

Implementation of FOREST, a PGF/TikZ-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

1	Identification	2
2	Package options	3
3	Patches	4
4	Utilities	5
4.1	Arrays	11
4.2	Testing for numbers and dimensions	15
4.3	forestmath	18
4.4	Sorting	24
5	The bracket representation parser	28
5.1	The user interface macros	28
5.2	Parsing	29
5.3	The tree-structure interface	33
6	Nodes	34
6.1	Option setting and retrieval	34
6.2	Tree structure	37
6.3	Node options	46
6.3.1	Option-declaration mechanism	46
7	Handlers	53
7.1	.process	54
7.1.1	Registers	63
7.1.2	Declaring options	69
7.1.3	Option propagation	76
7.2	Aggregate functions	78
7.2.1	pgfmath extensions	81
7.3	Nodewalk	83
7.4	Dynamic tree	103

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

8	Stages	108
8.1	Typesetting nodes	112
8.2	Packing	114
8.2.1	Tiers	126
8.2.2	Node boundary	129
8.3	Compute absolute positions	134
8.4	Drawing the tree	135
9	Geometry	137
9.1	Projections	137
9.2	Break path	140
9.3	Get tight edge of path	143
9.4	Get rectangle/band edge	148
9.5	Distance between paths	149
9.6	Utilities	152
10	The outer UI	154
10.1	Externalization	154
10.2	The forest environment	155
10.3	Standard node	159
10.4	ls coordinate system	160
10.5	Relative node names in TikZ	161
10.6	Anchors	162
11	Compatibility with previous versions	169

1 Identification

```

1 \ProvidesPackage{forest}[2017/07/14 v2.1.5 Drawing (linguistic) trees]
2
3 \RequirePackage{tikz}[2013/12/13]
4 \usetikzlibrary{shapes}
5 \usetikzlibrary{fit}
6 \usetikzlibrary{calc}
7 \usepgflibrary{intersections}
8
9 \RequirePackage{pgfplots}
10 \RequirePackage{etoolbox}[2010/08/21]
11 \RequirePackage{elocalloc}% for \locbox
12 \RequirePackage{environ}
13 \RequirePackage{xparse}
14
15 \RequirePackage{inlinedef}
16 \newtoks\ID@usercommands{}
17 \newcommand\NewInlineCommand[3][0]{%
18   \newcommand#2[#1]{#3}%
19   \ID@usercommands\x@{%
20     \the\ID@usercommands
21     \ifx\@foo#2%
22       \def\next{\ID@expandunsafe#2}%
23     \fi
24   }%
25 }
26 \def\@ExpandIfTF#1{%
27   \csname
28     % I'm not 100% sure if this plays well in every situation
29     \csname if#1\endcsname
30     @firstoftwo%
31   \else
32     @secondoftwo%

```

```

33 \fi
34 \endcsname
35 }
36 \patchcmd{\ID@switch}
37 {\ifcat\noexpand\@foo\space}
38 {\the\ID@usercommands\ifcat\noexpand\@foo\space}
39 {%
40 \NewInlineCommand[2]\ExpandIfT{%
41 \MultiExpand{3}{%
42 \@ExpandIfTF{#1}{#2}{}}%
43 }%
44 }
45 \NewInlineCommand[2]\ExpandIfF{%
46 \MultiExpand{3}{%
47 \@ExpandIfTF{#1}{}{#2}}%
48 }%
49 }
50 \NewInlineCommand[3]\ExpandIfTF{%
51 \MultiExpand{3}{%
52 \@ExpandIfTF{#1}{#2}{#3}}%
53 }%
54 }%
55 \newcommand\InlineNoDef[1]{%
56 \begingroup
57 % Define a few ‘quarks’
58 \def\Expand{\Expand}\def\Super{\Super}%
59 \def\UnsafeExpand{\UnsafeExpand}\def\MultiExpand{\MultiExpand}%
60 \def\Recurse{\Recurse}\def\NoExpand{\NoExpand}%
61 \def\Q@END{\Q@END}%
62 % Define a toks register
63 \ID@toks{}%
64 % Signal that we need to look for a star
65 \@testtrue\ID@starfalse\ID@starstarfalse\ID@bangfalse
66 % Start scanning for \def or \gdef
67 \ID@scan#1\Q@END{}%
68 \expandafter\endgroup
69 %\expandafter\@firstofone
70 \the\ID@toks
71 }%
72 }%
73 {%
74 \PackageWarning{forest}{Could not patch inlinedef! Disabling it. Except in some special situations (nested
75 \let\Inline\relax
76 \def\Expand#1{#1}%
77 \def\MultiExpand#1#2{#2}%
78 \def\InlineNoDef#1{#1}%
79 \def\ExpandIfT#1#2{\@ExpandIfTF{#1}{#2}{}}%
80 \def\ExpandIfF#1#2{\@ExpandIfTF{#1}{}{#2}}%
81 \def\ExpandIfTF#1#2#3{\@ExpandIfTF{#1}{#2}{#3}}%
82 }

```

/forest is the root of the key hierarchy.

```

83 \pgfkeys{/forest/.is family}
84 \def\forestset#1{\pgfkeys{/forest}{#1}}

```

2 Package options

```

85 \newif\ifforest@external@
86 \newif\ifforest@tikz@shack
87 \newif\ifforest@install@keys@to@tikz@path@
88 \newif\ifforest@debug@nodewalks

```

```

89 \newif\ifforestdebugdynamics
90 \newif\ifforestdebugprocess
91 \newif\ifforestdebugtemp
92 \newif\ifforestdebug
93 \def\forest@compat{}
94 \forestset{package@options/.cd,
95   external/.is if=forest@external@,
96   tikzcshack/.is if=foresttikzcshack,
97   tikzinstallkeys/.is if=forest@install@keys@to@tikz@path@,
98   compat/.code={\appto\forest@compat{#1}},
99   compat/.default=most,
100  .unknown/.code={% load library
101    \eappto\forest@loadlibrarieslater{%
102      \noexpand\useforestlibrary{\pgfkeyscurrentname}%
103      \noexpand\forestapplylibrarydefaults{\pgfkeyscurrentname}%
104    }%
105  },
106  debug/.code={\forestdebugtrue\pgfkeys{/forest/package@options/debug}{#1}},
107  debug/.default={nodewalks,dynamics,process},
108  debug/nodewalks/.is if=forestdebugnodewalks,
109  debug/dynamics/.is if=forestdebugdynamics,
110  debug/process/.is if=forestdebugprocess,
111 }
112 \forest@install@keys@to@tikz@path@true
113 \foresttikzcshacktrue
114 \def\forest@loadlibrarieslater{}
115 \AtEndOfPackage{\forest@loadlibrarieslater}
116 \NewDocumentCommand\useforestlibrary{s O{} m}{%
117   \def\useforestlibrary@##1{\useforestlibrary@{#2}{##1}}%
118   \forcsvlist\useforestlibrary@{#3}%
119   \IfBooleanT{#1}{\forestapplylibrarydefaults{#3}}%
120 }
121 \def\useforestlibrary@#1#2{%
122   \RequirePackage[#1]{forest-lib-#2}%
123   \csuse{forest@compat@libraries@#2}%
124 }
125 \def\forestapplylibrarydefaults#1{\forcsvlist\forestapplylibrarydefaults@{#1}}
126 \def\forestapplylibrarydefaults@#1{\forestset{libraries/#1/defaults/.try}}
127 \NewDocumentCommand\ProvidesForestLibrary{m O{}}{%
128   \ProvidesPackage{forest-lib-#1}[#2]%
129   \csdef{forest@libraries@loaded@#1}{%
130 }
131 \def\forest@iflibraryloaded#1#2#3{\ifcsdef{forest@libraries@loaded@#1}{#2}{#3}}
132 \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.)

```

133 \def\forest@patch#1#2#3#4#5{%
134   % #1 = cs to be patched
135   % #2 = purpose of the patch
136   % #3 = macro arguments
137   % #4 = original code
138   % #5 = patched code
139   \csdef{forest@original@#1}#3{#4}%
140   \csdef{forest@patched@#1}#3{#5}%
141   \ifcsequal{#1}{forest@original@#1}{%
142     \csletcs{#1}{forest@patched@#1}%

```

```

143 }{%
144   \ifcsequal{#1}{forest@patched@#1}{% all is good, the patch is in!
145   }{%
146     \PackageWarning{forest}{Failed patching '\expandafter\string\cname #1\endcsname'. Purpose of the patch
147     }{%
148   }%
149 }

```

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

```

150 \forest@patch{pgfgettransform}{fix a leaking space}{#1}{%
151   \edef#1{\pgf@pt@aa}{\pgf@pt@ab}{\pgf@pt@ba}{\pgf@pt@bb}{\the\pgf@pt@x}{\the\pgf@pt@y}}
152 }{%
153   \edef#1{\pgf@pt@aa}{\pgf@pt@ab}{\pgf@pt@ba}{\pgf@pt@bb}{\the\pgf@pt@x}{\the\pgf@pt@y}}%
154 }

```

4 Utilities

This is handy.

```

155 \def\forest@empty{}
    Escaping \ifs.
156 \long\def\@escapeif#1#2\fi{\fi#1}
157 \long\def\@escapeifif#1#2\fi#3\fi{\fi\fi#1}
158 \long\def\@escapeififif#1#2\fi#3\fi#4\fi{\fi\fi\fi#1}
159 \def\forest@repeat@n@times#1{% #1=n, #2=code
160   \expandafter\forest@repeat@n@times@\expandafter{\the\numexpr#1}}
161 \def\forest@repeat@n@times@#1{%
162   \ifnum#1>0
163     \@escapeif{%
164       \expandafter\forest@repeat@n@times@\@expandafter{\the\numexpr#1-1}}%
165     }%
166   \else
167     \expandafter@gobble
168   \fi
169 }
170 \def\forest@repeat@n@times@@#1#2{%
171   #2%
172   \forest@repeat@n@times@{#1}{#2}%
173 }

```

A factory for creating \dots loop\dots macros.

```

174 \def\newloop#1{%
175   \count@=\escapechar
176   \escapechar=-1
177   \expandafter\newloop@parse@loopname\string#1\newloop@end
178   \escapechar=\count@
179 }%
180 {\lccode'7='1 \lccode'8='o \lccode'9='p
181   \lowercase{\gdef\newloop@parse@loopname#17889#2\newloop@end{%
182     \edef\newloop@marshal{%
183       \noexpand\csdef{#1loop#2}###1\expandafter\noexpand\cename #1repeat#2\endcsname{%
184         \noexpand\csdef{#1iterate#2}{###1\relax\noexpand\expandafter\expandafter\noexpand\cename#1iterate#
185         \expandafter\noexpand\cename#1iterate#2\endcsname
186         \let\expandafter\noexpand\cename#1iterate#2\endcsname\relax
187       }%
188     }%
189     \newloop@marshal
190   }%
191 }%
192 }%

```

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.

```

193 \def\nnewsafeloop#1{%
194   \csdef{safeloop@#1}##1\saferepeat{%
195     \forest@temp@toks{##1}%
196     \csedef{safeiterate@#1}{%
197       \the\forest@temp@toks\relax
198       \noexpand\expandafter
199       \expandonce{\csname safeiterate@#1\endcsname}%
200       \noexpand\fi
201     }%
202     \csuse{safeiterate@#1}%
203     \advance\noexpand\safeloop@depth-1\relax
204     \cslet{safeiterate@#1}\relax
205   }%
206   \expandafter\newif\csname ifsafebreak@\the\safeloop@depth\endcsname
207 }%
208 \newcount\safeloop@depth
209 \def\safeloop{%
210   \advance\safeloop@depth1
211   \ifcsdef{safeloop@\the\safeloop@depth}{\expandafter\nnewsafeloop\expandafter{\the\safeloop@depth}}%
212   \csdef{safeloopn@\the\safeloop@depth}{0}%
213   \csuse{safeloop@\the\safeloop@depth}%
214   \csedef{safeloopn@\the\safeloop@depth}{\number\numexpr\csuse{safeloopn@\the\safeloop@depth}+1}%
215 }
216 \let\saferepeat\fi
217 \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.

```

218 \def\nnewsafeRKloop#1{%
219   \csdef{safeRKloop@#1}##1\safeRKrepeat{%
220     \forest@temp@toks{##1}%
221     \csedef{safeRKiterate@#1}{%
222       \the\forest@temp@toks\relax
223       \noexpand\expandafter
224       \expandonce{\csname safeRKiterate@#1\endcsname}%
225       \noexpand\fi
226     }%
227     \csuse{safeRKiterate@#1}%
228     \advance\noexpand\safeRKloop@depth-1\relax
229     \cslet{safeRKiterate@#1}\relax
230   }%
231   \expandafter\newif\csname ifsafesafeRKbreak@\the\safeRKloop@depth\endcsname
232 }%
233 \newcount\safeRKloop@depth
234 \def\safeRKloop{%
235   \advance\safeRKloop@depth1
236   \ifcsdef{safeRKloop@\the\safeRKloop@depth}{\expandafter\nnewsafeRKloop\expandafter{\the\safeRKloop@depth}}%
237   \csdef{safeRKloopn@\the\safeRKloop@depth}{0}%
238   \csuse{safeRKbreak@\the\safeRKloop@depth false}%
239   \csuse{safeRKloop@\the\safeRKloop@depth}%
240   \csedef{safeRKloopn@\the\safeRKloop@depth}{\number\numexpr\csuse{safeRKloopn@\the\safeRKloop@depth}+1}%
241 }
242 \let\safeRKrepeat\fi
243 \def\safeRKloopn{\csuse{safeRKloopn@\the\safeRKloop@depth}}%

```

Additional loops (for embedding).

```

244 \newloop\forest@loop

```

New counters, dimens, ifs.


```

300 \else
301 \escapeif\forest@cotu@nospaceB
302 \fi
303 }
304 \def\forest@cotu@nospaceB#1{%
305 \ifcat#1a%
306 \let\forest@cotu@next\forest@cotu@have@alpha
307 \else
308 \if!\ifnum9<1#1!\fi
309 \let\forest@cotu@next\forest@cotu@have@num
310 \else
311 \let\forest@cotu@next\forest@cotu@have@other
312 \fi
313 \fi
314 \forest@cotu@next#1%
315 }
316 \def\forest@cotu@have@alpha#1{%
317 \appto\forest@cotu@result{#1}%
318 \forest@cotu
319 }
320 \def\forest@cotu@have@other#1{%
321 \appto\forest@cotu@result{_%}
322 \forest@cotu
323 }

Additional list macros.
324 \def\forest@listedel#1#2{% #1 = list, #2 = item
325 \edef\forest@marshal{\noexpand\forest@listdel\noexpand#1{#2}}%
326 \forest@marshal
327 }
328 \def\forest@listcsdel#1#2{%
329 \expandafter\forest@listdel\csname #1\endcsname{#2}%
330 }
331 \def\forest@listcsedel#1#2{%
332 \expandafter\forest@listedel\csname #1\endcsname{#2}%
333 }
334 \edef\forest@restorelistsepcatcode{\noexpand\catcode'\the\catcode'\relax}%
335 \catcode'\|=3
336 \gdef\forest@listdel#1#2{%
337 \def\forest@listedel@A##1|#2|##2\forest@END{%
338 \forest@listedel@B##1|##2\forest@END%|
339 }%
340 \def\forest@listedel@B|##1\forest@END{%|
341 \def#1{##1}%
342 }%
343 \expandafter\forest@listedel@A\expandafter|#1\forest@END%|
344 }
345 \forest@restorelistsepcatcode

Strip (the first level of) braces from all the tokens in the argument.
346 \def\forest@strip@braces#1{%
347 \forest@strip@braces@A#1\forest@strip@braces@preend\forest@strip@braces@end
348 }
349 \def\forest@strip@braces@A#1#2\forest@strip@braces@end{%
350 #1\ifx\forest@strip@braces@preend#2\else\escapeif{\forest@strip@braces@A#2\forest@strip@braces@end}\fi
351 }

Utilities dealing with pgfkeys.
352 \def\forest@copycommandkey#1#2{% copies command of #1 into #2
353 \pgfkeysifdefined{#1/.@cmd}{-}{%
354 \PackageError{forest}{Key #1 is not a command key}{-}%
355 }%

```



```

356 \pgfkeysgetvalue{#1/.@cmd}\forest@temp
357 \pgfkeyslet{#2/.@cmd}\forest@temp
358 \pgfkeysifdefined{#1/.@args}{%
359   \pgfkeysgetvalue{#1/.@args}\forest@temp
360   \pgfkeyslet{#2/.@args}\forest@temp
361 }{}%
362 \pgfkeysifdefined{#1/.@body}{%
363   \pgfkeysgetvalue{#1/.@body}\forest@temp
364   \pgfkeyslet{#2/.@body}\forest@temp
365 }{}%
366 \pgfkeysifdefined{#1/.@body}{%
367   \pgfkeysgetvalue{#1/.@body}\forest@temp
368   \pgfkeyslet{#2/.@body}\forest@temp
369 }{}%
370 \pgfkeysifdefined{#1/.@def}{%
371   \pgfkeysgetvalue{#1/.@def}\forest@temp
372   \pgfkeyslet{#2/.@def}\forest@temp
373 }{}%
374 }
375 \forestset{
376   copy command key/.code 2 args={\forest@copycommandkey{#1}{#2}},
377   autoforward/.code 2 args={\forest@autoforward{#1}{#2={#1={##1}}}{true}},
378   autoforward'/ .code 2 args={\forest@autoforward{#1}{#2=#1, #2={#1={##1}}}{true}},
379   Autoforward/.code 2 args={\forest@autoforward{#1}{#2}{true}},
380   autoforward register/.code 2 args={\forest@autoforward{#1}{#2={#1={##1}}}{false}},
381   autoforward register'/ .code 2 args={\forest@autoforward{#1}{#2=#1, #2={#1={##1}}}{false}},
382   Autoforward register/.code 2 args={\forest@autoforward{#1}{#2}{false}},
383   copy command key@if it exists/.code 2 args={%
384     \pgfkeysifdefined{#1/.@cmd}{%
385       \forest@copycommandkey{#1}{#2}%
386     }{}%
387   },
388   unautoforward/.style={
389     typeout={unautoforwarding #1},
390     copy command key@if it exists={/forest/autoforwarded #1}{/forest/#1},
391     copy command key@if it exists={/forest/autoforwarded #1+}{/forest/#1+},
392     copy command key@if it exists={/forest/autoforwarded #1-}{/forest/#1-},
393     copy command key@if it exists={/forest/autoforwarded #1*}{/forest/#1*},
394     copy command key@if it exists={/forest/autoforwarded #1:}{/forest/#1:},
395     copy command key@if it exists={/forest/autoforwarded #1'}{/forest/#1'},
396     copy command key@if it exists={/forest/autoforwarded #1+'}{/forest/#1+'},
397     copy command key@if it exists={/forest/autoforwarded #1-'}{/forest/#1-'},
398     copy command key@if it exists={/forest/autoforwarded #1*'}{/forest/#1*'},
399     copy command key@if it exists={/forest/autoforwarded #1:'}{/forest/#1:'},
400     copy command key@if it exists={/forest/autoforwarded +#1}{/forest/+#1},
401   },
402   /handlers/.undef/.code={\csundef{pgfk@pgfkeyscurrentpath}},
403   undef option/.style={
404     /forest/#1/.undef,
405     /forest/#1/.@cmd/.undef,
406     /forest/#1+/.@cmd/.undef,
407     /forest/#1-/.@cmd/.undef,
408     /forest/#1*/.@cmd/.undef,
409     /forest/#1:/@cmd/.undef,
410     /forest/#1'/@cmd/.undef,
411     /forest/#1+'/.@cmd/.undef,
412     /forest/#1-'/.@cmd/.undef,
413     /forest/#1*'/.@cmd/.undef,
414     /forest/#1:'/.@cmd/.undef,
415     /forest/+#1/.@cmd/.undef,
416     /forest/TeX={\patchcmd{\forest@node@init}{\forest@init{#1}}{}{}},

```

```

417 },
418 undef register/.style={undef option={#1}},
419 }
420 \def\forest@autoforward#1#2#3{%
421 % #1 = option name
422 % #2 = code of a style taking one arg (new option value),
423 % which expands to whatever should be done with the new value
424 % autoforward(') adds to the keylist (arg#2)
425 % #3 = true=option, false=register
426 \forest@autoforward@createforwarder{}{#1}{}{#2}{#3}%
427 \forest@autoforward@createforwarder{}{#1}{+}{#2}{#3}%
428 \forest@autoforward@createforwarder{}{#1}{-}{#2}{#3}%
429 \forest@autoforward@createforwarder{}{#1}{*}{#2}{#3}%
430 \forest@autoforward@createforwarder{}{#1}{:}{#2}{#3}%
431 \forest@autoforward@createforwarder{}{#1}{'}{#2}{#3}%
432 \forest@autoforward@createforwarder{}{#1}{+'}{#2}{#3}%
433 \forest@autoforward@createforwarder{}{#1}{-'}{#2}{#3}%
434 \forest@autoforward@createforwarder{}{#1}{*''}{#2}{#3}%
435 \forest@autoforward@createforwarder{}{#1}{:''}{#2}{#3}%
436 \forest@autoforward@createforwarder{+}{#1}{}{#2}{#3}%
437 }
438 \def\forest@autoforward@createforwarder#1#2#3#4#5{%
439 % #1=prefix, #2=option name, #3=suffix, #4=macro code (#2 above), #5=option or register
440 \pgfkeysifdefined{/forest/#1#2#3/.@cmd}{%
441 \forest@copycommandkey{/forest/#1#2#3}{/forest/autoforwarded #1#2#3}%
442 \pgfkeyssetvalue{/forest/autoforwarded #1#2#3/option@name}{#2}%
443 \pgfkeysdef{/forest/#1#2#3}{%
444 \pgfkeysalso{autoforwarded #1#2#3={##1}}%
445 \def\forest@temp@macro###1{#4}%
446 \csname forest@temp#5\endcsname
447 \edef\forest@temp@value{\ifforest@temp\expandafter\forest0v\expandafter{\expandafter\forest@setter@node
448 %\expandafter\expandafter\expandafter\pgfkeysalso\expandafter\expandafter\expandafter{\expandafter\fore
449 \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\pgfkeysalso\expand
450 }%
451 }{}%
452 }
453 \def\forest@node@removekeysfromkeylist#1#2{% #1 = keys to remove, #2 = option name
454 \edef\forest@marshal{%
455 \noexpand\forest@removekeysfromkeylist{\unexpanded{#1}}{\forestov{#2}}\noexpand\forest@temp@toks}\forest@
456 \forestoaset{#2}{\the\forest@temp@toks}%
457 }
458 \def\forest@removekeysfromkeylist#1#2#3{%
459 % #1 = keys to remove (a keylist: an empty value means remove a key with any value)
460 % #2 = keylist
461 % #3 = toks cs for result
462 \forest@temp@toks{}%
463 \def\forestnovalue{\forestnovalue}%
464 \pgfqkeys{/forest/remove@key@installer}{#1}%
465 \let\forestnovalue\pgfkeysnovaluertext
466 \pgfqkeys{/forest/remove@key}{#2}%
467 \pgfqkeys{/forest/remove@key@uninstaller}{#1}%
468 #3\forest@temp@toks
469 }
470 \def\forest@remove@key@novalue{\forest@remove@key@novalue}%
471 \forestset{
472 remove@key@installer/.unknown/.code={% #1 = (outer) value
473 \def\forest@temp{#1}%
474 \ifx\forest@temp\pgfkeysnovalue@text
475 \pgfkeysdef{/forest/remove@key/\pgfkeyscurrentname}{}%
476 \else
477 \ifx\forest@temp\forestnovalue

```

```

478     \expandafter\forest@remove@key@installer@defwithvalue\expandafter{\pgfkeyscurrentname}{\pgfkeysnovalu
479     \else
480     \expandafter\forest@remove@key@installer@defwithvalue\expandafter{\pgfkeyscurrentname}{#1}%
481     \fi
482     \fi
483   },
484   remove@key/.unknown/.code={% #1 = (inner) value
485     \expandafter\apptotoks\expandafter\forest@temp@toks\expandafter{\pgfkeyscurrentname={#1},}%
486   },
487   remove@key@uninstaller/.unknown/.code={%
488     \pgfkeyslet{/forest/remove@key/\pgfkeyscurrentname/.@cmd}\@undefined},
489 }
490 \def\forest@remove@key@installer@defwithvalue#1#2{% #1=key name, #2 = outer value
491   \pgfkeysdef{/forest/remove@key/#1}{% ##1 = inner value
492     \def\forest@temp@outer{#2}%
493     \def\forest@temp@inner{##1}%
494     \ifx\forest@temp@outer\forest@temp@inner
495     \else
496       \apptotoks\forest@temp@toks{#1={##1},}%
497     \fi
498   }%
499 }
500 \forestset{
501   show register/.code={%
502     \forestrget{#1}\foresttemp
503     \typeout{Forest register "#1"=\expandafter\detokenize\expandafter{\foresttemp}}%
504   },
505 }

```

4.1 Arrays

```

506 \def\forest@newarray#1{%
507   \forest@tempfalse % non-global
508   {%
509     \escapechar=-1
510     \expandafter\escapechar\expandafter\count@\expandafter
511   }%
512   \expandafter\forest@newarray@\expandafter{\string#1}%
513 }
514 \def\forest@newglobalarray#1{%
515   \forest@temptrue % global
516   {%
517     \escapechar=-1
518     \expandafter\escapechar\expandafter\count@\expandafter
519   }%
520   \expandafter\forest@newarray@\expandafter{\string#1}%
521 }
522 \def\forest@array@empty@error#1{%
523   \PackageError{forest}{Cannot pop from empty array "#1".}{}}%
524 \def\forest@array@oub@error#1#2{%
525   \PackageError{forest}{#2 is out of bounds of array "#1"
526     (\the\csuse{#1M}--\the\csuse{#1N}).}{}}%

```

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

```

527 \def\forest@newarray@#1{%
528   % array bounds: M <= i < N
529   \expandafter\newcount\csname#1M\endcsname
530   \expandafter\newcount\csname#1N\endcsname
531   \csedef{#1clear}{%
532     \ifforest@temp\global\fi\expandonce{\csname#1M\endcsname}0

```

```

533 \ifforest@temp\global\fi\expandonce{\csname#1N\endcsname}0
534 }%
535 \csedef{#1ifempty}{%
536 \noexpand\ifnum\expandonce{\csname#1M\endcsname}<\expandonce{\csname#1N\endcsname}\relax
537 \unexpanded{\expandafter\@secondoftwo
538 \else
539 \expandafter\@firstoftwo
540 \fi}%
541 }%
542 \csedef{#1length}{% a numexpr
543 \noexpand\numexpr\expandonce{\csname#1N\endcsname}-\expandonce{\csname#1M\endcsname}\relax
544 }%
545 \csedef{#1checkrange}##1##2{% args can be \numexprs
546 \noexpand\forest@tempfalse
547 \noexpand\ifnum\numexpr##1<\expandonce{\csname#1M\endcsname}\relax
548 \noexpand\forest@temptrue
549 \noexpand\fi
550 \noexpand\ifnum\numexpr##2>\expandonce{\csname#1N\endcsname}\relax
551 \noexpand\forest@temptrue
552 \noexpand\fi
553 \noexpand\ifforest@temp
554 \noexpand\forest@array@oub@error{#1}{Range "\noexpand\number\noexpand\numexpr##1\relax--\noexpand\number\noexpand\numexpr##2\relax"}%
555 \noexpand\fi
556 }%
557 \csedef{#1checkindex}##1{% arg can be a \numexpr
558 \noexpand\forest@tempfalse
559 \noexpand\ifnum\numexpr##1<\expandonce{\csname#1M\endcsname}\relax
560 \noexpand\forest@temptrue
561 \noexpand\fi
562 \noexpand\ifnum\numexpr##1<\expandonce{\csname#1N\endcsname}\relax
563 \noexpand\else
564 \noexpand\forest@temptrue
565 \noexpand\fi
566 \noexpand\ifforest@temp
567 \noexpand\forest@array@oub@error{#1}{Index "\noexpand\number\noexpand\numexpr##1\relax"}%
568 \noexpand\fi
569 }%
570 \csedef{#1get}##1##2{% ##1 = index, ##2 = receiving cs
571 \expandonce{\csname#1checkindex\endcsname}{##1}%
572 \noexpand\letcs##2{##1}%
573 }%
574 \csedef{#1get@}##1##2{% ##1 = index, ##2 = receiving cs (don't check bounds)
575 \noexpand\letcs##2{##1}%
576 }%
577 \csedef{#1toppop}##1{% ##1 = receiving cs
578 \expandonce{\csname#1ifempty\endcsname}{%
579 \noexpand\forest@array@empty@error{#1}%
580 }{%
581 \ifforest@temp\global\fi\advance\expandonce{\csname#1N\endcsname}-1
582 \noexpand\letcs\noexpand##1{#1\noexpand\the\expandonce{\csname#1N\endcsname}}%
583 }%
584 }%
585 \InlineNoDef{\csdef{#1bottompop}##1{% ##1 = receiving cs
586 \Expand{\csname#1ifempty\endcsname}{%
587 \forest@array@empty@error{#1}%
588 }{%
589 \letcs##1{#1\the\Expand{\csname#1M\endcsname}}%
590 \ExpandIfT{forest@temp}\global\advance\Expand{\csname#1M\endcsname 1}%
591 }%
592 }}%
593 % \csdef{#1bottompop}##1{% we need this as \Inline chokes on \let\macro=\relax

```

```

594 % \expandafter\Inline\expandafter\def\csname#1bottompop\endcsname##1{% ##1 = receiving cs
595 %   \Expand{\csname#1ifempty\endcsname}{%
596 %     \forest@array@empty@error{#1}%
597 %   }{%
598 %     \letcs##1{#1\the\Expand{\csname#1M\endcsname}}}%
599 %   \ExpandIfT{\forest@temp}\global\advance\Expand{\csname#1M\endcsname 1}%
600 % }%
601 % }%
602 % \csedef{#1bottompop}##1{% ##1 = receiving cs
603 %   \expandonce{\csname#1ifempty\endcsname}{%
604 %     \noexpand\forest@array@empty@error{#1}%
605 %   }{%
606 %     \noexpand\letcs\noexpand##1{#1\noexpand\the\expandonce{\csname#1M\endcsname}}}%
607 %   \ifforest@temp\global\fi\advance\expandonce{\csname#1M\endcsname}1
608 % }%
609 % }%
610 \csedef{#1setappend}##1{% ##1 = definition
611   \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi
612   {#1\noexpand\the\expandonce{\csname#1N\endcsname}}}%
613   {\noexpand\unexpanded{##1}}}%
614   \ifforest@temp\global\fi\advance\expandonce{\csname#1N\endcsname}1
615 }%
616 \csedef{#1setappend@}##1##2{% ##1 = continue by, ##2 = definition
617   \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi
618   {#1\noexpand\the\expandonce{\csname#1N\endcsname}}}%
619   {\noexpand\unexpanded{##2}}}%
620   \ifforest@temp\global\fi\advance\expandonce{\csname#1N\endcsname}1
621   ##1%
622 }%
623 \csedef{#1setprepend}##1{% ##1 = definition
624   \ifforest@temp\global\fi\advance\expandonce{\csname#1M\endcsname}-1
625   \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi
626   {#1\noexpand\the\expandonce{\csname#1M\endcsname}}}%
627   {\noexpand\unexpanded{##1}}}%
628 }%
629 \csedef{#1esetappend}##1{% ##1 = definition
630   \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi{#1\noexpand\the\expandonce{\csname#1N\endcsname}}}-1
631   \ifforest@temp\global\fi\advance\expandonce{\csname#1N\endcsname}1
632 }%
633 \csedef{#1esetprepend}##1{% ##1 = definition
634   \ifforest@temp\global\fi\advance\expandonce{\csname#1M\endcsname}-1
635   \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi{#1\noexpand\the\expandonce{\csname#1M\endcsname}}}-1
636 }%
637 \csedef{#1letappend}##1{% ##1 = cs
638   \ifforest@temp\noexpand\expandafter\noexpand\global\fi\noexpand\expandafter\noexpand\let
639   \noexpand\csname#1\noexpand\the\expandonce{\csname#1N\endcsname}\noexpand\endcsname
640   ##1%
641   \ifforest@temp\global\fi\advance\expandonce{\csname#1N\endcsname}1
642 }%
643 \csedef{#1letprepend}##1{% ##1 = cs
644   \ifforest@temp\global\fi\advance\expandonce{\csname#1M\endcsname}-1
645   \ifforest@temp\noexpand\expandafter\noexpand\global\fi\noexpand\expandafter\noexpand\let
646   \noexpand\csname#1\noexpand\the\expandonce{\csname#1M\endcsname}\noexpand\endcsname
647   ##1%
648 }%

```

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}%
}%
}%

649 \csedef{#1values}{%
650 \noexpand\expandafter\noexpand\expandafter\noexpand\expandafter\expandonce{\csname#1valuesfromrange\endcsname}%
651 \noexpand\expandafter\noexpand\expandafter\noexpand\expandafter{%
652 \noexpand\expandafter\noexpand\the
653 \noexpand\expandafter\expandonce{\csname#1M\endcsname}%
654 \noexpand\expandafter}%
655 \noexpand\expandafter{\noexpand\the\expandonce{\csname#1N\endcsname}}%
656 }%

\def\arrayvaluesfromrange##1##2{% ##1/##2 = lower/upper bounds (we receive them expanded) <-- \csedef{#1values}
\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#1values}
\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 do this on your own using ...checkrange defined above.

```

657 \csedef{#1valuesfromrange}##1##2{% ##1/##2 = lower/upper bounds (we receive them expanded)
658 \noexpand\ifnum##1<##2
659 {\noexpand\expandafter\noexpand\expandonce\noexpand\expandafter{\noexpand\csname#1##1\noexpand\endcsname}%
660 \noexpand\expandafter\noexpand@escapeif\noexpand\expandafter{\noexpand\expandafter\expandonce{\csname#1##2\endcsname}%
661 \noexpand\expandafter{\noexpand\number\noexpand\numexpr##1+1}{##2}}%
662 \noexpand\fi
663 }%

```

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.

```

664 \csdef{#1topextend}{\def\forest@array@currentarray{#1}\forest@array@topextend}%
665 }
666 \def\forest@array@topextend{\futurelet\forest@ate@next@token\forest@ate@checkforspace}
667 \def\forest@ate@checkforspace{%
668 \expandafter\ifx\space\forest@ate@next@token
669 \expandafter\forest@ate@havespace
670 \else
671 \expandafter\forest@ate@checkforgroup
672 \fi
673 }
674 \def\forest@ate@havespace{\expandafter\forest@array@topextend\romannumeral-'0}%
675 \def\forest@ate@checkforgroup{%
676 \ifx\forest@ate@next@token\bgroup
677 \expandafter\forest@ate@appendgroup
678 \else
679 \expandafter\forest@ate@checkforeov
680 \fi
681 }
682 \def\forest@ate@appendgroup{%
683 \expandonce{\csname\forest@array@currentarray setappend@\endcsname}\forest@array@topextend
684 }
685 \def\forest@ate@checkforeov{%
686 \ifx\forest@ate@next@token\forest@eov
687 \expandafter\forest@ate@finish

```

```

688 \else
689 \expandafter\forest@ate@appendtoken
690 \fi
691 }
692 \def\forest@ate@appendtoken#1{%
693 \expandonce{\csname\forest@array@currentarray setappend\endcsname}#{1}%
694 \forest@array@topextend
695 }
696 \def\forest@ate@finish\forest@eov{\forest@topextend@next}
697 \let\forest@topextend@next\relax
698 \forest@newarray\forest@temparray@
699 \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 T_EX 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.

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

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

```

700 \def\forest@eon{ }
701 \newif\ifforest@isnum@minus
702 \newif\ifforest@isnum
703 \def\forest@isnum#1{%
704 \forest@isnum@minusfalse
705 \let\forest@isnum@next\forest@isnum@finish

```

Expand in advance, like `pgfmath` does.

```
706 \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.

```

707 \expandafter\forest@isnum@a\forest@isnum@temp\forest@eon\forest@eon\forest@END
708 \ifforest@isnum
709 \expandafter\@firstoftwo
710 \else
711 \expandafter\@secondoftwo
712 \fi
713 }

```

```
714 \def\forest@isnum@a{\futurelet\forest@isnum@token\forest@isnum@b}

```

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

```

715 \def\forest@isnum@minustoggle{%
716 \ifforest@isnum@minus\forest@isnum@minusfalse\else\forest@isnum@minustrue\fi
717 }
718 \def\forest@isnum@b{%
719 \let\forest@next\forest@isnum@p
720 \ifx-\forest@isnum@token
721 \forest@isnum@minustoggle
722 \let\forest@next\forest@isnum@c
723 \else
724 \ifx+\forest@isnum@token
725 \let\forest@next\forest@isnum@c
726 \else
727 \expandafter\ifx\space\forest@isnum@token
728 \let\forest@next\forest@isnum@s
729 \fi
730 \fi
731 \fi

```

```
732 \forest@next
733 }
```

Eat + and -.

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

Eat the space!

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

```
736 \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.

```
737 \def\forest@isnum@p{%
```

```
738 \ifx0\forest@isnum@token
```

```
739 \let\forest@next\forest@isnum@next
```

```
740 \else
```

```
741 \let\forest@next\forest@isnum@nz@
```

```
742 \fi
```

```
743 \forest@isnum>true
```

```
744 \afterassignment\forest@isnum@q\forest@isnum@count\ifforest@isnum@minus-\fi0%
```

```
745 }
```

```
746 \def\forest@isnum@q{%
```

```
747 \futurelet\forest@isnum@token\forest@next
```

```
748 }
```

```
749 \def\forest@isnum@nz@{%
```

```
750 \ifnum\forest@isnum@count=0
```

```
751 \forest@isnum>false
```

```
752 \fi
```

```
753 \forest@isnum@next
```

```
754 }
```

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

```
755 \def\forest@isnum@finish#1\forest@END{%
```

```
756 \ifx\forest@isnum@token\forest@eon
```

```
757 \else
```

```
758 \forest@isnum>false
```

```
759 \fi
```

```
760 }
```

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

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

```
761 \newcount\forest@isdim@nonintpart
```

```
762 \newif\ifforest@isdim
```

```
763 \def\forest@isdim#1{%
```

```
764 \forest@isdim>false
```

```
765 \forest@isnum@minus>false
```

```
766 \def\forest@isdim@leadingzeros{}%
```

```
767 \forest@isdim@nonintpart=0
```

```
768 \def\forest@isdim@unit{pt}%
```

```
769 \let\forest@isnum@next\forest@isdim@checkfordot
```

```
770 \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.

```
771 \expandafter\forest@isnum@a\forest@isnum@temp\forest@eon\forest@eon\forest@eon\forest@eon\forest@END
```

```
772 \ifforest@isdim
```

```
773 \expandafter\@firstoftwo
```

```
774 \else
```

```
775 \expandafter\@secondoftwo
```

```
776 \fi
```



```

777 }
778 \def\forest@isdim@checkfordot{%
779   \ifx.\forest@isnum@token
780     \expandafter\forest@isdim@dot
781   \else
782     \ifx,\forest@isnum@token
783       \expandafter\expandafter\expandafter\forest@isdim@dot
784     \else
785       \expandafter\expandafter\expandafter\forest@isdim@nodot
786     \fi
787   \fi
788 }
789 \def\forest@isdim@nodot{%
790   \ifforest@isnum
791     No number, no dot, so not a dimension.
792   \expandafter\forest@isdim@checkforunit
793   \else
794     \expandafter\forest@isdim@finish@nodim
795   \fi
796 }
797 \def\forest@isdim@dot#1{% #1=. or ,
798   \futurelet\forest@isnum@token\forest@isdim@collectzero
799 }
800 \def\forest@isdim@collectzero{%
801   \ifx0\forest@isnum@token
802     \expandafter\forest@isdim@collectzero@
803   \else
804     \expandafter\forest@isdim@getnonintpart
805   \fi
806 }
807 \def\forest@isdim@collectzero@#1{% #1 = 0
808   \appto\forest@isdim@leadingzeros{0}%
809   \futurelet\forest@isnum@token\forest@isdim@collectzero
810 }
811 \def\forest@isdim@getnonintpart{%
812   \afterassignment\forest@isdim@checkforunit\forest@isdim@nonintpart0%
813 }

```

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

```

813 \def\forest@def@unit#1{\csdef{forest@unit@#1}{#1}}
814 \forest@def@unit{em}
815 \forest@def@unit{ex}
816 \forest@def@unit{pt}
817 \forest@def@unit{pc}
818 \forest@def@unit{in}
819 \forest@def@unit{bp}
820 \forest@def@unit{cm}
821 \forest@def@unit{mm}
822 \forest@def@unit{dd}
823 \forest@def@unit{cc}
824 \forest@def@unit{sp}
825 \def\forest@isdim@checkforunit#1#2{%
826   \lowercase{\edef\forest@isnum@temp{\detokenize{#1#2}}}%
827   \ifcsname forest@unit@\forest@isnum@temp\endcsname
828     \let\forest@isdim@next\forest@isdim@finish@dim
829   \edef\forest@isdim@unit{\csname forest@unit@\forest@isnum@temp\endcsname}%
830   \else
831     \ifx#1\forest@eon
832       \let\forest@isdim@next\forest@isdim@finish@dim
833     \else
834       \let\forest@isdim@next\forest@isdim@finish@nodim

```

```

835 \fi
836 \fi
837 \forest@isdim@next
838 }
839 \def\forest@isdim@finish@dim{%
840 \futurelet\forest@isnum@token\forest@isdim@finish@dim@a
841 }
842 \def\forest@isdim@finish@dim@a{%
843 \expandafter\ifx\space\forest@isnum@token
844 \expandafter\forest@isdim@finish@dim@b
845 \else
846 \expandafter\forest@isdim@finish@dim@c
847 \fi
848 }
849 \expandafter\def\expandafter\forest@isdim@finish@dim@b\space{% eat one space
850 \futurelet\forest@isnum@token\forest@isdim@finish@dim@c
851 }
852 \def\forest@isdim@finish@dim@c#1\forest@END{%
853 \ifx\forest@isnum@token\forest@eon
854 \forest@isdim>true
855 \forest@isdim@dimen\the\forest@isnum@count.\forest@isdim@leadingzeros\the\forest@isdim@nonintpart\forest@
856 \else
857 \forest@isdim>false
858 \fi
859 }
860 \def\forest@isdim@finish@nodim#1\forest@END{%
861 \forest@isdim>false
862 }
863 \newdimen\forest@isdim@dimen
864 % \long\def\@firstofthree#1#2#3{#3} % defined by LaTeX
865 \long\def\@firstofthree#1#2#3{#1}
866 \long\def\@secondofthree#1#2#3{#2}
867 \def\forest@isnumdim#1{%
868 \forest@isdim{#1}{%
869 \forest@isnumdim@
870 }{%
871 \@thirdofthree
872 }%
873 }
874 \def\forest@isnumdim@{%
875 \ifforest@isnum
876 \expandafter\@firstofthree
877 \else
878 \expandafter\@secondofthree
879 \fi
880 }

```

4.3 forestmath

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

```

881 \def\forestmath@type@generic{ } % generic (token list)
882 \def\forestmath@type@count{n} % integer
883 \def\forestmath@type@dimen{d} % a dimension: <decimal> pt
884 \def\forestmath@type@unitless{P} % <decimal> (a unitless dimension) (P because pgfmath returns such numbers)
885 \def\forestmath@type@textasc{t} % text (ascending)
886 \def\forestmath@type@textdesc{T} % text (descending)
887 \def\forestmath@type@none{} % internal (for requests - means whatever)
888 \def\forestmath@result{}
889 \let\forestmath@resulttype\forestmath@type@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; if it doesn't, we execute the fourth argument.

```

890 \def\forest@tryprocess#1{%
891   \def\forest@tryprocess@returnarray{#1}%
892   \expandafter\forest@tryprocess@a\romannumeral-'0}
893 \def\forest@tryprocess@a{\futurelet\forest@temp@token\forest@tryprocess@b}
894 \def\forest@tryprocess@b{%
895   \ifx>\forest@temp@token
896     \expandafter\forest@tryprocess@yes
897   \else
898     \expandafter\forest@tryprocess@no
899   \fi
900 }
901 \def\forest@spacegen{ \forest@spacegen}
902 \def\forest@tryprocess@yes#1#2\forest@spacegen{%
903   \expandafter\forest@process\expandafter{\forest@tryprocess@returnarray}#2\forest@eov
904   \@firstoftwo
905 }
906 \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.)

```

907 \def\forestmathsetcount#1#2{%
908   \forest@tryprocess{false}#2\forest@spacegen{%
909     #1=\forest@process@result\relax
910   }{%
911     \forestmathsetcount@#1{#2}%
912   }%
913 }
914 \def\forestmathsetcount@#1#2{%
915   \forest@isnum{#2}{%
916     #1=\forest@isnum@count
917   }{%
918     \pgfmathsetcount#1{#2}%
919   }%
920 }
921 \def\forestmathsetlength#1#2{%
922   \forest@tryprocess{false}#2\forest@spacegen{%
923     #1=\forest@process@result\relax
924   }{%
925     \forestmathsetlength@#1{#2}%
926   }%
927 }
928 \def\forestmathsetlength@#1#2{%
929   \forest@isdim{#2}{%
930     #1=\forest@isdim@dimen
931   }{%
932     \pgfmathsetlength#1{#2}%
933   }%
934 }
935 \def\forestmathtruncatemacro#1#2{%
936   \forest@tryprocess{false}#2\forest@spacegen{%
937     \forest@temp@count=\forest@process@result\relax
938     \edef#1{\the\forest@temp@count}%
939   }{%
940     \forestmathtruncatemacro@#1{#2}%
941   }%

```

```

942 }
943 \def\forestmathtruncatemacro#1#2{%
944   \forest@isnum{#2}{%
945     \edef#1{\the\forest@isnum@count}%
946   }{%
947     \pgfmathtruncatemacro#1{#2}%
948   }%
949 }
950 \def\forestmathsetlengthmacro#1#2{%
951   \forest@tryprocess{false}#2\forest@spacegen{%
952     \forest@temp@dimen=\forest@process@result\relax
953     \edef#1{\the\forest@temp@dimen}%
954   }{%
955     \forestmathsetlengthmacro#1{#2}%
956   }%
957 }
958 \def\forestmathsetlengthmacro@#1#2{%
959   \forest@isdim{#2}{%
960     \edef#1{\the\forest@isdim@dimen}%
961   }{%
962     \pgfmathsetlengthmacro#1{#2}%
963   }%
964 }
965 \def\forestmathsetmacro#1#2{%
966   \forest@tryprocess{false}#2\forest@spacegen{%
967     \let#1\forest@process@result
968     \let\forestmathresulttype\forest@process@result@type
969   }{%
970     \forestmathsetmacro#1{#2}%
971     \let\forestmathresulttype\forestmath@type@unitless
972   }%
973 }
974 \def\forestmathsetmacro@#1#2{%
975   \forest@isdim{#2}{%
976     \edef#1{\expandafter\Pgf@geT\the\forest@isdim@dimen}%
977   }{%
978     \pgfmathsetmacro#1{#2}%
979   }%
980 }
981 \def\forestmathparse#1{%
982   \forest@tryprocess{false}#1\forest@spacegen{%
983     \let\forestmathresult\forest@process@result
984     \let\forestmathresulttype\forest@process@result@type
985   }{%
986     \forestmathparse@{#1}%
987     \let\forestmathresulttype\forestmath@type@unitless
988   }%
989 }
990 \def\forestmathparse@#1{%
991   \forest@isdim{#1}{%
992     \edef\forestmathresult{\expandafter\Pgf@geT\the\forest@isdim@dimen}%
993   }{%
994     \pgfmathsetmacro\forestmathresult{#1}%
995   }%
996 }

```

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.

```

997 %\def\forestmathparse@returnarray#1{% same as above, but returns the result as an array (used only internal
998 % \forest@tryprocess{true}#1\forest@spacegen}{%

```

```

999 % \forestmathparse@{#1}%
1000 % \let\forest@process@result@type\forestmathtype@unitless
1001 % \forest@process@result@clear
1002 % \forest@process@result@letappend\forestmathresult
1003 % }%
1004 %}

```

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

```

1005 \csdef{forest@bh@0}{0}
1006 \csdef{forest@bh@false}{0}
1007 \csdef{forest@bh@1}{1}
1008 \csdef{forest@bh@true}{1}
1009 \def\forestmath@if#1{%
1010 \ifcsdef{forest@bh@\detokenize{#1}}{%
1011 \let\forest@next\forestmath@if@fast
1012 }{%
1013 \let\forest@next\forestmath@if@slow
1014 }%
1015 \forest@next{#1}%
1016 \ifnum\forest@temp=0
1017 \expandafter\@secondoftwo
1018 \else
1019 \expandafter\@firstoftwo
1020 \fi
1021 }
1022 \def\forestmath@if@fast#1{\letcs\forest@temp{forest@bh@\detokenize{#1}}}
1023 \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).

