest@nminusone<\forest@n\relax\else
    \ifnum\csname forest@pi@\forest@nminusone @n\endcsname>0
      \forest@gettightedgeofpath@maybemoveto{\forest@nminusone}{0}%
    \fi
  \fi
  \pgfsyssoftpath@getcurrentpath#1%
  \pgfsyssoftpath@setcurrentpath\pgfutil@empty
}
```

The body of a loop containing an embedded loop must be put in a separate macro because it contains the `\if...` of the embedded `\forest@loop...` without the matching `\fi`: `\fi` is “hiding” in the embedded `\forest@loop`, which has not been expanded yet.

```
\def\forest@gettightedgeofpath@getedge@loopa{%
    \ifnum\csname forest@pi@\the\c@pgf@counta @n\endcsname>0
```

Degenerate case: a subpath of the edge is a point.

```
      \forest@gettightedgeofpath@maybemoveto{\the\c@pgf@counta}{0}%
```

Loop through points projecting to the current projection. The preparations above guarantee that the points are ordered (either in the ascending or the descending order) with respect to their distance to the grow line.

```
      \c@pgf@countb=0
      \safeloop
      \ifnum\c@pgf@countb<\csname forest@pi@\the\c@pgf@counta @n\endcsname\relax
        \forest@gettightedgeofpath@getedge@loopb
      \saferepeat
    \fi
    \advance\c@pgf@counta 1
}
```

Loop through points projecting to the next projection. Again, the points are ordered.

```
\def\forest@gettightedgeofpath@getedge@loopb{%
          \c@pgf@countc=0
          \advance\c@pgf@counta 1
          \edef\forest@aplusone{\the\c@pgf@counta}%
          \advance\c@pgf@counta -1
          \safeloop
          \ifnum\c@pgf@countc<\csname forest@pi@\forest@aplusone @n\endcsname\relax
```

Test whether [the current point]–[the next point] or [the next point]–[the current point] is a segment in the (broken) path. The first segment found is the one with the minimal/maximal distance (depending on the sort order of arrays of points projecting to the same projection) to the grow line.

Note that for this to work in all cases, the original path should have been broken on its self-intersections. However, a careful reader will probably remember that `\forest@breakpath` does *not* break the path at its self-intersections. This is omitted for performance reasons. Given the intended use of the algorithm (calculating edges of subtrees), self-intersecting paths cannot arise anyway, if only the node boundaries are non-self-intersecting. So, a warning: if you develop a new shape and write a macro computing its boundary, make sure that the computed boundary path is non-self-intersecting!

```
          \edef\forest@temp{%
            (\csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb x\endcsname,%
            \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb y\endcsname)--(%
            \csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname,%
            \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname)%
          }%
          \expandafter\expandafter\expandafter\pgfutil@in@\expandafter\expandafter\expandafter
            {\expandafter\forest@temp\expandafter}\expandafter
            {\the\forest@segment@toks}%
          \ifpgfutil@in@
          \else
            \edef\forest@temp{%
              (\csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname,%
              \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname)--(%
              \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb x\endcsname,%
              \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb y\endcsname)%
            }%
            \expandafter\expandafter\expandafter\pgfutil@in@\expandafter\expandafter\expandafter
              {\expandafter\forest@temp\expandafter}\expandafter
              {\the\forest@segment@toks}%
          \fi
          \ifpgfutil@in@
```

We have found the segment with the minimal/maximal distance to the grow line. So let's add it to the edge path.

First, deal with the start point of the edge: check if the current point is the last point. If that is the case (this happens if the current point was the end point of the last segment added to the edge), nothing needs to be done; otherwise (this happens if the current point will start a new subpath of the edge), move to the current point, and update the last-point macros.

```
            \forest@gettightedgeofpath@maybemoveto{\the\c@pgf@counta}{\the\c@pgf@countb}%
```

Second, create a line to the end point.

```
            \edef\forest@last@x{%
              \csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname}%
            \edef\forest@last@y{%
              \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname}%
            \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
```

Finally, "break" out of the innermost two loops.

```
            \c@pgf@countc=\csname forest@pi@\forest@aplusone @n\endcsname
            \c@pgf@countb=\csname forest@pi@\the\c@pgf@counta @n\endcsname
          \fi
          \advance\c@pgf@countc 1
```

```
        \saferepeat
        \advance\c@pgf@countb 1
}
```

`\forest@#1@` is an (ordered) array of points projecting to projection with index `#1`. Check if `#2`th point of that array equals the last point added to the edge: if not, add it.

```
\def\forest@gettightedgeofpath@maybemoveto#1#2{%
  \forest@temptrue
  \ifx\forest@last@x\relax\else
    \ifdim\forest@last@x=\csname forest@pi@#1@#2x\endcsname\relax
      \ifdim\forest@last@y=\csname forest@pi@#1@#2y\endcsname\relax
        \forest@tempfalse
      \fi
    \fi
  \fi
  \ifforest@temp
    \edef\forest@last@x{\csname forest@pi@#1@#2x\endcsname}%
    \edef\forest@last@y{\csname forest@pi@#1@#2y\endcsname}%
    \pgfsyssoftpath@moveto\forest@last@x\forest@last@y
  \fi
}
```

Simplify the resulting path by “unbreaking” segments where possible. (The macro itself is just a wrapper for path processing macros below.)

```
\def\forest@simplifypath#1#2{%
  \pgfsyssoftpath@setcurrentpath\pgfutil@empty
  \forest@save@pgfsyssoftpath@tokendefs
  \let\pgfsyssoftpath@movetotoken\forest@simplifypath@moveto
  \let\pgfsyssoftpath@linetotoken\forest@simplifypath@lineto
  \let\forest@last@x\relax
  \let\forest@last@y\relax
  \let\forest@last@atan\relax
  #1%
  \ifx\forest@last@x\relax\else
    \ifx\forest@last@atan\relax\else
      \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
    \fi
  \fi
  \forest@restore@pgfsyssoftpath@tokendefs
  \pgfsyssoftpath@getcurrentpath#2%
  \pgfsyssoftpath@setcurrentpath\pgfutil@empty
}
```

When a move-to is encountered, we flush whatever segment we were building, make the move, remember the last position, and set the slope to unknown.

```
\def\forest@simplifypath@moveto#1#2{%
  \ifx\forest@last@x\relax\else
    \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
  \fi
  \pgfsyssoftpath@moveto{#1}{#2}%
  \def\forest@last@x{#1}%
  \def\forest@last@y{#2}%
  \let\forest@last@atan\relax
}
```

How much may the segment slopes differ that we can still merge them? (Ignore `pt`, these are degrees.) Also, how good is this number?

```
\def\forest@getedgeofpath@precision{1pt}
```

When a line-to is encountered…

```
\def\forest@simplifypath@lineto#1#2{%
  \ifx\forest@last@x\relax
```

If we're not in the middle of a merger, we need to nothing but start it.

```
    \def\forest@last@x{#1}%
    \def\forest@last@y{#2}%
    \let\forest@last@atan\relax
  \else
```

Otherwise, we calculate the slope of the current segment (i.e. the segment between the last and the current point), . . .

```
    \pgfpointdiff{\pgfqpoint{#1}{#2}}{\pgfqpoint{\forest@last@x}{\forest@last@y}}%
    \ifdim\pgf@x<\pgfintersectiontolerance
      \ifdim-\pgf@x<\pgfintersectiontolerance
        \pgf@x=0pt
      \fi
    \fi
    \edef\forest@marshal{%
      \noexpand\pgfmathatantwo@
        {\expandafter\Pgf@geT\the\pgf@x}%
        {\expandafter\Pgf@geT\the\pgf@y}%
      }\forest@marshal
    \let\forest@current@atan\pgfmathresult
    \ifx\forest@last@atan\relax
```

If this is the first segment in the current merger, simply remember the slope and the last point.