```

1024 \def\forestmath@convert@fromto#1#2#3{%
1025 \edef\forestmathresult{\csname forestmath@convert@from@#1@to@#2\endcsname{#3}}
1026 \def\forestmath@convert@from#1{\forestmath@convert@fromto{#1}{\forestmathresulttype}}
1027 \def\forestmath@convert@to{\forestmath@convert@fromto{\forestmathresulttype}}
1028 \def\forestmath@convert@from@n@to@n#1{#1}
1029 \def\forestmath@convert@from@n@to@d#1{#1\pgfmath@pt}
1030 \def\forestmath@convert@from@n@to@P#1{#1}
1031 \def\forestmath@convert@from@d@to@n#1{%
1032 \expandafter\forestmath@convert@uptodot\pgf@geT#1.\forest@eov}
1033 \def\forestmath@convert@from@d@to@d#1{#1}
1034 \def\forestmath@convert@from@d@to@P#1{\pgf@geT#1}
1035 \def\forestmath@convert@from@P@to@n#1{%
1036 \forestmath@convert@uptodot#1.\forest@eov}
1037 \def\forestmath@convert@from@P@to@d#1{#1\pgfmath@pt}
1038 \def\forestmath@convert@from@P@to@P#1{#1}
1039 \def\forestmath@convert@uptodot#1.#2\forest@eov{#1}
1040 \def\forestmathzero{\forestmath@convert@from\forestmathtype@count{0}}

```

These defer conversion (see aggregates).

```

1041 \csdef{forestmath@convert@from@n@to@_}#1{\unexpanded{#1}}
1042 \csdef{forestmath@convert@from@d@to@_}#1{\unexpanded{#1}}
1043 \csdef{forestmath@convert@from@P@to@_}#1{\unexpanded{#1}}

```

Sets \pgfmathresulttype to the type of #1.

```

1044 \def\forestmathsettypefrom#1{%
1045 \forest@isnumdim{%
1046 \let\forestmathresulttype\forestmathtype@count
1047 }{%
1048 \let\forestmathresulttype\forestmathtype@dimen
1049 }{%
1050 \let\forestmathresulttype\forestmathtype@unitless

```

```
1051 }%
1052 }
```

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.

```
1053 \def\forestmathadd#1#2{\edef\forestmathresult{%
1054   \csname forestmathadd@\forestmathresulttype\endcsname{#1}{#2}}
1055 \def\forestmathadd@#1#2#3{\edef\forestmathresult{%
1056   \csname forestmathadd@#1\endcsname{#2}{#3}}
1057 \def\forestmathadd@n#1#2{\the\numexpr#1+#2\relax}
1058 \def\forestmathadd@d#1#2{\the\dimexpr#1+#2\relax}
1059 \def\forestmathadd@P#1#2{\expandafter\Pgf@geT\the\dimexpr#1pt+#2pt\relax}
1060 \def\forestmathmultiply#1#2{%
1061   \csname forestmathmultiply@\forestmathresulttype\endcsname{#1}{#2}}
1062 \def\forestmathmultiply@#1#2#3{%
1063   \csname forestmathmultiply@#1\endcsname{#2}{#3}}
1064 \def\forestmathmultiply@n#1#2{\edef\forestmathresult{%
1065   \the\numexpr#1*#2\relax}}
1066 \def\forestmathmultiply@d#1#2{%
1067   \edef\forestmath@marshal{\forestmathmultiply@d@{#1}{#2}}\forestmath@marshal
1068 }
1069 \def\forestmathmultiply@d@#1#2{%
1070   \edef\forestmath@marshal{%
1071     \noexpand\pgfmathmultiply@{\Pgf@geT#1}{\Pgf@geT#2}%
1072   }\forestmath@marshal
1073   \edef\forestmathresult{\pgfmathresult\pgfmath@pt}%
1074 }
1075 \def\forestmathmultiply@P#1#2{%
1076   \pgfmathmultiply@{#1}{#2}%
1077   \let\forestmathresult\pgfmathresult
1078 }
```

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.

```
1079 \def\forestmathdivide#1#2{%
1080   \csname forestmathdivide@\forestmathresulttype\endcsname{#1}{#2}}
1081 \def\forestmathdivide@#1#2#3{%
1082   \csname forestmathdivide@#1\endcsname{#2}{#3}}
1083 \def\forestmathdivide@n#1#2{\edef\forestmathresult{%
1084   \the\numexpr#1/#2\relax}}
1085 \def\forestmathdivide@d#1#2{\edef\forestmathresult{%
1086   \the\dimexpr#1/#2\relax}}
1087 \def\forestmathdivide@P#1#2{%
1088   \edef\forest@marshal{%
1089     \noexpand\pgfmathdivide{+#1}{+#2}%
1090   }\forest@marshal
1091   \let\forestmathresult\pgfmathresult
1092 }
```

Booleans.

```
1093 \def\forestmathtrue{%
1094   \def\forestmathresult{1}%
1095   \let\forestmathresulttype\forestmathtype@count}
1096 \def\forestmathfalse{%
1097   \def\forestmathresult{0}%
1098   \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 <.

```

1099 \def\forestmath@if@lt@n#1#2{\ifnum#1<#2\relax
1100 \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1101 \def\forestmath@if@lt@d#1#2{\ifdim#1<#2\relax
1102 \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1103 \def\forestmath@if@lt@P#1#2{\ifdim#1pt<#2pt
1104 \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1105 \def\forestmath@if@lt@t#1#2{\ifnum\pdfstrcmp{#1}{#2}<0
1106 \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1107 \def\forestmath@if@lt@T#1#2{\ifnum\pdfstrcmp{#1}{#2}>0
1108 \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1109 \def\forest@cmp@error#1#2{\PackageError{forest}{Comparison
1110 ("<" or ">") of generic type arguments "#1" and "#2"
1111 makes no sense}{Use one of argument processor instructions
1112 "n", "d", "P" or "t" to change the type. Use package option
1113 "debug=process" to see what's happening here.}}
1114 \cslet{forestmath@if@lt@_}\forest@cmp@error

```

Low level =.

```

1115 \def\forestmath@if@eq@n#1#2{\ifnum#1=#2\relax
1116 \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1117 \def\forestmath@if@eq@d#1#2{\ifdim#1=#2\relax
1118 \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1119 \def\forestmath@if@eq@P#1#2{\ifdim#1pt=#2pt
1120 \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1121 \def\forestmath@if@eq@t#1#2{\ifnum\pdfstrcmp{#1}{#2}=0
1122 \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1123 \let\forestmath@if@eq@T\forestmath@if@eq@t
1124 \csdef{forestmath@if@eq@_}#1#2{%
1125 \def\forestmath@tempa{#1}%
1126 \def\forestmath@tempb{#2}%
1127 \ifx\forestmath@tempa\forestmath@tempb
1128 \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}

```

High level <, > and =.

```

1129 \def\forestmathlt#1#2{%
1130 \csname forestmath@if@lt@\forestmathresulttype\endcsname{#1}{#2}%
1131 \forestmathtrue
1132 \forestmathfalse}
1133 \def\forestmathlt@#1#2#3{%
1134 \csname forestmath@if@lt@#1\endcsname{#2}{#3}%
1135 \forestmathtrue
1136 \forestmathfalse}
1137 \def\forestmathgt#1#2{%
1138 \csname forestmath@if@lt@\forestmathresulttype\endcsname{#2}{#1}%
1139 \forestmathtrue
1140 \forestmathfalse}
1141 \def\forestmathgt@#1#2#3{%
1142 \csname forestmath@if@lt@#1\endcsname{#3}{#2}%
1143 \forestmathtrue
1144 \forestmathfalse}
1145 \def\forestmatheq#1#2{%
1146 \csname forestmath@if@eq@\forestmathresulttype\endcsname{#1}{#2}%
1147 \forestmathtrue
1148 \forestmathfalse}
1149 \def\forestmatheq@#1#2#3{%
1150 \csname forestmath@if@eq@#1\endcsname{#2}{#3}%
1151 \forestmathtrue
1152 \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.

```

1153 \def\forestmathmin{\forestmath@minmax{min}{\forestmathresulttype}}
1154 \def\forestmathmax{\forestmath@minmax{max}{\forestmathresulttype}}
1155 \def\forestmathmin@{\forestmath@minmax{min}}
1156 \def\forestmathmax@{\forestmath@minmax{max}}
1157 \def\forestmath@minmax#1#2#3#4{% #1=min/max, #2=type, #3,#4=args
1158   \edef\forestmath@tempa{#3}%
1159   \edef\forestmath@tempb{#4}%
1160   \if\relax\detokenize\expandafter{\forestmath@tempa}\relax
1161     \forestmath@minmax@one{#1}{#2}\forestmath@tempb
1162   \else
1163     \if\relax\detokenize\expandafter{\forestmath@tempb}\relax
1164       \forestmath@minmax@one{#1}{#2}\forestmath@tempa
1165     \else
1166       \csname forestmath@#1\endcsname{#2}%
1167     \fi
1168   \fi
1169 }
1170 \def\forestmath@minmax@one#1#2#3{% #1=min/max, #2=type, #3 = the (possibly) non-empty arg
1171   \ifcsname forestmath@#1@one@#2\endcsname
1172     \csname forestmath@#1@one@#2\endcsname#3%
1173   \else
1174     \let\forestmathresult#3%
1175   \fi
1176 }
1177 \def\forestmath@min@one@t#1{\let\forestmathresult\forest@empty}
1178 \def\forestmath@max@one@t#1{\let\forestmathresult#1}
1179 \def\forestmath@min@one@T#1{\let\forestmathresult#1}
1180 \def\forestmath@max@one@T#1{\let\forestmathresult\forest@empty}
1181
1182 \def\forestmath@min#1{% #1 = type
1183   \csname forestmath@if@lt@#1\endcsname\forestmath@tempa\forestmath@tempb
1184   {\let\forestmathresult\forestmath@tempa}%
1185   {\let\forestmathresult\forestmath@tempb}%
1186 }
1187 \def\forestmath@max#1{% #1 = type
1188   \csname forestmath@if@lt@#1\endcsname\forestmath@tempa\forestmath@tempb
1189   {\let\forestmathresult\forestmath@tempb}%
1190   {\let\forestmathresult\forestmath@tempa}%
1191 }

```

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 given in #3: it can be 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.

```

1192 \def\forest@sort#1#2#3#4#5{%

```

¹In 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`.


```

1193 \let\forest@sort@cmp#1\relax
1194 \let\forest@sort@let#2\relax
1195 \let\forest@sort@direction#3\relax
1196 \forest@sort@{#4}{#5}%
1197 }

```

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.

```

1198 \def\forest@quicksort@minarraylength{10000}
1199 \def\forest@sort#1#2{%
1200   \ifnum#1<#2\relax\escapeif{%
1201     \forest@sort@m=#2
1202     \advance\forest@sort@m -#1
1203     \ifnum\forest@sort@m>\forest@quicksort@minarraylength\relax\escapeif{%
1204       \forest@quicksort{#1}{#2}%
1205     }\else\escapeif{%
1206       \forest@insertionsort{#1}{#2}%
1207     }\fi
1208   }\fi
1209 }

```

Various counters and macros needed by the sorting algorithms.

```

1210 \newcount\forest@sort@m\newcount\forest@sort@k\newcount\forest@sort@p
1211 \def\forest@sort@ascending{>}
1212 \def\forest@sort@descending{<}
1213 \def\forest@sort@cmp{%
1214   \PackageError{sort}{You must define forest@sort@cmp function before calling
1215     sort}{The macro must take two arguments, indices of the array
1216     elements to be compared, and return '=' if the elements are equal
1217     and '>'/ '<' if the first is greater /less than the secong element.}%
1218 }
1219 \def\forest@sort@cmp@gt{\def\forest@sort@cmp@result{>}}
1220 \def\forest@sort@cmp@lt{\def\forest@sort@cmp@result{<}}
1221 \def\forest@sort@cmp@eq{\def\forest@sort@cmp@result{=}}
1222 \def\forest@sort@let{%
1223   \PackageError{sort}{You must define forest@sort@let function before calling
1224     sort}{The macro must take two arguments, indices of the array:
1225     element 2 must be copied onto element 1.}%
1226 }

```

Quick sort macro (adapted from [laansort](#)).

```

1227 \newloop\forest@sort@loop
1228 \newloop\forest@sort@loopA
1229 \def\forest@quicksort#1#2{%

```

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

```

1230   \forest@sort@m=#2
1231   \advance\forest@sort@m -#1
1232   \ifodd\forest@sort@m\relax\advance\forest@sort@m1 \fi
1233   \divide\forest@sort@m 2
1234   \advance\forest@sort@m #1

```

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

```

1235   \forest@sort@cmp{#1}{#2}%
1236   \if\forest@sort@cmp@result=%
1237     \forest@sort@p=#1
1238   \else
1239     \if\forest@sort@cmp@result>%
1240       \forest@sort@p=#1\relax
1241     \else
1242       \forest@sort@p=#2\relax
1243     \fi
1244   \forest@sort@cmp{\the\forest@sort@p}{\the\forest@sort@m}%

```

```

1245     \if\forest@sort@cmp@result<%
1246     \else
1247         \forest@sort@p=\the\forest@sort@m
1248     \fi
1249 \fi

Exchange the pivot and the first element.
1250 \forest@sort@xch{#1}{\the\forest@sort@p}%

Counter \forest@sort@m will hold the final location of the pivot element.
1251 \forest@sort@m=#1\relax

Loop through the list.
1252 \forest@sort@k=#1\relax
1253 \forest@sort@loop
1254 \ifnum\forest@sort@k<#2\relax
1255     \advance\forest@sort@k 1

Compare the pivot and the current element.
1256     \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.
1257     \ifx\forest@sort@direction\forest@sort@cmp@result
1258         \advance\forest@sort@m 1
1259         \forest@sort@xch{\the\forest@sort@m}{\the\forest@sort@k}
1260     \fi
1261 \forest@sort@repeat

Move the pivot element into its final position.
1262 \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.
1263 \forest@sort@k=\forest@sort@m
1264 \advance\forest@sort@k -1
1265 \advance\forest@sort@m 1
1266 \edef\forest@sort@marshal{%
1267     \noexpand\forest@sort@{#1}{\the\forest@sort@k}%
1268     \noexpand\forest@sort@{\the\forest@sort@m}{#2}%
1269 }%
1270 \forest@sort@marshal
1271 }

1272 % We defines the item-exchange macro in terms of the (user-provided)
1273 % array let macro.
1274 %     \begin{macrocode}
1275 \def\forest@sort@aux{aux}
1276 \def\forest@sort@xch#1#2{%
1277     \forest@sort@let{\forest@sort@aux}{#1}%
1278     \forest@sort@let{#1}{#2}%
1279     \forest@sort@let{#2}{\forest@sort@aux}%
1280 }

Insertion sort.
1281 \def\forest@insertionsort#1#2{%
1282     \forest@sort@m=#1
1283     \edef\forest@insertionsort@low{#1}%
1284     \forest@sort@loopA
1285     \ifnum\forest@sort@m<#2
1286         \advance\forest@sort@m 1
1287         \forest@insertionsort@Qbody
1288     \forest@sort@repeatA
1289 }

```

```

1290 \newif\ifforest@insertionsort@loop
1291 \def\forest@insertionsort@qbody{%
1292   \forest@sort@let{\forest@sort@aux}{\the\forest@sort@m}%
1293   \forest@sort@k\forest@sort@m
1294   \advance\forest@sort@k -1
1295   \forest@insertionsort@looptrue
1296   \forest@sort@loop
1297   \ifforest@insertionsort@loop
1298     \forest@insertionsort@qbody
1299   \forest@sort@repeat
1300   \advance\forest@sort@k 1
1301   \forest@sort@let{\the\forest@sort@k}{\forest@sort@aux}%
1302 }
1303 \def\forest@insertionsort@qbody{%
1304   \forest@sort@cmp{\the\forest@sort@k}{\forest@sort@aux}%
1305   \ifx\forest@sort@direction\forest@sort@cmp@result\relax
1306     \forest@sort@p=\forest@sort@k
1307     \advance\forest@sort@p 1
1308     \forest@sort@let{\the\forest@sort@p}{\the\forest@sort@k}%
1309     \advance\forest@sort@k -1
1310     \ifnum\forest@sort@k<\forest@insertionsort@low\relax
1311       \forest@insertionsort@loopfalse
1312     \fi
1313   \else
1314     \forest@insertionsort@loopfalse
1315   \fi
1316 }

```

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

Compare numbers.

```

1317 \def\forest@sort@cmpnumcs#1#2{%
1318   \ifnum\csname#1\endcsname>\csname#2\endcsname\relax
1319     \forest@sort@cmp@gt
1320   \else
1321     \ifnum\csname#1\endcsname<\csname#2\endcsname\relax
1322       \forest@sort@cmp@lt
1323     \else
1324       \forest@sort@cmp@eq
1325     \fi
1326   \fi
1327 }

```

Compare dimensions.

```

1328 \def\forest@sort@cmpdimcs#1#2{%
1329   \ifdim\csname#1\endcsname>\csname#2\endcsname\relax
1330     \forest@sort@cmp@gt
1331   \else
1332     \ifdim\csname#1\endcsname<\csname#2\endcsname\relax
1333       \forest@sort@cmp@lt
1334     \else
1335       \forest@sort@cmp@eq
1336     \fi
1337   \fi
1338 }

```

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

```

1339 \def\forest@sort@cmptwodimcs#1#2#3#4{%
1340   \ifdim\csname#1\endcsname>\csname#3\endcsname\relax
1341     \forest@sort@cmp@gt
1342   \else
1343     \ifdim\csname#1\endcsname<\csname#3\endcsname\relax

```

```

1344     \forest@sort@cmp@lt
1345 \else
1346     \ifdim\csname#2\endcsname>\csname#4\endcsname\relax
1347     \forest@sort@cmp@gt
1348 \else
1349     \ifdim\csname#2\endcsname<\csname#4\endcsname\relax
1350     \forest@sort@cmp@lt
1351 \else
1352     \forest@sort@cmp@eq
1353 \fi
1354 \fi
1355 \fi
1356 \fi
1357 }

```

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).

```

1358 \def\forest@reversearray#1#2#3{%
1359 \let\forest@sort@let#1%
1360 \c@pgf@countc=#2
1361 \c@pgf@countd=#3
1362 \advance\c@pgf@countd -1
1363 \safeloop
1364 \ifnum\c@pgf@countc<\c@pgf@countd\relax
1365 \forest@sort@exch{\the\c@pgf@countc}{\the\c@pgf@countd}%
1366 \advance\c@pgf@countc 1
1367 \advance\c@pgf@countd -1
1368 \saferepeat
1369 }

```

5 The bracket representation parser

5.1 The user interface macros

Settings.

```

1370 \def\bracketset#1{\pgfkeys{/bracket}{#1}}%
1371 \bracketset{%
1372 /bracket/.is family,
1373 /handlers/.let/.style={\pgfkeyscurrentpath/.code={\let#1#1}},
1374 opening bracket/.let=\bracket@openingBracket,
1375 closing bracket/.let=\bracket@closingBracket,
1376 action character/.let=\bracket@actionCharacter,
1377 opening bracket=[,
1378 closing bracket=],
1379 action character,
1380 new node/.code n args={3}{% #1=preamble, #2=node spec, #3=cs receiving the id
1381 \forest@node@new#3%
1382 \forest@set{#3}{given options}{\forest@contentto=\unexpanded{#2}}%
1383 \ifblank{#1}{}{%
1384 \forestrset{preamble}{#1}%
1385 }%
1386 },
1387 set afterthought/.code 2 args={% #1=node id, #2=afterthought
1388 \ifblank{#2}{}{\forest@appto{#1}{given options}{,afterthought={#2}}}%
1389 }
1390 }

```

`\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 not a formal parameter of the macro.)

```
1391 \newtoks\bracket@content
1392 \newtoks\bracket@afterthought
1393 \def\bracketParse#1#2={%
1394   \def\bracketEndParsingHook{#1}%
1395   \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 afterthoughts.)

```
1396 \bracket@content={}%
1397 \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.

```
1398 \let\bracket@state\bracket@state@starting
1399 \bracket@ignorespacestrue
```

By default, don’t expand anything.

```
1400 \bracket@expandtokensfalse
```

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

```
1401 \def\bracket@parentNode{0}%
1402 \def\bracket@rootNode{0}%
1403 \def\bracket@newNode{0}%
1404 \def\bracket@afterthoughtNode{0}%
```

Finally, we start the parser.

```
1405 \bracket@Parse
1406 }
```

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.

```
1407 \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 T_EXs (undelimited) argument parsing algorithm. If a space is found, remember that, eat it up, and restart the parsing.

```
1408 \def\bracket@Parse{%
1409   \futurelet\bracket@next@token\bracket@Parse@checkForSpace
1410 }
1411 \def\bracket@Parse@checkForSpace{%
1412   \expandafter\ifx\space\bracket@next@token\@escapeif{%
1413     \ifbracket@ignorespaces\else
1414       \bracket@haveSpacetrue
1415       \fi
1416     \expandafter\bracket@Parse\romannumeral-‘0%
1417   }\else\@escapeif{%
1418     \bracket@Parse@maybeexpand
1419   }\fi
1420 }
```

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

```
1421 \newif\ifbracket@expandtokens
1422 \def\bracket@Parse@maybeexpand{%
1423   \ifbracket@expandtokens\@escapeif{%
1424     \expandafter\bracket@Parse@peekAhead\romannumeral-‘0%
1425   }\else\@escapeif{%
1426     \bracket@Parse@peekAhead
```

```
1427 }\fi
1428 }
```

We then look ahead to see what's coming.

```
1429 \def\bracket@Parse@peekAhead{%
1430 \futurelet\bracket@next@token\bracket@Parse@checkForTeXGroup
1431 }
```

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

```
1432 \def\bracket@Parse@checkForTeXGroup{%
1433 \ifx\bracket@next@token\bgroup%
1434 \escapeif{\bracket@Parse@appendGroup}%
1435 \else
1436 \escapeif{\bracket@Parse@token}%
1437 \fi
1438 }
```

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.

```
1439 \long\def\bracket@Parse@token#1{%
1440 \ifx#1\bracket@openingBracket
1441 \escapeif{\bracket@Parse@openingBracketFound}%
1442 \else
1443 \escapeif{%
1444 \ifx#1\bracket@closingBracket
1445 \escapeif{\bracket@Parse@closingBracketFound}%
1446 \else
1447 \escapeif{%
1448 \ifx#1\bracket@actionCharacter
1449 \escapeif{\futurelet\bracket@next@token\bracket@Parse@actionCharacterFound}%
1450 \else
1451 \escapeif{\bracket@Parse@appendToken#1}%
1452 \fi
1453 }%
1454 \fi
1455 }%
1456 \fi
1457 }
```

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

```
1458 \newif\ifbracket@haveSpace
1459 \newif\ifbracket@ignorespaces
1460 \def\bracket@Parse@appendSpace{%
1461 \ifbracket@haveSpace
1462 \ifcase\bracket@state\relax
1463 \eapptotoks\bracket@content\space
1464 \or
1465 \eapptotoks\bracket@afterthought\space
1466 \or
1467 \eapptotoks\bracket@afterthought\space
1468 \fi
1469 \bracket@haveSpacefalse
1470 \fi
1471 }
1472 \long\def\bracket@Parse@appendToken#1{%
1473 \bracket@Parse@appendSpace
1474 \ifcase\bracket@state\relax
1475 \lapptotoks\bracket@content{#1}%
1476 \or
1477 \lapptotoks\bracket@afterthought{#1}%
```

```

1478 \or
1479 \lapptotoks\bracket@afterthought{#1}%
1480 \fi
1481 \bracket@ignorespacesfalse
1482 \bracket@Parse
1483 }
1484 \def\bracket@Parse@appendGroup#1{%
1485 \bracket@Parse@appendSpace
1486 \ifcase\bracket@state\relax
1487 \apptotoks\bracket@content{#1}}%
1488 \or
1489 \apptotoks\bracket@afterthought{#1}}%
1490 \or
1491 \apptotoks\bracket@afterthought{#1}}%
1492 \fi
1493 \bracket@ignorespacesfalse
1494 \bracket@Parse
1495 }

```

Declare states.

```

1496 \def\bracket@state@inContent{0}
1497 \def\bracket@state@inAfterthought{1}
1498 \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.)

```

1499 \def\bracket@Parse@openingBracketFound{%
1500 \bracket@haveSpacefalse
1501 \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.

```

1502 \escapeif{%
1503 \bracket@createNode
1504 \ifnum\bracket@parentNode=0 \else
1505 \forest@node@Append{\bracket@parentNode}{\bracket@newNode}%
1506 \fi
1507 \let\bracket@parentNode\bracket@newNode
1508 \bracket@Parse
1509 }%
1510 \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.

```

1511 \escapeif{%
1512 \bracket@addAfterthought
1513 \let\bracket@state\bracket@state@inContent
1514 \bracket@Parse
1515 }%
1516 \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.

```

1517 \escapeif{%
1518 \let\bracket@state\bracket@state@inContent
1519 \bracket@Parse
1520 }%

```

```

1521 \fi
1522 }
1523 \def\bracket@Parse@closingBracketFound{%
1524 \bracket@haveSpacefalse
1525 \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.

```

1526 \escapeif{%
1527 \bracket@createNode
1528 \ifnum\bracket@parentNode=0
1529 \escapeif\bracket@endParsingHook
1530 \else
1531 \escapeif{%
1532 \let\bracket@afterthoughtNode\bracket@newNode
1533 \let\bracket@state\bracket@state@inAfterthought
1534 \forest@node@Append{\bracket@parentNode}{\bracket@newNode}%
1535 \bracket@Parse
1536 }%
1537 \fi
1538 }%
1539 \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.

```

1540 \escapeif{%
1541 \bracket@addAfterthought
1542 \let\bracket@afterthoughtNode\bracket@parentNode
1543 \edef\bracket@parentNode{\forestOve{\bracket@parentNode}{@parent}}%
1544 \ifnum\bracket@parentNode=0
1545 \expandafter\bracket@endParsingHook
1546 \else
1547 \expandafter\bracket@Parse
1548 \fi
1549 }%
1550 \else % starting

```

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

```

1551 \PackageError{forest}{You're attempting to start a bracket representation
1552 with a closing bracket}{}%
1553 \fi
1554 }

```

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

```

1555 \def\bracket@Parse@actionCharacterFound{%
  If a braced expression follows, its contents will be fully expanded.
1556 \ifx\bracket@next@token\bgroup\escapeif{%
1557 \bracket@Parse@action@expandgroup
1558 }\else\escapeif{%
1559 \bracket@Parse@action@notagroup
1560 }\fi
1561 }
1562 \def\bracket@Parse@action@expandgroup#1{%
1563 \edef\bracket@Parse@action@expandgroup@macro{#1}%
1564 \expandafter\bracket@Parse\bracket@Parse@action@expandgroup@macro
1565 }
1566 \let\bracket@action@fullyexpandCharacter+

```



```

1567 \let\bracket@action@dontexpandCharacter-
1568 \let\bracket@action@executeCharacter!
1569 \def\bracket@Parse@action@notagroup#1{%

```

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

```

1570 \ifx#1\bracket@action@fullyexpandCharacter\@escapeif{%
1571 \bracket@expandtokenstrue\bracket@Parse
1572 }\else\@escapeif{%

```

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

```

1573 \ifx#1\bracket@action@dontexpandCharacter\@escapeif{%
1574 \bracket@expandtokensfalse\bracket@Parse
1575 }\else\@escapeif{%

```

Inhibit expansion of the next token.

```

1576 \ifx#10\@escapeif{%
1577 \bracket@Parse@appendToken
1578 }\else\@escapeif{%

```

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

```

1579 \ifx#1\bracket@actionCharacter
1580 \else\@escapeif{%

```

Anything else will be expanded once.

```

1581 \expandafter\bracket@Parse#1%
1582 }\fi
1583 }\fi
1584 }\fi
1585 }\fi
1586 }

```

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.

```

1587 \def\bracket@createNode{%
1588 \ifnum\bracket@rootNode=0
1589 % root node
1590 \bracketset{new node/.expanded=%
1591 {\the\bracket@afterthought}}%
1592 {\the\bracket@content}}%
1593 \noexpand\bracket@newNode
1594 }%
1595 \bracket@afterthought={}%
1596 \let\bracket@rootNode\bracket@newNode
1597 \expandafter\let\bracket@saveRootNodeTo\bracket@newNode
1598 \else
1599 % other nodes
1600 \bracketset{new node/.expanded=%
1601 {}}%
1602 {\the\bracket@content}}%
1603 \noexpand\bracket@newNode
1604 }%
1605 \fi
1606 \bracket@content={}%
1607 }

```

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

```

1608 \def\bracket@addAfterthought{%

```

```

1609 \bracketset{%
1610   set afterthought/.expanded={\bracket@afterthoughtNode}{\the\bracket@afterthought}%
1611 }%
1612 \bracket@afterthought={}%
1613 }

```

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.

```

1614 % full expansion expands precisely to the value
1615 \def\forestov#1{\expandafter\expandonce\csname fRsT\forest@cn/#1\endcsname}
1616 % full expansion expands all the way
1617 \def\forestove#1{\csname fRsT\forest@cn/#1\endcsname}
1618 % full expansion expands to the cs holding the value
1619 \def\forestom#1{\expandonce{\csname fRsT\forest@cn/#1\endcsname}}
1620 \def\forestogget#1#2{\expandafter\let\expandafter#2\csname fRsT\forest@cn/#1\endcsname}
1621 \def\forestolet#1#2{\expandafter\let\csname fRsT\forest@cn/#1\endcsname#2}
1622 % \def\forestocslet#1#2{%
1623 %   \edef\forest@marshal{%
1624 %     \noexpand\pgfkeyslet{/forest/@node/\forest@cn/#1}{\expandonce{\csname#2\endcsname}}%
1625 %   }\forest@marshal
1626 % }
1627 \def\forestoset#1#2{\expandafter\edef\csname fRsT\forest@cn/#1\endcsname{\unexpanded{#2}}}
1628 \def\forestoeset#1#2
1629   {\expandafter\edef\csname fRsT\forest@cn/#1\endcsname
1630     % {#2}
1631   }
1632 \def\forestooppto#1#2{%
1633   \forestoeset{#1}{\forestov{#1}\unexpanded{#2}}%
1634 }
1635 \def\forestoiifdefined#1#2#3
1636   {%
1637   \ifcsdef{fRsT\forest@cn/#1}%{#2}{#3}%
1638 }

```

User macros for retrieving node options of the current node.

```

1639 \let\forestoption\forestov
1640 \let\foresteooption\forestove

```

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

```

1641 \def\forestOv#1#2{\expandafter\expandonce\csname fRsT#1/#2\endcsname}
1642 \def\forestOve#1#2{\csname fRsT#1/#2\endcsname}
1643 % full expansion expands to the cs holding the value
1644 \def\forestOm#1#2{\expandonce{\csname fRsT#1/#2\endcsname}}
1645 \def\forestOget#1#2#3{\expandafter\let\expandafter#3\csname fRsT#1/#2\endcsname}
1646 \def\forestOlet#1#2#3{\expandafter\let\csname fRsT#1/#2\endcsname#3}
1647 % \def\forestOcslet#1#2#3{%
1648 %   \edef\forest@marshal{%
1649 %     \noexpand\pgfkeyslet{/forest/@node/#1/#2}{\expandonce{\csname#3\endcsname}}%
1650 %   }\forest@marshal
1651 % }
1652 \def\forestOset#1#2#3{\expandafter\edef\csname fRsT#1/#2\endcsname{\unexpanded{#3}}}
1653 \def\forestOeset#1#2#3
1654   {\expandafter\edef\csname fRsT#1/#2\endcsname
1655     % {#3}

```

```

1656 }
1657 \def\forest0appto#1#2#3{%
1658   \forest0eset{#1}{#2}{\forest0v{#1}{#2}\unexpanded{#3}}%
1659 }
1660 \def\forest0eappto#1#2#3{%
1661   \forest0eset{#1}{#2}{\forest0v{#1}{#2}#3}%
1662 }
1663 \def\forest0preto#1#2#3{%
1664   \forest0eset{#1}{#2}{\unexpanded{#3}\forest0v{#1}{#2}}%
1665 }
1666 \def\forest0epreto#1#2#3{%
1667   \forest0eset{#1}{#2}{#3\forest0v{#1}{#2}}%
1668 }
1669 \def\forest0ifdefined#1#2%#3#4
1670 {%
1671   \ifcsdef{fRsT#1/#2}%{#3}{#4}%
1672 }
1673 \def\forest0let0#1#2#3#4{% option #2 of node #1 <-- option #4 of node #3
1674   \forest0get{#3}{#4}\forestoption@temp
1675   \forest0let{#1}{#2}\forestoption@temp}
1676 \def\forest0leto#1#2#3{%
1677   \forestoget{#3}\forestoption@temp
1678   \forest0let{#1}{#2}\forestoption@temp}
1679 \def\forest0let0#1#2#3{%
1680   \forest0get{#2}{#3}\forestoption@temp
1681   \forestolet{#1}\forestoption@temp}
1682 \def\forest0leto#1#2{%
1683   \forestoget{#2}\forestoption@temp
1684   \forestolet{#1}\forestoption@temp}

  Macros for retrieving node options given by (relative node name).(option).
1685 \def\forestRNOget#1#2{% #1=rn!option, #2 = receiving cs
1686   \pgfutil@in@{.}{#1}%
1687   \ifpgfutil@in@
1688     \forestRNOget@rn#2#1\forest@END
1689   \else
1690     \forestoget{#1}#2%
1691   \fi
1692 }
1693 \def\forestRNOget@rn#1#2.#3\forest@END{%
1694   \forest@forthis{%
1695     \forest@nameandgo{#2}%
1696     \forestoget{#3}#1%
1697   }%
1698 }
1699 \def\forestRNO@getvalueandtype#1#2#3{% #1=rn.option, #2,#3 = receiving css
1700   \pgfutil@in@{.}{#1}%
1701   \ifpgfutil@in@
1702     \forestRNO@getvalueandtype@rn#2#3#1\forest@END
1703   \else
1704     \forestoget{#1}#2%
1705     \pgfkeysgetvalue{/forest/#1/@type}#3%
1706   \fi
1707 }
1708 \def\forestRNO@getvalueandtype@rn#1#2#3.#4\forest@END{%
1709   % #1,#2=receiving css, #3=relative node name, #4=option name
1710   \forest@forthis{%
1711     \forest@nameandgo{#3}%
1712     \forestoget{#4}#1%
1713   }%
1714   \pgfkeysgetvalue{/forest/#4/@type}#2%

```

1715 }

Macros for retrieving/setting registers.

```
1716 % full expansion expands precisely to the value
1717 \def\forestrv#1{\expandafter\expandonce\csname fRsT/#1\endcsname}
1718 % full expansion expands all the way
1719 \def\forestrve#1{\csname fRsT/#1\endcsname}
1720 % full expansion expands to the cs holding the value
1721 \def\forestrm#1{\expandonce{\csname fRsT/#1\endcsname}}
1722 \def\forestrget#1#2{\expandafter\let\expandafter#2\csname fRsT/#1\endcsname}
1723 \def\forestrlet#1#2{\expandafter\let\csname fRsT/#1\endcsname#2}
1724 % \def\forestrcslet#1#2{%
1725 %   \edef\forest@marshal{%
1726 %     \noexpand\pgfkeyslet{/forest/@node/register/#1}{\expandonce{\csname#2\endcsname}}%
1727 %   }\forest@marshal
1728 % }
1729 \def\forestrset#1#2{\expandafter\edef\csname fRsT/#1\endcsname{\unexpanded{#2}}}
1730 \def\forestreset#1#2
1731   {\expandafter\edef\csname fRsT/#1\endcsname}{%#2}
1732 \def\forestrappto#1#2{%
1733   \forestreset{#1}{\forestrv{#1}\unexpanded{#2}}%
1734 }
1735 \def\forestrpreto#1#2{%
1736   \forestreset{#1}{\unexpanded{#2}\forestrv{#1}}%
1737 }
1738 \def\forestrifdefined#1#2#3
1739 {%
1740   \ifcsdef{fRsT/#1}{%#2}{%#3}%
1741 }
```

User macros for retrieving node options of the current node.

```
1742 \def\forestregister#1{\forestrv{#1}}
1743 \def\foresteregister#1{\forestrve{#1}}
```

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

```
1744 \def\forest@node@init{%
1745   \forestoset{@parent}{0}%
1746   \forestoset{@previous}{0}% previous sibling
1747   \forestoset{@next}{0}% next sibling
1748   \forestoset{@first}{0}% primary child
1749   \forestoset{@last}{0}% last child
1750 }
1751 \def\forestoinit#1{%
1752   \pgfkeysgetvalue{/forest/#1}\forestoinit@temp
1753   \forestolet{#1}\forestoinit@temp
1754 }
1755 \newcount\forest@node@maxid
1756 \def\forest@node@new#1{% #1 = cs receiving the new node id
1757   \advance\forest@node@maxid1
1758   \forest@fornode{\the\forest@node@maxid}{%
1759     \forest@node@init
1760     \forestoeset{id}{\forest@cn}%
1761     \forest@node@setname{node@\forest@cn}%
1762     \forest@initializefromstandardnode
1763     \edef#1{\forest@cn}%
1764   }%
1765 }
1766 \let\forestoinit@orig\forestoinit
1767 \def\forest@node@copy#1#2{% #1=from node id, cs receiving the new node id
1768   \advance\forest@node@maxid1
1769   \def\forestoinit##1{\ifstrequal{##1}{name}{\forestoset{name}{node@\forest@cn}}{\forestolet0{##1}{#1}{##1}}}
1770   \forest@fornode{\the\forest@node@maxid}{%
```

```

1771 \forest@node@init
1772 \forestoaset{id}{\forest@cn}%
1773 \forest@node@setname{\forest@copy@name@template{\forestOve{#1}{name}}}%
1774 \edef#2{\forest@cn}%
1775 }%
1776 \let\forest@init\forest@init@orig
1777 }
1778 \forestset{
1779 copy name template/.code={\def\forest@copy@name@template##1{#1}},
1780 copy name template/.default={node@the\forest@node@maxid},
1781 copy name template
1782 }
1783 \def\forest@tree@copy#1#2{% #1=from node id, #2=cs receiving the new node id
1784 \forest@node@copy{#1}\forest@node@copy@temp@id
1785 \forest@for@node{\forest@node@copy@temp@id}{%
1786 \expandafter\forest@tree@copy@\expandafter{\forest@node@copy@temp@id}{#1}%
1787 \edef#2{\forest@cn}%
1788 }%
1789 }
1790 \def\forest@tree@copy@#1#2{%
1791 \forest@node@Foreachchild{#2}{%
1792 \expandafter\forest@tree@copy@\expandafter{\forest@cn}\forest@node@copy@temp@childid
1793 \forest@node@Append{#1}{\forest@node@copy@temp@childid}%
1794 }%
1795 }

```

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.

```

1796 \def\forest@cn{0}
1797 \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.

```

1798 \def\forest@node@append#1{\expandtwonumberargs\forest@node@Append{\forest@cn}{#1}}
1799 \def\forest@node@prepend#1{\expandtwonumberargs\forest@node@Insertafter{\forest@cn}{#1}{0}}
1800 \def\forest@node@insertafter#1#2{%
1801 \expandthreenumberargs\forest@node@Insertafter{\forest@cn}{#1}{#2}}
1802 \def\forest@node@insertbefore#1#2{%
1803 \expandthreenumberargs\forest@node@Insertafter{\forest@cn}{#1}{\forestOve{#2}{@previous}}%
1804 }
1805 \def\forest@node@remove{\expandnumberarg\forest@node@Remove{\forest@cn}}
1806 \def\forest@node@Append#1#2{\expandtwonumberargs\forest@node@Append@{#1}{#2}}
1807 \def\forest@node@Prepend#1#2{\expandtwonumberargs\forest@node@Insertafter{#1}{#2}{0}}
1808 \def\forest@node@Insertafter#1#2#3{% #2 is inserted after #3
1809 \expandthreenumberargs\forest@node@Insertafter@{#1}{#2}{#3}%
1810 }
1811 \def\forest@node@Insertbefore#1#2#3{% #2 is inserted before #3
1812 \expandthreenumberargs\forest@node@Insertafter{#1}{#2}{\forestOve{#3}{@previous}}%
1813 }
1814 \def\forest@node@Remove#1{\expandnumberarg\forest@node@Remove@{#1}}
1815 \def\forest@node@Insertafter@#1#2#3{%
1816 \ifnum\forestOve{#2}{@parent}=0
1817 \else
1818 \PackageError{forest}{Insertafter(#1,#2,#3):
1819 node #2 already has a parent (\forestOve{#2}{@parent})}{}%
1820 \fi

```

```

1821 \ifnum#3=0
1822 \else
1823   \ifnum#1=\forestOve{#3}{@parent}
1824   \else
1825     \PackageError{forest}{Insertafter(#1,#2,#3): node #1 is not the parent of the
1826       intended sibling #3 (with parent \forestOve{#3}{@parent})}{}%
1827   \fi
1828 \fi
1829 \forestOset{#2}{@parent}{#1}%
1830 \forestOset{#2}{@previous}{#3}%
1831 \ifnum#3=0
1832   \forestOget{#1}{@first}\forest@node@temp
1833   \forestOset{#1}{@first}{#2}%
1834 \else
1835   \forestOget{#3}{@next}\forest@node@temp
1836   \forestOset{#3}{@next}{#2}%
1837 \fi
1838 \forestOset{#2}{@next}{\forest@node@temp}%
1839 \ifnum\forest@node@temp=0
1840   \forestOset{#1}{@last}{#2}%
1841 \else
1842   \forestOset{\forest@node@temp}{@previous}{#2}%
1843 \fi
1844 }
1845 \def\forest@node@Append@#1#2{%
1846   \ifnum\forestOve{#2}{@parent}=0
1847   \else
1848     \PackageError{forest}{Append(#1,#2):
1849       node #2 already has a parent (\forestOve{#2}{@parent})}{}%
1850   \fi
1851   \forestOset{#2}{@parent}{#1}%
1852   \forestOget{#1}{@last}\forest@node@temp
1853   \forestOset{#1}{@last}{#2}%
1854   \forestOset{#2}{@previous}{\forest@node@temp}%
1855   \ifnum\forest@node@temp=0
1856     \forestOset{#1}{@first}{#2}%
1857   \else
1858     \forestOset{\forest@node@temp}{@next}{#2}%
1859   \fi
1860 }
1861 \def\forest@node@Remove@#1{%
1862   \forestOget{#1}{@parent}\forest@node@temp@parent
1863   \ifnum\forest@node@temp@parent=0
1864   \else
1865     \forestOget{#1}{@previous}\forest@node@temp@previous
1866     \forestOget{#1}{@next}\forest@node@temp@next
1867     \ifnum\forest@node@temp@previous=0
1868       \forestOset{\forest@node@temp@parent}{@first}{\forest@node@temp@next}%
1869     \else
1870       \forestOset{\forest@node@temp@previous}{@next}{\forest@node@temp@next}%
1871     \fi
1872     \ifnum\forest@node@temp@next=0
1873       \forestOset{\forest@node@temp@parent}{@last}{\forest@node@temp@previous}%
1874     \else
1875       \forestOset{\forest@node@temp@next}{@previous}{\forest@node@temp@previous}%
1876     \fi
1877     \forestOset{#1}{@parent}{0}%
1878     \forestOset{#1}{@previous}{0}%
1879     \forestOset{#1}{@next}{0}%
1880   \fi
1881 }

```

Do some stuff and return to the current node.

```
1882 \def\forest@forthis#1{%
1883   \edef\forest@node@marshal{\unexpanded{#1}\def\noexpand\forest@cn}%
1884   \expandafter\forest@node@marshal\expandafter{\forest@cn}%
1885 }
1886 \def\forest@fornode#1#2{%
1887   \edef\forest@node@marshal{\edef\noexpand\forest@cn{#1}\unexpanded{#2}\def\noexpand\forest@cn}%
1888   \expandafter\forest@node@marshal\expandafter{\forest@cn}%
1889 }
```

Looping methods: children.

```
1890 \def\forest@node@foreachchild#1{\forest@node@Foreachchild{\forest@cn}{#1}}
1891 \def\forest@node@Foreachchild#1#2{%
1892   \forest@fornode{\forestOve{#1}{@first}}{\forest@node@@forselfandfollowingsiblings{#2}}%
1893 }
1894 \def\forest@node@@forselfandfollowingsiblings#1{%
1895   \ifnum\forest@cn=0
1896   \else
1897     \forest@forthis{#1}%
1898     \@escapeif{%
1899       \edef\forest@cn{\forestove{@next}}%
1900       \forest@node@@forselfandfollowingsiblings{#1}%
1901     }%
1902   \fi
1903 }
1904 \def\forest@node@@forselfandfollowingsiblings@reversed#1{%
1905   \ifnum\forest@cn=0
1906   \else
1907     \@escapeif{%
1908       \edef\forest@marshal{%
1909         \noexpand\def\noexpand\forest@cn{\forestove{@next}}%
1910         \noexpand\forest@node@@forselfandfollowingsiblings@reversed{\unexpanded{#1}}%
1911         \noexpand\forest@fornode{\forest@cn}{\unexpanded{#1}}%
1912       }\forest@marshal
1913     }%
1914   \fi
1915 }
1916 \def\forest@node@foreachchild@reversed#1{\forest@node@Foreachchild@reversed{\forest@cn}{#1}}
1917 \def\forest@node@Foreachchild@reversed#1#2{%
1918   \forest@fornode{\forestOve{#1}{@last}}{\forest@node@@forselfandprecedingsiblings@reversed{#2}}%
1919 }
1920 \def\forest@node@@forselfandprecedingsiblings@reversed#1{%
1921   \ifnum\forest@cn=0
1922   \else
1923     \forest@forthis{#1}%
1924     \@escapeif{%
1925       \edef\forest@cn{\forestove{@previous}}%
1926       \forest@node@@forselfandprecedingsiblings@reversed{#1}%
1927     }%
1928   \fi
1929 }
1930 \def\forest@node@@forselfandprecedingsiblings#1{%
1931   \ifnum\forest@cn=0
1932   \else
1933     \@escapeif{%
1934       \edef\forest@marshal{%
1935         \noexpand\def\noexpand\forest@cn{\forestove{@previous}}%
1936         \noexpand\forest@node@@forselfandprecedingsiblings{\unexpanded{#1}}%
1937         \noexpand\forest@fornode{\forest@cn}{\unexpanded{#1}}%
1938       }\forest@marshal
1939     }%
1939 }
```

```
1940 \fi
1941 }
```

Looping methods: (sub)tree and descendants.