```
      \def\forest@last@x{#1}%
      \def\forest@last@y{#2}%
      \let\forest@last@atan\forest@current@atan
    \else
```

Otherwise, compare the first and the current slope.

```
      \pgfutil@tempdima=\forest@current@atan pt
      \advance\pgfutil@tempdima -\forest@last@atan pt
      \ifdim\pgfutil@tempdima<0pt\relax
        \multiply\pgfutil@tempdima -1
      \fi
      \ifdim\pgfutil@tempdima<\forest@getedgeofpath@precision\relax
      \else
```

If the slopes differ too much, flush the path up to the previous segment, and set up a new first slope.

```
        \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
        \let\forest@last@atan\forest@current@atan
      \fi
```

In any event, update the last point.

```
      \def\forest@last@x{#1}%
      \def\forest@last@y{#2}%
    \fi
  \fi
}
```

9.4 Get rectangle/band edge

```
\def\forest@getnegativerectangleedgeofpath#1#2{%
  \forest@getnegativerectangleorbandedgeofpath{#1}{#2}{\the\pgf@xb}}
\def\forest@getpositiverectangleedgeofpath#1#2{%
  \forest@getpositiverectangleorbandedgeofpath{#1}{#2}{\the\pgf@xb}}
\def\forest@getbothrectangleedgesofpath#1#2#3{%
  \forest@getbothrectangleorbandedgesofpath{#1}{#2}{#3}{\the\pgf@xb}}
\def\forest@bandlength{5000pt} % something large (ca. 180cm), but still manageable for TeX without producing
\def\forest@getnegativebandedgeofpath#1#2{%
  \forest@getnegativerectangleorbandedgeofpath{#1}{#2}{\forest@bandlength}}
\def\forest@getpositivebandedgeofpath#1#2{%
  \forest@getpositiverectangleorbandedgeofpath{#1}{#2}{\forest@bandlength}}
```

```
\def\forest@getbothbandedgesofpath#1#2#3{%
  \forest@getbothrectangleorbandedgesofpath{#1}{#2}{#3}{\forest@bandlength}}
\def\forest@getnegativerectangleorbandedgeofpath#1#2#3{%
  \forest@path@getboundingrectangle@ls#1{\forest@grow}%
  \edef\forest@gre@path{%
    \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@ya}%
    \noexpand\pgfsyssoftpath@linetotoken{#3}{\the\pgf@ya}%
  }%
  {%
    \pgftransformreset
    \forest@pgfqtransformrotate{\forest@grow}%
    \forest@pgfpathtransformed\forest@gre@path
  }%
  \pgfsyssoftpath@getcurrentpath#2%
}
\def\forest@getpositiverectangleorbandedgeofpath#1#2#3{%
  \forest@path@getboundingrectangle@ls#1{\forest@grow}%
  \edef\forest@gre@path{%
    \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@yb}%
    \noexpand\pgfsyssoftpath@linetotoken{#3}{\the\pgf@yb}%
  }%
  {%
    \pgftransformreset
    \forest@pgfqtransformrotate{\forest@grow}%
    \forest@pgfpathtransformed\forest@gre@path
  }%
  \pgfsyssoftpath@getcurrentpath#2%
}
\def\forest@getbothrectangleorbandedgesofpath#1#2#3#4{%
  \forest@path@getboundingrectangle@ls#1{\forest@grow}%
  \edef\forest@gre@negpath{%
    \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@ya}%
    \noexpand\pgfsyssoftpath@linetotoken{#4}{\the\pgf@ya}%
  }%
  \edef\forest@gre@pospath{%
    \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@yb}%
    \noexpand\pgfsyssoftpath@linetotoken{#4}{\the\pgf@yb}%
  }%
  {%
    \pgftransformreset
    \forest@pgfqtransformrotate{\forest@grow}%
    \forest@pgfpathtransformed\forest@gre@negpath
  }%
  \pgfsyssoftpath@getcurrentpath#2%
  {%
    \pgftransformreset
    \forest@pgfqtransformrotate{\forest@grow}%
    \forest@pgfpathtransformed\forest@gre@pospath
  }%
  \pgfsyssoftpath@getcurrentpath#3%
}
```

9.5 Distance between paths

Another crucial part of the package.