```
1942 \def\forest@node@@foreach#1#2#3#4{%
1943   % #1 = do what
1944   % #2 = do that -1=before,1=after processing children
1945   % #3 & #4: normal or reversed order of children?
1946   %   #3 = @first/@last
1947   %   #4 = \forest@node@@forselfandfollowingsiblings / \forest@node@@forselfandprecedingsiblings@reversed
1948   \ifnum#2<0 \forest@forthis{#1}\fi
1949   \ifnum\forest@ve{#3}=0
1950   \else\@escapeif{%
1951     \forest@forthis{%
1952       \edef\forest@cn{\forest@ve{#3}}%
1953       #4{\forest@node@@foreach{#1}{#2}{#3}{#4}}%
1954     }%
1955   }\fi
1956   \ifnum#2>0 \forest@forthis{#1}\fi
1957 }
1958 \def\forest@node@foreach#1{%
1959   \forest@node@@foreach{#1}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1960 \def\forest@node@Foreach#1#2{%
1961   \forest@fornode{#1}{\forest@node@@foreach{#2}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1962 \def\forest@node@foreach@reversed#1{%
1963   \forest@node@@foreach{#1}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1964 \def\forest@node@Foreach@reversed#1#2{%
1965   \forest@fornode{#1}{\forest@node@@foreach{#2}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1966 \def\forest@node@foreach@childrenfirst#1{%
1967   \forest@node@@foreach{#1}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1968 \def\forest@node@Foreach@childrenfirst#1#2{%
1969   \forest@fornode{#1}{\forest@node@@foreach{#2}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1970 \def\forest@node@foreach@childrenfirst@reversed#1{%
1971   \forest@node@@foreach{#1}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1972 \def\forest@node@Foreach@childrenfirst@reversed#1#2{%
1973   \forest@fornode{#1}{\forest@node@@foreach{#2}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1974 \def\forest@node@foreach@descendant#1{%
1975   \forest@node@foreachchild{\forest@node@@foreach{#1}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1976 \def\forest@node@Foreach@descendant#1#2{%
1977   \forest@node@Foreachchild{#1}{\forest@node@@foreach{#2}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1978 \def\forest@node@foreach@descendant@reversed#1{%
1979   \forest@node@foreachchild@reversed{\forest@node@@foreach{#1}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1980 \def\forest@node@Foreach@descendant@reversed#1#2{%
1981   \forest@node@Foreachchild@reversed{#1}{\forest@node@@foreach{#2}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1982 \def\forest@node@foreach@descendant@childrenfirst#1{%
1983   \forest@node@foreachchild{\forest@node@@foreach{#1}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1984 \def\forest@node@Foreach@descendant@childrenfirst#1#2{%
1985   \forest@node@Foreachchild{#1}{\forest@node@@foreach{#2}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1986 \def\forest@node@foreach@descendant@childrenfirst@reversed#1{%
1987   \forest@node@foreachchild@reversed{\forest@node@@foreach{#1}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1988 \def\forest@node@Foreach@descendant@childrenfirst@reversed#1#2{%
1989   \forest@node@Foreachchild@reversed{#1}{\forest@node@@foreach{#2}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}

```

Looping methods: breadth-first.

```
1990 \def\forest@node@foreach@breadthfirst#1#2{% #1 = max level, #2 = code
1991   \forest@node@Foreach@breadthfirst@{\forest@cn}{@first}{@next}{#1}{#2}}
1992 \def\forest@node@foreach@breadthfirst@reversed#1#2{% #1 = max level, #2 = code
1993   \forest@node@Foreach@breadthfirst@{\forest@cn}{@last}{@previous}{#1}{#2}}
1994 \def\forest@node@Foreach@breadthfirst#1#2#3{% #1 = node id, #2 = max level, #3 = code
1995   \forest@node@Foreach@breadthfirst@{#1}{@first}{@next}{#2}{#3}}
1996 \def\forest@node@Foreach@breadthfirst@reversed#1#2#3{% #1 = node id, #2 = max level, #3 = code
1997   \forest@node@Foreach@breadthfirst@{#1}{@last}{@previous}{#2}{#3}}

```



```

1998 \def\forest@node@Foreach@breadthfirst@#1#2#3#4#5{%
1999 % #1 = root node,
2000 % #2 = @first/@last, #3 = @next/@previous (must be in sync with #2),
2001 % #4 = max level (< 0 means infinite)
2002 % #5 = code to execute at each node
2003 \forest@node@Foreach@breadthfirst@processqueue{#1,}{#2}{#3}{#4}{#5}%
2004 }
2005 \def\forest@node@Foreach@breadthfirst@processqueue#1#2#3#4#5{%
2006 % #1 = queue,
2007 % #2 = @first/@last, #3 = @next/@previous (must be in sync with #2),
2008 % #4 = max level (< 0 means infinite)
2009 % #5 = code to execute at each node
2010 \ifstrempy{#1}{}{%
2011   \forest@node@Foreach@breadthfirst@processqueue@#1\forest@node@Foreach@breadthfirst@processqueue@
2012     {#2}{#3}{#4}{#5}%
2013 }%
2014 }
2015 \def\forest@node@Foreach@breadthfirst@processqueue@#1,#2\forest@node@Foreach@breadthfirst@processqueue@#3#4#5{%
2016 % #1 = first,
2017 % #2 = rest,
2018 % #3 = @first/@last, #4 = next/previous (must be in sync with #2),
2019 % #5 = max level (< 0 means infinite)
2020 % #6 = code to execute at each node
2021 \forest@node@Foreach@breadthfirst@processqueue@#1{#2}{#3}{#4}{#5}{#6}%
2022 #6%
2023 \ifnum#5<0
2024   \forest@node@Getlistofchildren\forest@temp{#3}{#4}%
2025 \else
2026   \ifnum\forest@temp{#5}>#5\relax
2027     \def\forest@temp{#5}%
2028   \else
2029     \forest@node@Getlistofchildren\forest@temp{#3}{#4}%
2030   \fi
2031 \fi
2032 \edef\forest@marshal{%
2033   \noexpand\forest@node@Foreach@breadthfirst@processqueue@{\unexpanded{#2}\forest@temp}%
2034   {#3}{#4}{#5}{\unexpanded{#6}}%
2035 }%
2036 }%
2037 }
2038 \def\forest@node@Getlistofchildren#1#2#3{% #1 = list cs, #2 = @first/@last, #3 = @next/@previous
2039 \forest@node@Getlistofchildren{\forest@cn}{#1}{#2}{#3}%
2040 }
2041 \def\forest@node@Getlistofchildren#1#2#3#4{% #1 = node, #2 = list cs, #3 = @first/@last, #4 = @next/@previous
2042 \def#2{}%
2043 \ifnum\forest@temp{#3}=0
2044 \else
2045   \eappto#2{\forest@temp{#1}{#3},}%
2046   \escapeif{%
2047     \edef\forest@marshal{%
2048       \noexpand\forest@node@Getlistofchildren@{\forest@temp{#1}{#3}}\noexpand#2{#4}%
2049     }%
2050   }%
2051 \fi
2052 }
2053 \def\forest@node@Getlistofchildren@#1#2#3{% #1 = node, #2 = list cs, #3 = @next/@previous
2054 \ifnum\forest@temp{#1}{#3}=0
2055 \else
2056   \eappto#2{\forest@temp{#1}{#3},}%
2057   \escapeif{%
2058     \edef\forest@marshal{%

```

```

2059     \noexpand\forest@node@Getlistofchildren@{\forestOve{#1}{#3}}\noexpand#2{#3}%
2060     }\forest@marshal
2061   }%
2062 \fi
2063 }

    Compute n, n', n children and level.
2064 \def\forest@node@Compute@numeric@ts@info@#1{%
2065   \forest@node@Foreach{#1}{\forest@node@@compute@numeric@ts@info}%
2066   \ifnum\forestOve{#1}{@parent}=0
2067   \else
2068     \forest@fornode{#1}{\forest@node@@compute@numeric@ts@info@nbar}%
2069     % hack: the parent of the node we called the update for gets +1 for n_children
2070     \edef\forest@node@temp{\forestOve{#1}{@parent}}%
2071     \forestOset{\forest@node@temp}{n children}{%
2072       \number\numexpr\forestOve{\forest@node@temp}{n children}-1%
2073     }%
2074   \fi
2075   \forest@node@Foreachdescendant{#1}{\forest@node@@compute@numeric@ts@info@nbar}%
2076 }
2077 \def\forest@node@@compute@numeric@ts@info{%
2078   \forestoset{n children}{0}%
2079   %
2080   \edef\forest@node@temp{\forestove{@previous}}%
2081   \ifnum\forest@node@temp=0
2082     \forestoset{n}{1}%
2083   \else
2084     \forestoeset{n}{\number\numexpr\forestOve{\forest@node@temp}{n}+1}%
2085   \fi
2086   %
2087   \edef\forest@node@temp{\forestove{@parent}}%
2088   \ifnum\forest@node@temp=0
2089     \forestoset{n}{0}%
2090     \forestoset{n'}{0}%
2091     \forestoset{level}{0}%
2092   \else
2093     \forestOset{\forest@node@temp}{n children}{%
2094       \number\numexpr\forestOve{\forest@node@temp}{n children}+1%
2095     }%
2096     \forestoeset{level}{%
2097       \number\numexpr\forestOve{\forest@node@temp}{level}+1%
2098     }%
2099   \fi
2100 }
2101 \def\forest@node@@compute@numeric@ts@info@nbar{%
2102   \forestoeset{n'}{\number\numexpr\forestOve{\forestove{@parent}}{n children}-\forestove{n}+1}%
2103 }
2104 \def\forest@node@compute@numeric@ts@info#1{%
2105   \expandnumberarg\forest@node@Compute@numeric@ts@info@{\forest@cn}%
2106 }
2107 \def\forest@node@Compute@numeric@ts@info#1{%
2108   \expandnumberarg\forest@node@Compute@numeric@ts@info@{#1}%
2109 }

    Tree structure queries.
2110 \def\forest@node@rootid{%
2111   \expandnumberarg\forest@node@Rootid{\forest@cn}%
2112 }
2113 \def\forest@node@Rootid#1{% #1=node
2114   \ifnum\forestOve{#1}{@parent}=0
2115     #1%
2116   \else

```

```

2117 \escapeif{\expandnumberarg\forest@node@Rootid{\forestOve{#1}{@parent}}}%
2118 \fi
2119 }
2120 \def\forest@node@nthchildid#1{% #1=n
2121 \ifnum#1<1
2122 0%
2123 \else
2124 \expandnumberarg\forest@node@nthchildid@{\number\forestove{@first}}{#1}%
2125 \fi
2126 }
2127 \def\forest@node@nthchildid@#1#2{%
2128 \ifnum#1=0
2129 0%
2130 \else
2131 \ifnum#2>1
2132 \escapeifif{\expandtwonumberargs
2133 \forest@node@nthchildid@{\forestOve{#1}{@next}}{\numexpr#2-1}}%
2134 \else
2135 #1%
2136 \fi
2137 \fi
2138 }
2139 \def\forest@node@nbarthchildid#1{% #1=n
2140 \expandnumberarg\forest@node@nbarthchildid@{\number\forestove{@last}}{#1}%
2141 }
2142 \def\forest@node@nbarthchildid@#1#2{%
2143 \ifnum#1=0
2144 0%
2145 \else
2146 \ifnum#2>1
2147 \escapeifif{\expandtwonumberargs
2148 \forest@node@nbarthchildid@{\forestOve{#1}{@previous}}{\numexpr#2-1}}%
2149 \else
2150 #1%
2151 \fi
2152 \fi
2153 }
2154 \def\forest@node@nornbarthchildid#1{%
2155 \ifnum#1>0
2156 \forest@node@nthchildid{#1}%
2157 \else
2158 \ifnum#1<0
2159 \forest@node@nbarthchildid{-#1}%
2160 \else
2161 \forest@node@nornbarthchildid@error
2162 \fi
2163 \fi
2164 }
2165 \def\forest@node@nornbarthchildid@error{%
2166 \PackageError{forest}{In \string\forest@node@nornbarthchildid, n should !=0}{}%
2167 }
2168 \def\forest@node@previousleafid{%
2169 \expandnumberarg\forest@node@Previousleafid{\forest@cn}%
2170 }
2171 \def\forest@node@Previousleafid#1{%
2172 \ifnum\forestOve{#1}{@previous}=0
2173 \escapeif{\expandnumberarg\forest@node@previousleafid@Goup{#1}}%
2174 \else
2175 \expandnumberarg\forest@node@previousleafid@Godown{\forestOve{#1}{@previous}}%
2176 \fi
2177 }

```

```

2178 \def\forest@node@previousleafid@Goup#1{%
2179   \ifnum\forestOve{#1}{@parent}=0
2180     \PackageError{forest}{get previous leaf: this is the first leaf}{}%
2181   \else
2182     \@escapeif{\expandnumberarg\forest@node@Previousleafid{\forestOve{#1}{@parent}}}%
2183   \fi
2184 }
2185 \def\forest@node@previousleafid@Godown#1{%
2186   \ifnum\forestOve{#1}{@last}=0
2187     #1%
2188   \else
2189     \@escapeif{\expandnumberarg\forest@node@previousleafid@Godown{\forestOve{#1}{@last}}}%
2190   \fi
2191 }
2192 \def\forest@node@nextleafid{%
2193   \expandnumberarg\forest@node@Nextleafid{\forest@cn}%
2194 }
2195 \def\forest@node@Nextleafid#1{%
2196   \ifnum\forestOve{#1}{@next}=0
2197     \@escapeif{\expandnumberarg\forest@node@nextleafid@Goup{#1}}%
2198   \else
2199     \expandnumberarg\forest@node@nextleafid@Godown{\forestOve{#1}{@next}}%
2200   \fi
2201 }
2202 \def\forest@node@nextleafid@Goup#1{%
2203   \ifnum\forestOve{#1}{@parent}=0
2204     \PackageError{forest}{get next leaf: this is the last leaf}{}%
2205   \else
2206     \@escapeif{\expandnumberarg\forest@node@Nextleafid{\forestOve{#1}{@parent}}}%
2207   \fi
2208 }
2209 \def\forest@node@nextleafid@Godown#1{%
2210   \ifnum\forestOve{#1}{@first}=0
2211     #1%
2212   \else
2213     \@escapeif{\expandnumberarg\forest@node@nextleafid@Godown{\forestOve{#1}{@first}}}%
2214   \fi
2215 }
2216
2217
2218
2219 \def\forest@node@linearnextid{%
2220   \ifnum\forestove{@first}=0
2221     \expandafter\forest@node@linearnextnotdescendantid
2222   \else
2223     \forestove{@first}%
2224   \fi
2225 }
2226 \def\forest@node@linearnextnotdescendantid{%
2227   \expandnumberarg\forest@node@Linearnextnotdescendantid{\forest@cn}%
2228 }
2229 \def\forest@node@Linearnextnotdescendantid#1{%
2230   \ifnum\forestOve{#1}{@next}=0
2231     \ifnum\forestOve{#1}{@parent}=0
2232       0%
2233     \else
2234       \@escapeifif{\expandnumberarg\forest@node@Linearnextnotdescendantid{\forestOve{#1}{@parent}}}%
2235     \fi
2236   \else
2237     \forestOve{#1}{@next}%
2238   \fi

```

```

2239 }
2240 \def\forest@node@linearpreviousid{%
2241   \ifnum\forest@ve{@previous}=0
2242     \forest@ve{@parent}%
2243   \else
2244     \forest@node@previousleafid
2245   \fi
2246 }

```

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.

```

2247
2248 \def\forest@ifancestorof#1{% is the current node an ancestor of #1? Yes: #2, no: #3
2249   \begingroup
2250   \expandnumberarg\forest@ifancestorof{\forest@ve{#1}{@parent}}%
2251 }
2252 \def\forest@ifancestorof@#1{%
2253   \ifnum#1=0
2254     \def\forest@ifancestorof@next{\expandafter\endgroup\@secondoftwo}%
2255   \else
2256     \ifnum\forest@cn=#1
2257       \def\forest@ifancestorof@next{\expandafter\endgroup\@firstoftwo}%
2258     \else
2259       \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.

```

2260     \def\forest@ifancestorof@next{\expandafter\endgroup\@secondoftwo}%
2261   }{%
2262     \csdef{forest@circularity@used#1}{}%
2263     \def\forest@ifancestorof@next{\expandnumberarg\forest@ifancestorof{\forest@ve{#1}{@parent}}}%
2264   }%
2265   \fi
2266 \fi
2267 \forest@ifancestorof@next
2268 }

```

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

```

2269 \NewDocumentCommand\forestdebugtypeouttrees{o}{%
2270   \forestdebug@typeouttrees\forest@temp
2271   \typeout{%
2272     \forestdebugtypeouttreesprefix
2273     \IfValueTF{#1}{#1: }{}%
2274     \detokenize\expandafter{\forest@temp}%
2275     \forestdebugtypeouttreessuffix
2276   }%
2277 }
2278 \def\forestdebug@typeouttrees#1{% #1 = cs to store the result
2279   \begingroup
2280   \edef\forest@temp@message{}%
2281   \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.

```

2282   \loop
2283   \ifnum\forestdebug@typeouttrees@n<\forest@node@maxid
2284     \edef\forestdebug@typeouttrees@n{\number\numexpr\forestdebug@typeouttrees@n+1}%
2285     \ifcsdef{forestdebug@typeouttree@used@\forestdebug@typeouttrees@n}{}%
2286     \forest@fornode{\forestdebug@typeouttrees@n}{%

```

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

```

2287     \begingroup
2288     \forestdebug@typeouttrees@findroot
2289     \expandafter\endgroup
2290     \expandafter\edef\expandafter\forest@cn\expandafter{\forest@cn}%
2291     \forestdebug@typeouttree@build
2292     \appto\forest@temp@message{ }%
2293     }%
2294   }%
2295 \repeat
2296 \expandafter\endgroup
2297 \expandafter\def\expandafter#1\expandafter{\forest@temp@message}%
2298 }
2299 \def\forestdebug@typeouttrees@findroot{%
2300   \let\forestdebug@typeouttrees@next\relax
2301   \edef\forestdebug@typeouttrees@parent{\forestOve{\forest@cn}{@parent}}%
2302   \ifnum\forestdebug@typeouttrees@parent=0
2303     \else
2304       \ifcsdef{forestdebug@typeouttree@used@\forest@cn}{-}{%
2305         \csdef{forestdebug@typeouttree@used@\forest@cn}{-}%
2306         \edef\forest@cn{\forestdebug@typeouttrees@parent}%
2307         \let\forestdebug@typeouttrees@next\forestdebug@typeouttrees@findroot
2308       }%
2309     \fi
2310     \forestdebug@typeouttrees@next
2311   }
2312 \def\forestdebug@typeouttree#1#2{% #1=root id, #2=cs to receive result
2313   \begingroup
2314   \edef\forest@temp@message{%
2315     \forest@fornode{#1}{\forestdebug@typeouttree@build}%
2316     \expandafter\endgroup
2317     \expandafter\edef\expandafter#2\expandafter{\forest@temp@message}%
2318   }
2319   \NewDocumentCommand\forestdebugtypeouttree{d() O{\forest@cn}}{%
2320     \forestdebug@typeouttree{#2}\forest@temp
2321     \typeout{\IfValueTF{#1}{#1: }{\}\forest@temp}%
2322   }

```

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

```

2323 \def\forestdebug@typeouttree@build{%
2324   \eappto\forest@temp@message{\forestdebugtypeouttreenodeinfo%]
2325   \ifcsdef{forestdebug@typeouttree@used@\forest@cn}{*}{%
2326     }%
2327   \ifcsdef{forestdebug@typeouttree@used@\forest@cn}{-}{%
2328     \csdef{forestdebug@typeouttree@used@\forest@cn}{-}%
2329     \forest@node@foreachchild{\forestdebug@typeouttree@build}%
2330   }%
2331   \eappto\forest@temp@message{[%
2332     ]}%
2333 }
2334 \def\forestdebugtypeouttreenodeinfo{\forest@cn}
2335 \def\forestdebugtypeouttreesprefix{}
2336 \def\forestdebugtypeouttreessuffix{}

```

6.3 Node options

6.3.1 Option-declaration mechanism

Common code for declaring options.

```

2337 \def\forest@declarehandler#1#2#3{%#1=handler for specific type,#2=option name,#3=default value
2338   \pgfkeyssetvalue{/forest/#2}{#3}%
2339   \appto\forest@node@init{\forest@init{#2}}%

```

```

2340 \pgfkeyssetvalue{/forest/#2/node@or@reg}{\forest@cn}%
2341 \forest@convert@others@to@underscores{#2}\forest@pgfmathoptionname
2342 \edef\forest@marshal{%
2343   \noexpand#1{/forest/#2}{/forest}{#2}{\forest@pgfmathoptionname}%
2344 } \forest@marshal
2345 }
2346 \def\forest@def@with@pgfeov#1#2{% \pgfeov mustn't occur in the arg of the .code handler!!!
2347   \long\def#1##1\pgfeov{#2}%
2348 }

```

Option-declaration handlers.

```

2349 \def\forest@declaretoks@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2350   \forest@declaretoks@handler@A{#1}{#2}{#3}{#4}{}%
2351 }
2352 \def\forest@declarekeylist@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2353   \forest@declaretoks@handler@A{#1}{#2}{#3}{#4}{,}%
2354   \forest@copycommandkey{#1}{#1'}%
2355   \pgfkeyssetvalue{#1'/option@name}{#3}%
2356   \forest@copycommandkey{#1+}{#1}%
2357   \pgfkeysalso{#1-/.code={%
2358     \forest@fornode{\forest@setter@node}{%
2359       \forest@node@removekeysfromkeylist{##1}{#3}%
2360     }}%
2361   \pgfkeyssetvalue{#1-/option@name}{#3}%
2362 }
2363 \def\forest@declaretoks@handler@A#1#2#3#4#5{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
2364   \pgfkeysalso{%
2365     #1/.code={\forest0set{\forest@setter@node}{#3}{##1}},
2366     #2/if #3/.code n args={3}{%
2367       \forest0get{#3}\forest@temp@option@value
2368       \edef\forest@temp@compared@value{\unexpanded{##1}}%
2369       \ifx\forest@temp@option@value\forest@temp@compared@value
2370         \pgfkeysalso{##2}%
2371       \else
2372         \pgfkeysalso{##3}%
2373       \fi
2374     },
2375     #2/if in #3/.code n args={3}{%
2376       \forest0get{#3}\forest@temp@option@value
2377       \edef\forest@temp@compared@value{\unexpanded{##1}}%
2378       \expandafter\expandafter\expandafter\pgfutil@in@\expandafter\expandafter\expandafter{\forest@temp@option@value}
2379       \ifpgfutil@in@
2380         \pgfkeysalso{##2}%
2381       \else
2382         \pgfkeysalso{##3}%
2383       \fi
2384     },
2385     #2/where #3/.style n args={3}{for tree={#2/if #3={##1}{##2}{##3}},
2386     #2/where in #3/.style n args={3}{for tree={#2/if in #3={##1}{##2}{##3}}}
2387   }%
2388   \ifstrempy{#5}{%
2389     \pgfkeysalso{%
2390       #1+/.code={\forest0appto{\forest@setter@node}{#3}{#5##1}},
2391       #2/+#3/.code={\forest0preto{\forest@setter@node}{#3}{##1#5}},
2392     }%
2393   }{%
2394     \pgfkeysalso{%
2395       #1+/.code={%
2396         \forest0get{\forest@setter@node}{#3}\forest@temp
2397         \ifdefempty{\forest@temp}{%
2398           \forest0set{\forest@setter@node}{#3}{##1}%

```

```

2399     }{%
2400     \forestOappto{\forest@setter@node}{#3}{#5##1}%
2401     }%
2402   },
2403   #2/+#3/.code={%
2404     \forestOget{\forest@setter@node}{#3}\forest@temp
2405     \ifdefempty{\forest@temp}{%
2406       \forestOset{\forest@setter@node}{#3}{##1}%
2407     }{%
2408       \forestOpreto{\forest@setter@node}{#3}{##1#5}%
2409     }%
2410   }%
2411 }%
2412 }%
2413 \pgfkeyssetvalue{#1/option@name}{#3}%
2414 \pgfkeyssetvalue{#1+/option@name}{#3}%
2415 \pgfkeyssetvalue{#2/+#3/option@name}{#3}%
2416 \pgfkeyslet{#1/@type}\forest@math@type@generic % for .process & co
2417 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@toks{##1}{#3}}%
2418 }
2419 \def\forest@declareautowrappedtoks@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
2420 \forest@declaretoks@handler{#1}{#2}{#3}{#4}%
2421 \forest@copycommandkey{#1}{#1'}%
2422 \pgfkeysalso{#1/.style={#1'/.wrap value={##1}}}%
2423 \pgfkeyssetvalue{#1'/option@name}{#3}%
2424 \forest@copycommandkey{#1+}{#1+'}%
2425 \pgfkeysalso{#1+/.style={#1+'/.wrap value={##1}}}%
2426 \pgfkeyssetvalue{#1+'/option@name}{#3}%
2427 \forest@copycommandkey{#2/+#3}{#2/+#3'}%
2428 \pgfkeysalso{#2/+#3/.style={#2/+#3'/.wrap value={##1}}}%
2429 \pgfkeyssetvalue{#2/+#3'/option@name}{#3}%
2430 }
2431 \def\forest@declarereadonlydimen@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2432 % this is to have 'pt' with the correct category code
2433 \pgfutil@tempdima=\pgfkeysvalueof{/forest/#3}\relax
2434 \edef\forest@marshal{%
2435   \noexpand\pgfkeyssetvalue{/forest/#3}{\the\pgfutil@tempdima}%
2436 }%
2437 \forest@marshal
2438 \pgfkeysalso{%
2439   #2/if #3/.code n args={3}{%
2440     \forestOget{#3}\forest@temp@option@value
2441     \ifdim\forest@temp@option@value>##1\relax
2442       \pgfkeysalso{##2}%
2443     \else
2444       \pgfkeysalso{##3}%
2445     \fi
2446   },
2447   #2/if #3</.code n args={3}{%
2448     \forestOget{#3}\forest@temp@option@value
2449     \ifdim\forest@temp@option@value>##1\relax
2450       \pgfkeysalso{##3}%
2451     \else
2452       \pgfkeysalso{##2}%
2453     \fi
2454   },
2455   #2/if #3>/.code n args={3}{%
2456     \forestOget{#3}\forest@temp@option@value
2457     \ifdim\forest@temp@option@value<##1\relax
2458       \pgfkeysalso{##3}%
2459     \else
2460       \pgfkeysalso{##2}%

```



```

2460     \fi
2461   },
2462   #2/where #3/.style n args={3}{for tree={#2/if #3={##1}{##2}{##3}}},
2463   #2/where #3</.style n args={3}{for tree={#2/if #3<={##1}{##2}{##3}}},
2464   #2/where #3>/.style n args={3}{for tree={#2/if #3>={##1}{##2}{##3}}},
2465 }%
2466 \pgfkeyslet{#1/@type}\forestmathtype@dimen % for .process & co
2467 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@dimen{##1}{#3}}%
2468 }
2469 \def\forest@declaredimen@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2470 \forest@declarereadonlydimen@handler{#1}{#2}{#3}{#4}%
2471 \pgfkeysalso{%
2472   #1/.code={%
2473     \forestmathsetlengthmacro\forest@temp{##1}%
2474     \forestOlet{\forest@setter@node}{#3}\forest@temp
2475   },
2476   #1+/.code={%
2477     \forestmathsetlengthmacro\forest@temp{##1}%
2478     \pgfutil@tempdima=\forestove{#3}
2479     \advance\pgfutil@tempdima\forest@temp\relax
2480     \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2481   },
2482   #1-/.code={%
2483     \forestmathsetlengthmacro\forest@temp{##1}%
2484     \pgfutil@tempdima=\forestove{#3}
2485     \advance\pgfutil@tempdima-\forest@temp\relax
2486     \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2487   },
2488   #1*/.style={%
2489     #1={#4()*{##1}}%
2490   },
2491   #1:/ .style={%
2492     #1={#4()/ {##1}}%
2493   },
2494   #1'/.code={%
2495     \pgfutil@tempdima=##1\relax
2496     \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2497   },
2498   #1'+/.code={%
2499     \pgfutil@tempdima=\forestove{#3}\relax
2500     \advance\pgfutil@tempdima##1\relax
2501     \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2502   },
2503   #1'-/.code={%
2504     \pgfutil@tempdima=\forestove{#3}\relax
2505     \advance\pgfutil@tempdima-##1\relax
2506     \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2507   },
2508   #1'*/.style={%
2509     \pgfutil@tempdima=\forestove{#3}\relax
2510     \multiply\pgfutil@tempdima##1\relax
2511     \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2512   },
2513   #1':/.style={%
2514     \pgfutil@tempdima=\forestove{#3}\relax
2515     \divide\pgfutil@tempdima##1\relax
2516     \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2517   },
2518 }%
2519 \pgfkeyssetvalue{#1/option@name}{#3}%
2520 \pgfkeyssetvalue{#1+/option@name}{#3}%

```

```

2521 \pgfkeyssetvalue{#1-/option@name}{#3}%
2522 \pgfkeyssetvalue{#1*/option@name}{#3}%
2523 \pgfkeyssetvalue{#1:/option@name}{#3}%
2524 \pgfkeyssetvalue{#1'/option@name}{#3}%
2525 \pgfkeyssetvalue{#1'+/option@name}{#3}%
2526 \pgfkeyssetvalue{#1'-/option@name}{#3}%
2527 \pgfkeyssetvalue{#1'*option@name}{#3}%
2528 \pgfkeyssetvalue{#1':/option@name}{#3}%
2529 }
2530 \def\forest@declarereadonlycount@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2531 \pgfkeysalso{
2532 #2/if #3/.code n args={3}{%
2533 \forestoget{#3}\forest@temp@option@value
2534 \ifnum\forest@temp@option@value=#1\relax
2535 \pgfkeysalso{##2}%
2536 \else
2537 \pgfkeysalso{##3}%
2538 \fi
2539 },
2540 #2/if #3</.code n args={3}{%
2541 \forestoget{#3}\forest@temp@option@value
2542 \ifnum\forest@temp@option@value>#1\relax
2543 \pgfkeysalso{##3}%
2544 \else
2545 \pgfkeysalso{##2}%
2546 \fi
2547 },
2548 #2/if #3>/.code n args={3}{%
2549 \forestoget{#3}\forest@temp@option@value
2550 \ifnum\forest@temp@option@value<#1\relax
2551 \pgfkeysalso{##3}%
2552 \else
2553 \pgfkeysalso{##2}%
2554 \fi
2555 },
2556 #2/where #3/.style n args={3}{for tree={#2/if #3={##1}{##2}{##3}}},
2557 #2/where #3</.style n args={3}{for tree={#2/if #3<={##1}{##2}{##3}}},
2558 #2/where #3>/.style n args={3}{for tree={#2/if #3>={##1}{##2}{##3}}},
2559 }%
2560 \pgfkeyslet{#1/@type}\forest@math@type@count % for .process & co
2561 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@count{##1}{#3}}%
2562 }
2563 \def\forest@declarecount@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2564 \forest@declarereadonlycount@handler{#1}{#2}{#3}{#4}%
2565 \pgfkeysalso{
2566 #1/.code={%
2567 \forestmathtruncatemacro\forest@temp{##1}%
2568 \forest0let{\forest@setter@node}{#3}\forest@temp
2569 },
2570 #1+/.code={%
2571 \forestmathtruncatemacro\forest@temp{##1}%
2572 \c@pgf@counta=\forestove{#3}\relax
2573 \advance\c@pgf@counta\forest@temp\relax
2574 \forest0set{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2575 },
2576 #1-/.code={%
2577 \forestmathtruncatemacro\forest@temp{##1}%
2578 \c@pgf@counta=\forestove{#3}\relax
2579 \advance\c@pgf@counta-\forest@temp\relax
2580 \forest0set{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2581 },

```

```

2582 #1*/.code={%
2583   \forestmathtruncatemacro\forest@temp{##1}%
2584   \c@pgf@counta=\forestove{#3}\relax
2585   \multiply\c@pgf@counta\forest@temp\relax
2586   \forestOeset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2587 },
2588 #1:/.code={%
2589   \forestmathtruncatemacro\forest@temp{##1}%
2590   \c@pgf@counta=\forestove{#3}\relax
2591   \divide\c@pgf@counta\forest@temp\relax
2592   \forestOeset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2593 },
2594 #1'/.code={%
2595   \c@pgf@counta=##1\relax
2596   \forestOeset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2597 },
2598 #1'+/.code={%
2599   \c@pgf@counta=\forestove{#3}\relax
2600   \advance\c@pgf@counta##1\relax
2601   \forestOeset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2602 },
2603 #1'-/.code={%
2604   \c@pgf@counta=\forestove{#3}\relax
2605   \advance\c@pgf@counta-##1\relax
2606   \forestOeset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2607 },
2608 #1'*/.style={%
2609   \c@pgf@counta=\forestove{#3}\relax
2610   \multiply\c@pgf@counta##1\relax
2611   \forestOeset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2612 },
2613 #1':/.style={%
2614   \c@pgf@counta=\forestove{#3}\relax
2615   \divide\c@pgf@counta##1\relax
2616   \forestOeset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2617 },
2618 }%
2619 \pgfkeyssetvalue{#1/option@name}{#3}%
2620 \pgfkeyssetvalue{#1+/option@name}{#3}%
2621 \pgfkeyssetvalue{#1-/option@name}{#3}%
2622 \pgfkeyssetvalue{#1*/option@name}{#3}%
2623 \pgfkeyssetvalue{#1:/option@name}{#3}%
2624 \pgfkeyssetvalue{#1'/option@name}{#3}%
2625 \pgfkeyssetvalue{#1'+/option@name}{#3}%
2626 \pgfkeyssetvalue{#1'-/option@name}{#3}%
2627 \pgfkeyssetvalue{#1'*/option@name}{#3}%
2628 \pgfkeyssetvalue{#1':/option@name}{#3}%
2629 }

```

Nothing else should be defined in this namespace.

```

2630 \def\forest@declareboolean@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2631   \pgfkeysalso{%
2632     #1/.code={%
2633       \forestmath@if{##1}{%
2634         \def\forest@temp{1}%
2635       }{%
2636         \def\forest@temp{0}%
2637       }%
2638       \forestOlet{\forest@setter@node}{#3}\forest@temp
2639     },
2640     #1/.default=1,

```

```

2641 #2/not #3/.code={\forest0set{\forest@setter@node}{#3}{0}},
2642 #2/if #3/.code 2 args={%
2643   \forestoget{#3}\forest@temp@option@value
2644   \ifnum\forest@temp@option@value=0
2645     \pgfkeysalso{##2}%
2646   \else
2647     \pgfkeysalso{##1}%
2648   \fi
2649 },
2650 #2/where #3/.style 2 args={for tree={#2/if #3={##1}{##2}}}
2651 }%
2652 \pgfkeyssetvalue{#1/option@name}{#3}%
2653 \pgfkeyslet{#1/@type}\forest@mathtype@count % for .process & co
2654 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@count{##1}{#3}}%
2655 }
2656 \forestset{
2657   declare toks/.code 2 args={%
2658     \forest@declarehandler\forest@declaretoks@handler{#1}{#2}%
2659   },
2660   declare autowrapped toks/.code 2 args={%
2661     \forest@declarehandler\forest@declareautowrappedtoks@handler{#1}{#2}%
2662   },
2663   declare keylist/.code 2 args={%
2664     \forest@declarehandler\forest@declarekeylist@handler{#1}{#2}%
2665   },
2666   declare readonly dimen/.code 2 args={%
2667     \forestmathsetlengthmacro\forest@temp{#2}%
2668     \edef\forest@marshal{%
2669       \unexpanded{\forest@declarehandler\forest@declarereadonlydimen@handler{#1}}{\forest@temp}%
2670     }\forest@marshal
2671   },
2672   declare dimen/.code 2 args={%
2673     \forestmathsetlengthmacro\forest@temp{#2}%
2674     \edef\forest@marshal{%
2675       \unexpanded{\forest@declarehandler\forest@declaredimen@handler{#1}}{\forest@temp}%
2676     }\forest@marshal
2677   },
2678   declare readonly count/.code 2 args={%
2679     \forestmathtruncatemacro\forest@temp{#2}%
2680     \edef\forest@marshal{%
2681       \unexpanded{\forest@declarehandler\forest@declarereadonlycount@handler{#1}}{\forest@temp}%
2682     }\forest@marshal
2683   },
2684   declare count/.code 2 args={%
2685     \forestmathtruncatemacro\forest@temp{#2}%
2686     \edef\forest@marshal{%
2687       \unexpanded{\forest@declarehandler\forest@declarecount@handler{#1}}{\forest@temp}%
2688     }\forest@marshal
2689   },
2690   declare boolean/.code 2 args={%
2691     \forestmath@if{#2}{%
2692       \def\forest@temp{1}%
2693     }{%
2694       \def\forest@temp{0}%
2695     }%
2696     \edef\forest@marshal{%
2697       \unexpanded{\forest@declarehandler\forest@declareboolean@handler{#1}}{\forest@temp}%
2698     }\forest@marshal
2699   },

```



```

2759   \forest@wrap@n@pgfmath@do{#1}{6}},
2760 /handlers/.wrap 7 pgfmath args/.code n args={8}{-%
2761   \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#7}%
2762   \forest@wrap@n@pgfmath@do{#1}{7}},
2763 /handlers/.wrap 8 pgfmath args/.code n args={9}{-%
2764   \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}{8}%
2765   \forest@wrap@n@pgfmath@do{#1}{8}},
2766 }
2767 \def\forest@wrap@n@pgfmath@args#1#2#3#4#5#6#7#8#9{-%
2768   \forestmathparse{#1}\let\forest@wrap@arg@i\forestmathresult
2769   \ifnum#9>1 \forestmathparse{#2}\let\forest@wrap@arg@ii\forestmathresult\fi
2770   \ifnum#9>2 \forestmathparse{#3}\let\forest@wrap@arg@iii\forestmathresult\fi
2771   \ifnum#9>3 \forestmathparse{#4}\let\forest@wrap@arg@iv\forestmathresult\fi
2772   \ifnum#9>4 \forestmathparse{#5}\let\forest@wrap@arg@v\forestmathresult\fi
2773   \ifnum#9>5 \forestmathparse{#6}\let\forest@wrap@arg@vi\forestmathresult\fi
2774   \ifnum#9>6 \forestmathparse{#7}\let\forest@wrap@arg@vii\forestmathresult\fi
2775   \ifnum#9>7 \forestmathparse{#8}\let\forest@wrap@arg@viii\forestmathresult\fi
2776   \edef\forest@wrap@args{%-
2777     {\expandonce\forest@wrap@arg@i}
2778     \ifnum#9>1 {\expandonce\forest@wrap@arg@ii}\fi
2779     \ifnum#9>2 {\expandonce\forest@wrap@arg@iii}\fi
2780     \ifnum#9>3 {\expandonce\forest@wrap@arg@iv}\fi
2781     \ifnum#9>4 {\expandonce\forest@wrap@arg@v}\fi
2782     \ifnum#9>5 {\expandonce\forest@wrap@arg@vi}\fi
2783     \ifnum#9>6 {\expandonce\forest@wrap@arg@vii}\fi
2784     \ifnum#9>7 {\expandonce\forest@wrap@arg@viii}\fi
2785   }%-
2786 }
2787 \def\forest@wrap@n@pgfmath@do#1#2{-%
2788   \ifcase#2\relax
2789   \or\def\forest@wrap@code##1{#1}%
2790   \or\def\forest@wrap@code##1##2{#1}%
2791   \or\def\forest@wrap@code##1##2##3{#1}%
2792   \or\def\forest@wrap@code##1##2##3##4{#1}%
2793   \or\def\forest@wrap@code##1##2##3##4##5{#1}%
2794   \or\def\forest@wrap@code##1##2##3##4##5##6{#1}%
2795   \or\def\forest@wrap@code##1##2##3##4##5##6##7{#1}%
2796   \or\def\forest@wrap@code##1##2##3##4##5##6##7##8{#1}%
2797   \fi
2798   \forest@wrap@pgfmath@args@@@wrapandpasson
2799 }

```

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

```

2800 \def\forest@wrap@pgfmath@args@@@wrapandpasson{%-
2801   \expandafter\expandafter\expandafter\forest@temp@toks
2802   \expandafter\expandafter\expandafter{%-
2803     \expandafter\forest@wrap@code\forest@wrap@args}%
2804   \expandafter\pgfkeysalso\expandafter{%-
2805     \expandafter\pgfkeyscurrentpath\expandafter=\expandafter{%-
2806       \the\forest@temp@toks}}%-
2807 }

```

7.1 .process

```

2808 \def\forest@process@catregime{} % filled by processor defs
2809 \forest@newarray\forest@process@left@ % processed args
2810 \forest@newarray\forest@process@right@ % unprocessed args
2811 \forest@newarray\forest@process@saved@ % used by instructions |S| and |U|
2812 \let\forest@process@savedtype\forestmathtype@none
2813 \forest@newglobalarray\forest@process@result@
2814 \newif\ifforest@process@returnarray@

```

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

```

2815 \def\forest@process#1#2#{% #1 = true/false (should we return an array?)
2816 % #2 = processing instructions (if non-empty),
2817 % (initial) args follow
2818 \ifblank{#2}{\forest@process@a{#1}}{\forest@process@a{#1}{#2}}%
2819 }
2820 \Inline\def\forest@process@a#1#2{%
2821 \begingroup
2822 \forest@process@left@clear
2823 \forest@process@right@clear
2824 \forest@process@savetoclear
2825 \let\forest@process@savetype\forestmathtype@generic
2826 \csname forest@process@returnarray@#1\endcsname
2827 \def\forest@topextend@next{%
2828 \ExpandIfT{forestdebug}{%
2829 \edef\forest@process@debug@args{\unexpanded{#2}}%
2830 \forest@processor@debuginfo@template{Start "\unexpanded{#2}}%
2831 }%
2832 \forest@process@catregime
2833 \endlinechar=-1
2834 \scantokens{#2}%
2835 \forest@process@finish
2836 }%
2837 \forest@process@right@topextend
2838 }
2839 \pgfkeys{%
2840 /handlers/.process/.code={%
2841 \forest@process{true}#1\forest@eov
2842 \edef\forest@marshal{%
2843 \noexpand\pgfkeysalso{\noexpand\pgfkeyscurrentpath=\forest@process@result@values}%
2844 }\forest@marshal
2845 },
2846 /forest/copy command key={/handlers/.process}{/handlers/.process args},
2847 }
2848 \def\forest@process@finish{%
2849 \ifforest@process@returnarray@
2850 \forest@process@finish@array
2851 \else
2852 \forest@process@finish@single
2853 \fi
2854 \global\let\forest@process@result@type\forestmathresulttype
2855 \ifforestdebugprocess\forest@process@debug@end\fi
2856 \endgroup
2857 }
2858 \def\forest@process@finish@single{%
2859 \edef\forest@temp{forest@process@finish@single@%
2860 \the\numexpr\forest@process@left@N-\forest@process@left@M\relax
2861 \the\numexpr\forest@process@right@N-\forest@process@right@M\relax
2862 }%
2863 \ifcsname\forest@temp\endcsname
2864 \csname\forest@temp\endcsname
2865 \global\let\forest@process@result\forest@temp
2866 \else
2867 \forest@process@lengtherror
2868 \fi
2869 }
2870 \csdef{forest@process@finish@single@10}{\forest@process@left@toppop\forest@temp}
2871 \csdef{forest@process@finish@single@01}{\forest@process@right@toppop\forest@temp}
2872 \def\forest@process@finish@array{%
2873 \forest@process@result@clear
2874 \forest@temp@count\forest@process@left@M\relax
2875 \forest@loop

```

```

2876 \ifnum\forest@temp@count<\forest@process@left@N\relax
2877   \forest@process@left@get@{\the\forest@temp@count}\forest@temp
2878   \forest@process@result@letappend\forest@temp
2879   \advance\forest@temp@count1
2880 \forest@repeat
2881 \forest@temp@count\forest@process@right@M\relax
2882 \forest@loop
2883 \ifnum\forest@temp@count<\forest@process@right@N\relax
2884   \forest@process@right@get@{\the\forest@temp@count}\forest@temp
2885   \forest@process@result@letappend\forest@temp
2886   \advance\forest@temp@count1
2887 \forest@repeat
2888 }

```

Debugging and error messages.