```
\newtoks\forest@PIi@toks
\newtoks\forest@PIii@toks
\def\forest@distance@between@edge@paths#1#2#3{%
  \begingroup
  % #1, #2 = (edge) paths
  %
  % project paths
  \forest@projectpathtogrowline#1{forest@p1@}%
  \forest@projectpathtogrowline#2{forest@p2@}%
  % merge projections (the lists are sorted already, because edge
  % paths are |sorted|)
  \forest@dbep@mergeprojections
    {forest@p1@}{forest@p2@}%
    {forest@P1@}{forest@P2@}%
  % process projections
  \forest@processprojectioninfo{forest@P1@}{forest@PI1@}\forest@PIi@toks
  \forest@processprojectioninfo{forest@P2@}{forest@PI2@}\forest@PIii@toks
  % break paths
  \forest@breakpath#1{forest@PI1@}\forest@PIi@toks\forest@broken@one
  \forest@breakpath#2{forest@PI2@}\forest@PIii@toks\forest@broken@two
  % sort inner arrays ---optimize: it's enough to find max and min
  \forest@sort@inner@arrays{forest@PI1@}\forest@sort@descending
  \forest@sort@inner@arrays{forest@PI2@}\forest@sort@ascending
  % compute the distance
  \let\forest@distance\relax
  \c@pgf@countc=0
  \forest@loop
  \ifnum\c@pgf@countc<\csname forest@PI1@n\endcsname\relax
    \ifnum\csname forest@PI1@\the\c@pgf@countc @n\endcsname=0 \else
      \ifnum\csname forest@PI2@\the\c@pgf@countc @n\endcsname=0 \else
        \pgfutil@tempdima=\csname forest@PI2@\the\c@pgf@countc @0d\endcsname\relax
        \advance\pgfutil@tempdima -\csname forest@PI1@\the\c@pgf@countc @0d\endcsname\relax
        \ifx\forest@distance\relax
          \edef\forest@distance{\the\pgfutil@tempdima}%
        \else
          \ifdim\pgfutil@tempdima<\forest@distance\relax
            \edef\forest@distance{\the\pgfutil@tempdima}%
          \fi
        \fi
      \fi
    \fi
    \advance\c@pgf@countc 1
  \forest@repeat
  \global\let\forest@global@temp\forest@distance
  \endgroup
  \let#3\forest@global@temp
}
  % merge projections: we need two projection arrays, both containing
  % projection points from both paths, but each with the original
  % points from only one path
\def\forest@dbep@mergeprojections#1#2#3#4{%
  % TODO: optimize: v bistvu ni treba sortirat, ker je edge path e sortiran
  \forest@sortprojections{#1}%
  \forest@sortprojections{#2}%
  \c@pgf@counta=0
  \c@pgf@countb=0
  \c@pgf@countc=0
  \edef\forest@input@prefix@one{#1}%
  \edef\forest@input@prefix@two{#2}%
  \edef\forest@output@prefix@one{#3}%
  \edef\forest@output@prefix@two{#4}%
  \forest@dbep@mp@iterate
  \csedef{#3n}{\the\c@pgf@countc}%
  \csedef{#4n}{\the\c@pgf@countc}%
}
\def\forest@dbep@mp@iterate{%
  \let\forest@dbep@mp@next\forest@dbep@mp@iterate
  \ifnum\c@pgf@counta<\csname\forest@input@prefix@one n\endcsname\relax
    \ifnum\c@pgf@countb<\csname\forest@input@prefix@two n\endcsname\relax
      \let\forest@dbep@mp@next\forest@dbep@mp@do
    \else
      \let\forest@dbep@mp@next\forest@dbep@mp@iteratefirst
    \fi
  \else
    \ifnum\c@pgf@countb<\csname\forest@input@prefix@two n\endcsname\relax
      \let\forest@dbep@mp@next\forest@dbep@mp@iteratesecond
    \else
      \let\forest@dbep@mp@next\relax
    \fi
  \fi
  \forest@dbep@mp@next
}
\def\forest@dbep@mp@do{%
  \forest@sort@cmptwodimcs%
    {\forest@input@prefix@one\the\c@pgf@counta xp}%
    {\forest@input@prefix@one\the\c@pgf@counta yp}%
    {\forest@input@prefix@two\the\c@pgf@countb xp}%
    {\forest@input@prefix@two\the\c@pgf@countb yp}%
  \if\forest@sort@cmp@result=%
    \forest@dbep@mp@@store@p\forest@input@prefix@one\c@pgf@counta
    \forest@dbep@mp@@store@o\forest@input@prefix@one
        \c@pgf@counta\forest@output@prefix@one
    \forest@dbep@mp@@store@o\forest@input@prefix@two
        \c@pgf@countb\forest@output@prefix@two
    \advance\c@pgf@counta 1
    \advance\c@pgf@countb 1
  \else
    \if\forest@sort@cmp@result>%
      \forest@dbep@mp@@store@p\forest@input@prefix@two\c@pgf@countb
      \forest@dbep@mp@@store@o\forest@input@prefix@two
          \c@pgf@countb\forest@output@prefix@two
      \advance\c@pgf@countb 1
    \else%<
      \forest@dbep@mp@@store@p\forest@input@prefix@one\c@pgf@counta
      \forest@dbep@mp@@store@o\forest@input@prefix@one
          \c@pgf@counta\forest@output@prefix@one
      \advance\c@pgf@counta 1
    \fi
  \fi
  \advance\c@pgf@countc 1
  \forest@dbep@mp@iterate
}
\def\forest@dbep@mp@@store@p#1#2{%
  \csletcs
    {\forest@output@prefix@one\the\c@pgf@countc xp}%
    {#1\the#2xp}%
  \csletcs
    {\forest@output@prefix@one\the\c@pgf@countc yp}%
    {#1\the#2yp}%
  \csletcs
    {\forest@output@prefix@two\the\c@pgf@countc xp}%
    {#1\the#2xp}%
  \csletcs
    {\forest@output@prefix@two\the\c@pgf@countc yp}%
    {#1\the#2yp}%
}
\def\forest@dbep@mp@@store@o#1#2#3{%
  \csletcs{#3\the\c@pgf@countc xo}{#1\the#2xo}%
  \csletcs{#3\the\c@pgf@countc yo}{#1\the#2yo}%
}
\def\forest@dbep@mp@iteratefirst{%
  \forest@dbep@mp@iterateone\forest@input@prefix@one\c@pgf@counta\forest@output@prefix@one
}
\def\forest@dbep@mp@iteratesecond{%
  \forest@dbep@mp@iterateone\forest@input@prefix@two\c@pgf@countb\forest@output@prefix@two
}
\def\forest@dbep@mp@iterateone#1#2#3{%
  \forest@loop
  \ifnum#2<\csname#1n\endcsname\relax
    \forest@dbep@mp@@store@p#1#2%
    \forest@dbep@mp@@store@o#1#2#3%
    \advance\c@pgf@countc 1
    \advance#21
  \forest@repeat
}
```

9.6 Utilities

Equality test: points are considered equal if they differ less than `\pgfintersectiontolerance` in each coordinate.

```
\newif\ifforest@equaltotolerance
\def\forest@equaltotolerance#1#2{{%
  \pgfpointdiff{#1}{#2}%
  \ifdim\pgf@x<0pt \multiply\pgf@x -1 \fi
  \ifdim\pgf@y<0pt \multiply\pgf@y -1 \fi
  \global\forest@equaltotolerancefalse
  \ifdim\pgf@x<\pgfintersectiontolerance\relax
    \ifdim\pgf@y<\pgfintersectiontolerance\relax
      \global\forest@equaltotolerancetrue
    \fi
  \fi
}}
```

Save/restore pgfs `\pgfsyssoftpath@...token` definitions.

```
\def\forest@save@pgfsyssoftpath@tokendefs{%
  \let\forest@origmovetotoken\pgfsyssoftpath@movetotoken
  \let\forest@origlinetotoken\pgfsyssoftpath@linetotoken
  \let\forest@origcurvetosupportatoken\pgfsyssoftpath@curvetosupportatoken
  \let\forest@origcurvetosupportbtoken\pgfsyssoftpath@curvetosupportbtoken
  \let\forest@origcurvetotoken\pgfsyssoftpath@curvetototoken
  \let\forest@origrectcornertoken\pgfsyssoftpath@rectcornertoken
  \let\forest@origrectsizetoken\pgfsyssoftpath@rectsizetoken
  \let\forest@origclosepathtoken\pgfsyssoftpath@closepathtoken
  \let\pgfsyssoftpath@movetotoken\forest@badtoken
  \let\pgfsyssoftpath@linetotoken\forest@badtoken
  \let\pgfsyssoftpath@curvetosupportatoken\forest@badtoken
  \let\pgfsyssoftpath@curvetosupportbtoken\forest@badtoken
  \let\pgfsyssoftpath@curvetototoken\forest@badtoken
  \let\pgfsyssoftpath@rectcornertoken\forest@badtoken
  \let\pgfsyssoftpath@rectsizetoken\forest@badtoken
  \let\pgfsyssoftpath@closepathtoken\forest@badtoken
}
\def\forest@badtoken{%
  \PackageError{forest}{This token should not be in this path}{}%
}
\def\forest@restore@pgfsyssoftpath@tokendefs{%
  \let\pgfsyssoftpath@movetotoken\forest@origmovetotoken
  \let\pgfsyssoftpath@linetotoken\forest@origlinetotoken
  \let\pgfsyssoftpath@curvetosupportatoken\forest@origcurvetosupportatoken
  \let\pgfsyssoftpath@curvetosupportbtoken\forest@origcurvetosupportbtoken
  \let\pgfsyssoftpath@curvetototoken\forest@origcurvetotoken
  \let\pgfsyssoftpath@rectcornertoken\forest@origrectcornertoken
  \let\pgfsyssoftpath@rectsizetoken\forest@origrectsizetoken
  \let\pgfsyssoftpath@closepathtoken\forest@origclosepathtoken
}
```

Extend path `#1` with path `#2` translated by point `#3`.

```
\def\forest@extendpath#1#2#3{%
  \pgf@process{#3}%
  \pgfsyssoftpath@setcurrentpath#1%
  \forest@save@pgfsyssoftpath@tokendefs
  \let\pgfsyssoftpath@movetotoken\forest@extendpath@moveto
  \let\pgfsyssoftpath@linetotoken\forest@extendpath@lineto
  #2%
  \forest@restore@pgfsyssoftpath@tokendefs
  \pgfsyssoftpath@getcurrentpath#1%
}
\def\forest@extendpath@moveto#1#2{%
  \forest@extendpath@do{#1}{#2}\pgfsyssoftpath@moveto
}
\def\forest@extendpath@lineto#1#2{%
  \forest@extendpath@do{#1}{#2}\pgfsyssoftpath@lineto
}
\def\forest@extendpath@do#1#2#3{%
  {%
    \advance\pgf@x #1
    \advance\pgf@y #2
    #3{\the\pgf@x}{\the\pgf@y}%
  }%
}
```

Get bounding rectangle of the path. `#1` = the path, `#2` = grow. Returns (`\pgf@xa`=min x/l, `\pgf@ya`=max y/s, `\pgf@xb`=min x/l, `\pgf@yb`=max y/s). (If path `#1` is empty, the result is undefined.)