```

2889 \ifforestdebug
2890 \let\forest@process@d\forest@process@b
2891 \def\forest@process@b#1\forest@eov{% save and print initial arguments
2892   \edef\forest@process@debug@args{\unexpanded{#1}}%
2893   \typeout{[forest .process] Start "\unexpanded{#1}"}%
2894   \forest@process@d#1\forest@eov
2895 }
2896 \fi
2897 \def\forest@process@debug@end{%
2898   \typeout{[forest .process] End "\expandonce{\forest@process@debug@args}" -> "\forest@process@left@values\fo
2899 }
2900 \def\forest@process@lengtherror{%
2901   \PackageError{forest}{%
2902     The ".process" expression was expected to evaluate to a single argument,
2903     but the result is \the\forest@process@result@N
2904     \space items long.}{}%
2905 }

```

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

```

2906 \def\forest@def@processor#1{%
2907   {%
2908     \def\forest@dp@double##1{%
2909       \gdef\forest@global@temp{\forest@def@processor@{#1}{##1}}%
2910     }%
2911     \let\\\forest@dp@double
2912     \catcode'#1=13
2913     \scantokens{\#1}%
2914   }%
2915   \forest@global@temp
2916 }
2917 \def\forest@def@processor@#1#2{%
2918   % #1 = instruction char (normal catcode), #2 = instruction char (active)
2919   % #3 = default n (optional numeric arg, which precedes any other args;
2920   %           if the default is empty, this means no optional n)
2921   % #4 = args spec,
2922   % #5 = code
2923   \eappto\forest@process@catregime{%
2924     \unexpanded{\let#2}\expandonce{\cename forest@processor@#1\endcsname}%
2925     \unexpanded{\catcode'#1=13 }%
2926   }%
2927   \def\forest@def@processor@inschar#1}%
2928   \forest@def@processor@@
2929 }

```

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

```

2930 \def\forest@def@processor@@#1{%
2931   \ifstrempy{#1}{%
2932     \forest@def@processor@@non

```



```

2933 }{%
2934   \def\forest@def@processor@@default@n{#1}%
2935   \forest@def@processor@@n
2936 }%
2937 }

```

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:

```

2938 \def\forest@def@processor@@non#1#2{% #1=args spec, #2=code
2939   \csedef{forest@processor@\forest@def@processor@inschar}#1{%
2940     \relax %% we need this (see above)
2941     \unexpanded{#2}%
2942     \expandafter\forest@def@processor@debuginfo\expandafter{%
2943       \expandafter"\forest@def@processor@inschar"\ifstrempy{#1}{(##1)}}%
2944     \ignorespaces
2945   }%
2946 }

```

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

```

2947 \def\forest@def@processor@@n{%
2948   \@ifnextchar*%
2949     {\forest@temptrue\forest@def@processor@@n@}%
2950     {\forest@tempfalse\forest@def@processor@@n@@}%
2951 }
2952 \def\forest@def@processor@@n@*{\forest@def@processor@@n@@}
2953 \def\forest@def@processor@@n@@#1#2{% #1=args spec, #2=code
2954   \csedef{forest@processor@\forest@def@processor@inschar}{%
2955     \relax %% we need this (see above)
2956     \noexpand\forest@process@get@n
2957     {\forest@def@processor@@default@n}%
2958     \expandonce{\csname forest@processor@\forest@def@processor@inschar @\endcsname}%
2959   }%
2960   \ifforest@temp
2961     \csedef{forest@processor@\forest@def@processor@inschar @}{%
2962       \noexpand\forest@repeat@n@times{\forest@process@n}{%
2963         \expandonce{\csname forest@processor@\forest@def@processor@inschar @rep\endcsname}%
2964       }%
2965     }%
2966   \fi
2967   \edef\forest@temp{%
2968     \forest@def@processor@inschar
2969     \ifforest@temp\else\noexpand\the\forest@process@n\fi
2970   }%
2971   \csedef{forest@processor@\forest@def@processor@inschar @\ifforest@temp rep\fi}#1{%
2972     \unexpanded{#2}%
2973     \expandafter\forest@def@processor@debuginfo\expandafter{%
2974       \forest@temp
2975       \ifstrempy{#1}{(##1)}}%
2976   }%
2977 }
2978 \def\forest@def@processor@debuginfo#1{% #1 = instruction call
2979   \ifforest@debug
2980     \expandonce{\forest@processor@debuginfo@template{\space\space After #1}}%
2981   \fi
2982 }
2983 \def\forest@processor@debuginfo@template#1{%
2984   \ifforest@debugprocess
2985     \edef\forest@temp@left{\forest@process@left@values}%
2986     \edef\forest@temp@right{\forest@process@right@values}%
2987     \edef\forest@temp@saved{\forest@process@saved@values}%
2988     \typeout{[forest .process] #1: left="\expandonce{\forest@temp@left}", right="\expandonce{\forest@temp@right}"}

```

```

2989 \fi
2990 }

A helper macro which puts the optional numeric argument into count \forest@process@n (default being
#1) and then executes control sequence #2.
2991 \newcount\forest@process@n
2992 \def\forestprocess@get@n#1#2{%
2993 \def\forestprocess@default@n{#1}%
2994 \let\forestprocess@after@get@n#2%
2995 \afterassignment\forestprocess@get@n@\forest@process@n=0%
2996 }
2997 \def\forestprocess@get@n@{%
2998 \ifnum\forest@process@n=0
2999 \forest@process@n\forestprocess@default@n\relax
3000 \fi
3001 \forestprocess@after@get@n@
3002 }

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. Pro-
cessors 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.
3003 \forest@def@processor_{_}{1}*{}-% no processing, no type
3004 \forest@process@right@bottompop\forest@temp
3005 \forest@process@left@letappend\forest@temp
3006 }
3007 \forest@def@processor{n}{1}*{}-% numexpr
3008 \forest@process@right@bottompop\forest@temp
3009 \forest@process@left@esetappend{\number\numexpr\forest@temp}%
3010 \let\forestmathresulttype\forestmathtype@count
3011 }
3012 \forest@def@processor{d}{1}*{}-% dimexpr
3013 \forest@process@right@bottompop\forest@temp
3014 \forest@process@left@esetappend{\the\dimexpr\forest@temp}%
3015 \let\forestmathresulttype\forestmathtype@dimen
3016 }
3017 \forest@def@processor{P}{1}*{}-% pgfmath expression
3018 \forest@process@right@bottompop\forest@temp
3019 \pgfmathparse{\forest@temp}%
3020 \forest@process@left@letappend\pgfmathresult
3021 \let\forestmathresulttype\forestmathtype@unitless
3022 }
3023 \forest@def@processor{p}{1}*{}-% process expression
3024 \forest@process@right@bottompop\forest@temp@a
3025 \def\forest@temp{\forest@process{true}}%
3026 \expandafter\forest@temp\forest@temp@a\forest@eov
3027 \let\forest@topextend@next\relax
3028 \edef\forest@temp{\forest@process@result@values}%
3029 \expandafter\forest@process@left@topextend\forest@temp\forest@eov
3030 \let\forestmathresulttype\forest@process@result@type
3031 }
3032 \forest@def@processor{t}{1}*{}-% text
3033 \forest@process@right@bottompop\forest@temp
3034 \forest@process@left@letappend\forest@temp
3035 \let\forestmathresulttype\forestmathtype@textasc
3036 }
3037 \forest@def@processor{-}{1}*{}-% toggle ascending/descending
3038 \forest@process@left@toppop\forestmathresult
3039 \csname forest@processor@-\forestmathresulttype\endcsname
3040 \forest@process@left@letappend\forestmathresult
3041 }
3042 \cslet{forest@processor@-\forestmathtype@generic}\relax

```

```

3043 \csdef{forest@processor@-\forestmathtype@count}{%
3044   \forestmathadd{\forestmathzero}{-\forestmathresult}}
3045 \csletcs{forest@processor@-\forestmathtype@dimen}
3046   {forest@processor@-\forestmathtype@count}
3047 \csletcs{forest@processor@-\forestmathtype@unitless}
3048   {forest@processor@-\forestmathtype@count}
3049 \csdef{forest@processor@-\forestmathtype@textasc}{%
3050   \let\forestmathresulttype\forestmathtype@textdesc}
3051 \csdef{forest@processor@-\forestmathtype@textdesc}{%
3052   \let\forestmathresulttype\forestmathtype@textasc}
3053
3054 \forest@def@processor{c}{-}{% to lowercase
3055   \forest@process@right@bottompop\forest@temp
3056   \expandafter\lowercase\expandafter{\expandafter\def\expandafter\forest@temp\expandafter{\forest@temp}}%
3057   \forest@process@left@letappend\forest@temp
3058 }
3059 \forest@def@processor{C}{-}{% to uppercase
3060   \forest@process@right@bottompop\forest@temp
3061   \expandafter\uppercase\expandafter{\expandafter\def\expandafter\forest@temp\expandafter{\forest@temp}}%
3062   \forest@process@left@letappend\forest@temp
3063 }

```

Expansions:

```

3064 \forest@def@processor{x}{-}{% expand
3065   \forest@process@right@bottompop\forest@temp
3066   \forest@process@left@setappend{\forest@temp}%
3067   \let\forestmathresulttype\forestmathtype@generic
3068 }
3069 \forest@def@processor{o}{1}{% expand once (actually, \forest@count@n times)
3070   \forest@process@right@bottompop\forest@temp
3071   \forest@repeat@n@times{\forest@process@n}{%
3072     \expandafter\expandafter\expandafter\def
3073     \expandafter\expandafter\expandafter\forest@temp
3074     \expandafter\expandafter\expandafter{\forest@temp}}%
3075   }%
3076   \expandafter\forest@process@left@setappend\expandafter{\forest@temp}%
3077   \let\forestmathresulttype\forestmathtype@generic
3078 }

```

Access to FOREST data.

```

3079 \forest@def@processor{0}{1}{*}{% option
3080   \forest@process@right@bottompop\forest@temp
3081   \expandafter\forestRNO@getvalueandtype\expandafter{\forest@temp}\forest@tempvalue\forest@temp@type
3082   \let\forestmathresulttype\forest@temp@type
3083   \forest@process@left@letappend\forest@tempvalue
3084 }
3085 \forest@def@processor{R}{1}{*}{% register
3086   \forest@process@right@bottompop\forest@temp
3087   \forestRget{\forest@temp}\forest@tempvalue
3088   \forest@process@left@letappend\forest@tempvalue
3089   \pgfkeysgetvalue{/forest/\forest@temp/@type}\forest@temp@type
3090   \let\forestmathresulttype\forest@temp@type
3091 }

```

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

```

3092 \forest@def@processor{+}{1}{*}{% join processors = pop one from result
3093   \forest@process@left@toppop\forest@temp
3094   \forest@process@right@letprepend\forest@temp
3095 }
3096 \forest@def@processor{u}{-}{% ungroup: remove braces and leave in the argument list
3097   \forest@process@right@bottompop\forest@temp
3098   \forest@temparray@clear
3099   \let\forestmathresulttype\forestmathtype@generic

```

```

3100 \let\forest@topextend@next\forest@processor@u@
3101 \expandafter\forest@temparray@topextend\forest@temp\forest@eov
3102 }
3103 \def\forest@processor@u@{%
3104 \forest@loop
3105 \ifnum\forest@temparray@N>0
3106 \forest@temparray@topop\forest@temp
3107 \expandafter\forest@process@right@setprepend\expandafter{\forest@temp}%
3108 \forest@repeat
3109 }
3110 \def\forest@process@check@mn#1#2#3#4{%
3111 % #1 = processor, #2 = given n, #3/#4 = lower/upper bound (inclusive)
3112 \ifnum#3>#2\relax
3113 \forest@process@check@n@error{#1}{#2}{#3<=}{<=#4}%
3114 \else
3115 \ifnum#4<#2\relax
3116 \forest@process@check@n@error{#1}{#2}{#3<=}{<=#4}%
3117 \fi
3118 \fi
3119 }
3120 \def\forest@process@check@m#1#2#3{%
3121 % #1 = processor, #2 = given n, #3 = lower bound (inclusive)
3122 \ifnum#2<#3\relax
3123 \forest@process@check@n@error{#1}{#2}{#3<=}{}%
3124 \fi
3125 }
3126 \def\forest@process@check@n@error#1#2#3#4{%
3127 \PackageError{forest}{'.process' instruction '#1' requires a numeric modifier #3n#4, but n="#2" was given.}
3128 }
3129 \newif\ifforest@process@W
3130 \forest@def@processor@w@{1}{1}{% consuming wrap: first test 1<=#1<=#2
3131 \forest@process@Wtrue
3132 \forest@process@check@mn@w@{0}\the\forest@process@n@{1}{9}%
3133 \expandafter\forest@processor@wW@\expandafter{\the\forest@process@n@}%
3134 }
3135 \forest@def@processor@W@{1}{1}{% nonconsuming wrap: first test 1<=#1<=#2
3136 \forest@process@Wfalse
3137 \forest@process@check@mn@W@{0}\the\forest@process@n@{1}{9}%
3138 \expandafter\forest@processor@wW@\expandafter{\the\forest@process@n@}%
3139 }
3140 \def\forest@processor@wW@#1{%
3141 \forest@process@left@checkindex{\forest@process@left@N-#1}%
3142 \edef\forest@marshal{%
3143 \edef\noexpand\forest@temp@args{%
3144 \noexpand\forest@process@left@valuesfromrange
3145 {\number\numexpr\forest@process@left@N-#1}%
3146 {\the\forest@process@left@N}%
3147 }%
3148 }\forest@marshal
3149 \ifforest@process@W
3150 \advance\forest@process@left@N-#1\relax
3151 \fi
3152 \forest@process@right@bottompop\forest@temp@macrobody
3153 \expandafter\forest@def@n@expandafter\forest@process@temp@macro\expandafter{\expandafter#1\expandafter}\exp
3154 \expandafter\expandafter\expandafter\forest@process@left@setappend\expandafter\expandafter\expandafter{\exp
3155 \let\forestmathresulttype\forestmathtype@generic
3156 }
3157 \def\forest@def@n#1#2{\csname forest@def@n#2\endcsname#1}
3158 \csdef{forest@def@n@1}#1{\def#1##1}
3159 \csdef{forest@def@n@2}#1{\def#1##1##2}
3160 \csdef{forest@def@n@3}#1{\def#1##1##2##3}

```

```

3161 \csdef{forest@def@n@4}#1{\def#1##1##2##3##4}
3162 \csdef{forest@def@n@5}#1{\def#1##1##2##3##4##5}
3163 \csdef{forest@def@n@6}#1{\def#1##1##2##3##4##5##6}
3164 \csdef{forest@def@n@7}#1{\def#1##1##2##3##4##5##6##7}
3165 \csdef{forest@def@n@8}#1{\def#1##1##2##3##4##5##6##7##8}
3166 \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.

```

3167 \forest@def@processor{s}{1}{-}{%
3168   \forest@temptrue   % delete the originals
3169   \expandafter\forest@processor@save\expandafter{%
3170     \the\numexpr\forest@process@left@N-\forest@process@n}}
3171 \forest@def@processor{S}{1}{-}{%
3172   \forest@tempfalse  % keep the originals
3173   \expandafter\forest@processor@save\expandafter{%
3174     \the\numexpr\forest@process@left@N-\forest@process@n}}
3175 \def\forest@processor@save#1{%
3176   \forest@process@left@checkindex{#1}%
3177   \forest@temp@count#1
3178   \forest@loop
3179   \ifnum\forest@temp@count<\forest@process@left@N\relax
3180     \forest@process@left@get@{\the\forest@temp@count}\forest@temp
3181     \forest@process@saved@letappend\forest@temp
3182     \advance\forest@temp@count+1
3183   \forest@repeat
3184   \let\forest@process@savedtype\forestmathresulttype
3185   \ifforest@temp
3186     \forest@process@left@N=#1
3187   \fi
3188 }

```

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.

```

3189 \forest@def@processor{l}{0}{-}{%
3190   \forest@temptrue
3191   \forest@processor@U@@
3192 }
3193 \forest@def@processor{L}{0}{-}{%
3194   \forest@tempfalse
3195   \forest@processor@U@@
3196 }
3197
3198 \def\forest@processor@U@@{%
3199   \ifnum\forest@process@n=0
3200     \forest@process@n\forest@process@saved@N\relax
3201   \fi
3202   \expandafter\forest@processor@U@@@%
3203   \the\numexpr\forest@process@saved@N-\forest@process@n}%
3204 }
3205 \def\forest@processor@U@@@#1{%
3206   \forest@temp@count#1
3207   \forest@loop
3208   \ifnum\forest@temp@count<\forest@process@saved@N\relax
3209     \forest@process@saved@get@{\the\forest@temp@count}\forest@temp
3210     \forest@process@left@letappend\forest@temp
3211     \advance\forest@temp@count1
3212   \forest@repeat
3213   \let\forestmathresulttype\forest@process@savedtype
3214   \ifforest@temp
3215     \let\forest@process@savedtype\forestmathtype@none
3216   \forest@process@saved@N#1

```

```

3217 \fi
3218 }

  Boolean operations:
3219 \forest@def@processor{&}{2}{-}{% and
3220 \def\forest@tempa{1}%
3221 \forest@repeat@n@times{\forest@process@n}{%
3222 \forest@process@left@toppop\forest@tempb
3223 \edef\forest@tempa{\ifnum10<\forest@tempa\forest@tempb\space 1\else0\fi}%
3224 }%
3225 \forest@process@left@setappend{\forest@tempa}%
3226 \let\forestmathresulttype\forestmathtype@count
3227 }
3228 \forest@def@processor{|}{2}{-}{% or
3229 \def\forest@tempa{0}%
3230 \forest@repeat@n@times{\forest@process@n}{%
3231 \forest@process@left@toppop\forest@tempb
3232 \edef\forest@tempa{\ifnum0=\forest@tempa\forest@tempb\space 0\else1\fi}%
3233 }%
3234 \forest@process@left@setappend{\forest@tempa}%
3235 \let\forestmathresulttype\forestmathtype@count
3236 }
3237 \forest@def@processor{!}{-}{-}{% not
3238 \forest@process@left@toppop\forest@temp
3239 \forest@process@left@setappend{\ifnum0=\forest@temp\space 1\else0\fi}%
3240 \let\forestmathresulttype\forestmathtype@count
3241 }
3242 \forest@def@processor{?}{-}{-}{%
3243 \forest@process@left@toppop\forest@temp
3244 \forest@process@right@bottompop\forest@tempa
3245 \forest@process@right@bottompop\forest@tempb
3246 \ifnum\forest@temp=0
3247 \forest@process@right@letprepend\forest@tempb
3248 \else
3249 \forest@process@right@letprepend\forest@tempa
3250 \fi
3251 \let\forestmathresulttype\forestmathtype@generic
3252 }

```

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

```

3253 \forest@def@processor{=}{-}{-}{%
3254 \forest@process@left@toppop\forest@tempa
3255 \forest@process@left@toppop\forest@tempb
3256 \forest@process@left@setappend{\ifx\forest@tempa\forest@tempb 1\else0\fi}%
3257 \let\forestmathresulttype\forestmathtype@count
3258 }
3259 \forest@def@processor{<}{-}{-}{%
3260 \forest@process@left@toppop\forest@tempb
3261 \forest@process@left@toppop\forest@tempa
3262 \ifx\forestmathresulttype\forestmathtype@generic
3263 \forest@cmp@error\forest@tempa\forest@tempb
3264 \else
3265 \forestmathlt{\forest@tempa}{\forest@tempb}%
3266 \forest@process@left@setappend{\forestmathresult}%
3267 \fi
3268 }
3269 \forest@def@processor{>}{-}{-}{%
3270 \forest@process@left@toppop\forest@tempb
3271 \forest@process@left@toppop\forest@tempa
3272 \ifx\forestmathresulttype\forestmathtype@generic
3273 \forest@cmp@error\forest@tempa\forest@tempb

```

```

3274 \else
3275   \forestmathgt{\forest@tempa}{\forest@tempb}%
3276   \forest@process@left@esetappend{\forestmathresult}%
3277 \fi
3278 }

Various.
3279 \forest@def@processor{r}{-}{-}{% reverse keylist
3280 \forest@process@right@bottompop\forest@temp
3281 \expandafter\forest@processor@r@\expandafter{\forest@temp}%
3282 }
3283 \def\forest@processor@r#1{%
3284   \forest@process@left@esetappend{}%
3285   \def\forest@tempcomma{}%
3286   \pgfkeys{/forest}{split=#1}{,}{process@rk}}%
3287   \let\forestmathresulttype\forestmathtype@generic
3288 }
3289 \forestset{%
3290   process@rk/.code={%
3291     \forest@process@left@toppop\forest@temp
3292     \forest@temp@toks{#1}%
3293     \forest@process@left@esetappend{\the\forest@temp@toks\forest@tempcomma\expandonce{\forest@temp}}%
3294     \def\forest@tempcomma{,%}
3295   }%
3296 }

```

7.1.1 Registers

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

```

3297 \def\forest@pgfmathhelper@register@toks#1#2{% #1 is discarded: it is present only for analogy with options
3298   \forestrget{#2}\pgfmathresult
3299 }
3300 \def\forest@pgfmathhelper@register@dimen#1#2{%
3301   \forestrget{#2}\forest@temp
3302   \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
3303 }
3304 \def\forest@pgfmathhelper@register@count#1#2{%
3305   \forestrget{#2}\pgfmathresult
3306 }
3307 \def\forest@declareregisterhandler#1#2{%#1=handler for specific type,#2=option name
3308   \pgfkeyssetvalue{/forest/#2/node@or@reg}{}% empty = register (node id=node)
3309   \forest@convert@others@to@underscores{#2}\forest@pgfmathoptionname
3310   \edef\forest@marshal{%
3311     \noexpand#1{/forest/#2}{/forest}{#2}{\forest@pgfmathoptionname}%
3312   }\forest@marshal
3313 }
3314 \def\forest@declaretoksregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
3315   \forest@declaretoksregister@handler@A{#1}{#2}{#3}{#4}{}%
3316 }
3317 \def\forest@declarekeylistregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
3318   \forest@declaretoksregister@handler@A{#1}{#2}{#3}{#4}{,%}
3319   \forest@copycommandkey{#1}{#1}'%
3320   \pgfkeyssetvalue{#1'/option@name}{#3}%
3321   \forest@copycommandkey{#1+}{#1}%
3322   \pgfkeysalso{#1-/.code={%
3323     \forest@fornode{}{%
3324       \forest@node@removekeysfromkeylist{##1}{#3}%
3325     }}%
3326   \pgfkeyssetvalue{#1-/option@name}{#3}%
3327 }
3328 \def\forest@declaretoksregister@handler@A#1#2#3#4#5{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
3329   \pgfkeysalso{%

```

```

3330 #1/.code={\forestrset{#3}{##1}},
3331 #2/if #3/.code n args={3}{%
3332   \forestrget{#3}\forest@temp@option@value
3333   \edef\forest@temp@compared@value{\unexpanded{##1}}%
3334   \ifx\forest@temp@option@value\forest@temp@compared@value
3335     \pgfkeysalso{##2}%
3336   \else
3337     \pgfkeysalso{##3}%
3338   \fi
3339 },
3340 #2/if in #3/.code n args={3}{%
3341   \forestrget{#3}\forest@temp@option@value
3342   \edef\forest@temp@compared@value{\unexpanded{##1}}%
3343   \expandafter\expandafter\expandafter\pgfutil@in@\expandafter\expandafter\expandafter{\expandafter\fore
3344   \ifpgfutil@in@
3345     \pgfkeysalso{##2}%
3346   \else
3347     \pgfkeysalso{##3}%
3348   \fi
3349 },
3350 }%
3351 \ifstrempy{#5}{%
3352   \pgfkeysalso{%
3353     #1+/.code={\forestrappto{#3}{#5##1}},
3354     #2/+#3/.code={\forestrpreto{#3}{##1#5}},
3355   }%
3356 }{%
3357   \pgfkeysalso{%
3358     #1+/.code={%
3359       \forestrget{#3}\forest@temp
3360       \ifdefempty{\forest@temp}{%
3361         \forestrset{#3}{##1}%
3362       }{%
3363         \forestrappto{#3}{#5##1}%
3364       }%
3365     },
3366     #2/+#3/.code={%
3367       \forestrget{#3}\forest@temp
3368       \ifdefempty{\forest@temp}{%
3369         \forestrset{#3}{##1}%
3370       }{%
3371         \forestrpreto{#3}{##1#5}%
3372       }%
3373     }%
3374   }%
3375 }%
3376 \pgfkeyssetvalue{#1/option@name}{#3}%
3377 \pgfkeyssetvalue{#1+/option@name}{#3}%
3378 \pgfkeyssetvalue{#2/+#3/option@name}{#3}%
3379 \pgfkeyslet{#1/@type}\forest@math@type@generic % for .process & co
3380 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@toks{##1}{#3}}%
3381 }
3382 \def\forest@declareautowrappedtoksregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
3383   \forest@declaretoksregister@handler{#1}{#2}{#3}{#4}%
3384   \forest@copycommandkey{#1}{#1'}%
3385   \pgfkeysalso{#1/.style={#1'/.wrap value={##1}}}%
3386   \pgfkeyssetvalue{#1'/option@name}{#3}%
3387   \forest@copycommandkey{#1+}{#1+'}%
3388   \pgfkeysalso{#1+/.style={#1+'/.wrap value={##1}}}%
3389   \pgfkeyssetvalue{#1+'/option@name}{#3}%
3390   \forest@copycommandkey{#2/+#3}{#2/+#3'}%

```



```

3391 \pgfkeysalso{#2/+#3/.style={#2/+#3'/.wrap value={##1}}}%
3392 \pgfkeyssetvalue{#2/+#3'/option@name}{#3}%
3393 }
3394 \def\forest@declarereadonlydimenregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
3395 \pgfkeysalso{%
3396 #2/if #3/.code n args={3}{%
3397 \forestrget{#3}\forest@temp@option@value
3398 \ifdim\forest@temp@option@value>##1\relax
3399 \pgfkeysalso{##2}%
3400 \else
3401 \pgfkeysalso{##3}%
3402 \fi
3403 },
3404 #2/if #3</.code n args={3}{%
3405 \forestrget{#3}\forest@temp@option@value
3406 \ifdim\forest@temp@option@value>##1\relax
3407 \pgfkeysalso{##3}%
3408 \else
3409 \pgfkeysalso{##2}%
3410 \fi
3411 },
3412 #2/if #3>/.code n args={3}{%
3413 \forestrget{#3}\forest@temp@option@value
3414 \ifdim\forest@temp@option@value<##1\relax
3415 \pgfkeysalso{##3}%
3416 \else
3417 \pgfkeysalso{##2}%
3418 \fi
3419 },
3420 }%
3421 \pgfkeyslet{#1/@type}\forest@mathtype@dimen % for .process & co
3422 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@dimen{##1}{#3}}%
3423 }
3424 \def\forest@declaredimenregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
3425 \forest@declarereadonlydimenregister@handler{#1}{#2}{#3}{#4}%
3426 \pgfkeysalso{%
3427 #1/.code={%
3428 \forestmathsetlengthmacro\forest@temp{##1}%
3429 \forestrlet{#3}\forest@temp
3430 },
3431 #1+/.code={%
3432 \forestmathsetlengthmacro\forest@temp{##1}%
3433 \pgfutil@tempdima=\forestrve{#3}
3434 \advance\pgfutil@tempdima\forest@temp\relax
3435 \forestreset{#3}{\the\pgfutil@tempdima}%
3436 },
3437 #1-/.code={%
3438 \forestmathsetlengthmacro\forest@temp{##1}%
3439 \pgfutil@tempdima=\forestrve{#3}
3440 \advance\pgfutil@tempdima-\forest@temp\relax
3441 \forestreset{#3}{\the\pgfutil@tempdima}%
3442 },
3443 #1*/.style={%
3444 #1={#4()* (##1)}%
3445 },
3446 #1:/.style={%
3447 #1={#4()/ (##1)}%
3448 },
3449 #1'/.code={%
3450 \pgfutil@tempdima=##1\relax
3451 \forestreset{#3}{\the\pgfutil@tempdima}%

```

```

3452   },
3453   #1' +/.code={%
3454     \pgfutil@tempdima=\forestrve{#3}\relax
3455     \advance\pgfutil@tempdima##1\relax
3456     \forestreset{#3}{\the\pgfutil@tempdima}%
3457   },
3458   #1' -/.code={%
3459     \pgfutil@tempdima=\forestrve{#3}\relax
3460     \advance\pgfutil@tempdima-##1\relax
3461     \forestreset{#3}{\the\pgfutil@tempdima}%
3462   },
3463   #1' */.style={%
3464     \pgfutil@tempdima=\forestrve{#3}\relax
3465     \multiply\pgfutil@tempdima##1\relax
3466     \forestreset{#3}{\the\pgfutil@tempdima}%
3467   },
3468   #1' :/.style={%
3469     \pgfutil@tempdima=\forestrve{#3}\relax
3470     \divide\pgfutil@tempdima##1\relax
3471     \forestreset{#3}{\the\pgfutil@tempdima}%
3472   },
3473   }%
3474   \pgfkeyssetvalue{#1/option@name}{#3}%
3475   \pgfkeyssetvalue{#1+/option@name}{#3}%
3476   \pgfkeyssetvalue{#1-/option@name}{#3}%
3477   \pgfkeyssetvalue{#1*/option@name}{#3}%
3478   \pgfkeyssetvalue{#1:/option@name}{#3}%
3479   \pgfkeyssetvalue{#1'/option@name}{#3}%
3480   \pgfkeyssetvalue{#1'+/option@name}{#3}%
3481   \pgfkeyssetvalue{#1'-/option@name}{#3}%
3482   \pgfkeyssetvalue{#1'*/option@name}{#3}%
3483   \pgfkeyssetvalue{#1':/option@name}{#3}%
3484 }
3485 \def\forest@declarereadonlycountregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
3486   \pgfkeysalso{
3487     #2/if #3/.code n args={3}{%
3488       \forestrget{#3}\forest@temp@option@value
3489       \ifnum\forest@temp@option@value=##1\relax
3490         \pgfkeysalso{##2}%
3491       \else
3492         \pgfkeysalso{##3}%
3493       \fi
3494     },
3495     #2/if #3</.code n args={3}{%
3496       \forestrget{#3}\forest@temp@option@value
3497       \ifnum\forest@temp@option@value>##1\relax
3498         \pgfkeysalso{##3}%
3499       \else
3500         \pgfkeysalso{##2}%
3501       \fi
3502     },
3503     #2/if #3>/.code n args={3}{%
3504       \forestrget{#3}\forest@temp@option@value
3505       \ifnum\forest@temp@option@value<##1\relax
3506         \pgfkeysalso{##3}%
3507       \else
3508         \pgfkeysalso{##2}%
3509       \fi
3510     },
3511   }%
3512   \pgfkeyslet{#1/@type}\forestmathtype@count % for .process & co

```

```

3513 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@count{##1}{#3}}%
3514 }
3515 \def\forest@declarecountregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
3516 \forest@declarereadonlycountregister@handler{#1}{#2}{#3}{#4}%
3517 \pgfkeysalso{
3518 #1/.code={%
3519 \forestmathtruncatemacro\forest@temp{##1}%
3520 \forestrlet{#3}\forest@temp
3521 },
3522 #1+/.code={%
3523 \forestmathtruncatemacro\forest@temp{##1}%
3524 \c@pgf@counta=\forestrve{#3}\relax
3525 \advance\c@pgf@counta\forest@temp\relax
3526 \forestreset{#3}{\the\c@pgf@counta}%
3527 },
3528 #1-/.code={%
3529 \forestmathtruncatemacro\forest@temp{##1}%
3530 \c@pgf@counta=\forestrve{#3}\relax
3531 \advance\c@pgf@counta-\forest@temp\relax
3532 \forestreset{#3}{\the\c@pgf@counta}%
3533 },
3534 #1*/.code={%
3535 \forestmathtruncatemacro\forest@temp{##1}%
3536 \c@pgf@counta=\forestrve{#3}\relax
3537 \multiply\c@pgf@counta\forest@temp\relax
3538 \forestreset{#3}{\the\c@pgf@counta}%
3539 },
3540 #1:/ .code={%
3541 \forestmathtruncatemacro\forest@temp{##1}%
3542 \c@pgf@counta=\forestrve{#3}\relax
3543 \divide\c@pgf@counta\forest@temp\relax
3544 \forestreset{#3}{\the\c@pgf@counta}%
3545 },
3546 #1'/.code={%
3547 \c@pgf@counta=##1\relax
3548 \forestreset{#3}{\the\c@pgf@counta}%
3549 },
3550 #1'+/.code={%
3551 \c@pgf@counta=\forestrve{#3}\relax
3552 \advance\c@pgf@counta##1\relax
3553 \forestreset{#3}{\the\c@pgf@counta}%
3554 },
3555 #1'-/.code={%
3556 \c@pgf@counta=\forestrve{#3}\relax
3557 \advance\c@pgf@counta-##1\relax
3558 \forestreset{#3}{\the\c@pgf@counta}%
3559 },
3560 #1'*/.style={%
3561 \c@pgf@counta=\forestrve{#3}\relax
3562 \multiply\c@pgf@counta##1\relax
3563 \forestreset{#3}{\the\c@pgf@counta}%
3564 },
3565 #1':/.style={%
3566 \c@pgf@counta=\forestrve{#3}\relax
3567 \divide\c@pgf@counta##1\relax
3568 \forestreset{#3}{\the\c@pgf@counta}%
3569 },
3570 }%
3571 \pgfkeyssetvalue{#1/option@name}{#3}%
3572 \pgfkeyssetvalue{#1+/option@name}{#3}%
3573 \pgfkeyssetvalue{#1-/option@name}{#3}%

```

```

3574 \pgfkeyssetvalue{#1*/option@name}{#3}%
3575 \pgfkeyssetvalue{#1:/option@name}{#3}%
3576 \pgfkeyssetvalue{#1'/option@name}{#3}%
3577 \pgfkeyssetvalue{#1'+/option@name}{#3}%
3578 \pgfkeyssetvalue{#1'-/option@name}{#3}%
3579 \pgfkeyssetvalue{#1'*/*option@name}{#3}%
3580 \pgfkeyssetvalue{#1':/*option@name}{#3}%
3581 }
3582 \def\forest@declarebooleanregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
3583 \pgfkeysalso{%
3584 #1/.code={%
3585 \ifcsdef{forest@bh@\detokenize{##1}}{%
3586 \letcs\forest@temp{forest@bh@\detokenize{##1}}%
3587 }{%
3588 \forestmathtruncatemacro\forest@temp{##1}%
3589 \ifx\forest@temp0\else\def\forest@temp{1}\fi
3590 }%
3591 \forestrlet{#3}\forest@temp
3592 },
3593 #1/.default=1,
3594 #2/not #3/.code={\forestrset{#3}{0}},
3595 #2/if #3/.code 2 args={%
3596 \forestrget{#3}\forest@temp@option@value
3597 \ifnum\forest@temp@option@value=1
3598 \pgfkeysalso{##1}%
3599 \else
3600 \pgfkeysalso{##2}%
3601 \fi
3602 },
3603 }%
3604 \pgfkeyssetvalue{#1/option@name}{#3}%
3605 \pgfkeyslet{#1/@type}\forestmathtypecount % for .process & co
3606 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@count{##1}{#3}}%
3607 }
3608 \forestset{
3609 declare toks register/.code={%
3610 \forest@declareregisterhandler\forest@declaretoksregister@handler{#1}%
3611 \forestset{#1={}}%
3612 },
3613 declare autowrapped toks register/.code={%
3614 \forest@declareregisterhandler\forest@declareautowrappedtoksregister@handler{#1}%
3615 \forestset{#1={}}%
3616 },
3617 declare keylist register/.code={%
3618 \forest@declareregisterhandler\forest@declarekeylistregister@handler{#1}%
3619 \forestset{#1'={}}%
3620 },
3621 declare dimen register/.code={%
3622 \forest@declareregisterhandler\forest@declaredimenregister@handler{#1}%
3623 \forestset{#1'=0pt}%
3624 },
3625 declare count register/.code={%
3626 \forest@declareregisterhandler\forest@declarecountregister@handler{#1}%
3627 \forestset{#1'=0}%
3628 },
3629 declare boolean register/.code={%
3630 \forest@declareregisterhandler\forest@declarebooleanregister@handler{#1}%
3631 \forestset{#1=0}%
3632 },
3633 }

```

Declare some temporary registers.

```

3634 \forestset{
3635   declare toks register=temptoksa,temptoksa={},
3636   declare toks register=temptoksb,temptoksb={},
3637   declare toks register=temptoksc,temptoksc={},
3638   declare toks register=temptoksd,temptoksd={},
3639   declare keylist register=tempkeylista,tempkeylista'={},
3640   declare keylist register=tempkeylistb,tempkeylistb'={},
3641   declare keylist register=tempkeylistc,tempkeylistc'={},
3642   declare keylist register=tempkeylistd,tempkeylistd'={},
3643   declare dimen register=tempdima,tempdima'={Opt},
3644   declare dimen register=tempdimb,tempdimb'={Opt},
3645   declare dimen register=tempdimc,tempdimc'={Opt},
3646   declare dimen register=tempdimd,tempdimd'={Opt},
3647   declare dimen register=tempdimx,tempdimx'={Opt},
3648   declare dimen register=tempdimxa,tempdimxa'={Opt},
3649   declare dimen register=tempdimxb,tempdimxb'={Opt},
3650   declare dimen register=tempdimy,tempdimy'={Opt},
3651   declare dimen register=tempdimya,tempdimya'={Opt},
3652   declare dimen register=tempdimyb,tempdimyb'={Opt},
3653   declare dimen register=tempdiml,tempdiml'={Opt},
3654   declare dimen register=tempdimla,tempdimla'={Opt},
3655   declare dimen register=tempdimlb,tempdimlb'={Opt},
3656   declare dimen register=tempdims,tempdims'={Opt},
3657   declare dimen register=tempdimsa,tempdimsa'={Opt},
3658   declare dimen register=tempdimsb,tempdimsb'={Opt},
3659   declare count register=tempcounta,tempcounta'={0},
3660   declare count register=tempcountb,tempcountb'={0},
3661   declare count register=tempcountc,tempcountc'={0},
3662   declare count register=tempcountd,tempcountd'={0},
3663   declare boolean register=tempboola,tempboola={0},
3664   declare boolean register=tempboolb,tempboolb={0},
3665   declare boolean register=tempboolc,tempboolc={0},
3666   declare boolean register=tempboold,tempboold={0},
3667 }

```

7.1.2 Declaring options

```

3668 \def\forest@node@Nametoid#1{% #1 = name
3669   \csname forest@id@of@#1\endcsname
3670 }
3671 \def\forest@node@ifnamedefined#1#2#3{% #1 = name, #2=true,#3=false
3672   \ifcsvoid{forest@id@of@#1}{#3}{#2}%
3673 }
3674 \def\forest@node@setname#1{%
3675   \def\forest@temp@setname{y}%
3676   \def\forest@temp@silent{n}%
3677   \def\forest@temp@propagating{n}%
3678   \forest@node@setnameoralias{#1}%
3679 }
3680 \def\forest@node@setname@silent#1{%
3681   \def\forest@temp@setname{y}%
3682   \def\forest@temp@silent{y}%
3683   \def\forest@temp@propagating{n}%
3684   \forest@node@setnameoralias{#1}%
3685 }
3686 \def\forest@node@setalias#1{%
3687   \def\forest@temp@setname{n}%
3688   \def\forest@temp@silent{n}%
3689   \def\forest@temp@propagating{n}%

```

```

3690 \forest@node@setnameoralias{#1}%
3691 }
3692 \def\forest@node@setalias@silent#1{%
3693 \def\forest@temp@setname{n}%
3694 \def\forest@temp@silent{y}%
3695 \def\forest@temp@propagating{n}%
3696 \forest@node@setnameoralias{#1}%
3697 }
3698 \def\forest@node@setnameoralias#1{%
3699 \ifstrempy{#1}{%
3700 \forest@node@setnameoralias{node@\forest@cn}%
3701 }{%
3702 \forest@node@ifnamedefined{#1}{%
3703 \if y\forest@temp@propagating
3704 % this will find a unique name, eventually:
3705 \@escapeif{\forest@node@setnameoralias{#1@\forest@cn}}%
3706 \else\@escapeif{%
3707 \if y\forest@temp@setname
3708 \edef\forest@marshal{%
3709 \ifstrequal{\forestove{name}}{#1}%
3710 }\forest@marshal{%
3711 % same name, no problem
3712 }{%
3713 \@escapeif{\forest@node@setnameoralias@nameclash{#1}}%
3714 }%
3715 \else\@escapeif{% setting an alias: clashing with alias is not a problem
3716 \forest@get{\forest@node@Nametoid{#1}}{name}\forest@temp
3717 \expandafter\ifstrequal\expandafter{\forest@temp}{#1}{%
3718 \forest@node@setnameoralias@nameclash{#1}%
3719 }{%
3720 \forest@node@setnameoralias@do{#1}%
3721 }%
3722 }\fi
3723 }\fi
3724 }{%
3725 \forest@node@setnameoralias@do{#1}%
3726 }%
3727 }%
3728 }
3729 \def\forest@node@setnameoralias@nameclash#1{%
3730 \if y\forest@temp@silent
3731 \forest@fornode{\forest@node@Nametoid{#1}}{%
3732 \def\forest@temp@propagating{y}%
3733 \forest@node@setnameoralias{}%
3734 }%
3735 \forest@node@setnameoralias@do{#1}%
3736 \else
3737 \PackageError{forest}{Node name "#1" is already used}{}%
3738 \fi
3739 }
3740 \def\forest@node@setnameoralias@do#1{%
3741 \if y\forest@temp@setname
3742 \csdef{forest@id@of@\forestove{name}}{}%
3743 \forestoeset{name}{#1}%
3744 \fi
3745 \csedef{forest@id@of@#1}{\forest@cn}%
3746 }
3747 \forestset{
3748 TeX/.code={#1},
3749 TeX'/.code={\appto\forest@externalize@loadimages{#1}#1},
3750 TeX''/.code={\appto\forest@externalize@loadimages{#1}},

```

```

3751 options/.code={\forestset{#1}},
3752 also/.code={\pgfkeysalso{#1}},
3753 typeout/.style={TeX={\typeout{#1}}},
3754 declare toks={name}{},
3755 name/.code={% override the default setter
3756   \forest@fornode{\forest@setter@node}{\forest@node@setname{#1}}%
3757 },
3758 name/.default={},
3759 name'/.code={% override the default setter
3760   \forest@fornode{\forest@setter@node}{\forest@node@setname@silent{#1}}%
3761 },
3762 name'/.default={},
3763 alias/.code={\forest@fornode{\forest@setter@node}{\forest@node@setalias{#1}}},
3764 alias'/.code={\forest@fornode{\forest@setter@node}{\forest@node@setalias@silent{#1}}},
3765 begin draw/.code={\begin{tikzpicture}},
3766 end draw/.code={\end{tikzpicture}},
3767 declare keylist register=default preamble,
3768 default preamble'={},
3769 declare keylist register=preamble,
3770 preamble'={},
3771 declare autowrapped toks={content}{},
3772 % #1 = which option to split, #2 = separator (one char!), #3 = receiving options
3773 split option/.code n args=3{%
3774   \forestRNOget{#1}\forest@temp
3775   \edef\forest@marshal{%
3776     \noexpand\pgfkeysalso{split={\expandonce{\forest@temp}}\unexpanded{#2}{#3}}%
3777   }\forest@marshal
3778 },
3779 split register/.code n args=3{% #1 = which register to split, #2 = separator (one char!), #3 = receiving options
3780   \forestrget{#1}\forest@temp
3781   \edef\forest@marshal{%
3782     \noexpand\pgfkeysalso{split={\expandonce{\forest@temp}}\unexpanded{#2}{#3}}%
3783   }\forest@marshal
3784 },
3785 TeX={%
3786   \def\forest@split@sourcevalues{}%
3787   \def\forest@split@sourcevalue{}%
3788   \def\forest@split@receivingoptions{}%
3789   \def\forest@split@receivingoption{}%
3790 },
3791 split/.code n args=3{% #1 = string to split, #2 = separator (one char!), #3 = receiving options
3792   \forest@saveandrestoremacro\forest@split@sourcevalues{%
3793     \forest@saveandrestoremacro\forest@split@sourcevalue{%
3794       \forest@saveandrestoremacro\forest@split@receivingoptions{%
3795         \forest@saveandrestoremacro\forest@split@receivingoption{%
3796           \def\forest@split@sourcevalues{#1#2}%
3797           \edef\forest@split@receivingoptions{#3,%}
3798           \def\forest@split@receivingoption{}%
3799           \safeloop
3800             \expandafter\forest@split\expandafter{\forest@split@sourcevalues}{#2}\forest@split@sourcevalue\
3801             \ifdefempty\forest@split@receivingoptions{}{%
3802               \expandafter\forest@split\expandafter{\forest@split@receivingoptions}{,}\forest@temp\forest@split@receivingoption\
3803               \ifdefempty\forest@temp{}{\let\forest@split@receivingoption\forest@temp\def\forest@temp{}}%
3804             }%
3805           \edef\forest@marshal{%
3806             \noexpand\pgfkeysalso{\forest@split@receivingoption={\expandonce{\forest@split@sourcevalue}}%
3807           }\forest@marshal
3808           \ifdefempty\forest@split@sourcevalues{\forest@tempfalse}{\forest@temptrue}%
3809           \ifforest@temp
3810             \saferepeat
3811           }}}}%

```

```

3812 },
3813 declare count={grow}{270},
3814 TeX={% a hack for grow-reversed connection, and compass-based grow specification
3815   \forest@copycommandkey{/forest/grow}{/forest/grow@}%
3816   %\pgfkeysgetvalue{/forest/grow/.@cmd}\forest@temp
3817   %\pgfkeyslet{/forest/grow@/.@cmd}\forest@temp
3818 },
3819 grow/.style={grow@=#1,reversed=0},
3820 grow'/.style={grow@=#1,reversed=1},
3821 grow''/.style={grow@=#1}},
3822 grow@/.is choice,
3823 grow@/east/.style={/forest/grow@=0},
3824 grow@/north east/.style={/forest/grow@=45},
3825 grow@/north/.style={/forest/grow@=90},
3826 grow@/north west/.style={/forest/grow@=135},
3827 grow@/west/.style={/forest/grow@=180},
3828 grow@/south west/.style={/forest/grow@=225},
3829 grow@/south/.style={/forest/grow@=270},
3830 grow@/south east/.style={/forest/grow@=315},
3831 grow@/.unknown/.code={\let\forest@temp@grow\pgfkeyscurrentname
3832   \pgfkeysalso{/forest/grow@/.expand once=\forest@temp@grow}},
3833 declare boolean={reversed}{0},
3834 declare toks={parent anchor}{},
3835 declare toks={child anchor}{},
3836 declare toks={anchor}{base},
3837 Autoforward={anchor}{
3838   node options-=anchor,
3839   node options+={anchor=##1}}
3840 },
3841 anchor'/.style={anchor@no@compass=true,anchor=#1},
3842 anchor+/.style={anchor@no@compass=true,anchor+=#1},
3843 anchor-/.style={anchor@no@compass=true,anchor-=#1},
3844 anchor*'.style={anchor@no@compass=true,anchor*=#1},
3845 anchor:'.style={anchor@no@compass=true,anchor:=#1},
3846 anchor'+/.style={anchor@no@compass=true,anchor'+=#1},
3847 anchor'-/.style={anchor@no@compass=true,anchor'-=#1},
3848 anchor*'/.style={anchor@no@compass=true,anchor'*=#1},
3849 anchor':'.style={anchor@no@compass=true,anchor':=#1},
3850 % /tikz/forest anchor/.style={
3851 %   /forest/TeX={\forestanchortotikzanchor{#1}\forest@temp@anchor},
3852 %   anchor/.expand once=\forest@temp@anchor
3853 % },
3854 declare toks={calign}{midpoint},
3855 TeX={%
3856   \forest@copycommandkey{/forest/calign}{/forest/calign'}%
3857 },
3858 calign/.is choice,
3859 calign/child/.style={calign'=child},
3860 calign/first/.style={calign'=child,calign primary child=1},
3861 calign/last/.style={calign'=child,calign primary child=-1},
3862 calign with current/.style={for parent/.wrap pgfmath arg={calign=child,calign primary child=##1}{n}},
3863 calign with current edge/.style={for parent/.wrap pgfmath arg={calign=child edge,calign primary child=##1}{-}},
3864 calign/child edge/.style={calign'=child edge},
3865 calign/midpoint/.style={calign'=midpoint},
3866 calign/center/.style={calign'=midpoint,calign primary child=1,calign secondary child=-1},
3867 calign/edge midpoint/.style={calign'=edge midpoint},
3868 calign/fixed angles/.style={calign'=fixed angles},
3869 calign/fixed edge angles/.style={calign'=fixed edge angles},
3870 calign/.unknown/.code={\PackageError{forest}{unknown calign ' \pgfkeyscurrentname' }{}},
3871 declare count={calign primary child}{1},
3872 declare count={calign secondary child}{-1},