```
\def\forest@path@getboundingrectangle@ls#1#2{%
  {%
    \pgftransformreset
    \forest@pgfqtransformrotate{-#2}%
    \forest@pgfpathtransformed#1%
  }%
  \pgfsyssoftpath@getcurrentpath\forest@gbr@rotatedpath
  \forest@path@getboundingrectangle@xy\forest@gbr@rotatedpath
}
\def\forest@path@getboundingrectangle@xy#1{%
  \forest@save@pgfsyssoftpath@tokendefs
  \let\pgfsyssoftpath@movetotoken\forest@gbr@firstpoint
  \let\pgfsyssoftpath@linetotoken\forest@gbr@firstpoint
  #1%
  \forest@restore@pgfsyssoftpath@tokendefs
}
\def\forest@gbr@firstpoint#1#2{%
  \pgf@xa=#1 \pgf@xb=#1 \pgf@ya=#2 \pgf@yb=#2
  \let\pgfsyssoftpath@movetotoken\forest@gbr@point
  \let\pgfsyssoftpath@linetotoken\forest@gbr@point
}
\def\forest@gbr@point#1#2{%
  \ifdim#1<\pgf@xa\relax\pgf@xa=#1 \fi
  \ifdim#1>\pgf@xb\relax\pgf@xb=#1 \fi
  \ifdim#2<\pgf@ya\relax\pgf@ya=#2 \fi
  \ifdim#2>\pgf@yb\relax\pgf@yb=#2 \fi
}
```

Hack: create our own version of pgf's `\pgftransformrotate` which does not call `\pgfmathparse`. Nothing really bad happens if patch fails. We're just a bit slower.

```
\let\forest@pgfqtransformrotate\pgftransformrotate
\let\forest@pgftransformcm\pgftransformcm
\let\forest@pgf@transformcm\pgf@transformcm
\patchcmd{\forest@pgfqtransformrotate}{\pgfmathparse{#1}}{\edef\pgfmathresult{\number\numexpr#1}}{}{}
\patchcmd{\forest@pgfqtransformrotate}{\pgftransformcm}{\forest@pgftransformcm}{}{}
\patchcmd{\forest@pgftransformcm}{\pgf@transformcm}{\forest@pgf@transformcm}{}{}
\patchcmd{\forest@pgf@transformcm}{\pgfmathsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x
\patchcmd{\forest@pgf@transformcm}{\pgfmathsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x
\patchcmd{\forest@pgf@transformcm}{\pgfmathsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x
\patchcmd{\forest@pgf@transformcm}{\pgfmathsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x
\def\forest@pgf@transformcm@setlength#1#2{#1=#2pt}
```

10 The outer UI

10.1 Externalization

```
\pgfkeys{/forest/external/.cd,
  %copy command/.initial={cp "\source" "\target"},
  copy command/.initial={},
  optimize/.is if=forest@external@optimize@,
  context/.initial={%
    \forestOve{\csname forest@id@of@standard node\endcsname}{environment@formula}},
  depends on macro/.style={context/.append/.expanded={%
      \expandafter\detokenize\expandafter{#1}}},
}
\def\forest@file@copy#1#2{%
  \IfFileExists{#1}{%
    \pgfkeysgetvalue{/forest/external/copy command}\forest@copy@command
    \ifdefempty\forest@copy@command{%
      \forest@file@copy@{#1}{#2}%
    }{ % copy by external command
      \def\source{#1}%
      \def\target{#2}%
      \immediate\write18{\forest@copy@command}%
    }%
  }{}%
}
\newread\forest@copy@in
\newwrite\forest@copy@out
\def\forest@file@copy@#1#2{%
  \begingroup
  \openin\forest@copy@in=#1
  \immediate\openout\forest@copy@out#2
  \endlinechar-1
  \loop
  \unless\ifeof\forest@copy@in
    \readline\forest@copy@in to\forest@temp
    \immediate\write\forest@copy@out{\forest@temp}%
  \repeat
  \immediate\closeout\forest@copy@out
  \closein\forest@copy@in
  \endgroup
}
\newif\ifforest@external@optimize@
\forest@external@optimize@true
\ifforest@install@keys@to@tikz@path@
\tikzset{
  fit to/.style={
    /forest/for nodewalk=%
      {TeX={\def\forest@fitto{}},#1}%
      {TeX={\eappto\forest@fitto{(\forestove{name})}}},
    fit/.expanded={\forest@fitto}
  },
}
\fi
\ifforest@external@
  \ifdefined\tikzexternal@tikz@replacement\else
    \usetikzlibrary{external}%
  \fi
  \pgfkeys{%
    /tikz/external/failed ref warnings for={},
    /pgf/images/aux in dpth=false,
  }%
  \tikzifexternalizing{}{%
    \forest@file@copy{\jobname.aux}{\jobname.aux.copy}%
  }%
  \AtBeginDocument{%
    \tikzifexternalizing{%
      \IfFileExists{\tikzexternalrealjob.aux.copy}{%
        \makeatletter
        \input\tikzexternalrealjob.aux.copy\relax
        \makeatother
      }{}%
    }{%
      \newwrite\forest@auxout
      \immediate\openout\forest@auxout=\tikzexternalrealjob.for.tmp
    }%
    \IfFileExists{\tikzexternalrealjob.for}{%
      {%
        \makehashother\makeatletter
        \input\tikzexternalrealjob.for\relax
      }%
    }{}%
  }%
  \AtEndDocument{%
    \tikzifexternalizing{}{%
      \immediate\closeout\forest@auxout
      \forest@file@copy{\jobname.for.tmp}{\jobname.for}%
    }%
  }%
\fi
```

10.2 The `forest` environment

There are three ways to invoke FOREST: the environment and the starless and the starred version of the macro. The latter creates no group.

Most of the code in this section deals with externalization.

```
\NewDocumentEnvironment{forest}{D(){}}{%
  \forest@config{#1}%
  \Collect@Body
  \forest@env
}{}
\NewDocumentCommand{\Forest}{s D(){} m}{%
  \forest@config{#2}%
  \IfBooleanTF{#1}{\let\forest@next\forest@env}{\let\forest@next\forest@group@env}%
  \forest@next{#3}%
}
\def\forest@config#1{%
  \forest@defstages{stages}%
  \forestset{@config/.cd,#1}%
}
\def\forest@defstages#1{%
  \def\forest@stages{#1}%
}
\forestset{@config/.cd,
  %stages/.store in=\forest@stages,
  stages/.code={\forest@defstages{#1}},
  .unknown/.code={\PackageError{forest}{Unknown config option for forest environment/command.}{In Forest v2.0
}
\def\forest@group@env#1{{\forest@env{#1}}}
\newif\ifforest@externalize@tree@
\newif\ifforest@was@tikzexternalwasenable
\newcommand\forest@env[1]{%
  \let\forest@external@next\forest@begin
  \forest@was@tikzexternalwasenablefalse
  \ifdefined\tikzexternal@tikz@replacement
    \ifx\tikz\tikzexternal@tikz@replacement
      \forest@was@tikzexternalwasenabletrue
      \tikzexternaldisable
    \fi
  \fi
  \forest@externalize@tree@false
  \ifforest@external@
    \ifforest@was@tikzexternalwasenable
      \forest@env@
    \fi
  \fi
  \forest@standardnode@calibrate
  \forest@external@next{#1}%
}
\def\forest@env@{%
  \iftikzexternalexportnext
    \tikzifexternalizing{%
      \let\forest@external@next\forest@begin@externalizing
    }{%
      \let\forest@external@next\forest@begin@externalize
    }%
  \else
    \tikzexternalexportnexttrue
  \fi
}
```

We're externalizing, i.e. this code gets executed in the embedded call.

```
\long\def\forest@begin@externalizing#1{%
  \forest@external@setup{#1}%
  \let\forest@external@next\forest@begin
  \forest@externalize@inner@n=-1
  \ifforest@external@optimize@\forest@externalizing@maybeoptimize\fi
  \forest@external@next{#1}%
  \tikzexternalenable
}
\def\forest@externalizing@maybeoptimize{%
  \edef\forest@temp{\tikzexternalrealjob-forest-\forest@externalize@outer@n}%
  \edef\forest@marshal{%
    \noexpand\pgfutil@in@
      {\expandafter\detokenize\expandafter{\forest@temp}.}
      {\expandafter\detokenize\expandafter{\pgfactualjobname}.}%
  }\forest@marshal
  \ifpgfutil@in@
  \else
```

```
    \let\forest@external@next\@gobble
  \fi
}
```

Externalization is enabled, we're in the outer process, deciding if the picture is up-to-date.

```
\long\def\forest@begin@externalize#1{%
  \forest@external@setup{#1}%
  \iftikzexternal@file@isuptodate
    \setbox0=\hbox{%
      \csname forest@externalcheck@\forest@externalize@outer@n\endcsname
    }%
  \fi
  \iftikzexternal@file@isuptodate
    \csname forest@externalload@\forest@externalize@outer@n\endcsname
  \else
    \forest@externalize@tree@true
    \forest@externalize@inner@n=-1
    \forest@begin{#1}%
    \ifcsdef{forest@externalize@@\forest@externalize@id}{}{%
      \immediate\write\forest@auxout{%
        \noexpand\forest@external
        {\forest@externalize@outer@n}%
        {\expandafter\detokenize\expandafter{\forest@externalize@id}}%
        {\expandonce\forest@externalize@checkimages}%
        {\expandonce\forest@externalize@loadimages}%
      }%
    }%
  \fi
  \tikzexternalenable
}
\def\forest@includeexternal@check#1{%
  \tikzsetnextfilename{#1}%
  \IfFileExists{\tikzexternal@filenameprefix/#1}{\tikzexternal@file@isuptodatetrue}{\tikzexternal@file@isupto
}
\def\makehashother{\catcode`\#=12}%
\long\def\forest@external@setup#1{%
  % set up \forest@externalize@id and \forest@externalize@outer@n
  % we need to deal with #s correctly (\write doubles them)
  \setbox0=\hbox{\makehashother\makeatletter
    \scantokens{\forest@temp@toks{#1}}\expandafter
  }%
  \expandafter\forest@temp@toks\expandafter{\the\forest@temp@toks}%
  \edef\forest@temp{\pgfkeysvalueof{/forest/external/context}}%
  \edef\forest@externalize@id{%
    \expandafter\detokenize\expandafter{\forest@temp}%
    @@%
    \expandafter\detokenize\expandafter{\the\forest@temp@toks}%
  }%
  \letcs\forest@externalize@outer@n{forest@externalize@@\forest@externalize@id}%
  \ifdefined\forest@externalize@outer@n
    \global\tikzexternal@file@isuptodatetrue
  \else
    \global\advance\forest@externalize@max@outer@n 1
    \edef\forest@externalize@outer@n{\the\forest@externalize@max@outer@n}%
    \global\tikzexternal@file@isuptodatefalse
  \fi
  \def\forest@externalize@loadimages{}%
  \def\forest@externalize@checkimages{}%
}
\newcount\forest@externalize@max@outer@n
\global\forest@externalize@max@outer@n=0
```

```
\newcount\forest@externalize@inner@n
```

The `.for` file is a string of calls of this macro.

```
\long\def\forest@external#1#2#3#4{% #1=n,#2=context+source code,#3=update check code, #4=load code
  \ifnum\forest@externalize@max@outer@n<#1
    \global\forest@externalize@max@outer@n=#1
  \fi
  \global\csdef{forest@externalize@@\detokenize{#2}}{#1}%
  \global\csdef{forest@externalcheck@#1}{#3}%
  \global\csdef{forest@externalload@#1}{#4}%
  \tikzifexternalizing{}{%
    \immediate\write\forest@auxout{%
      \noexpand\forest@external{#1}%
      {\expandafter\detokenize\expandafter{#2}}%
      {\unexpanded{#3}}%
      {\unexpanded{#4}}%
    }%
  }%
}
```

These two macros include the external picture.

```
\def\forest@includeexternal#1{%
  \edef\forest@temp{\pgfkeysvalueof{/forest/external/context}}%
  %\typeout{forest: Including external picture '#1' for forest context+code: '\expandafter\detokenize\expanda
  {%
    %\def\pgf@declaredraftimage##1##2{\def\pgf@image{\hbox{}}}%
    \tikzsetnextfilename{#1}%
    \tikzexternalenable
    \tikz{}%
  }%
}
\def\forest@includeexternal@box#1#2{%
  \global\setbox#1=\hbox{\forest@includeexternal{#2}}%
}
```

This code runs the bracket parser and stage processing.

```
\long\def\forest@begin#1{%
  \iffalse{\fi\forest@parsebracket#1}%
}
\def\forest@parsebracket{%
  \bracketParse{\forest@get@root@afterthought}\forest@root=%
}
\def\forest@get@root@afterthought{%
  \expandafter\forest@get@root@afterthought@\expandafter{\iffalse}\fi
}
\long\def\forest@get@root@afterthought@#1{%
  \ifblank{#1}{}{%
    \forestOeappto{\forest@root}{given options}{,afterthought={\unexpanded{#1}}}%
  }%
  \forest@do
}
\def\forest@do{%
  \forest@node@Compute@numeric@ts@info{\forest@root}%
  \expandafter\forestset\expandafter{\forest@stages}%
  \ifforest@was@tikzexternalwasenable
    \tikzexternalenable
  \fi
}
```

10.3 Standard node

The standard node should be calibrated when entering the forest env: The standard node init does *not* initialize options from a(nother) standard node!

```
\def\forest@standardnode@new{%
  \advance\forest@node@maxid1
  \forest@fornode{\the\forest@node@maxid}{%
    \forest@node@init
    \forestoeset{id}{\forest@cn}%
    \forest@node@setname@silent{standard node}%
  }%
}
\def\forest@standardnode@calibrate{%
  \forest@fornode{\forest@node@Nametoid{standard node}}{%
    \edef\forest@environment{\forestove{environment@formula}}%
    \forestoget{previous@environment}\forest@previous@environment
    \ifx\forest@environment\forest@previous@environment\else
      \forestolet{previous@environment}\forest@environment
      \forest@node@typeset
      \forestoget{calibration@procedure}\forest@temp
      \expandafter\forestset\expandafter{\forest@temp}%
    \fi
  }%
}
```

Usage: `\forestStandardNode[#1]{#2}{#3}{#4}`. `#1` = standard node specification — specify it as any other node content (but without children, of course). `#2` = the environment fingerprint: list the values of parameters that influence the standard node's height and depth; the standard will be adjusted whenever any of these parameters changes. `#3` = the calibration procedure: a list of usual forest options which should calculating the values of exported options. `#4` = a comma-separated list of exported options: every newly created node receives the initial values of exported options from the standard node. (The standard node definition is local to the TeX group.)