```



```

3873 declare count={calign primary angle}{-35},
3874 declare count={calign secondary angle}{35},
3875 calign child/.style={calign primary child={#1}},
3876 calign angle/.style={calign primary angle=-{#1},calign secondary angle={#1}},
3877 declare toks={tier}{},
3878 declare toks={fit}{tight},
3879 declare boolean={ignore}{0},
3880 declare boolean={ignore edge}{0},
3881 no edge/.style={edge'={},ignore edge},
3882 declare keylist={edge}{draw},
3883 declare toks={edge path}{%
3884   \noexpand\path[\forestoption{edge}]%
3885   (\forestOve{\forestove{@parent}}{name}.parent anchor)--(\forestove{name}.child anchor)
3886   % =
3887   % (!u.parent anchor)--(.child anchor)\forestoption{edge label};
3888   \forestoption{edge label};%
3889 },
3890 edge path'/.style={
3891   edge path={%
3892     \noexpand\path[\forestoption{edge}]%
3893     #1%
3894     \forestoption{edge label};
3895   }
3896 },
3897 declare toks={edge label}{},
3898 declare boolean={phantom}{0},
3899 baseline/.style={alias={forest@baseline@node}},
3900 declare readonly count={id}{0},
3901 declare readonly count={n}{0},
3902 declare readonly count={n'}{0},
3903 declare readonly count={n children}{-1},
3904 declare readonly count={level}{-1},
3905 declare dimen=x{0pt},
3906 declare dimen=y{0pt},
3907 declare dimen={s}{0pt},
3908 declare dimen={l}{6ex}, % just in case: should be set by the calibration
3909 declare dimen={s sep}{0.6666em},
3910 declare dimen={l sep}{1ex}, % just in case: calibration!
3911 declare keylist={node options}{anchor=base},
3912 declare toks={tikz}{},
3913 afterthought/.style={tikz+={#1}},
3914 label/.style={tikz+={\path[late options={%
3915   name=\forestoption{name},label={#1}}];}},
3916 pin/.style={tikz+={\path[late options={%
3917   name=\forestoption{name},pin={#1}}];}},
3918 declare toks={content format}{\forestoption{content}},
3919 plain content/.style={content format={\forestoption{content}}},
3920 math content/.style={content format={\noexpand\ensuremath{\forestoption{content}}}},
3921 declare toks={node format}{%
3922   \noexpand\node
3923   (\forestoption{name})%
3924   [\forestoption{node options}]%
3925   {\forestoption{content format}};%
3926 },
3927 node format'/.style={
3928   node format={\noexpand\node(\forestoption{name})#1;}
3929 },
3930 tabular@environment/.style={content format={%
3931   \noexpand\begin{tabular}[\forestoption{base}]{\forestoption{align}}%
3932   \forestoption{content}%
3933   \noexpand\end{tabular}%

```

```

3934  }},
3935  declare toks={align}{},
3936  TeX={%
3937    \forest@copycommandkey{/forest/align}{/forest/align'}%
3938    %\pgfkeysgetvalue{/forest/align/.cmd}\forest@temp
3939    %\pgfkeyslet{/forest/align'/.cmd}\forest@temp
3940  },
3941  align/.is choice,
3942  align/.unknown/.code={%
3943    \edef\forest@marshal{%
3944      \noexpand\pgfkeysalso{%
3945        align'={\pgfkeyscurrentname},%
3946        tabular@environment
3947      }%
3948    }\forest@marshal
3949  },
3950  align/center/.style={align'=@{c@{}}},tabular@environment},
3951  align/left/.style={align'=@{l@{}}},tabular@environment},
3952  align/right/.style={align'=@{r@{}}},tabular@environment},
3953  declare toks={base}{t},
3954  TeX={%
3955    \forest@copycommandkey{/forest/base}{/forest/base'}%
3956    %\pgfkeysgetvalue{/forest/base/.cmd}\forest@temp
3957    %\pgfkeyslet{/forest/base'/.cmd}\forest@temp
3958  },
3959  base/.is choice,
3960  base/top/.style={base'=t},
3961  base/bottom/.style={base'=b},
3962  base/.unknown/.style={base'/.expand once=\pgfkeyscurrentname},
3963  unknown to/.store in=\forest@unknownto,
3964  unknown to=node options,
3965  unknown key error/.code={\PackageError{forest}{Unknown keyval: \detokenize{#1}}{}}},
3966  content to/.store in=\forest@contentto,
3967  content to=content,
3968  .unknown/.code={%
3969    \expandafter\pgfutil@in@\expandafter.\expandafter{\pgfkeyscurrentname}%
3970    \ifpgfutil@in@
3971      \expandafter\forest@relatednode@option@setter\pgfkeyscurrentname=#1\forest@END
3972    \else
3973      \edef\forest@marshal{%
3974        \noexpand\pgfkeysalso{\forest@unknownto={\pgfkeyscurrentname=\unexpanded{#1}}}%
3975      }\forest@marshal
3976    \fi
3977  },
3978  get node boundary/.code={%
3979    \forestoget{@boundary}\forest@node@boundary
3980    \def#1{}%
3981    \forest@extendpath#1\forest@node@boundary{\pgfqpoint{\forestove{x}}{\forestove{y}}}%
3982  },
3983  % get min l tree boundary/.code={%
3984  %   \forest@get@tree@boundary{negative}{\the\numexpr\forestove{grow}-90\relax}#1},
3985  % get max l tree boundary/.code={%
3986  %   \forest@get@tree@boundary{positive}{\the\numexpr\forestove{grow}-90\relax}#1},
3987  get min s tree boundary/.code={%
3988    \forest@get@tree@boundary{negative}{\forestove{grow}}#1},
3989  get max s tree boundary/.code={%
3990    \forest@get@tree@boundary{positive}{\forestove{grow}}#1},
3991  use as bounding box/.style={%
3992    before drawing tree={
3993      tikz+/.expanded={%
3994        \noexpand\pgfresetboundingbox

```

```

3995     \noexpand\useasboundingbox
3996     ($(.anchor)+(\forestoption{min x},\forestoption{min y})$)
3997     rectangle
3998     ($(.anchor)+(\forestoption{max x},\forestoption{max y})$)
3999     ;
4000   }
4001 }
4002 },
4003 use as bounding box'/.style={%
4004   before drawing tree={
4005     tikz+/.expanded={%
4006       \noexpand\pgfresetboundingbox
4007       \noexpand\useasboundingbox
4008       ($(.anchor)+(\forestoption{min x}+\pgfkeysvalueof{/pgf/outer xsep}/2+\pgfkeysvalueof{/pgf/inner xsep})
4009       rectangle
4010       ($(.anchor)+(\forestoption{max x}-\pgfkeysvalueof{/pgf/outer xsep}/2-\pgfkeysvalueof{/pgf/inner xsep})
4011       ;
4012     }
4013   }
4014 },
4015 }%
4016 \def\forest@iftikzkey#1#2#3{% #1 = key name, #2 = true code, #3 = false code
4017   \forest@temptrue
4018   \pgfkeysifdefined{/tikz/\pgfkeyscurrentname}{}%
4019   \pgfkeysifdefined{/tikz/\pgfkeyscurrentname/.@cmd}{}%
4020   \pgfkeysifdefined{/pgf/\pgfkeyscurrentname}{}%
4021   \pgfkeysifdefined{/pgf/\pgfkeyscurrentname/.@cmd}{}%
4022   \forest@tempfalse
4023   }%
4024 \ifforest@temp\@escapeif{#2}\else\@escapeif{#3}\fi
4025 }
4026 \def\forest@ifoptionortikzkey#1#2#3{% #1 = key name, #2 = true code, #3 = false code
4027   \forest@temptrue
4028   \pgfkeysifdefined{/forest/\pgfkeyscurrentname}{}%
4029   \pgfkeysifdefined{/forest/\pgfkeyscurrentname/.@cmd}{}%
4030   \forest@iftikzkey{#1}{}%
4031   }%
4032 \ifforest@temp\@escapeif{#2}\else\@escapeif{#3}\fi
4033 }
4034 \def\forest@get@tree@boundary#1#2#3{% #1=pos/neg,#2=grow,#3=receiving cs
4035   \def#3{}%
4036   \forest@node@getedge{#1}{#2}\forest@temp@boundary
4037   \forest@extendpath#3\forest@temp@boundary{\pgfpoint{\forestove{x}}{\forestove{y}}}%
4038 }
4039 \def\forest@setter@node{\forest@cn}%
4040 \def\forest@relatednode@option@compat@ignoreinvalidsteps#1{#1}
4041 \def\forest@relatednode@option@setter#1.#2=#3\forest@END{%
4042   \forest@forthis{%
4043     \forest@relatednode@option@compat@ignoreinvalidsteps{%
4044       \forest@nameandgo{#1}%
4045       \let\forest@setter@node\forest@cn
4046     }%
4047   }%
4048   \ifnum\forest@setter@node=0
4049   \else
4050     \forestset{#2={#3}}%
4051   \fi
4052   \def\forest@setter@node{\forest@cn}%
4053 }%
4054 \def\forest@split#1#2#3#4{% #1=list (assuming that the list is nonempty and finishes with the separator), #2
4055   \def\forest@split@##1#2##2\forest@split@##3##4{\def##3{##1}\def##4{##2}}%

```

```
4056 \forest@split@@#1\forest@split@@{#3}{#4}}
```

7.1.3 Option propagation

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

```
4057 \forestset{
4058   for tree'/.style 2 args={#1,for children={for tree'={#1}{#2}},#2},
4059   if/.code n args={3}{%
4060     \forestmathtruncatemacro\forest@temp{#1}%
4061     \ifnum\forest@temp=0
4062       \@escapeif{\pgfkeysalso{#3}}%
4063     \else
4064       \@escapeif{\pgfkeysalso{#2}}%
4065     \fi
4066   },
4067   %LaTeX if/.code n args={3}{#1{\pgfkeysalso{#2}}{\pgfkeysalso{#3}}},
4068   if nodewalk valid/.code n args={3}{%
4069     \forest@forthis{%
4070       \forest@configured@nodewalk{independent}{inherited}{fake}{%
4071         #1,
4072         TeX={\global\let\forest@global@temp\forest@cn}
4073       }{}%
4074     }%
4075     \ifnum\forest@global@temp=0
4076       \@escapeif{\pgfkeysalso{#3}}%
4077     \else
4078       \@escapeif{\pgfkeysalso{#2}}%
4079     \fi
4080   },
4081   if nodewalk empty/.code n args={3}{%
4082     \forest@forthis{%
4083       \forest@configured@nodewalk{independent}{independent}{fake}{%
4084         #1,
4085         TeX={\global\let\forest@global@temp\forest@nodewalk@n},
4086       }{}%
4087     }%
4088     \ifnum\forest@global@temp=0
4089       \@escapeif{\pgfkeysalso{#2}}%
4090     \else
4091       \@escapeif{\pgfkeysalso{#3}}%
4092     \fi
4093   },
4094   if current nodewalk empty/.code 2 args={%
4095     \ifnum\forest@nodewalk@n=0
4096       \@escapeif{\pgfkeysalso{#1}}%
4097     \else
4098       \@escapeif{\pgfkeysalso{#2}}%
4099     \fi
4100   },
4101   where/.style n args={3}{for tree={if={#1}{#2}{#3}},
4102   where nodewalk valid/.style n args={3}{for tree={if nodewalk valid={#1}{#2}{#3}}},
4103   where nodewalk empty/.style n args={3}{for tree={if nodewalk empty={#1}{#2}{#3}}},
4104   repeat/.code 2 args={%
4105     \forestmathtruncatemacro\forest@temp{#1}%
4106     \expandafter\forest@repeatkey\expandafter{\forest@temp}{#2}%
4107   },
4108   until/.code 2 args={%
4109     \ifstrempy{#1}{%
4110       \forest@untilkey{\ifnum\forest@cn=0\else\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
4111     }{%
4112       \forest@untilkey{\forestmath@if{#1}{\forestloopbreak}{}}{#2}%

```

```

4113 }%
4114 },
4115 while/.code 2 args={%
4116   \ifstrempy{#1}{%
4117     \forest@untilkey{\ifnum\forest@cn=0\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
4118   }{%
4119     \forest@untilkey{\forestmath@if{#1}{}\forestloopbreak}}{#2}%
4120   }%
4121 },
4122 do until/.code 2 args={%
4123   \ifstrempy{#1}{%
4124     \forest@dountilkey{\ifnum\forest@cn=0\else\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
4125   }{%
4126     \forest@dountilkey{\forestmath@if{#1}{}\forestloopbreak}}{#2}%
4127   }%
4128 },
4129 do while/.code 2 args={%
4130   \ifstrempy{#1}{%
4131     \forest@dountilkey{\ifnum\forest@cn=0\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
4132   }{%
4133     \forest@dountilkey{\forestmath@if{#1}{}\forestloopbreak}}{#2}%
4134   }%
4135 },
4136 until nodewalk valid/.code 2 args={%
4137   \forest@untilkey{\forest@forthis{%
4138     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\else\forestloopbreak\fi}}}{#2}
4139   },
4140 while nodewalk valid/.code 2 args={%
4141   \forest@untilkey{\forest@forthis{%
4142     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\forestloopbreak\fi}}}{#2}
4143   },
4144 do until nodewalk valid/.code 2 args={%
4145   \forest@dountilkey{\forest@forthis{%
4146     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\else\forestloopbreak\fi}}}{#2}
4147   },
4148 do while nodewalk valid/.code 2 args={%
4149   \forest@dountilkey{\forest@forthis{%
4150     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\forestloopbreak\fi}}}{#2}
4151   },
4152 until nodewalk empty/.code 2 args={%
4153   \forest@untilkey{\forest@forthis{%
4154     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\forestloopbreak\fi}}}{#2}
4155   },
4156 while nodewalk empty/.code 2 args={%
4157   \forest@untilkey{\forest@forthis{%
4158     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\else\forestloopbreak\fi}}
4159   },
4160 do until nodewalk empty/.code 2 args={%
4161   \forest@dountilkey{\forest@forthis{%
4162     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\forestloopbreak\fi}}}{#2}
4163   },
4164 do while nodewalk empty/.code 2 args={%
4165   \forest@dountilkey{\forest@forthis{%
4166     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\else\forestloopbreak\fi}}
4167   },
4168 break/.code={\forestloopBreak{#1}},
4169 break/.default=0,
4170 }
4171 \def\forest@repeatkey#1#2{%
4172   \safeRKloop
4173   \ifnum\safeRKloopn>#1\relax

```

```

4174 \csuse{safeRKbreak@the\safeRKloop@depth true}%
4175 \fi
4176 \expandafter\unless\csname ifsafeRKbreak@the\safeRKloop@depth\endcsname
4177 \pgfkeysalso{#2}%
4178 \safeRKrepeat
4179 }
4180 \def\forest@untilkey#1#2{% #1 = condition, #2 = keys
4181 \safeRKloop
4182 #1%
4183 \expandafter\unless\csname ifsafeRKbreak@the\safeRKloop@depth\endcsname
4184 \pgfkeysalso{#2}%
4185 \safeRKrepeat
4186 }
4187 \def\forest@dountilkey#1#2{% #1 = condition, #2 = keys
4188 \safeRKloop
4189 \pgfkeysalso{#2}%
4190 #1%
4191 \expandafter\unless\csname ifsafeRKbreak@the\safeRKloop@depth\endcsname
4192 \safeRKrepeat
4193 }
4194 \def\forestloopbreak{%
4195 \csname safeRKbreak@the\safeRKloop@depth true\endcsname
4196 }
4197 \def\forestloopBreak#1{%
4198 \csname safeRKbreak@number\numexpr\the\safeRKloop@depth-#1\relax true\endcsname
4199 }
4200 \def\forestloopcount{%
4201 \csname safeRKloopn@number\numexpr\the\safeRKloop@depth\endcsname
4202 }
4203 \def\forestloopCount#1{%
4204 \csname safeRKloopn@number\numexpr\the\safeRKloop@depth-#1\endcsname
4205 }
4206 \pgfmathdeclarefunction{forestloopcount}{1}{%
4207 \edef\pgfmathresult{\forestloopCount{\ifstrempty{#1}{0}{#1}}}%
4208 }
4209 \forest@copycommandkey{/forest/repeat}{/forest/nodewalk/repeat}
4210 \forest@copycommandkey{/forest/while}{/forest/nodewalk/while}
4211 \forest@copycommandkey{/forest/do while}{/forest/nodewalk/do while}
4212 \forest@copycommandkey{/forest/until}{/forest/nodewalk/until}
4213 \forest@copycommandkey{/forest/do until}{/forest/nodewalk/do until}
4214 \forest@copycommandkey{/forest/if}{/forest/nodewalk/if}
4215 \forest@copycommandkey{/forest/if nodewalk valid}{/forest/nodewalk/if nodewalk valid}
4216 %

```

7.2 Aggregate functions

```

4217 \forestset{
4218 aggregate postparse/.is choice,
4219 aggregate postparse/int/.code={%
4220 \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@toint},
4221 aggregate postparse/none/.code={%
4222 \let\forest@aggregate@pgfmathpostparse\relax},
4223 aggregate postparse/print/.code={%
4224 \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@print},
4225 aggregate postparse/macro/.code={%
4226 \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@usemacro},
4227 aggregate postparse macro/.store in=\forest@aggregate@pgfmathpostparse@macro,
4228 }
4229 \def\forest@aggregate@pgfmathpostparse@print{%
4230 \pgfmathprintnumberto{\pgfmathresult}{\pgfmathresult}%

```

```

4231 }
4232 \def\forest@aggregate@pgfmathpostparse@toint{%
4233   \expandafter\forest@split\expandafter{\pgfmathresult.}{.}\pgfmathresult\forest@temp
4234 }
4235 \def\forest@aggregate@pgfmathpostparse@usemacro{%
4236   \forest@aggregate@pgfmathpostparse@macro
4237 }
4238 \let\forest@aggregate@pgfmathpostparse\relax
4239 \forestset{
4240   /handlers/.aggregate/.code n args=4{%
4241     % #1 = start value (forestmath)
4242     % #2 = forestmath expression that calculates "aggregate result" at each step
4243     % #3 = forestmath expression that calculates "aggregate result" at the end of the nodewalk
4244     % #4 = nodewalk
4245     \forest@aggregate@handler{\forest@aggregate@generic{#1}{#2}{#3}{#4}}%
4246   },
4247   /handlers/.sum/.code 2 args={% #1=forestmath, #2=nodewalk
4248     \forest@aggregate@handler{\forest@aggregate@sum{#1}{#2}}%
4249   },
4250   /handlers/.count/.code={% #1=nodewalk
4251     \forest@aggregate@handler{\forest@aggregate@count{#1}}%
4252   },
4253   /handlers/.average/.code 2 args={% #1=forestmath, #2=nodewalk
4254     \forest@aggregate@handler{\forest@aggregate@average{#1}{#2}}%
4255   },
4256   /handlers/.product/.code 2 args={% #1=forestmath, #2=nodewalk
4257     \forest@aggregate@handler{\forest@aggregate@product{#1}{#2}}%
4258   },
4259   /handlers/.min/.code 2 args={% #1=forestmath, #2=nodewalk
4260     \forest@aggregate@handler{\forest@aggregate@min{#1}{#2}}%
4261   },
4262   /handlers/.max/.code 2 args={% #1=forestmath, #2=nodewalk
4263     \forest@aggregate@handler{\forest@aggregate@max{#1}{#2}}%
4264   },
4265   declare count register={aggregate n},
4266   declare toks register={aggregate value},
4267   declare toks register={aggregate result},
4268   aggregate result={},
4269 }
4270 \def\forest@aggregate@handler#1{%
4271   \edef\forest@marshal{%
4272     \unexpanded{%
4273       #1%
4274     }%
4275     \noexpand\pgfkeysalso{\pgfkeyscurrentpath/.register=aggregate result}%
4276   }%
4277   }\forest@marshal
4278 }
4279 \def\forest@aggregate@pgfmathfunction@finish{%
4280   \forestget{aggregate result}\pgfmathresult
4281 }
4282 \pgfmathdeclarefunction{aggregate}{4}{%
4283   \forest@aggregate@generic{#1}{#2}{#3}{#4}%
4284   \forest@aggregate@pgfmathfunction@finish
4285 }
4286 \pgfmathdeclarefunction{aggregate_count}{1}{%
4287   \forest@aggregate@sum{#1}%
4288   \forest@aggregate@pgfmathfunction@finish
4289 }
4290 \pgfmathdeclarefunction{aggregate_sum}{2}{%
4291   \forest@aggregate@sum{#1}{#2}%

```

```

4292 \forest@aggregate@pgfmathfunction@finish
4293 }
4294 \pgfmathdeclarefunction{aggregate_product}{2}{%
4295 \forest@aggregate@product{#1}{#2}%
4296 \forest@aggregate@pgfmathfunction@finish
4297 }
4298 \pgfmathdeclarefunction{aggregate_average}{2}{%
4299 \forest@aggregate@average{#1}{#2}%
4300 \forest@aggregate@pgfmathfunction@finish
4301 }
4302 \pgfmathdeclarefunction{aggregate_min}{2}{%
4303 \forest@aggregate@min{#1}{#2}%
4304 \forest@aggregate@pgfmathfunction@finish
4305 }
4306 \pgfmathdeclarefunction{aggregate_max}{2}{%
4307 \forest@aggregate@max{#1}{#2}%
4308 \forest@aggregate@pgfmathfunction@finish
4309 }

Define particular aggregate functions.
4310 \def\forest@aggregate#1#2#3#4#5#6{% #1...#5=real args,
4311 % #6=what to do with |aggregate result| register
4312 % #1 = start value (forestmath)
4313 % #2 = forestmath expression that calculates "aggregate current" at each step
4314 % #3 = forestmath expression that calculates "aggregate result" at each step
4315 % #4 = forestmath expression that calculates "aggregate result" at the end of the nodewalk
4316 % #5 = nodewalk
4317 \forest@saveandrestorereregister{aggregate result}{%
4318 \forest@saveandrestorereregister{aggregate n}{%
4319 \forest@aggregate@{#1}{#2}{#3}{#4}{#5}%
4320 #6%
4321 }%
4322 }%
4323 }
4324 \def\forest@aggregate@generic#1#2#3#4{\forest@aggregate
4325 {\forestmathparse{#1}}%
4326 {}%
4327 {\forestmathparse{#2}}%
4328 {\forestmathparse{#3}}%
4329 {#4}%
4330 }
4331 \def\forest@aggregate@sum#1#2{\forest@aggregate
4332 {\forestmath@convert@fromto\forestmathtype@count\forestmathtype@generic{0}}%
4333 {\forestmathparse{#1}}%
4334 {\forestmathadd{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
4335 {\forestregister{aggregate result}\forestmathresult}%
4336 {#2}%
4337 }
4338 \def\forest@aggregate@count#1{\forest@aggregate
4339 {\def\forestmathresult{0}\let\forestmathresulttype\forestmathtype@count}%
4340 {\def\forestmathresult{1}\let\forestmathresulttype\forestmathtype@count}%
4341 {\edef\forestmathresult{\the\numexpr\forestregister{aggregate result}+1}\let\forestmathresulttype\forestmat
4342 {\forestregister{aggregate result}\forestmathresult\let\forestmathresulttype\forestmathtype@count}%
4343 {#1}%
4344 }
4345 \def\forest@aggregate@average#1#2{\forest@aggregate
4346 {\forestmath@convert@fromto\forestmathtype@count\forestmathtype@generic{0}}%
4347 {\forestmathparse{#1}}%
4348 {\forestmathadd{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
4349 {\forestmathdivideP{\forestregister{aggregate result}}{\forestregister{aggregate n}}}%
4350 {#2}%

```



```

4351 }
4352 \def\forest@aggregate@product#1#2{\forest@aggregate
4353 {\forestmath@convert@fromto\forestmath@type@count\forestmath@type@generic{1}}%
4354 {\forestmath@parse{#1}}%
4355 {\forestmath@multiply{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
4356 {\forestregister{aggregate result}\forestmathresult}%
4357 {#2}%
4358 }
4359 \def\forest@aggregate@min#1#2{\forest@aggregate
4360 {\def\forestmathresult{}}%
4361 {\forestmath@parse{#1}}%
4362 {\forestmath@min{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
4363 {\forestregister{aggregate result}\forestmathresult}%
4364 {#2}%
4365 }
4366 \def\forest@aggregate@max#1#2{\forest@aggregate
4367 {\def\forestmathresult{}}%
4368 {\forestmath@parse{#1}}%
4369 {\forestmath@max{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
4370 {\forestregister{aggregate result}\forestmathresult}%
4371 {#2}%
4372 }

```

Actual computation.

```

4373 \def\forest@aggregate@#1#2#3#4#5{%
4374   % #1 = start value (forestmath)
4375   % #2 = forestmath expression that calculates "aggregate current" at each step
4376   % #3 = forestmath expression that calculates "aggregate result" at each step
4377   % #4 = forestmath expression that calculates "aggregate result" at the end of the nodewalk
4378   % #5 = nodewalk
4379   #1%
4380   \forestregister{aggregate result}\forestmathresult
4381   \forestregister{aggregate value}{}%
4382   \forestregister{aggregate n}{0}%
4383   \forest@forthis{%
4384     \forest@nodewalk{#5}{%
4385       TeX={%
4386         \forestregister{aggregate n}{\number\numexpr\forestregister{aggregate n}+1}%
4387         #2%
4388         \forestregister{aggregate value}\forestmathresult
4389         #3%
4390         \forestregister{aggregate result}\forestmathresult
4391       }%
4392     }{}%
4393   }%
4394   #4%
4395   \let\forest@temp@pgfmathpostparse\pgfmathpostparse
4396   \let\pgfmathpostparse\forest@aggregate@pgfmathpostparse
4397   \forestmath@convert@to\forestmath@type@dimen{\forestmathresult}%
4398   \pgfmathqparse{\forestmathresult}%
4399   \let\pgfmathpostparse\forest@temp@pgfmathpostparse
4400   \forestregister{aggregate result}\pgfmathresult
4401 }

```

7.2.1 pgfmath extensions

```

4402 \pgfmathdeclarefunction{strequal}{2}{%
4403   \ifstrequal{#1}{#2}{\def\pgfmathresult{1}}{\def\pgfmathresult{0}}%
4404 }
4405 \pgfmathdeclarefunction{instr}{2}{%
4406   \pgfutil@in@{#1}{#2}%
4407   \ifpgfutil@in@\def\pgfmathresult{1}\else\def\pgfmathresult{0}\fi

```

```

4408 }
4409 \pgfmathdeclarefunction{strcat}{...}{%
4410   \edef\pgfmathresult{\forest@strip@braces{#1}}%
4411 }
4412 \pgfmathdeclarefunction{min_s}{2}{% #1 = node, #2 = context node (for growth rotation)
4413   \forest@forthis{%
4414     \forest@nameandgo{#1}%
4415     \forest@compute@minmax@ls{#2}%
4416     \edef\forest@temp{\forestove{min@s}}%
4417     \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
4418   }%
4419 }
4420 \pgfmathdeclarefunction{min_l}{2}{% #1 = node, #2 = context node (for growth rotation)
4421   \forest@forthis{%
4422     \forest@nameandgo{#1}%
4423     \forest@compute@minmax@ls{#2}%
4424     \edef\forest@temp{\forestove{min@l}}%
4425     \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
4426   }%
4427 }
4428 \pgfmathdeclarefunction{max_s}{2}{% #1 = node, #2 = context node (for growth rotation)
4429   \forest@forthis{%
4430     \forest@nameandgo{#1}%
4431     \forest@compute@minmax@ls{#2}%
4432     \edef\forest@temp{\forestove{max@s}}%
4433     \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
4434   }%
4435 }
4436 \pgfmathdeclarefunction{max_l}{2}{% #1 = node, #2 = context node (for growth rotation)
4437   \forest@forthis{%
4438     \forest@nameandgo{#1}%
4439     \forest@compute@minmax@ls{#2}%
4440     \edef\forest@temp{\forestove{max@l}}%
4441     \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
4442   }%
4443 }
4444 \def\forest@compute@minmax@ls#1{% #1 = nodewalk; in the context of which node?
4445   {%
4446     \pgftransformreset
4447     \forest@forthis{%
4448       \forest@nameandgo{#1}%
4449       \forest@pgfqtransformrotate{-\forestove{grow}}%
4450     }%
4451     \forestoget{min x}\forest@temp@minx
4452     \forestoget{min y}\forest@temp@miny
4453     \forestoget{max x}\forest@temp@maxx
4454     \forestoget{max y}\forest@temp@maxy
4455     \pgfpointransformed{\pgfqpoint{\forest@temp@minx}{\forest@temp@miny}}%
4456     \forestoeset{min@l}{\the\pgf@x}%
4457     \forestoeset{min@s}{\the\pgf@y}%
4458     \forestoeset{max@l}{\the\pgf@x}%
4459     \forestoeset{max@s}{\the\pgf@y}%
4460     \pgfpointransformed{\pgfqpoint{\forest@temp@minx}{\forest@temp@maxy}}%
4461     \ifdim\pgf@x<\forestove{min@l}\relax\forestoeset{min@l}{\the\pgf@x}\fi
4462     \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
4463     \ifdim\pgf@x>\forestove{max@l}\relax\forestoeset{max@l}{\the\pgf@x}\fi
4464     \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
4465     \pgfpointransformed{\pgfqpoint{\forest@temp@maxx}{\forest@temp@miny}}%
4466     \ifdim\pgf@x<\forestove{min@l}\relax\forestoeset{min@l}{\the\pgf@x}\fi
4467     \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
4468     \ifdim\pgf@x>\forestove{max@l}\relax\forestoeset{max@l}{\the\pgf@x}\fi

```

```

4469 \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
4470 \pgfpointtransformed{\pgfqpoint{\forest@temp@maxx}{\forest@temp@maxy}}}%
4471 \ifdim\pgf@x<\forestove{min@l}\relax\forestoeset{min@l}{\the\pgf@x}\fi
4472 \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
4473 \ifdim\pgf@x>\forestove{max@l}\relax\forestoeset{max@l}{\the\pgf@x}\fi
4474 \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
4475 % smuggle out
4476 \edef\forest@marshal{%
4477   \noexpand\forestoeset{min@l}{\forestove{min@l}}}%
4478   \noexpand\forestoeset{min@s}{\forestove{min@s}}}%
4479   \noexpand\forestoeset{max@l}{\forestove{max@l}}}%
4480   \noexpand\forestoeset{max@s}{\forestove{max@s}}}%
4481 } \expandafter
4482 } \forest@marshal
4483 }
4484 \def\forest@pgfmathhelper@attribute@toks#1#2{%
4485   \forest@forthis{%
4486     \forest@nameandgo{#1}%
4487     \ifnum\forest@cn=0
4488       \def\pgfmathresult{%
4489         \else
4490         \forestoget{#2}\pgfmathresult
4491       \fi
4492     }%
4493   }
4494 \def\forest@pgfmathhelper@attribute@dimen#1#2{%
4495   \forest@forthis{%
4496     \forest@nameandgo{#1}%
4497     \ifnum\forest@cn=0
4498       \def\pgfmathresult{0}%
4499     \else
4500       \forestoget{#2}\forest@temp
4501       \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
4502     \fi
4503   }%
4504 }
4505 \def\forest@pgfmathhelper@attribute@count#1#2{%
4506   \forest@forthis{%
4507     \forest@nameandgo{#1}%
4508     \ifnum\forest@cn=0
4509       \def\pgfmathresult{0}%
4510     \else
4511       \forestoget{#2}\pgfmathresult
4512     \fi
4513   }%
4514 }
4515 \pgfmathdeclarefunction*{id}{1}{%
4516   \forest@forthis{%
4517     \forest@nameandgo{#1}%
4518     \let\pgfmathresult\forest@cn
4519   }%
4520 }

```

7.3 Nodewalk

Setup machinery.

```

4521 \def\forest@nodewalk@n{0}
4522 \def\forest@nodewalk@historyback{0,}
4523 \def\forest@nodewalk@historyforward{0,}
4524 \def\forest@nodewalk@origin{0}
4525 \def\forest@nodewalk@config@everystep@independent@before#1{% #1 = every step keylist

```

```

4526 \forestrrset{every step}{#1}%
4527 }
4528 \def\forest@nodewalk@config@everystep@independent@after{%
4529 \noexpand\forestrrset{every step}{\forestrv{every step}}%
4530 }
4531 \def\forest@nodewalk@config@history@independent@before{%
4532 \def\forest@nodewalk@n{0}%
4533 \edef\forest@nodewalk@origin{\forest@cn}%
4534 \def\forest@nodewalk@historyback{0,}%
4535 \def\forest@nodewalk@historyforward{0,}%
4536 }
4537 \def\forest@nodewalk@config@history@independent@after{%
4538 \edef\noexpand\forest@nodewalk@n{\expandonce{\forest@nodewalk@n}}%
4539 \edef\noexpand\forest@nodewalk@origin{\expandonce{\forest@nodewalk@origin}}%
4540 \edef\noexpand\forest@nodewalk@historyback{\expandonce{\forest@nodewalk@historyback}}%
4541 \edef\noexpand\forest@nodewalk@historyforward{\expandonce{\forest@nodewalk@historyforward}}%
4542 }
4543 \def\forest@nodewalk@config@everystep@shared@before#1{% #1 = every step keylist
4544 \def\forest@nodewalk@config@everystep@shared@after{}
4545 \def\forest@nodewalk@config@history@shared@before{}
4546 \def\forest@nodewalk@config@history@shared@after{}
4547 \def\forest@nodewalk@config@everystep@inherited@before#1{% #1 = every step keylist
4548 \let\forest@nodewalk@config@everystep@inherited@after\forest@nodewalk@config@everystep@independent@after
4549 \def\forest@nodewalk@config@history@inherited@before{}
4550 \let\forest@nodewalk@config@history@inherited@after\forest@nodewalk@config@history@independent@after
4551 \def\forest@nodewalk#1#2{% #1 = nodewalk, #2 = every step keylist
4552 \forest@configured@nodewalk{independent}{independent}{inherited}{#1}{#2}%
4553 }
4554 \def\forest@configured@nodewalk#1#2#3#4#5{%
4555 % #1 = every step method, #2 = history method, #3 = on invalid
4556 % #4 = nodewalk, #5 = every step keylist
4557 \def\forest@nodewalk@config@everystep@method{#1}%
4558 \def\forest@nodewalk@config@history@method{#2}%
4559 \def\forest@nodewalk@config@oninvalid{#3}%
4560 \forest@Nodewalk{#4}{#5}%
4561 }
4562 \def\forest@nodewalk@oninvalid@inherited@text{inherited}
4563 \def\forest@Nodewalk#1#2{% #1 = nodewalk, #2 = every step keylist
4564 \ifx\forest@nodewalk@config@oninvalid\forest@nodewalk@oninvalid@inherited@text
4565 \edef\forest@nodewalk@config@oninvalid{\forest@nodewalk@oninvalid}%
4566 \fi
4567 \edef\forest@nw@marshal{%
4568 \noexpand\pgfqkeys{/forest/nodewalk}{\unexpanded{#1}}%
4569 \csname forest@nodewalk@config@everystep@\forest@nodewalk@config@everystep@method @after\endcsname
4570 \csname forest@nodewalk@config@history@\forest@nodewalk@config@history@method @after\endcsname
4571 \edef\noexpand\forest@nodewalk@oninvalid{\forest@nodewalk@oninvalid}%
4572 }%
4573 \csname forest@nodewalk@config@everystep@\forest@nodewalk@config@everystep@method @before\endcsname{#2}%
4574 \csname forest@nodewalk@config@history@\forest@nodewalk@config@history@method @before\endcsname
4575 \edef\forest@nodewalk@oninvalid{\forest@nodewalk@config@oninvalid}%
4576 \forest@saveandrestoreifcs{forest@nodewalk@fake}{%
4577 \forest@nodewalk@fakefalse
4578 \forest@nw@marshal
4579 }%
4580 }
4581 \pgfmathdeclarefunction{valid}{1}{%
4582 \forest@forthis{%
4583 \forest@nameandgo{#1}%
4584 \edef\pgfmathresult{\ifnum\forest@cn=0 0\else 1\fi}%
4585 }%
4586 }

```

```

4587 \pgfmathdeclarefunction{invalid}{1}{%
4588   \forest@forthis{%
4589     \forest@nameandgo{#1}%
4590     \edef\pgfmathresult{\ifnum\forest@cn=0 1\else 0\fi}%
4591   }%
4592 }
4593 \newif\ifforest@nodewalk@fake
4594 \def\forest@nodewalk@oninvalid{error}
4595 \def\forest@nodewalk@makestep{%
4596   \ifnum\forest@cn=0
4597     \csname forest@nodewalk@makestep@oninvalid\forest@nodewalk@oninvalid\endcsname
4598   \else
4599     \forest@nodewalk@makestep@
4600   \fi
4601 }
4602 \csdef{forest@nodewalk@makestep@oninvalid@error if real}{\ifforest@nodewalk@fake\expandafter\forest@nodewalk@
4603 \csdef{forest@nodewalk@makestep@oninvalid@last valid}{%
4604   \forest@nodewalk@tolastvalid
4605   \ifforestdebugnodewalks\forest@nodewalk@makestep@invalidtolastvalid@debug\fi}%
4606 \def\forest@nodewalk@makestep@oninvalid@error{\PackageError{forest}{nodewalk stepped to the invalid node\Mess
4607 \let\forest@nodewalk@makestep@oninvalid@fake\relax
4608 \def\forest@nodewalk@makestep@oninvalid@compatfake{%
4609   \forest@deprecated{last step in stack "\forest@nodewalk@currentstepname", which stepped on an invalid node;
4610 }%
4611 \def\forest@nodewalk@makestep@{%
4612   \ifforestdebugnodewalks\forest@nodewalk@makestep@debug\fi
4613   \ifforest@nodewalk@fake
4614   \else
4615     \edef\forest@nodewalk@n{\number\numexpr\forest@nodewalk@n+1}%
4616     \epreto\forest@nodewalk@historyback{\forest@cn,}%
4617     \def\forest@nodewalk@historyforward{0,}%
4618     \forest@process@keylist@register{every step}%
4619   \fi
4620 }
4621 \def\forest@nodewalk@makestep@debug{%
4622   \edef\forest@marshal{%
4623     \noexpand\typeout{\ifforest@nodewalk@fake fake \fi "\forest@nodewalk@currentstepname" step to node id=\fo
4624   }\forest@marshal
4625 }%
4626 \def\forest@nodewalk@makestep@invalidtolastvalid@debug{%
4627   \edef\forest@marshal{%
4628     \noexpand\typeout{\ifforest@nodewalk@fake fake \fi "\forest@nodewalk@currentstepname" step to invalid nod
4629   }\forest@marshal
4630 }%
4631 \def\forest@handlers@savecurrentpath{%
4632   \edef\pgfkeyscurrentkey{\pgfkeyscurrentpath}%
4633   \let\forest@currentkey\pgfkeyscurrentkey
4634   \pgfkeys@split@path
4635   \edef\forest@currentpath{\pgfkeyscurrentpath}%
4636   \let\forest@currentname\pgfkeyscurrentname
4637 }
4638 \pgfkeys{/handlers/save current path/.code={\forest@handlers@savecurrentpath}}
4639 \newif\ifforest@nodewalkstephandler@style
4640 \newif\ifforest@nodewalkstephandler@autostep
4641 \newif\ifforest@nodewalkstephandler@stripfakesteps
4642 \newif\ifforest@nodewalkstephandler@muststartatvalidnode
4643 \newif\ifforest@nodewalkstephandler@makefor
4644 \let\forest@nodewalkstephandler@styletrueorfalse\forest@nodewalkstephandler@stylefalse
4645 \def\forest@nodewalk@currentstepname{}
4646 \forestset{
4647   /forest/define@step/style/.is if=forest@nodewalkstephandler@style,

```

```

4648 /forest/define@step/autostep/.is if=forest@nodewalkstephandler@autostep,
4649 % the following is useful because some macros use grouping (by \forest@forthis or similar) and therefore, a
4650 /forest/define@step/strip fake steps/.is if=forest@nodewalkstephandler@stripfakesteps,
4651 % this can never happen with autosteps ...
4652 /forest/define@step/autostep/.append code={%
4653   \ifforest@nodewalkstephandler@autostep
4654   \forest@nodewalkstephandler@stripfakestepsfalse
4655   \fi
4656 },
4657 /forest/define@step/must start at valid node/.is if=forest@nodewalkstephandler@muststartatvalidnode,
4658 /forest/define@step/n args/.store in=\forest@nodewalkstephandler@nargs,
4659 /forest/define@step/make for/.is if=forest@nodewalkstephandler@makefor,
4660 /forest/define@step/@bare/.style={strip fake steps=false,must start at valid node=false,make for=false},
4661 define long step/.code n args=3{%
4662   \forest@nodewalkstephandler@styletrueorfalse % true for end users; but in the package, most of steps are
4663   \forest@nodewalkstephandler@autostepfalse
4664   \forest@nodewalkstephandler@stripfakestepstrue
4665   \forest@nodewalkstephandler@muststartatvalidnodetrue % most steps can only start at a valid node
4666   \forest@nodewalkstephandler@makefortrue % make for prefix?
4667   \def\forest@nodewalkstephandler@nargs{0}%
4668   \pgfqkeys{/forest/define@step}{#2}%
4669   \forest@temp@toks{#3}% handler code
4670   \ifforest@nodewalkstephandler@style
4671     \expandafter\forest@temp@toks\expandafter{%
4672       \expandafter\pgfqkeysalso\expandafter{\the\forest@temp@toks}%
4673     }%
4674   \fi
4675   \ifforest@nodewalkstephandler@autostep
4676     \apptotoks\forest@temp@toks{\forest@nodewalk@makestep}%
4677   \fi
4678   \ifforest@nodewalkstephandler@stripfakesteps
4679     \expandafter\forest@temp@toks\expandafter{\expandafter\forest@nodewalk@stripfakesteps\expandafter{\the\forest@temp@toks}}%
4680   \fi
4681   \ifforest@nodewalkstephandler@muststartatvalidnode
4682     \edef\forest@marshal{%
4683       \noexpand\forest@temp@toks{%
4684         \unexpanded{%
4685           \ifnum\forest@cn=0
4686             \csname forest@nodewalk@start@oninvalid@\forest@nodewalk@oninvalid\endcsname{#1}%
4687           \else
4688             }%
4689           \noexpand\@escapeif{\the\forest@temp@toks}%
4690           \noexpand\fi
4691         }%
4692       }\forest@marshal
4693     \fi
4694     \pretotoks\forest@temp@toks{\appto\forest@nodewalk@currentstepname{, #1}}%
4695     \expandafter\forest@temp@toks\expandafter{\expandafter\forest@saveandrestoremacro\expandafter\forest@node
4696     \ifforestdebugnodewalks
4697       \epretotoks\forest@temp@toks{\noexpand\typeout{Starting step "#1" from id=\noexpand\forest@cn
4698         \ifnum\forest@nodewalkstephandler@nargs>0 \space with args \noexpand\unexpanded{###1}\fi
4699         \ifnum\forest@nodewalkstephandler@nargs>1 ,\noexpand\unexpanded{###2}\fi
4700         \ifnum\forest@nodewalkstephandler@nargs>2 ,\noexpand\unexpanded{###3}\fi
4701         \ifnum\forest@nodewalkstephandler@nargs>3 ,\noexpand\unexpanded{###4}\fi
4702         \ifnum\forest@nodewalkstephandler@nargs>4 ,\noexpand\unexpanded{###5}\fi
4703         \ifnum\forest@nodewalkstephandler@nargs>5 ,\noexpand\unexpanded{###6}\fi
4704         \ifnum\forest@nodewalkstephandler@nargs>6 ,\noexpand\unexpanded{###7}\fi
4705         \ifnum\forest@nodewalkstephandler@nargs>7 ,\noexpand\unexpanded{###8}\fi
4706         \ifnum\forest@nodewalkstephandler@nargs>8 ,\noexpand\unexpanded{###9}\fi
4707       }}%
4708   \fi

```

```

4709 \def\forest@temp{/forest/nodewalk/#1/.code}%
4710 \ifnum\forest@nodewalkstephandler@nargs<2
4711 \eappto\forest@temp{=}
4712 \else\ifnum\forest@nodewalkstephandler@nargs=2
4713 \eappto\forest@temp{ 2 args=}
4714 \else
4715 \eappto\forest@temp{ n args={\forest@nodewalkstephandler@nargs}}
4716 \fi\fi
4717 \eappto\forest@temp{{\the\forest@temp@toks}}
4718 \expandafter\pgfkeysalso\expandafter{\forest@temp}%
4719 \ifforest@nodewalkstephandler@makefor
4720 \ifnum\forest@nodewalkstephandler@nargs=0
4721 \forestset%
4722 for #1/.code={\forest@forstepwrapper{#1}{##1}},
4723 }%
4724 \else\ifnum\forest@nodewalkstephandler@nargs=1
4725 \forestset%
4726 for #1/.code 2 args={\forest@forstepwrapper{#1={##1}}{##2}},
4727 }%
4728 \else
4729 \forestset%
4730 for #1/.code n args/.expanded=%
4731 {\number\numexpr\forest@nodewalkstephandler@nargs+1}%
4732 {\noexpand\forest@forstepwrapper{#1\ifnum\forest@nodewalkstephandler@nargs>0=\fi\forest@util@narg
4733 }}%
4734 \fi\fi
4735 \fi
4736 },
4737 }
4738 {\csname forest@@doc@hook@bigbadforlist\endcsname}%
4739 \pgfqkeys{/handlers}{
4740 .nodewalk style/.code={\forest@handlers@savecurrentpath\pgfkeysalso%
4741 \forest@currentpath/nodewalk/\forest@currentname/.style={#1}%
4742 }},
4743 }

```

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

```

4744 \def\forest@forstepwrapper#1#2{\forest@forthis{\forest@nodewalk{#1}{#2}}}
4745 \def\forest@util@nargs#1#2#3{% #1 = prefix (#, ##, ...), #2 = n args, #3=start; returns {#start+1}...{#start+
4746 \ifnum#2>0 {#1\number\numexpr#3+1}\fi
4747 \ifnum#2>1 {#1\number\numexpr#3+2}\fi
4748 \ifnum#2>2 {#1\number\numexpr#3+3}\fi
4749 \ifnum#2>3 {#1\number\numexpr#3+4}\fi
4750 \ifnum#2>4 {#1\number\numexpr#3+5}\fi
4751 \ifnum#2>5 {#1\number\numexpr#3+6}\fi
4752 \ifnum#2>6 {#1\number\numexpr#3+7}\fi
4753 \ifnum#2>7 {#1\number\numexpr#3+8}\fi
4754 \ifnum#2>8 {#1\number\numexpr#3+9}\fi
4755 }
4756 \def\forest@nodewalk@start@oninvalid@fake#1{}
4757 \def\forest@nodewalk@start@oninvalid@compatfake#1{%
4758 \forest@deprecated{last step in stack "\forest@nodewalk@currentstepname", which started from an invalid nod
4759 }%
4760 \let\forest@nodewalk@start@oninvalid@errorifreal\forest@nodewalk@start@oninvalid@fake % the step will be to a
4761 \let\forest@nodewalk@start@oninvalid@lastvalid\forest@nodewalk@start@oninvalid@fake
4762 \def\forest@nodewalk@start@oninvalid@error#1{\PackageError{forest}{nodewalk step "#1" cannot start at the inv

```

Define long-form single-step walks.

```

4763 \forestset{
4764 define long step={current}{autostep}{},