```
\def\forestStandardNode[#1]#2#3#4{%
  \let\forest@standardnode@restoretikzexternal\relax
  \ifdefined\tikzexternaldisable
    \ifx\tikz\tikzexternal@tikz@replacement
      \tikzexternaldisable
      \let\forest@standardnode@restoretikzexternal\tikzexternalenable
    \fi
  \fi
  \forest@standardnode@new
  \forest@fornode{\forest@node@Nametoid{standard node}}{%
    \forestset{content=#1}%
    \forestoset{environment@formula}{#2}%
    \edef\forest@temp{\unexpanded{#3}}%
    \forestolet{calibration@procedure}\forest@temp
    \def\forest@calibration@initializing@code{}%
    \pgfqkeys{/forest/initializing@code}{#4}%
    \forestolet{initializing@code}\forest@calibration@initializing@code
    \forest@standardnode@restoretikzexternal
  }
}
\forestset{initializing@code/.unknown/.code={%
    \eappto\forest@calibration@initializing@code{%
      \noexpand\forestOget{\forest@node@Nametoid{standard node}}{\pgfkeyscurrentname}\noexpand\forest@temp
      \noexpand\forestolet{\pgfkeyscurrentname}\noexpand\forest@temp
    }%
  }
}
```

This macro is called from a new (non-standard) node's init.

```
\def\forest@initializefromstandardnode{%
  \forestOve{\forest@node@Nametoid{standard node}}{initializing@code}%
}
```

Define the default standard node. Standard content: dj — in Computer Modern font, d is the highest and j the deepest letter (not character!). Environment fingerprint: the height of the strut and the values of inner and outer seps. Calibration procedure: (i) `l sep` equals the height of the strut plus the value of `inner ysep`, implementing both font-size and inner sep dependency; (ii) The effect of `l` on the standard node should be the same as the effect of `l sep`, thus, we derive `l` from `l sep` by adding to the latter the total height of the standard node (plus the double outer sep, one for the parent and one for the child). (iii) s sep is straightforward: a double inner xsep. Exported options: options, calculated in the calibration. (Tricks: to change the default anchor, set it in `#1` and export it; to set a non-forest node option (such as `draw` or `blue`) as default, set it in `#1` and export the (internal) option `node options`.)

```
\forestStandardNode[dj]
  {%
    \forestOve{\forest@node@Nametoid{standard node}}{content},%
    \the\ht\strutbox,\the\pgflinewidth,%
    \pgfkeysvalueof{/pgf/inner ysep},\pgfkeysvalueof{/pgf/outer ysep},%
    \pgfkeysvalueof{/pgf/inner xsep},\pgfkeysvalueof{/pgf/outer xsep}%
  }
  {
    l sep'/.expanded={\the\dimexpr\the\ht\strutbox+\pgfkeysvalueof{/pgf/inner ysep}},
    l={l_sep()+abs(max_y()-min_y())+2*\pgfkeysvalueof{/pgf/outer ysep}},
    s sep'/.expanded={\the\dimexpr \pgfkeysvalueof{/pgf/inner xsep}*2}
  }
  {l sep,l,s sep}
```

10.4 `ls` coordinate system

```
\pgfqkeys{/forest/@cs}{%
  name/.code={%
    \edef\forest@cn{\forest@node@Nametoid{#1}}%
    \forest@forestcs@resetxy},
  id/.code={%
    \edef\forest@cn{#1}%
    \forest@forestcs@resetxy},
  go/.code={%
    \forest@go{#1}%
    \forest@forestcs@resetxy},
  anchor/.code={\forest@forestcs@anchor{#1}},
  l/.code={%
    \forestmathsetlengthmacro\forest@forestcs@l{#1}%
    \forest@forestcs@ls
  },
  s/.code={%
    \forestmathsetlengthmacro\forest@forestcs@s{#1}%
    \forest@forestcs@ls
  },
  .unknown/.code={%
    \expandafter\pgfutil@in@\expandafter.\expandafter{\pgfkeyscurrentname}%
    \ifpgfutil@in@
      \expandafter\forest@forestcs@namegoanchor\pgfkeyscurrentname\forest@end
    \else
      \expandafter\forest@nameandgo\expandafter{\pgfkeyscurrentname}%
      \forest@forestcs@resetxy
    \fi
  }
}
\def\forest@forestcs@resetxy{%
  \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
  \global\pgf@x\forestove{x}\relax
  \global\pgf@y\forestove{y}\relax
}
\def\forest@forestcs@ls{%
  \ifdefined\forest@forestcs@l
    \ifdefined\forest@forestcs@s
      {%
        \pgftransformreset
        \forest@pgfqtransformrotate{\forestove{grow}}%
        \pgfpointtransformed{\pgfqpoint{\forest@forestcs@l}{\forest@forestcs@s}}%
      }%
      \global\advance\pgf@x\forestove{x}%
      \global\advance\pgf@y\forestove{y}%
    \fi
  \fi
}
\def\forest@forestcs@anchor#1{%
  \edef\forest@marshal{%
    \noexpand\forest@original@tikz@parse@node\relax
    (\forestove{name}\ifx\relax#1\relax\else.\fi#1)%
  }\forest@marshal
}
\def\forest@forestcs@namegoanchor#1.#2\forest@end{%
  \forest@nameandgo{#1}%
  \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
  \forest@forestcs@anchor{#2}%
}
\def\forest@cs@invalidnodeerror{%
  \PackageError{forest}{Attempt to refer to the invalid node by "forest cs"}{}%
}
\tikzdeclarecoordinatesystem{forest}{%
  \forest@forthis{%
    \forest@forestcs@resetxy
    \ifdefined\forest@forestcs@l\undef\forest@forestcs@l\fi
    \ifdefined\forest@forestcs@s\undef\forest@forestcs@s\fi
    \pgfqkeys{/forest/@cs}{#1}%
  }%
}
```

10.5 Relative node names in Ti*k*Z

A hack into Ti*k*Z's coordinate parser: implements relative node names!

```
\def\forest@tikz@parse@node#1(#2){%
  \pgfutil@in@.{#2}%
  \ifpgfutil@in@
    \expandafter\forest@tikz@parse@node@checkiftikzname@withdot
  \else%
    \expandafter\forest@tikz@parse@node@checkiftikzname@withoutdot
  \fi%
  #1(#2)\forest@end
}
\def\forest@tikz@parse@node@checkiftikzname@withdot#1(#2.#3)\forest@end{%
  \forest@tikz@parse@node@checkiftikzname#1{#2}{.#3}}
\def\forest@tikz@parse@node@checkiftikzname@withoutdot#1(#2)\forest@end{%
  \forest@tikz@parse@node@checkiftikzname#1{#2}{}}
\def\forest@tikz@parse@node@checkiftikzname#1#2#3{%
  \expandafter\ifx\csname pgf@sh@ns@#2\endcsname\relax
    \forest@forthis{%
      \forest@nameandgo{#2}%
      \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
      \edef\forest@temp@relativenodename{\forestove{name}}%
    }%
  \else
    \def\forest@temp@relativenodename{#2}%
  \fi
  \expandafter\forest@original@tikz@parse@node\expandafter#1\expandafter(\forest@temp@relativenodename#3)%
}
\def\forest@nameandgo#1{%
  \pgfutil@in@!{#1}%
  \ifpgfutil@in@
    \forest@nameandgo@(#1)%
  \else
    \ifstrempty{#1}{}{\edef\forest@cn{\forest@node@Nametoid{#1}}}%
  \fi
}
\def\forest@nameandgo@(#1!#2){%
  \ifstrempty{#1}{}{\edef\forest@cn{\forest@node@Nametoid{#1}}}%
  \forest@go{#2}%
}
```

10.6 Anchors

Forest anchors are `(child/parent)_anchor` and growth anchors `parent/children_first/last`. The following code resolves them into Ti*k*Z anchors, based on the value of option `(child/parent)_anchor` and values of `grow` and `reversed`.

We need to access `rotate` for the anchors below to work in general.

```
\forestset{
  declare count={rotate}{0},
  autoforward'={rotate}{node options},
}
```

Variants of `parent/children_first/last` without `'` snap border anchors to the closest compass direction.

```
\newif\ifforest@anchor@snapbordertocompass
```

The code is used both in generic anchors (then, the result should be forwarded to Ti*k*Z for evaluation into coordinates) and in the UI macro `\forestanchortotikzanchor`.

```
\newif\ifforest@anchor@forwardtotikz
```

Growth-based anchors set this to true to signal that the result is a border anchor.

```
\newif\ifforest@anchor@isborder
```

The UI macro.

```
\def\forestanchortotikzanchor#1#2{% #1 = forest anchor, #2 = macro to receive the tikz anchor
  \forest@anchor@forwardtotikzfalse
  \forest@anchor@do{}{#1}{\forest@cn}%
  \let#2\forest@temp@anchor
}
```

Generic anchors.

```
\pgfdeclaregenericanchor{child anchor}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{child anchor}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{parent anchor}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{parent anchor}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{anchor}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{anchor}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{children}{%
```

```
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{children}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{-children}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{-children}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{children first}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{children first}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{-children first}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{-children first}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{first}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{first}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{parent first}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{parent first}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{-parent first}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{-parent first}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{parent}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{parent}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{-parent}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{-parent}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{parent last}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{parent last}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{-parent last}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{-parent last}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{last}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{last}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{children last}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{children last}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{-children last}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{-children last}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{children'}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{children'}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{-children'}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{-children'}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{children first'}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{children first'}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{-children first'}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{-children first'}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{first'}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{first'}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{parent first'}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{parent first'}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{-parent first'}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{-parent first'}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{parent'}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{parent'}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{-parent'}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{-parent'}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{parent last'}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{parent last'}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{-parent last'}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{-parent last'}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{last'}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{last'}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{children last'}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{children last'}{\forest@referencednodeid}%
}
\pgfdeclaregenericanchor{-children last'}{%
  \forest@anchor@forwardtotikztrue
  \forest@anchor@do{#1}{-children last'}{\forest@referencednodeid}%
}
```

The driver. The result is being passed around in `\forest@temp@anchor`.

```
\def\forest@anchor@do#1#2#3{% #1 = shape name, #2 = (potentially) forest anchor, #3 = node id
  \forest@fornode{#3}{%
    \def\forest@temp@anchor{#2}%
    \forest@anchor@snapbordertocompassfalse
    \forest@anchor@isborderfalse
    \forest@anchor@to@tikz@anchor
    \forest@anchor@border@to@compass
    \ifforest@anchor@forwardtotikz
      \forest@anchor@forward{#1}%
    \else
    \fi
  }%
}
```

This macro will loop (resolving the anchor) until the result is not a Forest macro.

```
\def\forest@anchor@to@tikz@anchor{%
  \ifcsdef{forest@anchor@@\forest@temp@anchor}{%
    \csuse{forest@anchor@@\forest@temp@anchor}%
    \forest@anchor@to@tikz@anchor
  }{}%
}
```

Actual computation.

```
\csdef{forest@anchor@@parent anchor}{%
  \forestoget{parent anchor}\forest@temp@anchor}
\csdef{forest@anchor@@child anchor}{%
  \forestoget{child anchor}\forest@temp@anchor}
\csdef{forest@anchor@@anchor}{%
  \forestoget{anchor}\forest@temp@anchor}
\csdef{forest@anchor@@children'}{%
  \forest@anchor@isbordertrue
  \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}}%
}
\csdef{forest@anchor@@-children'}{%
  \forest@anchor@isbordertrue
  \edef\forest@temp@anchor{\number\numexpr 180+\forestove{grow}-\forestove{rotate}}%
}
\csdef{forest@anchor@@parent'}{%
  \forest@anchor@isbordertrue
  \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestOve{\forestove{@parent}}{gro
  \edef\forest@temp@anchor{\number\numexpr\forest@temp@grow-\forestove{rotate}+180}%
}
\csdef{forest@anchor@@-parent'}{%
  \forest@anchor@isbordertrue
  \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestOve{\forestove{@parent}}{gro
  \edef\forest@temp@anchor{\number\numexpr\forest@temp@grow-\forestove{rotate}}%
}
\csdef{forest@anchor@@first'}{%
  \forest@anchor@isbordertrue
  \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=0 -\e
}
\csdef{forest@anchor@@last'}{%
  \forest@anchor@isbordertrue
  \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=0 +\e
}
\csdef{forest@anchor@@parent first'}{%
  \forest@anchor@isbordertrue
  \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestOve{\forestove{@parent}}{gro
  \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestOve{\forestove{@pare
  \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}+180}%
  \edef\forest@temp@anchor@first{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@revers
  \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@first}\forest@temp@anchor
}
\csdef{forest@anchor@@-parent first'}{%
  \forest@anchor@isbordertrue
  \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestOve{\forestove{@parent}}{gro
  \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestOve{\forestove{@pare
  \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}}%
  \edef\forest@temp@anchor@first{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@revers
  \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@first}\forest@temp@anchor
}
```

```
\csdef{forest@anchor@@parent last'}{%
  \forest@anchor@isbordertrue
  \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestOve{\forestove{@parent}}{gro
  \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestOve{\forestove{@pare
  \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}+180}%
  \edef\forest@temp@anchor@last{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@reverse
  \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@last}\forest@temp@anchor
}
\csdef{forest@anchor@@-parent last'}{%
  \forest@anchor@isbordertrue
  \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestOve{\forestove{@parent}}{gro
  \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestOve{\forestove{@pare
  \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}}%
  \edef\forest@temp@anchor@last{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@reverse
  \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@last}\forest@temp@anchor
}
\csdef{forest@anchor@@children first'}{%
  \forest@anchor@isbordertrue
  \edef\forest@temp@anchor@first{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}
  \forest@getaverageangle{\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@first}\forest@temp@anchor
}
\csdef{forest@anchor@@-children first'}{%
  \forest@anchor@isbordertrue
  \edef\forest@temp@anchor@first{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}
  \forest@getaverageangle{180+\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@first}\forest@temp@anc
}
\csdef{forest@anchor@@children last'}{%
  \forest@anchor@isbordertrue
  \edef\forest@temp@anchor@last{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=
  \forest@getaverageangle{\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@last}\forest@temp@anchor
}
\csdef{forest@anchor@@-children last'}{%
  \forest@anchor@isbordertrue
  \edef\forest@temp@anchor@last{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=
  \forest@getaverageangle{180+\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@last}\forest@temp@anch
}
\csdef{forest@anchor@@children}{%
  \forest@anchor@snapbordertocompasstrue
  \csuse{forest@anchor@@children'}%
}
\csdef{forest@anchor@@-children}{%
  \forest@anchor@snapbordertocompasstrue
  \csuse{forest@anchor@@-children'}%
}
\csdef{forest@anchor@@parent}{%
  \forest@anchor@snapbordertocompasstrue
  \csuse{forest@anchor@@parent'}%
}
\csdef{forest@anchor@@-parent}{%
  \forest@anchor@snapbordertocompasstrue
  \csuse{forest@anchor@@-parent'}%
}
\csdef{forest@anchor@@first}{%
  \forest@anchor@snapbordertocompasstrue
  \csuse{forest@anchor@@first'}%
}
\csdef{forest@anchor@@last}{%
  \forest@anchor@snapbordertocompasstrue
  \csuse{forest@anchor@@last'}%
}
\csdef{forest@anchor@@parent first}{%
```