```

```

4765 define long step={next}{autostep}{\edef\forest@cn{\forestove{@next}}},
4766 define long step={previous}{autostep}{\edef\forest@cn{\forestove{@previous}}},
4767 define long step={parent}{autostep}{\edef\forest@cn{\forestove{@parent}}},
4768 define long step={first}{autostep}{\edef\forest@cn{\forestove{@first}}},
4769 define long step={last}{autostep}{\edef\forest@cn{\forestove{@last}}},
4770 define long step={sibling}{autostep}{%
4771   \edef\forest@cn{%
4772     \ifnum\forestove{@previous}=0
4773       \forestove{@next}%
4774     \else
4775       \forestove{@previous}%
4776     \fi
4777   }%
4778 },
4779 define long step={next node}{autostep}{\edef\forest@cn{\forest@node@linearnextid}},
4780 define long step={previous node}{autostep}{\edef\forest@cn{\forest@node@linearpreviousid}},
4781 define long step={first leaf}{autostep}{%
4782   \safeloop
4783   \edef\forest@cn{\forestove{@first}}%
4784   \unless\ifnum\forestove{@first}=0
4785     \saferepeat
4786   },
4787 define long step={first leaf'}{autostep}{%
4788   \safeloop
4789   \unless\ifnum\forestove{@first}=0
4790     \edef\forest@cn{\forestove{@first}}%
4791     \saferepeat
4792   },
4793 define long step={last leaf}{autostep}{%
4794   \safeloop
4795   \edef\forest@cn{\forestove{@last}}%
4796   \unless\ifnum\forestove{@last}=0
4797     \saferepeat
4798   },
4799 define long step={last leaf'}{autostep}{%
4800   \safeloop
4801   \unless\ifnum\forestove{@last}=0
4802     \edef\forest@cn{\forestove{@last}}%
4803     \saferepeat
4804   },
4805 define long step={next leaf}{style,strip fake steps=false}{group={do until={n_children()==0}{next node}}},
4806 define long step={previous leaf}{style,strip fake steps=false}{group={do until={n_children()==0}{previous node}}},
4807 define long step={next on tier}{autostep,n args=1}{%
4808   \def\forest@temp{#1}%
4809   \ifx\forest@temp\pgfkeysnovalue@text
4810     \forestoget{tier}\forest@nodewalk@giventier
4811   \else
4812     \def\forest@nodewalk@giventier{#1}%
4813   \fi
4814   \edef\forest@cn{\forest@node@linearnextid}%
4815   \safeloop
4816   \forest@nodewalk@gettier
4817   \ifforest@temp
4818     \edef\forest@cn{\forest@node@linearnextid}%
4819   \saferepeat
4820   },
4821 define long step={previous on tier}{autostep,n args=1}{%
4822   \def\forest@temp{#1}%
4823   \ifx\forest@temp\pgfkeysnovalue@text
4824     \forestoget{tier}\forest@nodewalk@giventier
4825   \else

```



```

4826     \def\forest@nodewalk@giventier{#1}%
4827     \fi
4828     \safeloop
4829     \edef\forest@cn{\forest@node@linearpreviousid}%
4830     \forest@nodewalk@gettier
4831     \ifforest@temp
4832     \saferepeat
4833   },
4834   TeX={%
4835     \def\forest@nodewalk@gettier{%
4836       \ifnum\forest@cn=0
4837         \forest@tempfalse
4838       \else
4839         \foresttoget{tier}\forest@temp
4840         \ifx\forest@temp\forest@nodewalk@giventier
4841           \forest@tempfalse
4842         \else
4843           \forest@temptrue
4844         \fi
4845       \fi
4846     }%
4847   },
4848   %
4849   define long step={root}{autostep,must start at valid node=false}{%
4850     \edef\forest@cn{\forest@node@rootid}},
4851   define long step={root'}{autostep,must start at valid node=false}{%
4852     \forestOifdefined{\forest@root}{@parent}\edef\forest@cn{\forest@root}}{\edef\forest@cn{0}}%
4853   },
4854   define long step={origin}{autostep,must start at valid node=false}{\edef\forest@cn{\forest@nodewalk@origin}}%
4855   %
4856   define long step={n}{autostep,n args=1}{%
4857     \forestmathtruncatemacro\forest@temp@n{#1}%
4858     \edef\forest@cn{\forest@node@nthchildid{\forest@temp@n}}%
4859   },
4860   define long step={n}{autostep,make for=false,n args=1}{%
4861     % Yes, twice. ;- )
4862     % n=1 and n(ext)
4863     \def\forest@nodewalk@temp{#1}%
4864     \ifx\forest@nodewalk@temp\pgfkeysnovalue@text
4865       \edef\forest@cn{\forestove{@next}}%
4866     \else
4867       \forestmathtruncatemacro\forest@temp@n{#1}%
4868       \edef\forest@cn{\forest@node@nthchildid{\forest@temp@n}}%
4869     \fi
4870   },
4871   define long step={n'}{autostep,n args=1}{%
4872     \forestmathtruncatemacro\forest@temp@n{#1}%
4873     \edef\forest@cn{\forest@node@nbarthchildid{\forest@temp@n}}%
4874   },
4875   define long step={to tier}{autostep,n args=1}{%
4876     \def\forest@nodewalk@giventier{#1}%
4877     \safeloop
4878     \forest@nodewalk@gettier
4879     \ifforest@temp
4880     \foresttoget{@parent}\forest@cn
4881     \saferepeat
4882   },
4883   %
4884   define long step={name}{autostep,n args=1,must start at valid node=false}{%
4885     \edef\forest@cn{%
4886       \forest@node@ifnamedefined{#1}{\forest@node@Nametoid{#1}}{0}}%

```

```

4887     }%
4888 },
4889 define long step={id}{autostep,n args=1,must start at valid node=false}{%
4890     \forestOifdefined{#1}{@parent}{\edef\forest@cn{#1}}{\edef\forest@cn{0}}%
4891 },
4892 define long step={Nodewalk}{n args=3,@bare}{% #1 = config, #2 = nodewalk
4893     \def\forest@nodewalk@config@everystep@method{independent}%
4894     \def\forest@nodewalk@config@history@method{shared}%
4895     \def\forest@nodewalk@config@oninvalid{inherited}%
4896     \pgfqkeys{/forest/nodewalk@config}{#1}%
4897     \forest@Nodewalk{#2}{#3}%
4898 },
4899 define long step={nodewalk}{n args=2,@bare}{% #1 = nodewalk, #2 = every step
4900     \forest@nodewalk{#1}{#2}%
4901 },
4902 define long step={nodewalk'}{n args=1,@bare}{% #1 = nodewalk
4903     \forest@configured@nodewalk{inherited}{independent}{inherited}{#1}{}%
4904 },
4905 % these "for ..." keys must be defined explicitly
4906 % (and copied into node keyspace manually),
4907 % as prefix "for" normally introduces the every-step keylist
4908 define long step={for nodewalk}{n args=2,@bare}{% #1 = nodewalk, #2 = every step
4909     \forest@forthis{\forest@nodewalk{#1}{#2}}},
4910 define long step={for nodewalk'}{n args=1,@bare}{% #1 = nodewalk
4911     \forest@forthis{%
4912         \forest@configured@nodewalk{inherited}{independent}{inherited}{#1}{}%
4913     }%
4914 },
4915 define long step={for Nodewalk}{n args=3,@bare}{% #1 = config, #2 = nodewalk, #3 = every-step
4916     \def\forest@nodewalk@config@everystep@method{independent}%
4917     \def\forest@nodewalk@config@history@method{shared}%
4918     \def\forest@nodewalk@config@oninvalid{inherited}%
4919     \pgfqkeys{/forest/nodewalk@config}{#1}%
4920     \forest@forthis{\forest@Nodewalk{#2}{#3}}%
4921 },
4922 copy command key={/forest/nodewalk/Nodewalk}{/forest/Nodewalk},
4923 copy command key={/forest/nodewalk/for nodewalk}{/forest/for nodewalk},
4924 copy command key={/forest/nodewalk/for Nodewalk}{/forest/for Nodewalk},
4925 declare keylist register=every step,
4926 every step'={},
4927 %%% begin nodewalk config
4928 nodewalk@config/.cd,
4929 every@step/.is choice,
4930 every@step/independent/.code={},
4931 every@step/inherited/.code={},
4932 every@step/shared/.code={},
4933 every step/.store in=\forest@nodewalk@config@everystep@method,
4934 every step/.prefix style={every@step=#1},
4935 @history/.is choice,
4936 @history/independent/.code={},
4937 @history/inherited/.code={},
4938 @history/shared/.code={},
4939 history/.store in=\forest@nodewalk@config@history@method,
4940 history/.prefix style={@history=#1},
4941 on@invalid/.is choice,
4942 on@invalid/error/.code={},
4943 on@invalid/fake/.code={},
4944 on@invalid/error if real/.code={},
4945 on@invalid/last valid/.code={},
4946 on@invalid/inherited/.code={},
4947 on invalid/.store in=\forest@nodewalk@config@oninvalid,

```

```

4948 on invalid/.prefix style={on@invalid=#1},
4949 %%% end nodewalk config
4950 }
4951 \newtoks\forest@nodewalk@branch@toks
4952 \forestset{
4953 declare toks register=branch@temp@toks,
4954 branch@temp@toks={},
4955 declare keylist register=branched@nodewalk,
4956 branched@nodewalk={},
4957 define long step={branch}{n args=1,@bare,make for,style}{@branch={#1}{branch@build@realstep,branch@build@fa
4958 define long step={branch'}{n args=1,@bare,make for,style}{@branch={#1}{branch@build@realstep}},
4959 @branch/.style 2 args={%
4960 save and restore register={branched@nodewalk}{
4961 branch@temp@toks={},
4962 split/.process={r}{#1}{,}{#2},
4963 also/.register=branch@temp@toks,
4964 also/.register=branched@nodewalk,
4965 }
4966 },
4967 nodewalk/branch@build@realstep/.style={% #1 = nodewalk for this branch
4968 branch@temp@toks/.expanded={for nodewalk={\unexpanded{#1}}{
4969 branched@nodewalk+/.expanded={id=\noexpand\forestoption{id}},
4970 \forestregister{branch@temp@toks}}},
4971 },
4972 nodewalk/branch@build@fakestep/.style={% #1 = nodewalk for this branch
4973 branch@temp@toks/.expanded={for nodewalk={\unexpanded{#1}}{
4974 \forestregister{branch@temp@toks}}},
4975 },
4976 define long step={group}{autostep,n args=1}{\forest@go{#1}},
4977 nodewalk/fake/.code={%
4978 \forest@saveandrestoreifcs{forest@nodewalk@fake}{%
4979 \forest@nodewalk@faketrue
4980 \pgfkeysalso{#1}%
4981 }%
4982 },
4983 nodewalk/real/.code={%
4984 \forest@saveandrestoreifcs{forest@nodewalk@fake}{%
4985 \forest@nodewalk@fakefalse
4986 \pgfkeysalso{#1}%
4987 }%
4988 },
4989 declare keylist register=filtered@nodewalk,
4990 filtered@nodewalk={},
4991 define long step={filter}{n args=2,@bare,make for,style}{% #1 = nodewalk, #2 = condition
4992 save and restore register={filtered@nodewalk}{
4993 filtered@nodewalk'={},
4994 Nodewalk=%
4995 {history=inherited}%
4996 {#1}%
4997 {if={#2}{filtered@nodewalk+/.expanded={id=\forestoption{id}}}{}},
4998 filtered@nodewalk@style/.style/.register=filtered@nodewalk,
4999 filtered@nodewalk@style
5000 },
5001 },
5002 on@invalid/.is choice,
5003 on@invalid/error/.code={},
5004 on@invalid/fake/.code={},
5005 on@invalid/error if real/.code={},
5006 on@invalid/last valid/.code={},
5007 on invalid/.code 2 args={%
5008 \pgfkeysalso{/forest/on@invalid={#1}}%

```

```

5009 \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
5010 \def\forest@nodewalk@oninvalid{#1}%
5011 \pgfkeysalso{#2}%
5012 }%
5013 },
5014 define long step={strip fake steps}{n args=1,@bare}{%
5015 \forest@nodewalk@stripfakesteps{\pgfkeysalso{#1}}},
5016 define long step={unique}{n args=1}{%
5017 \begingroup
5018 \def\forest@nodewalk@unique@temp{}%
5019 \forest@nodewalk{#1}{%
5020 TeX={%
5021 \foresttoget{unique@visited}\forest@temp
5022 \ifx\forest@temp\relax
5023 \forestoset{unique@visited}{1}%
5024 \eappto\forest@nodewalk@unique@temp{id=\forest@cn}%
5025 \fi
5026 }%
5027 }%
5028 \global\let\forest@global@temp\forest@nodewalk@unique@temp
5029 \endgroup
5030 \pgfkeysalsofrom{\forest@global@temp}%
5031 },
5032 define long step={walk back}{n args=1,@bare}{%
5033 \forestmathtruncatemacro\forest@temp@n{#1}%
5034 \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\ifnum\forest@cn=0}{\forest@temp@n-1}}
5035 \forest@nodewalk@back@updatehistory
5036 },
5037 nodewalk/walk back/.default=1,
5038 define long step={jump back}{n args=1,@bare}{%
5039 \forestmathtruncatemacro\forest@temp@n{(#1)+\ifnum\forest@cn=0 0\else 1\fi}%
5040 \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\forest@temp@n-1}}
5041 \forest@nodewalk@back@updatehistory
5042 },
5043 nodewalk/jump back/.default=1,
5044 define long step={back}{n args=1,@bare}{%
5045 \forestmathtruncatemacro\forest@temp@n{#1}%
5046 \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\ifnum\forest@cn=0}{\forest@temp@n-1}}
5047 \forest@nodewalk@back@updatehistory
5048 },
5049 nodewalk/back/.default=1,
5050 define long step={walk forward}{n args=1,@bare}{%
5051 \forestmathtruncatemacro\forest@temp@n{#1}%
5052 \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{0}{\forest@temp@n-1}}
5053 \forest@nodewalk@forward@updatehistory
5054 },
5055 nodewalk/walk forward/.default=1,
5056 define long step={jump forward}{n args=1,@bare}{%
5057 \forestmathtruncatemacro\forest@temp@n{#1}%
5058 \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{\forest@temp@n-1}}
5059 \forest@nodewalk@forward@updatehistory
5060 },
5061 nodewalk/jump forward/.default=1,
5062 define long step={forward}{n args=1,@bare}{%
5063 \forestmathtruncatemacro\forest@temp@n{#1}%
5064 \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{0}{\forest@temp@n-1}}
5065 \forest@nodewalk@forward@updatehistory
5066 },
5067 nodewalk/forward/.default=1,
5068 define long step={last valid'}{@bare}{%
5069 \ifnum\forest@cn=0

```

```

5070     \forest@nodewalk@tolastvalid
5071     \forest@nodewalk@makestep
5072     \fi
5073   },
5074   define long step={last valid}{@bare}{%
5075     \forest@nodewalk@tolastvalid
5076   },
5077   define long step={reverse}{n args=1,@bare,make for}{%
5078     \forest@nodewalk{#1,TeX={%
5079       \global\let\forest@global@temp\forest@nodewalk@historyback
5080       \global\let\forest@global@tempn\forest@nodewalk@n
5081     }}{%
5082     \forest@nodewalk@walklist}{\forest@global@temp}{0}{\forest@global@tempn}{\let\forest@cn\forest@nodewalk@
5083   },
5084   define long step={walk and reverse}{n args=1,@bare,make for}{%
5085     \edef\forest@marshal{%
5086       \noexpand\pgfkeysalso{\unexpanded{#1}}%
5087       \noexpand\forest@nodewalk@walklist}{\noexpand\forest@nodewalk@historyback}{0}{\noexpand\forest@nodewalk@
5088     }\forest@marshal
5089   },
5090   define long step={sort}{n args=1,@bare,make for}{%
5091     \forest@nodewalk{#1,TeX={%
5092       \global\let\forest@global@temp\forest@nodewalk@historyback
5093       \global\let\forest@global@tempn\forest@nodewalk@n
5094     }}{%
5095     \forest@nodewalk@sortlist{\forest@global@temp}{\forest@global@tempn}\forest@sort@ascending
5096   },
5097   define long step={sort'}{n args=1,@bare,make for}{%
5098     \forest@nodewalk{#1,TeX={%
5099       \global\let\forest@global@temp\forest@nodewalk@historyback
5100       \global\let\forest@global@tempn\forest@nodewalk@n
5101     }}{%
5102     \forest@nodewalk@sortlist{\forest@global@temp}{\forest@global@tempn}\forest@sort@descending
5103   },
5104   define long step={walk and sort}{n args=1,@bare,make for}{% walk as given, then walk sorted
5105     \edef\forest@marshal{%
5106       \noexpand\pgfkeysalso{\unexpanded{#1}}%
5107       \noexpand\forest@nodewalk@sortlist{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-
5108     }\forest@marshal
5109   },
5110   define long step={walk and sort'}{n args=1,@bare,make for}{%
5111     \edef\forest@marshal{%
5112       \noexpand\pgfkeysalso{\unexpanded{#1}}%
5113       \noexpand\forest@nodewalk@sortlist{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-
5114     }\forest@marshal
5115   },
5116   declare keylist register=sort by,
5117   copy command key={/forest/sort by'}{/forest/sort by},
5118   sort by={},
5119   define long step={save}{n args=2,@bare,make for}{% #1 = name, #2 = nodewalk
5120     \forest@forthis{%
5121       \forest@nodewalk{#2,TeX={%
5122         \global\let\forest@global@temp\forest@nodewalk@historyback
5123         \global\let\forest@global@tempn\forest@nodewalk@n
5124       }}{%
5125     }%
5126     \forest@nodewalk@walklist}{\forest@global@temp}{0}{\forest@global@tempn}\relax
5127     \csedef{forest@nodewalk@sav#1}{\forest@nodewalk@walklist@walked}%
5128   },
5129   define long step={walk and save}{n args=2,@bare,make for}{% #1 = name, #2 = nodewalk
5130     \edef\forest@marshal{%

```

```

5131     \noexpand\pgfkeysalso{\unexpanded{#2}}%
5132     \noexpand\forest@nodewalk@walklist}{\noexpand\forest@nodewalk@historyback}{0}{\noexpand\forest@nodewalk
5133 } \forest@marshal
5134     \csedef{forest@nodewalk@samed@#1}{\forest@nodewalk@walklist@walked}%
5135 },
5136 define long step={save append}{style,n args=2,@bare,make for}{% #1 = nodewalk name, #2 = nodewalk
5137 save@append@prepend={#1}{#2}{save}{\cseappto}},
5138 define long step={save prepend}{style,n args=2,@bare,make for}{% #1 = nodewalk name, #2 = nodewalk
5139 save@append@prepend={#1}{#2}{save}{\csepreto}},
5140 define long step={walk and save append}{style,n args=2,@bare,make for}{% #1 = nodewalk name, #2 = nodewalk
5141 save@append@prepend={#1}{#2}{walk and save}{\cseappto}},
5142 define long step={walk and save prepend}{style,n args=2,@bare,make for}{% #1 = nodewalk name, #2 = nodewalk
5143 save@append@prepend={#1}{#2}{walk and save}{\csepreto}},
5144 nodewalk/save@append@prepend/.code n args=4{%
5145 % #1 = nodewalk name, #2 = nodewalk
5146 % #3 = "(walk and) save" #4 = \cseappto/\csepreto
5147 \pgfkeysalso{#3={@temp}{#2}}%
5148 \letcs\forest@temp{forest@nodewalk@samed@temp}%
5149 #4{forest@nodewalk@samed@#1}{\expandonce{forest@temp}}%
5150 },
5151 nodewalk/save history/.code 2 args={% #1 = back, forward
5152 \csedef{forest@nodewalk@samed@#1}{\forest@nodewalk@historyback}%
5153 \csedef{forest@nodewalk@samed@#2}{\forest@nodewalk@historyforward}%
5154 },
5155 define long step={load}{n args=1,@bare,make for}{%
5156 \forest@nodewalk@walklist}{\csuse{forest@nodewalk@samed@#1}0,}{0}{-1}{\ifnum\forest@nodewalk@cn=0 \else\
5157 },
5158 if in saved nodewalk/.code n args=4{% is node #1 in nodewalk #2; yes: #3, no: #4
5159 \forest@forthis{%
5160 \forest@go{#1}%
5161 \edef\forest@marshal{%
5162 \noexpand\pgfutil@in@{\forest@cn,}{\csuse{forest@nodewalk@samed@#2},}%
5163 } \forest@marshal
5164 }%
5165 \ifpgfutil@in@
5166 \@escapeif{\pgfkeysalso{#3}}%
5167 \else
5168 \@escapeif{\pgfkeysalso{#4}}%
5169 \fi
5170 },
5171 where in saved nodewalk/.style n args=4{
5172 for tree={if in saved nodewalk={#1}{#2}{#3}{#4}}
5173 },
5174 nodewalk/options/.code={\forestset{#1}},
5175 nodewalk/TeX/.code={#1},
5176 nodewalk/TeX'/.code={\appto\forest@externalize@loadimages{#1}{#1},
5177 nodewalk/TeX''/.code={\appto\forest@externalize@loadimages{#1}},
5178 nodewalk/typeout/.style={TeX={\typeout{#1}}},
5179 % repeat is taken later from /forest/repeat
5180 }
5181 \def\forest@nodewalk@walklist#1#2#3#4#5{%
5182 % #1 = list of preceding, #2 = list to walk
5183 % #3 = from, #4 = to
5184 % #5 = every step code
5185 \let\forest@nodewalk@cn\forest@cn
5186 \edef\forest@marshal{%
5187 \noexpand\forest@nodewalk@walklist@{#1}{#2}{\number\numexpr#3}{\number\numexpr#4}{1}{0}{\unexpanded{#5}}%
5188 } \forest@marshal
5189 }
5190 \def\forest@nodewalk@walklist@#1#2#3#4#5#6#7{%
5191 % #1 = list of walked, #2 = list to walk

```

```

5192 % #3 = from, #4 = to
5193 % #5 = current step n, #6 = steps made
5194 % #7 = every step code
5195 \def\forest@nodewalk@walklist@walked{#1}%
5196 \def\forest@nodewalk@walklist@rest{#2}%
5197 \edef\forest@nodewalk@walklist@stepsmade{#6}%
5198 \ifnum#4<0
5199   \forest@temptrue
5200 \else
5201   \ifnum#5>#4\relax
5202     \forest@tempfalse
5203   \else
5204     \forest@temptrue
5205   \fi
5206 \fi
5207 \ifforest@temp
5208   \edef\forest@nodewalk@cn{\forest@csvlist@getfirst@{#2}}%
5209   \ifnum\forest@nodewalk@cn=0
5210     #7%
5211   \else
5212     \ifnum#5>#3\relax
5213       #7%
5214       \edef\forest@nodewalk@walklist@stepsmade{\number\numexpr#6+1}%
5215     \fi
5216     \forest@csvlist@getfirstrest@{#2}\forest@nodewalk@cn\forest@nodewalk@walklist@rest
5217     \@escapeifif{%
5218       \edef\forest@marshal{%
5219         \noexpand\forest@nodewalk@walklist@
5220         {\forest@nodewalk@cn,#1}{\forest@nodewalk@walklist@rest}{#3}{#4}{\number\numexpr#5+1}{\forest@nod
5221         }\forest@marshal
5222       }%
5223     \fi
5224 \fi
5225 }
5226
5227 \def\forest@nodewalk@back@updatehistory{%
5228   \ifnum\forest@cn=0
5229     \let\forest@nodewalk@historyback\forest@nodewalk@walklist@rest
5230     \let\forest@nodewalk@historyforward\forest@nodewalk@walklist@walked
5231   \else
5232     \expandafter\forest@csvlist@getfirstrest@\expandafter{\forest@nodewalk@walklist@walked}\forest@temp\forest
5233     \edef\forest@nodewalk@historyback{\forest@temp,\forest@nodewalk@walklist@rest}%
5234   \fi
5235 }
5236 \def\forest@nodewalk@forward@updatehistory{%
5237   \let\forest@nodewalk@historyforward\forest@nodewalk@walklist@rest
5238   \let\forest@nodewalk@historyback\forest@nodewalk@walklist@walked
5239 }
5240 \def\forest@go#1{%
5241   \forest@configured@nodewalk{independent}{inherited}{#1}{}%
5242 }
5243 \def\forest@csvlist@getfirst@#1{% assuming that the list is nonempty and finishes with a comma
5244   \forest@csvlist@getfirst@@#1\forest@csvlist@getfirst@@}
5245 \def\forest@csvlist@getfirst@@#1,#2\forest@csvlist@getfirst@@{#1}
5246 \def\forest@csvlist@getrest@#1{% assuming that the list is nonempty and finishes with a comma
5247   \forest@csvlist@getrest@@#1\forest@csvlist@getrest@@}
5248 \def\forest@csvlist@getrest@@#1,#2\forest@csvlist@getrest@@{#2}
5249 \def\forest@csvlist@getfirstrest@#1#2#3{% assuming that the list is nonempty and finishes with a comma
5250   % #1 = list, #2 = cs receiving first, #3 = cs receiving rest
5251   \forest@csvlist@getfirstrest@@#1\forest@csvlist@getfirstrest@@{#2}{#3}}
5252 \def\forest@csvlist@getfirstrest@@#1,#2\forest@csvlist@getfirstrest@@#3#4{%

```

```

5253 \def#3{#1}%
5254 \def#4{#2}%
5255 }
5256 \def\forest@nodewalk@stripfakesteps#1{%
5257 % go to the last valid node if the walk contained any nodes, otherwise restore the current node
5258 \edef\forest@marshal{%
5259 \unexpanded{#1}%
5260 \noexpand\ifnum\noexpand\forest@nodewalk@n=\forest@nodewalk@n\relax
5261 \def\noexpand\forest@cn{\forest@cn}%
5262 \noexpand\else
5263 \unexpanded{%
5264 \edef\forest@cn{%
5265 \expandafter\forest@csvglist@getfirst@\expandafter{\forest@nodewalk@historyback}%
5266 }%
5267 }%
5268 \noexpand\fi
5269 }\forest@marshal
5270 }
5271 \def\forest@nodewalk@tolastvalid{%
5272 \ifnum\forest@cn=0
5273 \edef\forest@cn{\expandafter\forest@csvglist@getfirst@\expandafter{\forest@nodewalk@historyback}}%
5274 \ifnum\forest@cn=0
5275 \let\forest@cn\forest@nodewalk@origin
5276 \fi
5277 \fi
5278 }
5279 \def\forest@nodewalk@sortlist#1#2#3{#1=list,#2=to,#3=asc/desc
5280 \edef\forest@nodewalksort@list{#1}%
5281 \expandafter\forest@nodewalk@sortlist@\expandafter{\number\numexpr#2}{#3}%
5282 }
5283 \def\forest@nodewalk@sortlist@#1#2{#1=to,#2=asc/desc
5284 \safeloop
5285 \unless\ifnum\safeloopn>#1\relax
5286 \expandafter\forest@csvglist@getfirstrest@\expandafter{\forest@nodewalksort@list}\forest@nodewalksort@cn\fi
5287 \csedef{forest@nodesort@\safeloopn}{\forest@nodewalksort@cn}%
5288 \saferepeat
5289 \forestrget{sort by}\forest@nodesort@sortkey
5290 \forest@sort\forest@nodesort@cmpnodes\forest@nodesort@let#2{1}{#1}%
5291 \def\forest@nodewalksort@sorted{}%
5292 \safeloop
5293 \unless\ifnum\safeloopn>#1\relax
5294 \edef\forest@cn{\csname forest@nodesort@\safeloopn\endcsname}%
5295 \forest@nodewalk@makestep
5296 \saferepeat
5297 }

```

Find minimal/maximal node in a walk.

```

5298 \forestset{
5299 define long step={min}{n args=1,@bare,make for}{% the first min in the argument nodewalk
5300 \forest@nodewalk{#1,TeX={%
5301 \global\let\forest@global@temp\forest@nodewalk@historyback
5302 }}}%
5303 \forest@nodewalk@minmax{\forest@global@temp}{-1}{<}{\forest@nodewalk@minmax@node,}%
5304 },
5305 define long step={mins}{n args=1,@bare,make for}{% all mins in the argument nodewalk
5306 \forest@nodewalk{#1,TeX={%
5307 \global\let\forest@global@temp\forest@nodewalk@historyback
5308 }}}%
5309 \forest@nodewalk@minmax{\forest@global@temp}{-1}{<}{\forest@nodewalk@minmax@nodes}%
5310 },
5311 define long step={walk and min}{n args=1,@bare}{%

```



```

5312 \edef\forest@marshal{%
5313   \noexpand\pgfkeysalso{\unexpanded{#1}}%
5314   \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
5315 } \forest@marshal
5316 },
5317 define long step={walk and mins}{n args=1,@bare}{%
5318   \edef\forest@marshal{%
5319     \noexpand\pgfkeysalso{\unexpanded{#1}}%
5320     \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
5321 } \forest@marshal
5322 },
5323 define long step={min in nodewalk}{@bare}{% find the first min in the preceding nodewalk, step to it
5324   \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{\forest@nodewalk@minmax@node,}%
5325 },
5326 define long step={mins in nodewalk}{@bare}{% find mins in the preceding nodewalk, step to mins
5327   \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{\forest@nodewalk@minmax@nodes}%
5328 },
5329 define long step={min in nodewalk'}{@bare}{% find the first min in the preceding nodewalk, step to min in h
5330   \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{}%
5331 },
5332 %
5333 define long step={max}{n args=1,@bare,make for}{% the first max in the argument nodewalk
5334   \forest@nodewalk{#1,TeX={%
5335     \global\let\forest@global@temp\forest@nodewalk@historyback
5336   }}{}%
5337   \forest@nodewalk@minmax{\forest@global@temp}{-1}{>}{\forest@nodewalk@minmax@node,}%
5338 },
5339 define long step={maxs}{n args=1,@bare,make for}{% all maxs in the argument nodewalk
5340   \forest@nodewalk{#1,TeX={%
5341     \global\let\forest@global@temp\forest@nodewalk@historyback
5342   }}{}%
5343   \forest@nodewalk@minmax{\forest@global@temp}{-1}{>}{\forest@nodewalk@minmax@nodes}%
5344 },
5345 define long step={walk and max}{n args=1,@bare}{%
5346   \edef\forest@marshal{%
5347     \noexpand\pgfkeysalso{\unexpanded{#1}}%
5348     \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
5349 } \forest@marshal
5350 },
5351 define long step={walk and maxs}{n args=1,@bare}{%
5352   \edef\forest@marshal{%
5353     \noexpand\pgfkeysalso{\unexpanded{#1}}%
5354     \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
5355 } \forest@marshal
5356 },
5357 define long step={max in nodewalk}{@bare}{% find the first max in the preceding nodewalk, step to it
5358   \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{\forest@nodewalk@minmax@node,}%
5359 },
5360 define long step={maxs in nodewalk}{@bare}{% find maxs in the preceding nodewalk, step to maxs
5361   \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{\forest@nodewalk@minmax@nodes}%
5362 },
5363 define long step={max in nodewalk'}{@bare}{% find the first max in the preceding nodewalk, step to max in h
5364   \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{}%
5365 },
5366 }
5367
5368 \def\forest@nodewalk@minmax#1#2#3#4{%
5369   % #1 = list of nodes
5370   % #2 = max index in list (start with 1)
5371   % #3 = min/max = ascending/descending = </>
5372   % #4 = how many results? 1 = {\forest@nodewalk@minmax@node,}, all={\forest@nodewalk@minmax@nodes}, walk in

```

```

5373 \forestrget{sort by}\forest@nodesort@sortkey
5374 \edef\forest@nodewalk@minmax@N{\number\numexpr#2}%
5375 \edef\forest@nodewalk@minmax@n{ }%
5376 \edef\forest@nodewalk@minmax@list{#1}%
5377 \def\forest@nodewalk@minmax@nodes{ }%
5378 \def\forest@nodewalk@minmax@node{ }%
5379 \ifdefempty{\forest@nodewalk@minmax@list}{%
5380 }{%
5381   \safeloop
5382     \expandafter\forest@csvglist@getfirstrest\expandafter{\forest@nodewalk@minmax@list}\forest@nodewalk@minmax@cn
5383     \ifnum\forest@nodewalk@minmax@cn=0 \else
5384       \ifdefempty{\forest@nodewalk@minmax@node}{%
5385         \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
5386         \edef\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,}%
5387         \edef\forest@nodewalk@minmax@n{\safeloopn}%
5388       }{%
5389         \csedef{forest@nodesort@1}{\forest@nodewalk@minmax@node}%
5390         \csedef{forest@nodesort@2}{\forest@nodewalk@minmax@cn}%
5391         \forest@nodesort@cmpnodes{2}{1}%
5392         \if=\forest@sort@cmp@result
5393           \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
5394           \epreto\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,}%
5395           \edef\forest@nodewalk@minmax@n{\safeloopn}%
5396         \else
5397           \if#3\forest@sort@cmp@result
5398             \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
5399             \edef\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,}%
5400             \edef\forest@nodewalk@minmax@n{\safeloopn}%
5401           \fi
5402         \fi
5403       }%
5404     \fi
5405     \ifdefempty{\forest@nodewalk@minmax@list}{\forest@tempfalse}{\forest@temptrue}%
5406     \ifnum\safeloopn=\forest@nodewalk@minmax@N\relax\forest@temptrue\fi
5407   \ifforest@temp
5408     \saferepeat
5409     \edef\forest@nodewalk@minmax@list{#4}%
5410     \ifdefempty\forest@nodewalk@minmax@list{%
5411       \forestset{nodewalk/jump back=\forest@nodewalk@minmax@n-1}% CHECK
5412     }{%
5413       \safeloop
5414         \expandafter\forest@csvglist@getfirstrest\expandafter{\forest@nodewalk@minmax@list}\forest@cn\forest@nodewalk@minmax@cn
5415         \forest@nodewalk@makestep
5416         \ifdefempty{\forest@nodewalk@minmax@list}{\forest@tempfalse}{\forest@temptrue}%
5417       \ifforest@temp
5418         \saferepeat
5419       }%
5420     }%
5421 }

```

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.

```

5422 \newtoks\forest@nodewalk@shortsteps@resolution
5423 \newif\ifforest@nodewalk@areshortsteps
5424 \pgfqkeys{/forest/nodewalk}{
5425   .unknown/.code={%
5426     \forest@nodewalk@areshortstepsfalse
5427     \ifx\pgfkeyscurrentvalue\pgfkeysnovalue@text % no value, so possibly short steps
5428       \forest@nodewalk@shortsteps@resolution{ }%
5429     \forest@nodewalk@areshortstepstrue

```

```

5430     \expandafter\forest@nodewalk@shortsteps\pgfkeyscurrentname=====,% "=" and "," cannot be short step
5431     \fi
5432     \ifforest@nodewalk@aeshortsteps
5433     \@escapeif{\expandafter\pgfkeysalso\expandafter{\the\forest@nodewalk@shortsteps@resolution}}%
5434     \else
5435     \@escapeif{\pgfkeysalso{/forest/\pgfkeyscurrentname={#1}}}%
5436     \fi
5437   },
5438 }
5439 \def\forest@nodewalk@shortsteps{%
5440   \futurelet\forest@nodewalk@nexttoken\forest@nodewalk@shortsteps@
5441 }
5442 \def\forest@nodewalk@shortsteps@{%
5443   \ifx\forest@nodewalk@nexttoken=%
5444     \let\forest@nodewalk@nextop\forest@nodewalk@shortsteps@end
5445   \else
5446     \ifx\forest@nodewalk@nexttoken\bgroup
5447       \letcs\forest@nodewalk@nextop{forest@shortstep@group}%
5448     \else
5449       \let\forest@nodewalk@nextop\forest@nodewalk@shortsteps@@
5450     \fi
5451   \fi
5452   \forest@nodewalk@nextop
5453 }
5454 \def\forest@nodewalk@shortsteps@@#1{%
5455   \ifcsdef{forest@shortstep@#1}{%
5456     \csname forest@shortstep@#1\endcsname
5457   }{%
5458     \forest@nodewalk@aeshortstepsfalse
5459     \forest@nodewalk@shortsteps@end
5460   }%
5461 }
5462 % in the following definitions:
5463 % #1 = short step
5464 % #2 = (long) step, or a style in /forest/nodewalk (taking n args)
5465 \csdef{forest@nodewalk@defshortstep@0@args}#1#2{%
5466   \csdef{forest@shortstep@#1}{%
5467     \apptotoks\forest@nodewalk@shortsteps@resolution{,#2}%
5468     \forest@nodewalk@shortsteps}}
5469 \csdef{forest@nodewalk@defshortstep@1@args}#1#2{%
5470   \csdef{forest@shortstep@#1}##1{%
5471     \edef\forest@marshal####1{#2}%
5472     \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}}%
5473     \forest@nodewalk@shortsteps}}
5474 \csdef{forest@nodewalk@defshortstep@2@args}#1#2{%
5475   \csdef{forest@shortstep@#1}##1##2{%
5476     \edef\forest@marshal####1####2{#2}%
5477     \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}}%
5478     \forest@nodewalk@shortsteps}}
5479 \csdef{forest@nodewalk@defshortstep@3@args}#1#2{%
5480   \csdef{forest@shortstep@#1}##1##2##3{%
5481     \edef\forest@marshal####1####2####3{#2}%
5482     \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}}%
5483     \forest@nodewalk@shortsteps}}
5484 \csdef{forest@nodewalk@defshortstep@4@args}#1#2{%
5485   \csdef{forest@shortstep@#1}##1##2##3##4{%
5486     \edef\forest@marshal####1####2####3####4{#2}%
5487     \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}}%
5488     \forest@nodewalk@shortsteps}}
5489 \csdef{forest@nodewalk@defshortstep@5@args}#1#2{%
5490   \csdef{forest@shortstep@#1}##1##2##3##4##5{%

```

```

5491 \edef\forest@marshal####1####2####3####4####5{#2}%
5492 \eapptotoks\forest@nodewalk@shortsteps@resolution{\forest@marshal{##1}{##2}{##3}{##4}{##5}}%
5493 \forest@nodewalk@shortsteps}}
5494 \csdef{forest@nodewalk@defshortstep@6@args}#1#2{%
5495 \csdef{forest@shortstep@#1}##1##2##3##4##5##6{%
5496 \edef\forest@marshal####1####2####3####4####5####6{#2}%
5497 \eapptotoks\forest@nodewalk@shortsteps@resolution{\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}}%
5498 \forest@nodewalk@shortsteps}}
5499 \csdef{forest@nodewalk@defshortstep@7@args}#1#2{%
5500 \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7{%
5501 \edef\forest@marshal####1####2####3####4####5####6####7{#2}%
5502 \eapptotoks\forest@nodewalk@shortsteps@resolution{\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}}%
5503 \forest@nodewalk@shortsteps}}
5504 \csdef{forest@nodewalk@defshortstep@8@args}#1#2{%
5505 \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7##8{%
5506 \edef\forest@marshal####1####2####3####4####5####6####7####8{#2}%
5507 \eapptotoks\forest@nodewalk@shortsteps@resolution{\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}{##8}}%
5508 \forest@nodewalk@shortsteps}}
5509 \csdef{forest@nodewalk@defshortstep@9@args}#1#2{%
5510 \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7##8##9{%
5511 \edef\forest@marshal####1####2####3####4####5####6####7####8####9{#2}%
5512 \eapptotoks\forest@nodewalk@shortsteps@resolution{\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}{##8}{##9}}%
5513 \forest@nodewalk@shortsteps}}
5514 \forestset{
5515 define short step/.code n args=3{% #1 = short step, #2 = n args, #3 = long step
5516 \csname forest@nodewalk@defshortstep@#2@args\endcsname{##1}{##3}%
5517 },
5518 }
5519 \def\forest@nodewalk@shortsteps@end#1,{
    Define short-form steps.
5520 \forestset{
5521 define short step={group}{1}{group={#1}}, % {braces} are special
5522 define short step={p}{0}{previous},
5523 define short step={n}{0}{next},
5524 define short step={u}{0}{parent},
5525 define short step={s}{0}{sibling},
5526 define short step={c}{0}{current},
5527 define short step={o}{0}{origin},
5528 define short step={r}{0}{root},
5529 define short step={R}{0}{root'},
5530 define short step={P}{0}{previous leaf},
5531 define short step={N}{0}{next leaf},
5532 define short step={F}{0}{first leaf},
5533 define short step={L}{0}{last leaf},
5534 define short step={>}{0}{next on tier},
5535 define short step={<}{0}{previous on tier},
5536 define short step={1}{0}{n=1},
5537 define short step={2}{0}{n=2},
5538 define short step={3}{0}{n=3},
5539 define short step={4}{0}{n=4},
5540 define short step={5}{0}{n=5},
5541 define short step={6}{0}{n=6},
5542 define short step={7}{0}{n=7},
5543 define short step={8}{0}{n=8},
5544 define short step={9}{0}{n=9},
5545 define short step={l}{0}{last},
5546 define short step={b}{0}{back},
5547 define short step={f}{0}{forward},
5548 define short step={v}{0}{last valid},
5549 define short step={*}{2}{repeat={#1}{#2}},

```

```

5550 for 1/.style={for nodewalk={n=1}{#1}},
5551 for 2/.style={for nodewalk={n=2}{#1}},
5552 for 3/.style={for nodewalk={n=3}{#1}},
5553 for 4/.style={for nodewalk={n=4}{#1}},
5554 for 5/.style={for nodewalk={n=5}{#1}},
5555 for 6/.style={for nodewalk={n=6}{#1}},
5556 for 7/.style={for nodewalk={n=7}{#1}},
5557 for 8/.style={for nodewalk={n=8}{#1}},
5558 for 9/.style={for nodewalk={n=9}{#1}},
5559 for -1/.style={for nodewalk={n'=1}{#1}},
5560 for -2/.style={for nodewalk={n'=2}{#1}},
5561 for -3/.style={for nodewalk={n'=3}{#1}},
5562 for -4/.style={for nodewalk={n'=4}{#1}},
5563 for -5/.style={for nodewalk={n'=5}{#1}},
5564 for -6/.style={for nodewalk={n'=6}{#1}},
5565 for -7/.style={for nodewalk={n'=7}{#1}},
5566 for -8/.style={for nodewalk={n'=8}{#1}},
5567 for -9/.style={for nodewalk={n'=9}{#1}},
5568 }

```

Define multiple-step walks.

```

5569 \forestset{
5570   define long step={tree}{-}{\forest@node@foreach{\forest@nodewalk@makestep}},
5571   define long step={tree reversed}{-}{\forest@node@foreach@reversed{\forest@nodewalk@makestep}},
5572   define long step={tree children-first}{-}{\forest@node@foreach@childrenfirst{\forest@nodewalk@makestep}},
5573   define long step={tree children-first reversed}{-}{\forest@node@foreach@childrenfirst@reversed{\forest@nodewalk@makestep}},
5574   define long step={tree breadth-first}{-}{\forest@node@foreach@breadthfirst{-1}{\forest@nodewalk@makestep}},
5575   define long step={tree breadth-first reversed}{-}{\forest@node@foreach@breadthfirst@reversed{-1}{\forest@nodewalk@makestep}},
5576   define long step={descendants}{-}{\forest@node@foreachdescendant{\forest@nodewalk@makestep}},
5577   define long step={descendants reversed}{-}{\forest@node@foreachdescendant@reversed{\forest@nodewalk@makestep}},
5578   define long step={descendants children-first}{-}{\forest@node@foreachdescendant@childrenfirst{\forest@nodewalk@makestep}},
5579   define long step={descendants children-first reversed}{-}{\forest@node@foreachdescendant@childrenfirst@reversed{\forest@nodewalk@makestep}},
5580   define long step={descendants breadth-first}{-}{\forest@node@foreach@breadthfirst{0}{\forest@nodewalk@makestep}},
5581   define long step={descendants breadth-first reversed}{-}{\forest@node@foreach@breadthfirst@reversed{0}{\forest@nodewalk@makestep}},
5582   define long step={level}{n args=1}{%
5583     \forestmathtruncatemacro\forest@temp{#1}%
5584     \edef\forest@marshal{%
5585       \noexpand\forest@node@foreach@breadthfirst
5586         {\forest@temp}%
5587       {\noexpand\ifnum\noexpand\forest@level=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexpand\forest@temp}
5588     }\forest@marshal
5589   },
5590   define long step={level>}{n args=1}{%
5591     \forestmathtruncatemacro\forest@temp{#1}%
5592     \edef\forest@marshal{%
5593       \noexpand\forest@node@foreach@breadthfirst
5594         {-1}%
5595       {\noexpand\ifnum\noexpand\forest@level<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@makestep\noexpand\forest@temp}
5596     }\forest@marshal
5597   },
5598   define long step={level<}{n args=1}{%
5599     \forestmathtruncatemacro\forest@temp{(#1)-1}%
5600     \ifnum\forest@temp=-1
5601       % special case, as \forest@node@foreach@breadthfirst uses level<0 as a signal for unlimited max level
5602       \ifnum\forest@level=0
5603         \forest@nodewalk@makestep
5604       \fi
5605     \else
5606       \edef\forest@marshal{%
5607         \noexpand\forest@node@foreach@breadthfirst
5608           {\forest@temp}%

```

```

5609     {\noexpand\forest@nodewalk@makestep}%
5610   }\forest@marshal
5611 \fi
5612 },
5613 define long step={level reversed}{n args=1}{%
5614   \forestmathtruncatemacro\forest@temp{#1}%
5615   \edef\forest@marshal{%
5616     \noexpand\forest@node@foreach@breadthfirst@reversed
5617     {\forest@temp}%
5618     {\noexpand\ifnum\noexpand\forestove{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexp
5619   }\forest@marshal
5620 },
5621 define long step={level reversed>}{n args=1}{%
5622   \forestmathtruncatemacro\forest@temp{#1}%
5623   \edef\forest@marshal{%
5624     \noexpand\forest@node@foreach@breadthfirst@reversed
5625     {-1}%
5626     {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
5627   }\forest@marshal
5628 },
5629 define long step={level reversed<}{n args=1}{%
5630   \forestmathtruncatemacro\forest@temp{(#1)-1}%
5631   \edef\forest@marshal{%
5632     \noexpand\forest@node@foreach@breadthfirst@reversed
5633     {\forest@temp}%
5634     {\noexpand\forest@nodewalk@makestep}%
5635   }\forest@marshal
5636 },
5637 %
5638 define long step={relative level}{n args=1}{%
5639   \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
5640   \edef\forest@marshal{%
5641     \noexpand\forest@node@foreach@breadthfirst
5642     {\forest@temp}%
5643     {\noexpand\ifnum\noexpand\forestove{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexp
5644   }\forest@marshal
5645 },
5646 define long step={relative level>}{n args=1}{%
5647   \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
5648   \edef\forest@marshal{%
5649     \noexpand\forest@node@foreach@breadthfirst
5650     {-1}%
5651     {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
5652   }\forest@marshal
5653 },
5654 define long step={relative level<}{n args=1}{%
5655   \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}-1}%
5656   \edef\forest@marshal{%
5657     \noexpand\forest@node@foreach@breadthfirst
5658     {\forest@temp}%
5659     {\noexpand\forest@nodewalk@makestep}%
5660   }\forest@marshal
5661 },
5662 define long step={relative level reversed}{n args=1}{%
5663   \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
5664   \edef\forest@marshal{%
5665     \noexpand\forest@node@foreach@breadthfirst@reversed
5666     {\forest@temp}%
5667     {\noexpand\ifnum\noexpand\forestove{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexp
5668   }\forest@marshal
5669 },