```
  \forest@anchor@snapbordertocompasstrue
  \csuse{forest@anchor@@parent first’}%
}
\csdef{forest@anchor@@-parent first}{%
  \forest@anchor@snapbordertocompasstrue
  \csuse{forest@anchor@@-parent first’}%
}
\csdef{forest@anchor@@parent last}{%
  \forest@anchor@snapbordertocompasstrue
  \csuse{forest@anchor@@parent last’}%
}
\csdef{forest@anchor@@-parent last}{%
  \forest@anchor@snapbordertocompasstrue
  \csuse{forest@anchor@@-parent last’}%
}
\csdef{forest@anchor@@children first}{%
  \forest@anchor@snapbordertocompasstrue
  \csuse{forest@anchor@@children first’}%
}
\csdef{forest@anchor@@-children first}{%
  \forest@anchor@snapbordertocompasstrue
  \csuse{forest@anchor@@-children first’}%
}
\csdef{forest@anchor@@children last}{%
  \forest@anchor@snapbordertocompasstrue
  \csuse{forest@anchor@@children last’}%
}
\csdef{forest@anchor@@-children last}{%
  \forest@anchor@snapbordertocompasstrue
  \csuse{forest@anchor@@-children last’}%
}
```

This macro computes the ”average” angle of `#1` and `#2` and stores in into `#3`. The angle computed is the geometrically ”closer” one. The formula is adapted from http://stackoverflow.com/a/1159336/624872.

```
\def\forest@getaverageangle#1#2#3{%
  \edef\forest@temp{\number\numexpr #1-#2+540}%
  \expandafter\pgfmathMod@\expandafter{\forest@temp}{360}%
  \forest@truncatepgfmathresult
  \edef\forest@temp{\number\numexpr 360+#2+((\pgfmathresult-180)/2)}%
  \expandafter\pgfmathMod@\expandafter{\forest@temp}{360}%
  \forest@truncatepgfmathresult
  \let#3\pgfmathresult
}
\def\forest@truncatepgfmathresult{%
  \afterassignment\forest@gobbletoEND
  \forest@temp@count=\pgfmathresult\forest@END
  \def\pgfmathresult{\the\forest@temp@count}%
}
\def\forest@gobbletoEND#1\forest@END{}
```

The first macro changes border anchor to compass anchor. The second one does this only if the node shape allows it.

```
\def\forest@anchor@border@to@compass{%
  \ifforest@anchor@isborder % snap to 45 deg, to range 0-360
    \ifforest@anchor@snapbordertocompass
      \forest@anchor@snap@border@to@compass
    \else % to range 0-360
      \pgfmathMod@{\forest@temp@anchor}{360}%
      \forest@truncatepgfmathresult
      \let\forest@temp@anchor\pgfmathresult
    \fi
    \ifforest@anchor@snapbordertocompass
      \ifforest@anchor@forwardtotikz
        \ifcsname pgf@anchor%
            @\csname pgf@sh@ns@\pgfreferencednodename\endcsname
            @\csname forest@compass@\forest@temp@anchor\endcsname
        \endcsname
          \letcs\forest@temp@anchor{forest@compass@\forest@temp@anchor}%
        \fi
      \else
        \letcs\forest@temp@anchor{forest@compass@\forest@temp@anchor}%
      \fi
    \fi
  \fi
}
\csdef{forest@compass@0}{east}
\csdef{forest@compass@45}{north east}
\csdef{forest@compass@90}{north}
\csdef{forest@compass@135}{north west}
\csdef{forest@compass@180}{west}
\csdef{forest@compass@225}{south west}
\csdef{forest@compass@270}{south}
\csdef{forest@compass@315}{south east}
\csdef{forest@compass@360}{east}
```

This macro approximates an angle (stored in `\forest@temp@anchor`) with a compass direction (stores it in the same macro).

```
\def\forest@anchor@snap@border@to@compass{%
  \pgfmathMod@{\forest@temp@anchor}{360}%
  \pgfmathdivide@{\pgfmathresult}{45}%
  \pgfmathround@{\pgfmathresult}%
  \pgfmathmultiply@{\pgfmathresult}{45}%
  \forest@truncatepgfmathresult
  \let\forest@temp@anchor\pgfmathresult
}
```

This macro forwards the resulting anchor to Ti*k*Z.

```
\def\forest@anchor@forward#1{% #1 = shape name
  \ifdefempty\forest@temp@anchor{%
    \pgf@sh@reanchor{#1}{center}%
    \xdef\forest@hack@tikzshapeborder{%
      \noexpand\tikz@shapebordertrue
      \def\noexpand\tikz@shapeborder@name{\pgfreferencednodename}%
    }\aftergroup\forest@hack@tikzshapeborder
  }{%
    \pgf@sh@reanchor{#1}{\forest@temp@anchor}%
  }%
}
```

Expandably strip ”not yet positionedPGFINTERNAL” from `\pgfreferencednodename` if it is there.

```
\def\forest@referencednodeid{\forest@node@Nametoid{\forest@referencednodename}}%
\def\forest@referencednodename{%
  \expandafter\expandafter\expandafter\forest@referencednodename@\expandafter\pgfreferencednodename\forest@pg
}
\expandafter\def\expandafter\forest@referencednodename@\expandafter#\expandafter1\forest@pgf@notyetpositioned
  \if\relax#1\relax\forest@referencednodename@stripafter#2\relax\fi
  \if\relax#2\relax#1\fi
}
\expandafter\def\expandafter\forest@referencednodename@stripafter\expandafter#\expandafter1\forest@pgf@notyet
```

This macro sets up `\pgf@x` and `\pgf@y` to the given anchor's coordinates, within the node's coordinate system. It works even before the node was positioned. If the anchor is empty, i.e. if is the implicit border anchor, we return the coordinates for the center.

```
\def\forest@pointanchor#1{% #1 = anchor
  \forest@Pointanchor{\forest@cn}{#1}%
}
\def\forest@Pointanchor#1#2{% #1 = node id, #2 = anchor
  \def\forest@pa@temp@name{name}%
  \forestOifdefined{#1}{@box}{%
    \forestOget{#1}{@box}\forest@temp
    \ifdefempty\forest@temp{}{%
      \def\forest@pa@temp@name{later@name}%
    }%
  }{}%
  \setbox0\hbox{%
    \begin{pgfpicture}%
      \if\relax\forestOve{#1}{#2}\relax
        \pgfpointanchor{\forestOve{#1}{\forest@pa@temp@name}}{center}%
      \else
        \pgfpointanchor{\forestOve{#1}{\forest@pa@temp@name}}{\forestOve{#1}{#2}}%
      \fi
      \xdef\forest@global@marshal{%
        \noexpand\global\noexpand\pgf@x=\the\pgf@x\relax
        \noexpand\global\noexpand\pgf@y=\the\pgf@y\relax\relax
      }%
    \end{pgfpicture}%
  }%
  \forest@global@marshal
}
```

Fill in the values of the invalid node. (It's now easy to test for `id=0`.)

```
\forest@node@init
```

11 Compatibility with previous versions

```
\ifdefempty{\forest@compat}{}{%
  \RequirePackage{forest-compat}
}
```