```

```

5670 define long step={relative level reversed>}{n args=1}{%
5671   \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
5672   \edef\forest@marshal{%
5673     \noexpand\forest@node@foreach@breadthfirst@reversed
5674     {-1}%
5675     {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
5676     }\forest@marshal
5677   },
5678 define long step={relative level reversed<}{n args=1}{%
5679   \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}-1}%
5680   \edef\forest@marshal{%
5681     \noexpand\forest@node@foreach@breadthfirst@reversed
5682     {\forest@temp}%
5683     {\noexpand\forest@nodewalk@makestep}%
5684   }\forest@marshal
5685 },
5686 define long step={leaves}{}{%
5687   \forest@node@foreach{%
5688     \ifnum\forestove{n children}=0
5689     \forest@nodewalk@makestep
5690   \fi
5691   }%
5692 },
5693 define long step={-level}{n args=1,style}{%
5694   unique={branch={leaves,{group={repeat={#1}{parent}}}}}
5695 },
5696 define long step={-level'}{n args=1,style}{%
5697   unique={on invalid={fake}{branch={leaves,{group={repeat={#1}{parent}}}}}
5698 },
5699 define long step={children}{}{\forest@node@foreachchild{\forest@nodewalk@makestep}},
5700 define long step={children reversed}{}{\forest@node@foreachchild@reversed{\forest@nodewalk@makestep}},
5701 define long step={current and following siblings}{}{\forest@node@@forselfandfollowingsiblings{\forest@nodew
5702 define long step={following siblings}{style}{if nodewalk valid={next}{fake=next,current and following sibli
5703 define long step={current and preceding siblings}{}{\forest@node@@forselfandprecedingsiblings{\forest@nodew
5704 define long step={preceding siblings}{style}{if nodewalk valid={previous}{fake=previous,current and precedi
5705 define long step={current and following siblings reversed}{}{\forest@node@@forselfandfollowingsiblings@re
5706 define long step={following siblings reversed}{style}{fake=next,current and following siblings reversed},
5707 define long step={current and preceding siblings reversed}{}{\forest@node@@forselfandprecedingsiblings@re
5708 define long step={preceding siblings reversed}{style}{fake=previous,current and preceding siblings reversed
5709 define long step={siblings}{style}{for nodewalk'={preceding siblings},following siblings},
5710 define long step={siblings reversed}{style}{for nodewalk'={following siblings reversed},preceding siblings
5711 define long step={current and siblings}{style}{for nodewalk'={preceding siblings},current and following sib
5712 define long step={current and siblings reversed}{style}{for nodewalk'={current and following siblings rever
5713 define long step={ancestors}{style}{while={}{parent},last valid},
5714 define long step={current and ancestors}{style}{current,ancestors},
5715 define long step={following nodes}{style}{while={}{next node},last valid},
5716 define long step={preceding nodes}{style}{while={}{previous node},last valid},
5717 define long step={current and following nodes}{style}{current,following nodes},
5718 define long step={current and preceding nodes}{style}{current,preceding nodes},
5719 }
5720 \let\forest@nodewalkstephandler@styletrueorfalse\forest@nodewalkstephandler@styletrue

```

7.4 Dynamic tree

```

5721 \def\forest@last@node{0}
5722 \csdef\forest@nodewalk@samed@dynamic nodes{}
5723 \def\forest@nodehandleby@name@nodewalk@or@bracket#1{%
5724   \ifx\pgfkeysnovalue#1%
5725     \edef\forest@last@node{\forest@node@Nametoid{forest@last@node}}%
5726   \else

```

```

5727 \forest@nodehandleby@nmb@checkfirst#1\forest@END
5728 \fi
5729 }
5730 \def\forest@nodehandleby@nmb@checkfirst#1#2\forest@END{%
5731 \ifx[#1]
5732 \forest@create@node{#1#2}%
5733 \cseappto{forest@nodewalk@saved@dynamic nodes}{\forest@last@node,}%
5734 \else
5735 \forest@forthis{%
5736 \forest@nameandgo{#1#2}%
5737 \ifnum\forest@cn=0
5738 \PackageError{forest}{Cannot use a dynamic key on the invalid node}{}%
5739 \fi
5740 \let\forest@last@node\forest@cn
5741 }%
5742 \fi
5743 }
5744 \def\forest@create@node#1{% #1=bracket representation
5745 \bracketParse{\forest@create@collectafterthought}%
5746 \forest@last@node=#1\forest@end@create@node
5747 }
5748 \def\forest@create@collectafterthought#1\forest@end@create@node{%
5749 \forest@node@Foreach{\forest@last@node}{%
5750 \forest@toletto{delay}{given options}%
5751 \forest@setto{given options}{}%
5752 }%
5753 \forest@oappto{\forest@last@node}{delay}{,\unexpanded{#1}}%
5754 \forest@oasetto{\forest@last@node}{given options}{delay={}}%
5755 }
5756 \def\forest@create@node@and@process@given@options#1{% #1=bracket representation
5757 \bracketParse{\forest@createandprocess@collectafterthought}%
5758 \forest@last@node=#1\forest@end@create@node
5759 }
5760 \def\forest@createandprocess@collectafterthought#1\forest@end@create@node{%
5761 \forest@node@Compute@numeric@ts@info{\forest@last@node}%
5762 \forest@saveandrestoremacro\forest@root{%
5763 \let\forest@root\forest@last@node
5764 \forest@setto{process keylist=given options}%
5765 }%
5766 }
5767 \def\forest@saveandrestoremacro#1#2{% #1 = the (zero-arg) macro to save before and restore after processing c
5768 \edef\forest@marshal{%
5769 \unexpanded{#2}%
5770 \noexpand\def\noexpand#1{\expandonce{#1}}%
5771 }\forest@marshal
5772 }
5773 \def\forest@saveandrestoreifcs#1#2{% #1 = the if cs to save before and restore after processing code in #2
5774 \edef\forest@marshal{%
5775 \unexpanded{#2}%
5776 \ifbool{#1}{\noexpand\setbool{#1}{true}}{\noexpand\setbool{#1}{false}}%
5777 }\forest@marshal
5778 }
5779 \def\forest@globalsaveandrestoreifcs#1#2{% #1 = the if cs to save before and restore after processing code in #2
5780 \edef\forest@marshal{%
5781 \unexpanded{#2}%
5782 \ifbool{#1}{\global\noexpand\setbool{#1}{true}}{\global\noexpand\setbool{#1}{false}}%
5783 }\forest@marshal
5784 }
5785 \def\forest@saveandrestoretoks#1#2{% #1 = the toks to save before and restore after processing code in #2
5786 \edef\forest@marshal{%
5787 \unexpanded{#2}%

```



```

5788     \noexpand#1{\the#1}%
5789   }\forest@marshal
5790 }
5791 \def\forest@saveandrestoreregister#1#2{% #1 = the register to save before and restore after processing code i
5792   \edef\forest@marshal{%
5793     \unexpanded{#2}%
5794     \noexpand\forest@rset{#1}{\forestregister{#1}}%
5795   }\forest@marshal
5796 }
5797 \forestset{
5798   save and restore register/.code 2 args={%
5799     \forest@saveandrestoreregister{#1}{%
5800       \pgfkeysalso{#2}%
5801     }%
5802   },
5803 }
5804 \def\forest@remove@node#1{%
5805   \ifforestdebugdynamics\forestdebug@dynamics{before removing #1}\fi
5806   \forest@node@Remove{#1}%
5807 }
5808 \def\forest@append@node#1#2{%
5809   \ifforestdebugdynamics\forestdebug@dynamics{before appending #2 to #1}\fi
5810   \forest@dynamic@circularitytest{#2}{#1}{append}%
5811   \forest@node@Remove{#2}%
5812   \forest@node@Append{#1}{#2}%
5813 }
5814 \def\forest@prepend@node#1#2{%
5815   \ifforestdebugdynamics\forestdebug@dynamics{before prepending #2 to #1}\fi
5816   \forest@dynamic@circularitytest{#2}{#1}{prepend}%
5817   \forest@node@Remove{#2}%
5818   \forest@node@Prepend{#1}{#2}%
5819 }
5820 \def\forest@insertafter@node#1#2{%
5821   \ifforestdebugdynamics\forestdebug@dynamics{before inserting #2 after #1}\fi
5822   \forest@node@Remove{#2}%
5823   \forest@node@Insertafter{\forest@ve{#1}{@parent}}{#2}{#1}%
5824 }
5825 \def\forest@insertbefore@node#1#2{%
5826   \ifforestdebugdynamics\forestdebug@dynamics{before inserting #2 before #1}\fi
5827   \forest@node@Remove{#2}%
5828   \forest@node@Insertbefore{\forest@ve{#1}{@parent}}{#2}{#1}%
5829 }
5830 \def\forest@set@root#1#2{%
5831   \ifforestdebugdynamics\forestdebug@dynamics{before setting #1 as root}\fi
5832   \def\forest@root{#2}%
5833 }
5834 \def\forest@dynamic@circularitytest#1#2#3{%
5835   % #1=potential ancestor,#2=potential descendant, #3=message prefix
5836   \ifnum#1=#2
5837     \forest@circularityerror{#1}{#2}{#3}%
5838   \else
5839     \forest@fornode{#1}{%
5840       \forest@ifancestorof{#2}{\forest@circularityerror{#1}{#2}{#3}}{%
5841       }%
5842     \fi
5843 }
5844 \def\forest@circularityerror#1#2#3{%
5845   \forestdebug@typeouttrees{\forest@temp}%
5846   \PackageError{forest}{#3ing node id=#1 to id=#2 would result in a circular tree\MessageBreak forest of ids:
5847 }%
5848 \def\forestdebug@dynamics#1{%

```

```

5849 \forestdebug@typeouttrees\forest@temp
5850 \typeout{#1: \forest@temp}%
5851 }
5852 \def\forest@appto@do@dynamics#1#2{%
5853   \forest@nodehandleby@name@nodewalk@or@bracket{#2}%
5854   \ifcase\forest@dynamics@copyhow\relax\or
5855     \forest@tree@copy{\forest@last@node}\forest@last@node
5856   \or
5857     \forest@node@copy{\forest@last@node}\forest@last@node
5858   \fi
5859   \forest@node@ifnamedefined{forest@last@node}{%
5860     \forest@depreto{\forest@last@node}{delay}
5861     {for id={\forest@node@Nametoid{forest@last@node}}{alias=forest@last@node},}%
5862   }{}%
5863   \edef\forest@marshal{%
5864     \noexpand\apptotoks\noexpand\forest@do@dynamics{%
5865       \noexpand#1{\forest@cn}{\forest@last@node}}%
5866   }\forest@marshal
5867 }
5868 \forestset{%
5869   create/.code={%
5870     \forest@create@node{#1}%
5871     \forest@fornode{\forest@last@node}{%
5872       \forest@node@setalias{forest@last@node}%
5873       \cseappto{forest@nodewalk@saved@dynamic nodes}{\forest@last@node},}%
5874   }%
5875 },
5876   create'/.code={%
5877     \forest@create@node@and@process@given@options{#1}%
5878     \forest@fornode{\forest@last@node}{%
5879       \forest@node@setalias{forest@last@node}%
5880       \cseappto{forest@nodewalk@saved@dynamic nodes}{\forest@last@node},}%
5881   }%
5882 },
5883   append/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@append@node{#1}},
5884   prepend/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@prepend@node{#1}},
5885   insert after/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@insertafter@node{#1}},
5886   insert before/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@insertbefore@node{#1}},
5887   append'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@append@node{#1}},
5888   prepend'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@prepend@node{#1}},
5889   insert after'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@insertafter@node{#1}},
5890   insert before'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@insertbefore@node{#1}},
5891   append''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@append@node{#1}},
5892   prepend''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@prepend@node{#1}},
5893   insert after''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@insertafter@node{#1}},
5894   insert before''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@insertbefore@node{#1}},
5895   remove/.code={%
5896     \pgfkeysalso{alias=forest@last@node}%
5897     \cseappto{forest@nodewalk@saved@dynamic nodes}{\forest@cn},}%
5898     \expandafter\apptotoks\expandafter\forest@do@dynamics\expandafter{%
5899       \expandafter\forest@remove@node\expandafter{\forest@cn}}%
5900   },
5901   set root/.code={%
5902     \def\forest@dynamics@copyhow{0}%
5903     \forest@appto@do@dynamics\forest@set@root{#1}%
5904   },
5905   replace by/.code={\forest@replaceby@code{#1}{insert after}},
5906   replace by'/.code={\forest@replaceby@code{#1}{insert after'}},
5907   replace by''/.code={\forest@replaceby@code{#1}{insert after''}},
5908   sort/.code={%
5909     \eapptotoks\forest@do@dynamics{%

```

```

5910     \def\noexpand\forest@nodesort@sortkey{\forestrv{sort by}}%
5911     \noexpand\forest@nodesort\noexpand\forest@sort@ascending{\forest@cn}
5912   }%
5913 },
5914 sort'/.code={%
5915   \eapptotoks\forest@do@dynamic{%
5916     \def\noexpand\forest@nodesort@sortkey{\forestrv{sort by}}%
5917     \noexpand\forest@nodesort\noexpand\forest@sort@descending{\forest@cn}
5918   }%
5919 },
5920 }
5921 \def\forest@replaceby@code#1#2{%#1=node spec,#2=insert after['][']
5922   \ifnum\forestove{@parent}=0
5923     \cseappto{\forest@nodewalk@saved@dynamic nodes}{\forest@cn,}%
5924     \pgfkeysalso{alias=forest@last@node,set root={#1}}%
5925   \else
5926     \cseappto{\forest@nodewalk@saved@dynamic nodes}{\forest@cn,}%
5927     \pgfkeysalso{alias=forest@last@node,#2={#1}}%
5928     \eapptotoks\forest@do@dynamic{%
5929       \noexpand\ifnum\noexpand\forestOve{\forest@cn}{@parent}=\forestove{@parent}
5930       \noexpand\forest@remove@node{\forest@cn}%
5931     \noexpand\fi
5932   }%
5933   \fi
5934 }
5935 \def\forest@nodesort#1#2{% #1 = direction, #2 = parent node
5936   \ifforestdebugdynamics\forestdebug@dynamic{before sorting children of #2}\fi
5937   \forest@for@node{#2}{\forest@nodesort@#1}%
5938   \ifforestdebugdynamics\forestdebug@dynamic{after sorting children of #2}\fi
5939 }
5940 \def\forest@nodesort@#1{%
5941   % prepare the array of child ids
5942   \c@pgf@counta=0
5943   \forest@toget{@first}\forest@nodesort@id
5944   \forest@loop
5945   \ifnum\forest@nodesort@id>0
5946     \advance\c@pgf@counta 1
5947     \csedef\forest@nodesort@the\c@pgf@counta{\forest@nodesort@id}%
5948     \forest@toget{\forest@nodesort@id}{@next}\forest@nodesort@id
5949   \forest@repeat
5950   % sort
5951   \forest@toget{n children}\forest@nodesort@n
5952   \forest@sort\forest@nodesort@cmpnodes\forest@nodesort@let#1{1}{\forest@nodesort@n}%
5953   % remove all children
5954   \forest@toget{@first}\forest@nodesort@id
5955   \forest@loop
5956   \ifnum\forest@nodesort@id>0
5957     \forest@node@Remove{\forest@nodesort@id}%
5958     \forest@toget{@first}\forest@nodesort@id
5959   \forest@repeat
5960   % insert the children in new order
5961   \c@pgf@counta=0
5962   \forest@loop
5963   \ifnum\c@pgf@counta<\forest@nodesort@n\relax
5964     \advance\c@pgf@counta 1
5965     \forest@node@append{\csname forest@nodesort@the\c@pgf@counta\endcsname}%
5966   \forest@repeat
5967 }
5968 \def\forest@nodesort@cmpnodes#1#2{%
5969   \expandafter\forest@nodesort@cmpnodes\forest@nodesort@sortkey,\forest@END{#1}{#2}%
5970 }

```

```

5971 \def\forest@nodesort@cmpnodes@#1,#2\forest@END#3#4{%
5972 % #1=process ins+arg for this dimension, #2=for next dimensions
5973 % #3, #4 = node ids
5974 {%
5975   \forest@fornode{\csname forest@nodesort@#3\endcsname}{%
5976     \forestmathsetmacro\forest@nodesort@resulta{#1}%
5977   }%
5978   \forest@fornode{\csname forest@nodesort@#4\endcsname}{%
5979     \forestmathsetmacro\forest@nodesort@resultb{#1}%
5980   }%
5981   \ifx\forestmathresulttype\forestmathtype@generic
5982     \forest@cmp@error{\forest@nodesort@resulta}{\forest@nodesort@resultb}%
5983   \fi
5984   \edef\forest@temp{%
5985     \noexpand\forest@nodesort@cmp
5986     {\expandonce{\forest@nodesort@resulta}}%
5987     {\expandonce{\forest@nodesort@resultb}}%
5988   }%
5989   \xdef\forest@global@temp{\forest@temp}%
5990 }%
5991 \if=\forest@global@temp
5992   \let\forest@next\forest@nodesort@cmpnodes@
5993 \else
5994   \let\forest@next\forest@nodesort@cmpnodes@finish
5995 \fi
5996 \ifstrempy{#2}{\let\forest@next\forest@nodesort@cmpnodes@finish}{}%
5997 \forest@next#2\forest@END{#3}{#4}%
5998 }
5999 \def\forest@nodesort@cmpnodes@finish#1\forest@END#2#3{%
6000 \let\forest@sort@cmp@result\forest@global@temp
6001 }

```

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

```

6002 \def\forest@nodesort@cmp{\csname fRsT@nsc@\forestmathresulttype\endcsname}
6003 \def\fRsT@nsc@#1{\csname fRsT@nsc@#1\endcsname}
6004 \def\fRsT@nsc@n#1#2{\ifnum#1<#2 <\else\ifnum#1=#2 =\else>\fi\fi}
6005 \def\fRsT@nsc@d#1#2{\ifdim#1<#2 <\else\ifdim#1=#2 =\else>\fi\fi}
6006 \def\fRsT@nsc@P#1#2{\ifdim#1pt<#2pt <\else\ifdim#1pt=#2pt =\else>\fi\fi}
6007 \def\fRsT@nsc@t#1#2{\csname fRsT@nsc@\pdfstrcmp{#1}{#2}\endcsname}
6008 \def\fRsT@nsc@T#1#2{\csname fRsT@nsc@\pdfstrcmp{#2}{#1}\endcsname}
6009 \csdef{fRsT@nsc@-1}{<}
6010 \csdef{fRsT@nsc@0}{=}
6011 \csdef{fRsT@nsc@1}{>}
6012 \def\forest@nodesort@let#1#2{%
6013   \csletcs{forest@nodesort@#1}{forest@nodesort@#2}%
6014 }
6015 \forestset{
6016   define long step={last dynamic node}{style,must start at valid node=false}{%
6017     name=forest@last@node
6018   }
6019 }

```

8 Stages

```

6020 \def\forest@root{0}
6021 %%% begin listing region: stages
6022 \forestset{
6023   stages/.style={
6024     for root'={
6025       process keylist register=default preamble,

```

```

6026     process keylist register=preamble
6027   },
6028   process keylist=given options,
6029   process keylist=before typesetting nodes,
6030   typeset nodes stage,
6031   process keylist=before packing,
6032   pack stage,
6033   process keylist=before computing xy,
6034   compute xy stage,
6035   process keylist=before drawing tree,
6036   draw tree stage
6037 },
6038 typeset nodes stage/.style={for root'=typeset nodes},
6039 pack stage/.style={for root'=pack},
6040 compute xy stage/.style={for root'=compute xy},
6041 draw tree stage/.style={for root'=draw tree},
6042 }
6043 %%% end listing region: stages
6044 \forestset{
6045   process keylist/.code={%
6046     \forest@process@hook@keylist{#1}{#1 processing order/.try,processing order/.lastretry}},
6047   process keylist'/ .code 2 args={\forest@process@hook@keylist@nodynamics{#1}{#2}},
6048   process keylist''/.code 2 args={\forest@process@hook@keylist@{#1}{#2}},
6049   process keylist register/.code={\forest@process@keylist@register{#1}},
6050   process delayed/.code={%
6051     \forest@havedelayedoptions{@delay}{#1}%
6052     \ifforest@havedelayedoptions
6053       \forest@process@hook@keylist@nodynamics{@delay}{#1}%
6054     \fi
6055   },
6056   do dynamics/.code={%
6057     \the\forest@do@dynamics
6058     \forest@do@dynamics{}%
6059     \forest@node@Compute@numeric@ts@info{\forest@root}%
6060   },
6061   declare keylist={given options}{},
6062   declare keylist={before typesetting nodes}{},
6063   declare keylist={before packing}{},
6064   declare keylist={before packing node}{},
6065   declare keylist={after packing node}{},
6066   declare keylist={before computing xy}{},
6067   declare keylist={before drawing tree}{},
6068   declare keylist={delay}{},
6069   delay n/.code 2 args={%
6070     \forestmathsetcount\forest@temp@count{#1}%
6071     \pgfkeysalso{delay n'={\forest@temp@count}{#2}}%
6072   },
6073   delay n'/ .code 2 args={
6074     \ifnum#1=0
6075       \pgfkeysalso{#2}%
6076     \else
6077       \pgfkeysalso{delay={delay n'/ .expand once=\expandafter{\number\numexpr#1-1\relax}{#2}}}%
6078     \fi
6079   },
6080   if have delayed/.style 2 args={if have delayed'={processing order}{#1}{#2}},
6081   if have delayed'/ .code n args=3{%
6082     \forest@havedelayedoptionsfalse
6083     \forest@forthis{%
6084       \forest@nodewalk{#1}{%
6085         TeX={%
6086           \foresttoget{delay}\forest@temp@delayed

```

```

6087     \ifdefempty\forest@temp@delayed{}{\forest@havedelayedoptionstrue}%
6088     }%
6089     }%
6090     }%
6091     \ifforest@havedelayedoptions\pgfkeysalso{#2}\else\pgfkeysalso{#3}\fi
6092   },
6093   typeset nodes/.code={%
6094     \forest@drawtree@preservenodeboxes@false
6095     \forest@nodewalk
6096     {typeset nodes processing order/.try,processing order/.lastretry}%
6097     {TeX={\forest@node@typeset}}}%
6098   },
6099   typeset nodes'/.code={%
6100     \forest@drawtree@preservenodeboxes@true
6101     \forest@nodewalk
6102     {typeset nodes processing order/.try,processing order/.lastretry}%
6103     {TeX={\forest@node@typeset}}}%
6104   },
6105   typeset node/.code={%
6106     \forest@drawtree@preservenodeboxes@false
6107     \forest@node@typeset
6108   },
6109   pack/.code={\forest@pack},
6110   pack'/.code={\forest@pack@onlythisnode},
6111   compute xy/.code={\forest@node@computeabsolutepositions},
6112   draw tree box/.store in=\forest@drawtreebox,
6113   draw tree box,
6114   draw tree/.code={%
6115     \forest@drawtree@preservenodeboxes@false
6116     \forest@node@drawtree
6117   },
6118   draw tree'/.code={%
6119     \forest@drawtree@preservenodeboxes@true
6120     \forest@node@drawtree
6121   },
6122   %%% begin listing region: draw_tree_method
6123   draw tree method/.style={
6124     for nodewalk={
6125       draw tree nodes processing order/.try,
6126       draw tree processing order/.retry,
6127       processing order/.lastretry
6128     }{draw tree node},
6129     for nodewalk={
6130       draw tree edges processing order/.try,
6131       draw tree processing order/.retry,
6132       processing order/.lastretry
6133     }{draw tree edge},
6134     for nodewalk={
6135       draw tree tikz processing order/.try,
6136       draw tree processing order/.retry,
6137       processing order/.lastretry
6138     }{draw tree tikz}
6139   },
6140   %%% end listing region: draw_tree_method
6141   draw tree node/.code={\forest@draw@node},
6142   draw tree node'/.code={\forest@draw@node@},
6143   if node drawn/.code n args={3}{%
6144     \forest@forthis{%
6145       \forest@configured@nodewalk{independent}{inherited}{fake}{#1}{}%
6146       \ifnum\forest@cn=0
6147         \forest@tempfalse

```

```

6148     \else
6149     \ifcsdef{forest@drawn@forest@cn}{\forest@temptrue}{\forest@tempfalse}%
6150     \fi
6151     }%
6152     \ifforest@temp\pgfkeysalso{#2}\else\pgfkeysalso{#3}\fi
6153   },
6154   draw tree edge/.code={\forest@draw@edge},
6155   draw tree edge'/.code={\forest@draw@edge@},
6156   draw tree tikz/.code={\forest@draw@tikz@}, % always!
6157   draw tree tikz'/.code={\forest@draw@tikz@},
6158   processing order/.nodewalk style={tree},
6159   %given options processing order/.style={processing order},
6160   %before typesetting nodes processing order/.style={processing order},
6161   %before packing processing order/.style={processing order},
6162   %before computing xy processing order/.style={processing order},
6163   %before drawing tree processing order/.style={processing order},
6164 }
6165 \newtoks\forest@do@dynamics
6166 \newif\ifforest@havedelayedoptions
6167 \def\forest@process@hook@keylist#1#2{%,#1=keylist,#2=processing order nodewalk
6168   \safeloop
6169     \forest@fornode{\forest@root}{\forest@process@hook@keylist@{#1}{#2}}%
6170     \expandafter\ifstreempty\expandafter{\the\forest@do@dynamics}{}%
6171     \the\forest@do@dynamics
6172     \forest@do@dynamics={}%
6173     \forest@node@Compute@numeric@ts@info{\forest@root}%
6174     }%
6175   \forest@fornode{\forest@root}{\forest@havedelayedoptions{#1}{#2}}%
6176   \ifforest@havedelayedoptions
6177   \saferepeat
6178 }
6179 \def\forest@process@hook@keylist@nodynamics#1#2{%,#1=keylist,#2=processing order nodewalk
6180   % note: this macro works on (nodewalk starting at) the current node
6181   \safeloop
6182     \forest@forthis{\forest@process@hook@keylist@{#1}{#2}}%
6183     \forest@havedelayedoptions{#1}{#2}%
6184     \ifforest@havedelayedoptions
6185     \saferepeat
6186 }
6187 \def\forest@process@hook@keylist@#1#2{%,#1=keylist,#2=processing order nodewalk
6188   \forest@nodewalk{#2}{%
6189     TeX={%
6190       \forest@toget{#1}\forest@temp@keys
6191       \ifdefvoid\forest@temp@keys}{%
6192         \forest@set{#1}{}%
6193         \expandafter\forest@set\expandafter{\forest@temp@keys}%
6194       }%
6195     }%
6196   }%
6197 }
6198 \def\forest@process@keylist@register#1{%
6199   \edef\forest@marshal{%
6200     \noexpand\forest@set{\forest@register{#1}}%
6201   }\forest@marshal
6202 }

```

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

```

6203 \def\forest@havedelayedoptions#1#2{%,#1 = keylist, #2=nodewalk
6204   \forest@havedelayedoptionsfalse
6205   \forest@forthis{%
6206     \forest@nodewalk{#2}{%

```

```

6207     TeX={%
6208         \foresttoget{delay}\forest@temp@delayed
6209         \ifdefempty\forest@temp@delayed{}\forest@havedelayedoptionstrue}%
6210         \forestolet{#1}\forest@temp@delayed
6211         \forestoset{delay}{}%
6212     }%
6213 }%
6214 }%
6215 }

```

8.1 Typesetting nodes

```

6216 \def\forest@node@typeset{%
6217     \let\forest@next\forest@node@typeset@
6218     \forestoifdefined{@box}{%
6219         \foresttoget{@box}\forest@temp
6220         \ifdefempty\forest@temp{%
6221             \locbox\forest@temp@box
6222             \forestolet{@box}\forest@temp@box
6223         }{%
6224             \ifforest@drawtree@preservenodeboxes@
6225                 \let\forest@next\relax
6226             \fi
6227         }%
6228     }{%
6229         \locbox\forest@temp@box
6230         \forestolet{@box}\forest@temp@box
6231     }%
6232     \def\forest@node@typeset@restore{}%
6233     \ifdefined\ifsa@tikz\forest@standalone@hack\fi
6234     \forest@next
6235     \forest@node@typeset@restore
6236 }
6237 \def\forest@standalone@hack{%
6238     \ifsa@tikz
6239         \let\forest@standalone@tikzpicture\tikzpicture
6240         \let\forest@standalone@endtikzpicture\endtikzpicture
6241         \let\tikzpicture\sa@orig@tikzpicture
6242         \let\endtikzpicture\sa@orig@endtikzpicture
6243         \def\forest@node@typeset@restore{%
6244             \let\tikzpicture\forest@standalone@tikzpicture
6245             \let\endtikzpicture\forest@standalone@endtikzpicture
6246         }%
6247     \fi
6248 }
6249 \newbox\forest@box
6250 \def\forest@pgf@notyetpositioned{not yet positionedPGFINTERNAL}
6251 \def\forest@node@typeset@{%
6252     \forestanchortotikzanchor{anchor}\forest@temp
6253     \edef\forest@marshal{%
6254         \noexpand\forestolet{anchor}\noexpand\forest@temp
6255         \noexpand\forest@node@typeset@
6256         \noexpand\forestoset{anchor}{\forestov{anchor}}%
6257     }\forest@marshal
6258 }
6259 \def\forest@node@typeset@@{%
6260     \foresttoget{name}\forest@nodename
6261     \edef\forest@temp@nodeformat{\forestove{node format}}%
6262     \gdef\forest@smuggle{}%
6263     \setbox0=\hbox{%
6264         \begin{tikzpicture}[%
6265             /forest/copy command key={/tikz/anchor}/{/tikz/forest@orig@anchor},

```



```

6266     anchor/.style={%
6267         /forest/TeX={\forestanchortotikzanchor{##1}\forest@temp@anchor},
6268         forest@orig@anchor/.expand once=\forest@temp@anchor
6269     }]
6270     \pgfpositionnodelater{\forest@positionnodelater@save}%
6271     \forest@temp@nodeformat
6272     \pgfinterruptpath
6273     \pgfpointanchor{\forest@pgf@notyetpositioned\forest@nodename}{forestcomputenodeboundary}%
6274     \endpgfinterruptpath
6275     \end{tikzpicture}%
6276 }%
6277 \setbox\forestove{@box}=\box\forest@box % smuggle the box
6278 \forestolet{@boundary}\forest@global@boundary
6279 \forest@smuggle % ... and the rest
6280 }
6281
6282
6283 \forestset{
6284     declare readonly dimen={min x}{0pt},
6285     declare readonly dimen={min y}{0pt},
6286     declare readonly dimen={max x}{0pt},
6287     declare readonly dimen={max y}{0pt},
6288 }
6289 \def\forest@patch@enormouscoordinateboxbounds@plus#1{%
6290     \expandafter\ifstrequal\expandafter{#1}{16000.0pt}{\edef#1{0.0\pgfmath@pt}}{}}%
6291 }
6292 \def\forest@patch@enormouscoordinateboxbounds@minus#1{%
6293     \expandafter\ifstrequal\expandafter{#1}{-16000.0pt}{\edef#1{0.0\pgfmath@pt}}{}}%
6294 }
6295 \def\forest@positionnodelater@save{%
6296     \global\setbox\forest@box=\box\pgfpositionnodelaterbox
6297     \xappto\forest@smuggle{\noexpand\forestoset{later@name}{\pgfpositionnodelatername}}%
6298     % a bug in pgf? ---well, here's a patch
6299     \forest@patch@enormouscoordinateboxbounds@plus\pgfpositionnodelaterminx
6300     \forest@patch@enormouscoordinateboxbounds@plus\pgfpositionnodelaterminy
6301     \forest@patch@enormouscoordinateboxbounds@minus\pgfpositionnodelatermaxx
6302     \forest@patch@enormouscoordinateboxbounds@minus\pgfpositionnodelatermaxy
6303     % end of patch
6304     % when shape=coordinate, pgf returns \pgfpositionnodelater... with wrong catcode of pt
6305     \forest@pnn@addtosmuggle{min x}\pgfpositionnodelaterminx
6306     \forest@pnn@addtosmuggle{max x}\pgfpositionnodelatermaxx
6307     \forest@pnn@addtosmuggle{min y}\pgfpositionnodelaterminy
6308     \forest@pnn@addtosmuggle{max y}\pgfpositionnodelatermaxy
6309 }
6310 \def\forest@pnn@addtosmuggle#1#2{%
6311     {%
6312         \pgfutil@tempdima=#2\relax
6313         \xappto\forest@smuggle{\noexpand\forestoset{#1}{\the\pgfutil@tempdima}}%
6314     }%
6315 }
6316 \def\forest@node@forest@positionnodelater@restore{%
6317     \ifforest@drawtree@preservenodeboxes@
6318         \let\forest@boxorcopy\copy
6319     \else
6320         \let\forest@boxorcopy\box
6321     \fi
6322     \forestoget{@box}\forest@temp
6323     \setbox\pgfpositionnodelaterbox=\forest@boxorcopy\forest@temp
6324     \edef\pgfpositionnodelatername{\forestove{later@name}}%
6325     \edef\pgfpositionnodelaterminx{\forestove{min x}}%
6326     \edef\pgfpositionnodelaterminy{\forestove{min y}}%

```

```

6327 \edef\pgfpositionnodelatermaxx{\forestove{max x}}%
6328 \edef\pgfpositionnodelatermaxy{\forestove{max y}}%
6329 \ifforest@drawtree@preservenodeboxes@
6330 \else
6331 \forestoset{@box}{}%
6332 \fi
6333 }

```

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.

```

6334 \def\forest@pack{%
6335 \pgfsyssoftpath@getcurrentpath\forest@pack@original@path
6336 \forest@pack@computetiers
6337 \forest@pack@computegrowthuniformity
6338 \forest@@pack
6339 \pgfsyssoftpath@setcurrentpath\forest@pack@original@path
6340 }
6341 \def\forest@@pack{%
6342 \ifnum\forestove{uniform growth}>0
6343 \ifnum\forestove{n children}>0
6344 \forest@pack@level@uniform
6345 \forest@pack@aligtiers@ofsubtree
6346 \forest@pack@sibling@uniform@recursive
6347 \fi
6348 \else
6349 \forest@node@foreachchild{\forest@@pack}%
6350 \forest@process@hook@keylist@nodynamics{before packing node}{current}%
6351 \ifnum\forestove{n children}>0
6352 \forest@pack@level@nonuniform
6353 \forest@pack@aligtiers
6354 \forest@pack@sibling@uniform@applyreversed
6355 \fi
6356 \forest@toget{after packing node}\forest@temp@keys
6357 \forest@process@hook@keylist@nodynamics{after packing node}{current}%
6358 \fi
6359 }
6360 % \forestset{recalculate tree boundary/.code={\forest@node@recalculate@edges}}
6361 % \def\forest@node@recalculate@edges{%
6362 % \edef\forest@marshal{%
6363 % \noexpand\forest@forthis{\noexpand\forest@node@getedges{\forestove{grow}}}%
6364 % }\forest@marshal
6365 % }
6366 \def\forest@pack@onlythisnode{%
6367 \ifnum\forestove{n children}>0
6368 \forest@pack@computetiers
6369 \forest@pack@level@nonuniform
6370 \forest@pack@aligtiers
6371 \forest@node@foreachchild{\forestoset{s}{0\pgfmath@pt}}%
6372 \forest@pack@sibling@uniform@applyreversed
6373 \fi
6374 }

```

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

```

6375 \def\forest@pack@computegrowthuniformity{%
6376 \forest@node@foreachchild{\forest@pack@computegrowthuniformity}%
6377 \edef\forest@pack@cgu@uniformity{%
6378 \ifnum\forestove{n children}=0
6379 2\else 1\fi

```

```

6380 }%
6381 \foresttoget{grow}\forest@pack@cgu@parentgrow
6382 \forest@node@foreachchild{%
6383   \ifnum\forestove{uniform growth}=0
6384     \def\forest@pack@cgu@uniformity{0}%
6385   \else
6386     \ifnum\forestove{uniform growth}=1
6387       \ifnum\forestove{grow}=\forest@pack@cgu@parentgrow\relax\else
6388         \def\forest@pack@cgu@uniformity{0}%
6389       \fi
6390     \fi
6391   \fi
6392 }%
6393 \foresttoget{before packing node}\forest@temp@a
6394 \foresttoget{after packing node}\forest@temp@b
6395 \expandafter\expandafter\expandafter\ifstrempty\expandafter\expandafter\expandafter{\expandafter\forest@temp@
6396   \forestolet{uniform growth}\forest@pack@cgu@uniformity
6397 }{%
6398   \forestoset{uniform growth}{0}%
6399 }%
6400 }

```

Pack children in the level dimension in a uniform tree.

```

6401 \def\forest@pack@level@uniform{%
6402   \let\forest@plu@minchildl\relax
6403   \foresttoget{grow}\forest@plu@grow
6404   \forest@node@foreachchild{%
6405     \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
6406     \advance\pgf@xa\forestove{1}\relax
6407     \ifx\forest@plu@minchildl\relax
6408       \edef\forest@plu@minchildl{\the\pgf@xa}%
6409     \else
6410       \ifdim\pgf@xa<\forest@plu@minchildl\relax
6411         \edef\forest@plu@minchildl{\the\pgf@xa}%
6412       \fi
6413     \fi
6414   }%
6415   \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
6416   \pgfutil@tempdima=\pgf@xb\relax
6417   \advance\pgfutil@tempdima-\forest@plu@minchildl\relax
6418   \advance\pgfutil@tempdima\forestove{1 sep}\relax
6419   \ifdim\pgfutil@tempdima>0pt
6420     \forest@node@foreachchild{%
6421       \forestoeset{1}{\the\dimexpr\forestove{1}+\the\pgfutil@tempdima}%
6422     }%
6423   \fi
6424   \forest@node@foreachchild{%
6425     \ifnum\forestove{n children}>0
6426       \forest@pack@level@uniform
6427     \fi
6428   }%
6429 }

```

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

```

6430 \def\forest@pack@level@nonuniform{%
6431   \let\forest@plu@minchildl\relax
6432   \foresttoget{grow}\forest@plu@grow
6433   \forest@node@foreachchild{%
6434     \forest@node@getedge{negative}{\forest@plu@grow}{\forest@plnu@negativechildedge}%
6435     \forest@node@getedge{positive}{\forest@plu@grow}{\forest@plnu@positivechildedge}%
6436     \def\forest@plnu@childedge{\forest@plnu@negativechildedge\forest@plnu@positivechildedge}%
6437     \forest@path@getboundingrectangle@ls\forest@plnu@childedge{\forest@plu@grow}%

```

```

6438 \advance\pgf@xa\forestove{1}\relax
6439 \ifx\forest@plu@minchildl\relax
6440 \edef\forest@plu@minchildl{\the\pgf@xa}%
6441 \else
6442 \ifdim\pgf@xa<\forest@plu@minchildl\relax
6443 \edef\forest@plu@minchildl{\the\pgf@xa}%
6444 \fi
6445 \fi
6446 }%
6447 \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
6448 \pgfutil@tempdima=\pgf@xb\relax
6449 \advance\pgfutil@tempdima-\forest@plu@minchildl\relax
6450 \advance\pgfutil@tempdima \forestove{1 sep}\relax
6451 \ifdim\pgfutil@tempdima>0pt
6452 \forest@node@foreachchild{%
6453 \forestoeset{1}{\the\dimexpr\the\pgfutil@tempdima+\forestove{1}}%
6454 }%
6455 \fi
6456 }

Align tiers.
6457 \def\forest@pack@aligntiers{%
6458 \forestoget{grow}\forest@temp@parentgrow
6459 \forestoget{@tiers}\forest@temp@tiers
6460 \forlistloop\forest@pack@aligtier@\forest@temp@tiers
6461 }
6462 \def\forest@pack@aligntiers@ofsubtree{%
6463 \forest@node@foreach{\forest@pack@aligntiers}%
6464 }
6465 \def\forest@pack@aligntiers@computeabsl{%
6466 \forestoleto{abs@1}{1}%
6467 \forest@node@foreachdescendant{\forest@pack@aligntiers@computeabsl}%
6468 }
6469 \def\forest@pack@aligntiers@computeabsl@{%
6470 \forestoeset{abs@1}{\the\dimexpr\forestove{1}+\forestove{\forestove{@parent}}{abs@1}}%
6471 }
6472 \def\forest@pack@aligtier@#1{%
6473 \forest@pack@aligntiers@computeabsl
6474 \pgfutil@tempdima=-\maxdimen\relax
6475 \def\forest@temp@currenttier{#1}%
6476 \forest@node@foreach{%
6477 \forestoget{tier}\forest@temp@tier
6478 \ifx\forest@temp@currenttier\forest@temp@tier
6479 \ifdim\pgfutil@tempdima<\forestove{abs@1}\relax
6480 \pgfutil@tempdima=\forestove{abs@1}\relax
6481 \fi
6482 \fi
6483 }%
6484 \ifdim\pgfutil@tempdima=-\maxdimen\relax\else
6485 \forest@node@foreach{%
6486 \forestoget{tier}\forest@temp@tier
6487 \ifx\forest@temp@currenttier\forest@temp@tier
6488 \forestoeset{1}{\the\dimexpr\pgfutil@tempdima-\forestove{abs@1}+\forestove{1}}%
6489 \fi
6490 }%
6491 \fi
6492 }

Pack children in the sibling dimension in a uniform tree: recursion.
6493 \def\forest@pack@sibling@uniform@recursive{%
6494 \forest@node@foreachchild{\forest@pack@sibling@uniform@recursive}%
6495 \forest@pack@sibling@uniform@applyreversed

```

6496 }

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

```
6497 \def\forest@pack@sibling@uniform@applyreversed{%
6498   \ifnum\forest@stove{n children}>1
6499     \ifnum\forest@stove{reversed}=0
6500       \forest@pack@sibling@uniform@main{first}{last}{next}{previous}%
6501     \else
6502       \forest@pack@sibling@uniform@main{last}{first}{previous}{next}%
6503     \fi
6504   \else
6505     \ifnum\forest@stove{n children}=1
```

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

```
6506     \csname forest@calign@\forest@stove{calign}\endcsname
6507   \fi
6508 \fi
6509 }
```

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

```
6510 \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.

```
6511   \forestoget{@#1}\forest@child
6512   \edef\forest@marshal{%
6513     \noexpand\forest@fornode{\forest@stove{@#1}}{%
6514       \noexpand\forest@node@getedge
6515         {positive}%
6516         {\forest@stove{grow}}%
6517       \noexpand\forest@temp@edge
6518     }%
6519   }\forest@marshal
6520   \forest@pack@pgfpoint@child@position\forest@child
6521   \let\forest@previous@positive@edge\pgfutil@empty
6522   \forest@extendpath\forest@previous@positive@edge\forest@temp@edge{%
6523     \forest@get{\forest@child}{@#3}\forest@child
```

Loop until the current child is the null node.

```
6524   \edef\forest@previous@child@s{0\pgfmath@pt}%
6525   \safeloop
6526   \unless\ifnum\forest@child=0
```

Get the negative edge of the child.

```
6527     \edef\forest@temp{%
6528       \noexpand\forest@fornode{\forest@child}{%
6529         \noexpand\forest@node@getedge
6530           {negative}%
6531           {\forest@stove{grow}}%
6532         \noexpand\forest@temp@edge
6533       }%
6534     }\forest@temp
```

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

```
6535     \forest@pack@pgfpoint@child@position\forest@child
```

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

```
6536     \let\forest@child@negative@edge\pgfutil@empty
6537     \forest@extendpath\forest@child@negative@edge\forest@temp@edge{%
```

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

```
6538 \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!)

```
6539 \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`.

```
6540 \ifx\forest@csdistance\relax
```

```
6541 \forest@set{\forest@child}{s}{\forest@previous@child@s}%
```

```
6542 \else
```

```
6543 \advance\pgfutil@tempdimb-\forest@csdistance\relax
```

```
6544 \advance\pgfutil@tempdimb\forestove{s_sep}\relax
```

```
6545 \forest@set{\forest@child}{s}{\the\dimexpr\forestove{\forest@child}{s}-\forest@csdistance+\forestove{s}
```

```
6546 \fi
```

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

```
6547 \ifdim\forestove{\forest@child}{s}<\forest@previous@child@s\relax
```

```
6548 \forest@set{\forest@child}{s}{\forest@previous@child@s}%
```

```
6549 \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.

```
6550 \forest@get{\forest@child}{s}\forest@child@s
```

```
6551 \edef\forest@previous@child@s{\forest@child@s}%
```

```
6552 \edef\forest@temp{%
```

```
6553 \noexpand\forest@fornode{\forest@child}{%
```

```
6554 \noexpand\forest@node@getedge
```

```
6555 {positive}%
```

```
6556 {\forestove{grow}}%
```

```
6557 \noexpand\forest@temp@edge
```

```
6558 }%
```

```
6559 }\forest@temp
```

```
6560 \forest@pack@pgfpoint@child@position\forest@child
```

```
6561 \forest@extendpath\forest@previous@positive@edge\forest@temp@edge-{}%
```

```
6562 \forest@getpositivetightedgeofpath\forest@previous@positive@edge\forest@previous@positive@edge
```

```
6563 \forest@get{\forest@child}{#3}\forest@child
```

```
6564 \saferepeat
```

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

```
6565 \csname forest@calign@\forestove{calign}\endcsname
```

```
6566 }
```

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

```
6567 \def\forest@pack@pgfpoint@child@position#1{%
```

```
6568 {%
```

```
6569 \pgftransformreset
```

```
6570 \forest@pgfqtransformrotate{\forestove{grow}}%
```

```
6571 \forest@fornode{#1}{%
```

```
6572 \pgfpointtransformed{\pgfqpoint{\forestove{1}}{\forestove{s}}}%
```

```
6573 }%
```

```
6574 }%
```

```
6575 }
```

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

```
6576 \def\forest@pack@pgfpoint@positioningrow#1{%
```

```
6577 {%
```

```
6578 \pgftransformreset
```

```
6579 \forest@pgfqtransformrotate{#1}%
```

```
6580 \pgfpointtransformed{\pgfqpoint{\forestove{1}}{\forestove{s}}}%
```

```
6581 }%
```

```
6582 }
```

Child alignment.

```

6583 \def\forest@calign@s@shift#1{%
6584   \pgfutil@tempdima=#1\relax
6585   \forest@node@foreachchild{%
6586     \forestoeset{s}{\the\dimexpr\forestove{s}+\pgfutil@tempdima}%
6587   }%
6588 }
6589 \def\forest@calign@child{%
6590   \forest@calign@s@shift{-\forestOve{\forest@node@nornbarthchildid{\forestove{calign primary child}}}{s}}%
6591 }
6592 \csdef{forest@calign@child edge}{%
6593   {%
6594     \edef\forest@temp@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
6595     \pgftransformreset
6596     \forest@pgfqtransformrotate{\forestove{grow}}%
6597     \pgfpointransformed{\pgfqpoint{\forestOve{\forest@temp@child}{1}}{\forestOve{\forest@temp@child}{s}}}%
6598     \pgf@xa=\pgf@x\relax\pgf@ya=\pgf@y\relax
6599     \forest@Pointanchor{\forest@temp@child}{child anchor}%
6600     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
6601     \forest@pointanchor{parent anchor}%
6602     \advance\pgf@xa-\pgf@x\relax\advance\pgf@ya-\pgf@y\relax
6603     \edef\forest@marshal{%
6604       \noexpand\pgftransformreset
6605       \noexpand\forest@pgfqtransformrotate{-\forestove{grow}}%
6606       \noexpand\pgfpointransformed{\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}%
6607     }\forest@marshal
6608   }%
6609   \forest@calign@s@shift{\the\dimexpr-\the\pgf@y}%
6610 }
6611 \csdef{forest@calign@midpoint}{%
6612   \forest@calign@s@shift{\the\dimexpr Opt -%
6613     (\forestOve{\forest@node@nornbarthchildid{\forestove{calign primary child}}}{s}%
6614     +\forestOve{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}{s}%
6615     )/2\relax
6616   }%
6617 }
6618 \csdef{forest@calign@edge midpoint}{%
6619   {%
6620     \edef\forest@temp@firstchild{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
6621     \edef\forest@temp@secondchild{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
6622     \pgftransformreset
6623     \forest@pgfqtransformrotate{\forestove{grow}}%
6624     \pgfpointransformed{\pgfqpoint{\forestOve{\forest@temp@firstchild}{1}}{\forestOve{\forest@temp@firstchild}{s}}}%
6625     \pgf@xa=\pgf@x\relax\pgf@ya=\pgf@y\relax
6626     \forest@Pointanchor{\forest@temp@firstchild}{child anchor}%
6627     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
6628     \edef\forest@marshal{%
6629       \noexpand\pgfpointransformed{\noexpand\pgfqpoint{\forestOve{\forest@temp@secondchild}{1}}{\forestOve{\forest@temp@secondchild}{s}}}%
6630     }\forest@marshal
6631     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
6632     \forest@Pointanchor{\forest@temp@secondchild}{child anchor}%
6633     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
6634     \divide\pgf@xa2 \divide\pgf@ya2
6635     \forest@pointanchor{parent anchor}%
6636     \advance\pgf@xa-\pgf@x\relax\advance\pgf@ya-\pgf@y\relax
6637     \edef\forest@marshal{%
6638       \noexpand\pgftransformreset
6639       \noexpand\forest@pgfqtransformrotate{-\forestove{grow}}%
6640       \noexpand\pgfpointransformed{\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}%
6641     }\forest@marshal

```

```

6642 }%
6643 \forest@calign@s@shift{\the\dimexpr-\the\pgf@y}%
6644 }

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.

6645 \def\forest@edef@strippt#1#2{%
6646 \edef#1{#2}%
6647 \edef#1{\expandafter\Pgf@geT#1}%
6648 }
6649 \csdef{forest@calign@fixed angles}{%
6650 \ifnum\forest@ve{n children}>1
6651 \edef\forest@ca@first@child{\forest@node@normbarthchildid{\forest@ve{calign primary child}}}%
6652 \edef\forest@ca@second@child{\forest@node@normbarthchildid{\forest@ve{calign secondary child}}}%
6653 \ifnum\forest@ve{reversed}=1
6654 \let\forest@temp\forest@ca@first@child
6655 \let\forest@ca@first@child\forest@ca@second@child
6656 \let\forest@ca@second@child\forest@temp
6657 \fi
6658 \forest@get{\forest@ca@first@child}{1}\forest@ca@first@l
6659 \edef\forest@ca@first@l{\expandafter\Pgf@geT\forest@ca@first@l}%
6660 \forest@get{\forest@ca@second@child}{1}\forest@ca@second@l
6661 \edef\forest@ca@second@l{\expandafter\Pgf@geT\forest@ca@second@l}%
6662 \pgfmathatan{\forest@ve{calign secondary angle}}%
6663 \pgfmathmultiply{\pgfmathresult}{\forest@ca@second@l}%
6664 \let\forest@calign@temp\pgfmathresult
6665 \pgfmathatan{\forest@ve{calign primary angle}}%
6666 \pgfmathmultiply{\pgfmathresult}{\forest@ca@first@l}%
6667 \edef\forest@ca@desired@s@distance{\the\dimexpr
6668 \forest@calign@temp pt-\pgfmathresult pt}%
6669 % \pgfmathsetlengthmacro\forest@ca@desired@s@distance{%
6670 % tan(\forest@ve{calign secondary angle})*\forest@ca@second@l
6671 % -tan(\forest@ve{calign primary angle})*\forest@ca@first@l
6672 % }%
6673 \forest@get{\forest@ca@first@child}{s}\forest@ca@first@s
6674 \forest@get{\forest@ca@second@child}{s}\forest@ca@second@s
6675 \edef\forest@ca@actual@s@distance{\the\dimexpr
6676 \forest@ca@second@s-\forest@ca@first@s}%
6677 % \pgfmathsetlengthmacro\forest@ca@actual@s@distance{%
6678 % \forest@ca@second@s-\forest@ca@first@s}%
6679 \ifdim\forest@ca@desired@s@distance>\forest@ca@actual@s@distance\relax
6680 \ifdim\forest@ca@actual@s@distance=0pt
6681 \pgfmathatan{\forest@ve{calign primary angle}}%
6682 \pgfmathmultiply{\pgfmathresult}{\forest@ca@second@l}%
6683 \pgfutil@tempdima=\pgfmathresult pt
6684 % \pgfmathsetlength\pgfutil@tempdima{tan(\forest@ve{calign primary angle})*\forest@ca@second@l}%
6685 \pgfutil@tempdimb=\dimexpr
6686 \forest@ca@desired@s@distance/(\forest@ve{n children}-1)\relax%
6687 %\pgfmathsetlength\pgfutil@tempdimb{\forest@ca@desired@s@distance/(\forest@ve{n children}-1)}%
6688 \forest@node@foreachchild{%
6689 \forest@set{s}{\the\pgfutil@tempdima}%
6690 \advance\pgfutil@tempdima\pgfutil@tempdimb
6691 }%
6692 \def\forest@calign@anchor{0pt}%
6693 \else
6694 \edef\forest@marshal{\noexpand\pgfmathdivide@
6695 {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
6696 {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
6697 }\forest@marshal

```



```

6698 \let\forest@ca@ratio\pgfmathresult
6699 %\pgfmathsetmacro\forest@ca@ratio{%
6700 % \forest@ca@desired@s@distance/\forest@ca@actual@s@distance}%
6701 \forest@node@foreachchild{%
6702 \forest@edef\strippt\forest@temp{\forestove{s}}%
6703 \pgfmathmultiply@{\forest@ca@ratio}{\forest@temp}%
6704 \forestoaset{s}{\the\dimexpr\pgfmathresult pt}%
6705 %\pgfmathsetlengthmacro\forest@temp{\forest@ca@ratio*\forestove{s}}%
6706 %\forestolet{s}\forest@temp
6707 }%
6708 \pgfmathatan@{\forestove{calign primary angle}}%
6709 \pgfmathmultiply@{\pgfmathresult}{\forest@ca@first@l}%
6710 \edef\forest@calign@anchor{\the\dimexpr-\pgfmathresult pt}%
6711 %\pgfmathsetlengthmacro\forest@calign@anchor{%
6712 % -tan(\forestove{calign primary angle})*\forest@ca@first@l}%
6713 \fi
6714 \else
6715 \ifdim\forest@ca@desired@s@distance<\forest@ca@actual@s@distance\relax
6716 \edef\forest@marshal{\noexpand\pgfmathdivide@
6717 {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
6718 {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
6719 }\forest@marshal
6720 \let\forest@ca@ratio\pgfmathresult
6721 %\pgfmathsetlengthmacro\forest@ca@ratio{%
6722 % \forest@ca@actual@s@distance/\forest@ca@desired@s@distance}%
6723 \forest@node@foreachchild{%
6724 \forest@edef\strippt\forest@temp{\forestove{l}}%
6725 \pgfmathmultiply@{\forest@ca@ratio}{\forest@temp}%
6726 \forestoaset{l}{\the\dimexpr\pgfmathresult pt}%
6727 %\pgfmathsetlengthmacro\forest@temp{\forest@ca@ratio*\forestove{l}}%
6728 %\forestolet{l}\forest@temp
6729 }%
6730 \forest@get{\forest@ca@first@child}{l}\forest@ca@first@l
6731 \edef\forest@ca@first@l{\expandafter\Pgf@geT\forest@ca@first@l}%
6732 \pgfmathatan@{\forestove{calign primary angle}}%
6733 \pgfmathmultiply@{\pgfmathresult}{\forest@ca@first@l}%
6734 \edef\forest@calign@anchor{\the\dimexpr-\pgfmathresult pt}%
6735 %\pgfmathsetlengthmacro\forest@calign@anchor{%
6736 % -tan(\forestove{calign primary angle})*\forest@ca@first@l}%
6737 \fi
6738 \fi
6739 \forest@calign@s@shift{-\forest@calign@anchor}%
6740 \fi
6741 }
6742 \csdef{forest@calign@fixed edge angles}{%
6743 \ifnum\forestove{n children}>1
6744 \edef\forest@ca@first@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
6745 \edef\forest@ca@second@child{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
6746 \ifnum\forestove{reversed}=1
6747 \let\forest@temp\forest@ca@first@child
6748 \let\forest@ca@first@child\forest@ca@second@child
6749 \let\forest@ca@second@child\forest@temp
6750 \fi
6751 \forest@get{\forest@ca@first@child}{l}\forest@ca@first@l
6752 \forest@get{\forest@ca@second@child}{l}\forest@ca@second@l
6753 \forest@pointanchor{parent anchor}%
6754 \edef\forest@ca@parent@anchor@s{\the\pgf@x}%
6755 \edef\forest@ca@parent@anchor@l{\the\pgf@y}%
6756 \forest@Pointanchor{\forest@ca@first@child}{child anchor}%
6757 \edef\forest@ca@first@child@anchor@s{\the\pgf@x}%
6758 \edef\forest@ca@first@child@anchor@l{\the\pgf@y}%

```

```

6759 \forest@Pointanchor{\forest@ca@second@child}{child anchor}%
6760 \edef\forest@ca@second@child@anchor@s{\the\pgf@x}%
6761 \edef\forest@ca@second@child@anchor@l{\the\pgf@y}%
6762 \pgfmathatan@{\forest@ve{calign secondary angle}}%
6763 \edef\forest@temp{\the\dimexpr
6764 \forest@ca@second@l-\forest@ca@second@child@anchor@l+\forest@ca@parent@anchor@l}%
6765 \pgfmathmultiply@{\pgfmathresult}{\expandafter\Pgf@geT\forest@temp}%
6766 \edef\forest@ca@desired@second@edge@s{\the\dimexpr\pgfmathresult pt}%
6767 %\pgfmathsetlengthmacro\forest@ca@desired@second@edge@s{%
6768 % \tan(\forest@ve{calign secondary angle})*%
6769 % (\forest@ca@second@l-\forest@ca@second@child@anchor@l+\forest@ca@parent@anchor@l)}%
6770 \pgfmathatan@{\forest@ve{calign primary angle}}%
6771 \edef\forest@temp{\the\dimexpr
6772 \forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@l}%
6773 \pgfmathmultiply@{\pgfmathresult}{\expandafter\Pgf@geT\forest@temp}%
6774 \edef\forest@ca@desired@first@edge@s{\the\dimexpr\pgfmathresult pt}%
6775 %\pgfmathsetlengthmacro\forest@ca@desired@first@edge@s{%
6776 % \tan(\forest@ve{calign primary angle})*%
6777 % (\forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@l)}%
6778 \edef\forest@ca@desired@s@distance{\the\dimexpr
6779 \forest@ca@desired@second@edge@s-\forest@ca@desired@first@edge@s}%
6780 %\pgfmathsetlengthmacro\forest@ca@desired@s@distance{\forest@ca@desired@second@edge@s-\forest@ca@desired@
6781 \forest@l@get{\forest@ca@first@child}{s}\forest@ca@first@s
6782 \forest@l@get{\forest@ca@second@child}{s}\forest@ca@second@s
6783 \edef\forest@ca@actual@s@distance{\the\dimexpr
6784 \forest@ca@second@s+\forest@ca@second@child@anchor@s
6785 -\forest@ca@first@s-\forest@ca@first@child@anchor@s}%
6786 %\pgfmathsetlengthmacro\forest@ca@actual@s@distance{%
6787 % \forest@ca@second@s+\forest@ca@second@child@anchor@s
6788 % -\forest@ca@first@s-\forest@ca@first@child@anchor@s}%
6789 \ifdim\forest@ca@desired@s@distance>\forest@ca@actual@s@distance\relax
6790 \ifdim\forest@ca@actual@s@distance=0pt
6791 \forest@toget{n children}\forest@temp@n@children
6792 \forest@node@foreachchild{%
6793 \forest@pointanchor{child anchor}%
6794 \edef\forest@temp@child@anchor@s{\the\pgf@x}%
6795 \forest@toeset{s}{\the\dimexpr
6796 \forest@ca@desired@first@edge@s+\forest@ca@desired@s@distance*(\forest@ve{n}-1)/(\forest@temp@n@
6797 %\pgfmathsetlengthmacro\forest@temp@{
6798 % \forest@ca@desired@first@edge@s+(\forest@ve{n}-1)*\forest@ca@desired@s@distance/(\forest@temp@n@
6799 %\forest@tolet{s}\forest@temp
6800 }%
6801 \def\forest@calign@anchor{0pt}%
6802 \else
6803 \edef\forest@marshal{\noexpand\pgfmathdivide@
6804 {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
6805 {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
6806 }\forest@marshal
6807 \let\forest@ca@ratio\pgfmathresult
6808 %\pgfmathsetmacro\forest@ca@ratio{%
6809 % \forest@ca@desired@s@distance/\forest@ca@actual@s@distance}%
6810 \forest@node@foreachchild{%
6811 \forest@pointanchor{child anchor}%
6812 \edef\forest@temp@child@anchor@s{\the\pgf@x}%
6813 \edef\forest@marshal{\noexpand\pgfmathmultiply@
6814 {\forest@ca@ratio}%
6815 {\expandafter\Pgf@geT\the\dimexpr
6816 \forest@ve{s}-\forest@ca@first@s+%
6817 \forest@temp@child@anchor@s-\forest@ca@first@child@anchor@s}%
6818 }\forest@marshal
6819 \forest@toeset{s}{\the\dimexpr\pgfmathresult pt+\forest@ca@first@s

```

```

6820     +\forest@ca@first@child@anchor@s-\forest@temp@child@anchor@s}%
6821     % \pgfmathsetlengthmacro\forest@temp{%
6822     %   \forest@ca@ratio*(%
6823     %     \forestove{s}-\forest@ca@first@s
6824     %     +\forest@temp@child@anchor@s-\forest@ca@first@child@anchor@s)%
6825     %   +\forest@ca@first@s
6826     %   +\forest@ca@first@child@anchor@s-\forest@temp@child@anchor@s}%
6827     % \forestolet{s}\forest@temp
6828   }%
6829   \pgfmathatan@{\forestove{calign primary angle}}%
6830   \edef\forest@marshal{\noexpand\pgfmathmultiply@
6831     {\pgfmathresult}%
6832     {\expandafter\Pgf@geT\the\dimexpr
6833       \forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@l}%
6834   }\forest@marshal
6835   \edef\forest@calign@anchor{\the\dimexpr
6836     -\pgfmathresult pt+\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s}%
6837   % \pgfmathsetlengthmacro\forest@calign@anchor{%
6838   %   -tan(\forestove{calign primary angle})*(\forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@l)%
6839   %   +\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s
6840   % }%
6841   \fi
6842 \else
6843   \ifdim\forest@ca@desired@s@distance<\forest@ca@actual@s@distance\relax
6844   \edef\forest@marshal{\noexpand\pgfmathdivide@
6845     {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
6846     {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
6847   }\forest@marshal
6848   \let\forest@ca@ratio\pgfmathresult
6849   %\pgfmathsetlengthmacro\forest@ca@ratio{%
6850   %   \forest@ca@actual@s@distance/\forest@ca@desired@s@distance}%
6851   \forest@node@foreachchild{%
6852     \forest@pointanchor{child anchor}%
6853     \edef\forest@temp@child@anchor@l{\the\pgf@y}%
6854     \edef\forest@marshal{\noexpand\pgfmathmultiply@
6855       {\forest@ca@ratio}%
6856       {\expandafter\Pgf@geT\the\dimexpr\forestove{l}+\forest@ca@parent@anchor@l-\forest@temp@child@anchor@l}%
6857     }\forest@marshal
6858     \forestoaset{l}{\the\dimexpr
6859       \pgfmathresult pt-\forest@ca@parent@anchor@l+\forest@temp@child@anchor@l}%
6860     % \pgfmathsetlengthmacro\forest@temp{%
6861     %   \forest@ca@ratio*(%
6862     %     \forestove{l}+\forest@ca@parent@anchor@l-\forest@temp@child@anchor@l)
6863     %     -\forest@ca@parent@anchor@l+\forest@temp@child@anchor@l}%
6864     % \forestolet{l}\forest@temp
6865   }%
6866   \forest@get{\forest@ca@first@child}{l}\forest@ca@first@l
6867   \pgfmathatan@{\forestove{calign primary angle}}%
6868   \edef\forest@marshal{\noexpand\pgfmathmultiply@
6869     {\pgfmathresult}%
6870     {\expandafter\Pgf@geT\the\dimexpr
6871       \forest@ca@first@l+\forest@ca@parent@anchor@l-\forest@temp@child@anchor@l}%
6872   }\forest@marshal
6873   \edef\forest@calign@anchor{\the\dimexpr
6874     -\pgfmathresult pt+\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s}%
6875   % \pgfmathsetlengthmacro\forest@calign@anchor{%
6876   %   -tan(\forestove{calign primary angle})*(\forest@ca@first@l+\forest@ca@parent@anchor@l-\forest@temp@child@anchor@l)%
6877   %   +\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s
6878   % }%
6879   \fi
6880 \fi

```

```

6881 \forest@calign@s@shift{-\forest@calign@anchor}%
6882 \fi
6883 }

```

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.

```

6884 \def\forest@node@getedge#1#2#3{%
6885 \forestoget{#1@edge@#2}#3%
6886 \ifx#3\relax
6887 \forest@node@foreachchild{%
6888 \forest@node@getedge{#1}{#2}{\forest@temp@edge}%
6889 }%
6890 \forest@forthis{\forest@node@getedges{#2}}%
6891 \forestoget{#1@edge@#2}#3%
6892 \fi
6893 }

```

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.

```

6894 \def\forest@node@getedges#1{%

```

Run the computation in a T_EX group.

```

6895 %{}

```

Setup the grow line.

```

6896 \forest@setupgrowline{#1}%

```

Get the edge of the node itself.

```

6897 \ifnum\forestove{ignore}=0
6898 \forestoget{@boundary}\forest@node@boundary
6899 \else
6900 \def\forest@node@boundary{}%
6901 \fi
6902 \csname forest@getboth\forestove{fit}edgesofpath\endcsname
6903 \forest@node@boundary\forest@negative@node@edge\forest@positive@node@edge
6904 \forestolet{negative@edge@#1}\forest@negative@node@edge
6905 \forestolet{positive@edge@#1}\forest@positive@node@edge

```

Add the edges of the children.

```

6906 \forest@get@edges@merge{negative}{#1}%
6907 \forest@get@edges@merge{positive}{#1}%
6908 }{}%
6909 }

```

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

```

6910 \def\forest@get@edges@merge#1#2{%
6911 \ifnum\forestove{n children}>0
6912 \forestoget{#1@edge@#2}\forest@node@edge

```

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

```

6913 \forest@pointanchor{parent anchor}%
6914 \edef\forest@getedge@pa1{\the\pgf@x}%
6915 \edef\forest@getedge@pa{s}{\the\pgf@y}%
6916 \eappto\forest@node@edge{\noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@pa1}{\forest@getedge@pa@s}}

```

Switch to this node's (1,s) coordinate system (origin at the node's anchor).

```

6917 \pgfgettransform\forest@temp@transform
6918 \pgftransformreset
6919 \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.

```

6920 \def\forest@all@edges{%
6921 \forest@node@foreachchild{%
6922 \forest@toget{#1@edge@#2}\forest@temp@edge
6923 \pgfpointtransformed{\pgfpoint{\forest@ve{1}}{\forest@ve{s}}}%
6924 \forest@extendpath\forest@node@edge\forest@temp@edge}%
6925 \ifnum\forest@ve{ignore edge}=0
6926 \pgfpointadd
6927 {\pgfpointtransformed{\pgfpoint{\forest@ve{1}}{\forest@ve{s}}}}%
6928 {\forest@pointanchor{child anchor}}%
6929 \pgfgetlastxy{\forest@getedge@ca@1}{\forest@getedge@ca@s}%
6930 \eappto\forest@all@edges{%
6931 \noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@pa@1}{\forest@getedge@pa@s}%
6932 \noexpand\pgfsyssoftpath@linetotoken{\forest@getedge@ca@1}{\forest@getedge@ca@s}%
6933 }%
6934 % this deals with potential overlap of the edges:
6935 \eappto\forest@node@edge{\noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@ca@1}{\forest@getedge@ca@s}}%
6936 \fi
6937 }%
6938 \ifdefempty{\forest@all@edges}{-}%
6939 \pgfintersectionofpaths{\pgfsetpath\forest@all@edges}{\pgfsetpath\forest@node@edge}%
6940 \def\forest@edgenode@intersections{%
6941 \forest@merge@intersectionloop
6942 \eappto\forest@node@edge{\expandonce{\forest@all@edges}\expandonce{\forest@edgenode@intersections}}%
6943 }%
6944 \pgfsettransform\forest@temp@transform

```

Process the path into an edge and store the edge.

```

6945 \csname forest@get#1\forest@ve{fit}edgeofpath\endcsname\forest@node@edge\forest@node@edge
6946 \forest@let{#1@edge@#2}\forest@node@edge
6947 \fi
6948 }
6949 %\newloop\forest@merge@loop
6950 \def\forest@merge@intersectionloop{%
6951 \c@pgf@counta=0
6952 \forest@loop
6953 \ifnum\c@pgf@counta<\pgfintersectionsolutions\relax
6954 \advance\c@pgf@counta1
6955 \pgfpointintersectionsolution{\the\c@pgf@counta}%
6956 \eappto\forest@edgenode@intersections{\noexpand\pgfsyssoftpath@movetotoken
6957 {\the\pgf@x}{\the\pgf@y}}%
6958 \forest@repeat
6959 }

```

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

```

6960 \def\forest@node@getboundingrectangle@ls#1{%
6961 \forest@toget{@boundary}\forest@node@boundary
6962 \forest@path@getboundingrectangle@ls\forest@node@boundary{#1}%
6963 }

```

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).

```

6964 \def\forest@pgfpathtransformed#1{%
6965 \forest@save@pgfsyssoftpath@tokendefs
6966 \let\pgfsyssoftpath@movetotoken\forest@pgfpathtransformed@moveto
6967 \let\pgfsyssoftpath@linetotoken\forest@pgfpathtransformed@lineto
6968 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
6969 #1%
6970 \forest@restore@pgfsyssoftpath@tokendefs
6971 }

```

```

6972 \def\forest@pgfpathtransformed@moveto#1#2{%
6973   \forest@pgfpathtransformed@op\pgfsyssoftpath@moveto{#1}{#2}%
6974 }
6975 \def\forest@pgfpathtransformed@lineto#1#2{%
6976   \forest@pgfpathtransformed@op\pgfsyssoftpath@lineto{#1}{#2}%
6977 }
6978 \def\forest@pgfpathtransformed@op#1#2#3{%
6979   \pgfpointtransformed{\pgfqpoint{#2}{#3}}%
6980   \edef\forest@temp{%
6981     \noexpand#1{\the\pgf@x}{\the\pgf@y}%
6982   }%
6983   \forest@temp
6984 }

```

8.2.1 Tiers

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

```

6985 \def\forest@pack@computetiers{%
6986   {%
6987     \forest@pack@tiers@getalltiersinsubtree
6988     \forest@pack@tiers@computetierhierarchy
6989     \forest@pack@tiers@findcontainers
6990     \forest@pack@tiers@raisecontainers
6991     \forest@pack@tiers@computeprocessingorder
6992     \gdef\forest@smuggle{}%
6993     \forest@pack@tiers@write
6994   }%
6995   \forest@node@foreach{\forest@set{@tiers}{}}%
6996   \forest@smuggle
6997 }

```

Puts all tiers contained in the subtree into attribute tiers.

```

6998 \def\forest@pack@tiers@getalltiersinsubtree{%
6999   \ifnum\forest@ve{n children}>0
7000     \forest@node@foreachchild{\forest@pack@tiers@getalltiersinsubtree}%
7001     \fi
7002     \forest@get{tier}\forest@temp@mytier
7003     \def\forest@temp@mytiers{}%
7004     \ifdefempty\forest@temp@mytier{}{%
7005       \listadd\forest@temp@mytiers\forest@temp@mytier
7006     }%
7007     \ifnum\forest@ve{n children}>0
7008       \forest@node@foreachchild{%
7009         \forest@get{tiers}\forest@temp@tiers
7010         \forlistloop\forest@pack@tiers@forhandlerA\forest@temp@tiers
7011       }%
7012     \fi
7013     \forest@let{tiers}\forest@temp@mytiers
7014 }
7015 \def\forest@pack@tiers@forhandlerA#1{%
7016   \ifinlist{#1}\forest@temp@mytiers{}{%
7017     \listadd\forest@temp@mytiers{#1}%
7018   }%
7019 }

```

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.

```

7020 \def\forest@pack@tiers@computetierhierarchy{%
7021   \def\forest@tiers@ancestors{}%
7022   \forest@get{tiers}\forest@temp@mytiers
7023   \forlistloop\forest@pack@tiers@cth@init\forest@temp@mytiers

```

```

7024 \forest@pack@tiers@computetierhierarchy@
7025 }
7026 \def\forest@pack@tiers@cth@init#1{%
7027 \csdef{forest@tiers@higher@#1}{}%
7028 \csdef{forest@tiers@lower@#1}{}%
7029 }
7030 \def\forest@pack@tiers@computetierhierarchy@{%
7031 \foresttoget{tier}\forest@temp@mytier
7032 \ifdefempty\forest@temp@mytier}{-%
7033 \forlistloop\forest@pack@tiers@forhandlerB\forest@tiers@ancestors
7034 \listead\forest@tiers@ancestors\forest@temp@mytier
7035 }%
7036 \forest@node@foreachchild{%
7037 \forest@pack@tiers@computetierhierarchy@
7038 }%
7039 \foresttoget{tier}\forest@temp@mytier
7040 \ifdefempty\forest@temp@mytier}{-%
7041 \forest@listedel\forest@tiers@ancestors\forest@temp@mytier
7042 }%
7043 }
7044 \def\forest@pack@tiers@forhandlerB#1{%
7045 \def\forest@temp@tier{#1}%
7046 \ifx\forest@temp@tier\forest@temp@mytier
7047 \PackageError{forest}{Circular tier hierarchy (tier \forest@temp@mytier)}{-%
7048 \fi
7049 \ifinlistcs{#1}{forest@tiers@higher@\forest@temp@mytier}{-%
7050 \listcsadd{forest@tiers@higher@\forest@temp@mytier}{#1}}%
7051 \xifinlistcs\forest@temp@mytier{forest@tiers@lower@#1}{-%
7052 \listcseadd{forest@tiers@lower@#1}{\forest@temp@mytier}}%
7053 }
7054 \def\forest@pack@tiers@findcontainers{%
7055 \foresttoget{tiers}\forest@temp@tiers
7056 \forlistloop\forest@pack@tiers@findcontainer\forest@temp@tiers
7057 }
7058 \def\forest@pack@tiers@findcontainer#1{%
7059 \def\forest@temp@tier{#1}%
7060 \foresttoget{tier}\forest@temp@mytier
7061 \ifx\forest@temp@tier\forest@temp@mytier
7062 \csedef{forest@tiers@container@#1}{\forest@cn}%
7063 \else\@escapeif{%
7064 \forest@pack@tiers@findcontainerA{#1}%
7065 }\fi%
7066 }
7067 \def\forest@pack@tiers@findcontainerA#1{%
7068 \c@pgf@counta=0
7069 \forest@node@foreachchild{%
7070 \foresttoget{tiers}\forest@temp@tiers
7071 \ifinlist{#1}\forest@temp@tiers{%
7072 \advance\c@pgf@counta 1
7073 \let\forest@temp@child\forest@cn
7074 }{}}%
7075 }%
7076 \ifnum\c@pgf@counta>1
7077 \csedef{forest@tiers@container@#1}{\forest@cn}%
7078 \else\@escapeif% surely =1
7079 \forest@fornode{\forest@temp@child}{-%
7080 \forest@pack@tiers@findcontainer{#1}%
7081 }%
7082 }\fi
7083 }
7084 \def\forest@pack@tiers@raisecontainers{%

```

```

7085 \foresttoget{tiers}\forest@temp@mytiers
7086 \forlistloop\forest@pack@tiers@rc@forhandlerA\forest@temp@mytiers
7087 }
7088 \def\forest@pack@tiers@rc@forhandlerA#1{%
7089 \edef\forest@tiers@temptier{#1}%
7090 \letcs\forest@tiers@containernodeoftier{forest@tiers@container@#1}%
7091 \letcs\forest@temp@lowertiers{forest@tiers@lower@#1}%
7092 \forlistloop\forest@pack@tiers@rc@forhandlerB\forest@temp@lowertiers
7093 }
7094 \def\forest@pack@tiers@rc@forhandlerB#1{%
7095 \letcs\forest@tiers@containernodeoflowertier{forest@tiers@container@#1}%
7096 \forest@get{\forest@tiers@containernodeoflowertier}{content}\lowercontent
7097 \forest@get{\forest@tiers@containernodeoftier}{content}\uppercontent
7098 \forest@fornode{\forest@tiers@containernodeoflowertier}{%
7099 \forest@ifancestorof
7100 {\forest@tiers@containernodeoftier}
7101 {\csletcs{forest@tiers@container@forest@tiers@temptier}{forest@tiers@container@#1}}%
7102 {}%
7103 }%
7104 }
7105 \def\forest@pack@tiers@computeprocessingorder{%
7106 \def\forest@tiers@processingorder{}%
7107 \foresttoget{tiers}\forest@tiers@cpo@tierstodo
7108 \safeloop
7109 \ifdefempty\forest@tiers@cpo@tierstodo{\forest@tempfalse}{\forest@temptrue}%
7110 \ifforest@temp
7111 \def\forest@tiers@cpo@tiersremaining{}%
7112 \def\forest@tiers@cpo@tiersindependent{}%
7113 \forlistloop\forest@pack@tiers@cpo@forhandlerA\forest@tiers@cpo@tierstodo
7114 \ifdefempty\forest@tiers@cpo@tiersindependent{%
7115 \PackageError{forest}{Circular tiers!}{}}{}%
7116 \forlistloop\forest@pack@tiers@cpo@forhandlerB\forest@tiers@cpo@tiersremaining
7117 \let\forest@tiers@cpo@tierstodo\forest@tiers@cpo@tiersremaining
7118 \saferepeat
7119 }
7120 \def\forest@pack@tiers@cpo@forhandlerA#1{%
7121 \ifcempty{forest@tiers@higher@#1}{%
7122 \listadd\forest@tiers@cpo@tiersindependent{#1}%
7123 \listadd\forest@tiers@processingorder{#1}%
7124 }{}%
7125 \listadd\forest@tiers@cpo@tiersremaining{#1}%
7126 }%
7127 }
7128 \def\forest@pack@tiers@cpo@forhandlerB#1{%
7129 \def\forest@pack@tiers@cpo@aremainingtier{#1}%
7130 \forlistloop\forest@pack@tiers@cpo@forhandlerC\forest@tiers@cpo@tiersindependent
7131 }
7132 \def\forest@pack@tiers@cpo@forhandlerC#1{%
7133 \ifinlistcs{#1}{forest@tiers@higher@\forest@pack@tiers@cpo@aremainingtier}{%
7134 \forest@listcsdel{forest@tiers@higher@\forest@pack@tiers@cpo@aremainingtier}{#1}%
7135 }{}%
7136 }
7137 \def\forest@pack@tiers@write{%
7138 \forlistloop\forest@pack@tiers@write@forhandler\forest@tiers@processingorder
7139 }
7140 \def\forest@pack@tiers@write@forhandler#1{%
7141 \forest@fornode{\cename forest@tiers@container@#1\endcename}{%
7142 \forest@pack@tiers@check{#1}%
7143 }%
7144 \xappto\forest@smuggle{%
7145 \noexpand\listadd

```



```

7146     \forestOm{\csname forest@tiers@container@#1\endcsname}{@tiers}%
7147     {#1}%
7148 }%
7149 }
7150 % checks if the tier is compatible with growth changes and calign=node/edge angle
7151 \def\forest@pack@tiers@check#1{%
7152   \def\forest@temp@currenttier{#1}%
7153   \forest@node@foreachdescendant{%
7154     \ifnum\forestove{grow}=\forestOve{\forestove{@parent}}{grow}
7155     \else
7156       \forest@pack@tiers@check@grow
7157     \fi
7158     \ifnum\forestove{n children}>1
7159       \foresttoget{calign}\forest@temp
7160       \ifx\forest@temp\forest@pack@tiers@check@nodeangle
7161         \forest@pack@tiers@check@calign
7162       \fi
7163       \ifx\forest@temp\forest@pack@tiers@check@edgeangle
7164         \forest@pack@tiers@check@calign
7165       \fi
7166     \fi
7167   }%
7168 }
7169 \def\forest@pack@tiers@check@nodeangle{node angle}%
7170 \def\forest@pack@tiers@check@edgeangle{edge angle}%
7171 \def\forest@pack@tiers@check@grow{%
7172   \foresttoget{content}\forest@temp@content
7173   \let\forest@temp@currentnode\forest@cn
7174   \forest@node@foreachdescendant{%
7175     \foresttoget{tier}\forest@temp
7176     \ifx\forest@temp@currenttier\forest@temp
7177       \forest@pack@tiers@check@grow@error
7178     \fi
7179   }%
7180 }
7181 \def\forest@pack@tiers@check@grow@error{%
7182   \PackageError{forest}{Tree growth direction changes in node \forest@temp@currentnode\space
7183     (content: \forest@temp@content), while tier '\forest@temp' is specified for nodes both
7184     out- and inside the subtree rooted in node \forest@temp@currentnode. This will not work.}{}%
7185 }
7186 \def\forest@pack@tiers@check@calign{%
7187   \forest@node@foreachchild{%
7188     \foresttoget{tier}\forest@temp
7189     \ifx\forest@temp@currenttier\forest@temp
7190       \forest@pack@tiers@check@calign@warning
7191     \fi
7192   }%
7193 }
7194 \def\forest@pack@tiers@check@calign@warning{%
7195   \PackageWarning{forest}{Potential option conflict: node \forestove{@parent} (content:
7196     '\forestOve{\forestove{@parent}}{content}') was given 'calign=\forestove{calign}', while its
7197     child \forest@cn\space (content: '\forestove{content}') was given 'tier=\forestove{tier}'.
7198     The parent's 'calign' will only work if the child was the lowest node on its tier before the
7199     alignment.}%
7200 }

```

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.

```

7201 \pgfdeclaregenericanchor{forestcomputenodeboundary}{%
7202 \letcs\forest@temp@boundary@macro{forest@compute@node@boundary@#1}%
7203 \ifcsname forest@compute@node@boundary@#1\endcsname
7204 \csname forest@compute@node@boundary@#1\endcsname
7205 \else
7206 \forest@compute@node@boundary@rectangle
7207 \fi
7208 \pgfsyssoftpath@getcurrentpath\forest@temp
7209 \global\let\forest@global@boundary\forest@temp
7210 }
7211 \def\forest@mt#1{%
7212 \expandafter\pgfpointanchor\expandafter{\pgfreferencednodename}{#1}%
7213 \pgfsyssoftpath@moveto{\the\pgf@x}{\the\pgf@y}%
7214 }%
7215 \def\forest@lt#1{%
7216 \expandafter\pgfpointanchor\expandafter{\pgfreferencednodename}{#1}%
7217 \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
7218 }%
7219 \def\forest@compute@node@boundary@coordinate{%
7220 \forest@mt{center}}%
7221 }
7222 \def\forest@compute@node@boundary@circle{%
7223 \forest@mt{east}}%
7224 \forest@lt{north east}}%
7225 \forest@lt{north}}%
7226 \forest@lt{north west}}%
7227 \forest@lt{west}}%
7228 \forest@lt{south west}}%
7229 \forest@lt{south}}%
7230 \forest@lt{south east}}%
7231 \forest@lt{east}}%
7232 }
7233 \def\forest@compute@node@boundary@rectangle{%
7234 \forest@mt{south west}}%
7235 \forest@lt{south east}}%
7236 \forest@lt{north east}}%
7237 \forest@lt{north west}}%
7238 \forest@lt{south west}}%
7239 }
7240 \def\forest@compute@node@boundary@diamond{%
7241 \forest@mt{east}}%
7242 \forest@lt{north}}%
7243 \forest@lt{west}}%
7244 \forest@lt{south}}%
7245 \forest@lt{east}}%
7246 }
7247 \let\forest@compute@node@boundary@ellipse\forest@compute@node@boundary@circle
7248 \def\forest@compute@node@boundary@trapezium{%
7249 \forest@mt{top right corner}}%
7250 \forest@lt{top left corner}}%
7251 \forest@lt{bottom left corner}}%
7252 \forest@lt{bottom right corner}}%
7253 \forest@lt{top right corner}}%
7254 }
7255 \def\forest@compute@node@boundary@semicircle{%
7256 \forest@mt{arc start}}%
7257 \forest@lt{north}}%
7258 \forest@lt{east}}%
7259 \forest@lt{north east}}%
7260 \forest@lt{apex}}%
7261 \forest@lt{north west}}%

```

```

7262 \forest@lt{west}%
7263 \forest@lt{arc end}%
7264 \forest@lt{arc start}%
7265 }
7266 %\newloop\forest@computenodeboundary@loop
7267 \csdef{forest@compute@node@boundary@regular polygon}{%
7268 \forest@mt{corner 1}%
7269 \c@pgf@counta=\sides\relax
7270 \forest@loop
7271 \ifnum\c@pgf@counta>0
7272 \forest@lt{corner \the\c@pgf@counta}%
7273 \advance\c@pgf@counta-1
7274 \forest@repeat
7275 }%
7276 \def\forest@compute@node@boundary@star{%
7277 \forest@mt{outer point 1}%
7278 \c@pgf@counta=\totalstarpoints\relax
7279 \divide\c@pgf@counta2
7280 \forest@loop
7281 \ifnum\c@pgf@counta>0
7282 \forest@lt{inner point \the\c@pgf@counta}%
7283 \forest@lt{outer point \the\c@pgf@counta}%
7284 \advance\c@pgf@counta-1
7285 \forest@repeat
7286 }%
7287 \csdef{forest@compute@node@boundary@isosceles triangle}{%
7288 \forest@mt{apex}%
7289 \forest@lt{left corner}%
7290 \forest@lt{right corner}%
7291 \forest@lt{apex}%
7292 }
7293 \def\forest@compute@node@boundary@kite{%
7294 \forest@mt{upper vertex}%
7295 \forest@lt{left vertex}%
7296 \forest@lt{lower vertex}%
7297 \forest@lt{right vertex}%
7298 \forest@lt{upper vertex}%
7299 }
7300 \def\forest@compute@node@boundary@dart{%
7301 \forest@mt{tip}%
7302 \forest@lt{left tail}%
7303 \forest@lt{tail center}%
7304 \forest@lt{right tail}%
7305 \forest@lt{tip}%
7306 }
7307 \csdef{forest@compute@node@boundary@circular sector}{%
7308 \forest@mt{sector center}%
7309 \forest@lt{arc start}%
7310 \forest@lt{arc center}%
7311 \forest@lt{arc end}%
7312 \forest@lt{sector center}%
7313 }
7314 \def\forest@compute@node@boundary@cylinder{%
7315 \forest@mt{top}%
7316 \forest@lt{after top}%
7317 \forest@lt{before bottom}%
7318 \forest@lt{bottom}%
7319 \forest@lt{after bottom}%
7320 \forest@lt{before top}%
7321 \forest@lt{top}%
7322 }

```

```

7323 \cslet{forest@compute@node@boundary@forbidden sign}\forest@compute@node@boundary@circle
7324 \cslet{forest@compute@node@boundary@magnifying glass}\forest@compute@node@boundary@circle
7325 \def\forest@compute@node@boundary@cloud{%
7326   \getradii
7327   \forest@mt{puff 1}%
7328   \c@pgf@counta=\puffs\relax
7329   \forest@loop
7330   \ifnum\c@pgf@counta>0
7331     \forest@lt{puff \the\c@pgf@counta}%
7332     \advance\c@pgf@counta-1
7333   \forest@repeat
7334 }
7335 \def\forest@compute@node@boundary@starburst{
7336   \calculatestarburstpoints
7337   \forest@mt{outer point 1}%
7338   \c@pgf@counta=\totalpoints\relax
7339   \divide\c@pgf@counta2
7340   \forest@loop
7341   \ifnum\c@pgf@counta>0
7342     \forest@lt{inner point \the\c@pgf@counta}%
7343     \forest@lt{outer point \the\c@pgf@counta}%
7344     \advance\c@pgf@counta-1
7345   \forest@repeat
7346 }%
7347 \def\forest@compute@node@boundary@signal{%
7348   \forest@mt{east}%
7349   \forest@lt{south east}%
7350   \forest@lt{south west}%
7351   \forest@lt{west}%
7352   \forest@lt{north west}%
7353   \forest@lt{north east}%
7354   \forest@lt{east}%
7355 }
7356 \def\forest@compute@node@boundary@tape{%
7357   \forest@mt{north east}%
7358   \forest@lt{60}%
7359   \forest@lt{north}%
7360   \forest@lt{120}%
7361   \forest@lt{north west}%
7362   \forest@lt{south west}%
7363   \forest@lt{240}%
7364   \forest@lt{south}%
7365   \forest@lt{310}%
7366   \forest@lt{south east}%
7367   \forest@lt{north east}%
7368 }
7369 \csdef{forest@compute@node@boundary@single arrow}{%
7370   \forest@mt{tip}%
7371   \forest@lt{after tip}%
7372   \forest@lt{after head}%
7373   \forest@lt{before tail}%
7374   \forest@lt{after tail}%
7375   \forest@lt{before head}%
7376   \forest@lt{before tip}%
7377   \forest@lt{tip}%
7378 }
7379 \csdef{forest@compute@node@boundary@double arrow}{%
7380   \forest@mt{tip 1}%
7381   \forest@lt{after tip 1}%
7382   \forest@lt{after head 1}%
7383   \forest@lt{before head 2}%

```

```

7384 \forest@lt{before tip 2}%
7385 \forest@mt{tip 2}%
7386 \forest@lt{after tip 2}%
7387 \forest@lt{after head 2}%
7388 \forest@lt{before head 1}%
7389 \forest@lt{before tip 1}%
7390 \forest@lt{tip 1}%
7391 }
7392 \csdef{forest@compute@node@boundary@arrow box}{%
7393 \forest@mt{before north arrow}%
7394 \forest@lt{before north arrow head}%
7395 \forest@lt{before north arrow tip}%
7396 \forest@lt{north arrow tip}%
7397 \forest@lt{after north arrow tip}%
7398 \forest@lt{after north arrow head}%
7399 \forest@lt{after north arrow}%
7400 \forest@lt{north east}%
7401 \forest@lt{before east arrow}%
7402 \forest@lt{before east arrow head}%
7403 \forest@lt{before east arrow tip}%
7404 \forest@lt{east arrow tip}%
7405 \forest@lt{after east arrow tip}%
7406 \forest@lt{after east arrow head}%
7407 \forest@lt{after east arrow}%
7408 \forest@lt{south east}%
7409 \forest@lt{before south arrow}%
7410 \forest@lt{before south arrow head}%
7411 \forest@lt{before south arrow tip}%
7412 \forest@lt{south arrow tip}%
7413 \forest@lt{after south arrow tip}%
7414 \forest@lt{after south arrow head}%
7415 \forest@lt{after south arrow}%
7416 \forest@lt{south west}%
7417 \forest@lt{before west arrow}%
7418 \forest@lt{before west arrow head}%
7419 \forest@lt{before west arrow tip}%
7420 \forest@lt{west arrow tip}%
7421 \forest@lt{after west arrow tip}%
7422 \forest@lt{after west arrow head}%
7423 \forest@lt{after west arrow}%
7424 \forest@lt{north west}%
7425 \forest@lt{before north arrow}%
7426 }
7427 \cslet{forest@compute@node@boundary@circle split}\forest@compute@node@boundary@circle
7428 \cslet{forest@compute@node@boundary@circle solidus}\forest@compute@node@boundary@circle
7429 \cslet{forest@compute@node@boundary@ellipse split}\forest@compute@node@boundary@ellipse
7430 \cslet{forest@compute@node@boundary@rectangle split}\forest@compute@node@boundary@rectangle
7431 \def\forest@compute@node@boundary@@callout{%
7432 \beforecalloutpointer
7433 \pgfsyssoftpath@moveto{\the\pgf@x}{\the\pgf@y}%
7434 \calloutpointeranchor
7435 \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
7436 \aftercalloutpointer
7437 \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
7438 }
7439 \csdef{forest@compute@node@boundary@rectangle callout}{%
7440 \forest@compute@node@boundary@rectangle
7441 \rectanglecalloutpoints
7442 \forest@compute@node@boundary@@callout
7443 }
7444 \csdef{forest@compute@node@boundary@ellipse callout}{%

```

```

7445 \forest@compute@node@boundary@ellipse
7446 \ellipsecalloutpoints
7447 \forest@compute@node@boundary@@callout
7448 }
7449 \csdef{forest@compute@node@boundary@cloud callout}{%
7450 \forest@compute@node@boundary@cloud
7451 % at least a first approx...
7452 \forest@mt{center}%
7453 \forest@lt{pointer}%
7454 }%
7455 \csdef{forest@compute@node@boundary@cross out}{%
7456 \forest@mt{south east}%
7457 \forest@lt{north west}%
7458 \forest@mt{south west}%
7459 \forest@lt{north east}%
7460 }%
7461 \csdef{forest@compute@node@boundary@strike out}{%
7462 \forest@mt{north east}%
7463 \forest@lt{south west}%
7464 }%
7465 \csdef{forest@compute@node@boundary@rounded rectangle}{%
7466 \forest@mt{east}%
7467 \forest@lt{north east}%
7468 \forest@lt{north}%
7469 \forest@lt{north west}%
7470 \forest@lt{west}%
7471 \forest@lt{south west}%
7472 \forest@lt{south}%
7473 \forest@lt{south east}%
7474 \forest@lt{east}%
7475 }%
7476 \csdef{forest@compute@node@boundary@chamfered rectangle}{%
7477 \forest@mt{before south west}%
7478 \forest@mt{after south west}%
7479 \forest@lt{before south east}%
7480 \forest@lt{after south east}%
7481 \forest@lt{before north east}%
7482 \forest@lt{after north east}%
7483 \forest@lt{before north west}%
7484 \forest@lt{after north west}%
7485 \forest@lt{before south west}%
7486 }%

```

8.3 Compute absolute positions

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

```

7487 \def\forest@node@computeabsolutepositions{%
7488 \edef\forest@marshal{%
7489 \noexpand\forest@node@foreachchild{%
7490 \noexpand\forest@node@computeabsolutepositions@{\forestove{x}}{\forestove{y}}{\forestove{grow}}%
7491 }%
7492 }\forest@marshal
7493 }
7494 \def\forest@node@computeabsolutepositions@#1#2#3{%
7495 \pgfpointadd
7496 {\pgfpoint{#1}{#2}}%
7497 {\pgfpointadd
7498 {\pgfpointpolar{#3}{\forestove{1}}}%
7499 {\pgfpointpolar{\numexpr 90+#3\relax}{\forestove{s}}}%

```

```

7500 }%
7501 \pgfgetlastxy\forest@temp@x\forest@temp@y
7502 \forestolet{x}\forest@temp@x
7503 \forestolet{y}\forest@temp@y
7504 \edef\forest@marshal{%
7505   \noexpand\forest@node@foreachchild{%
7506     \noexpand\forest@node@computeabsolutepositions@{\forest@temp@x}{\forest@temp@y}{\forestove{grow}}}%
7507   }%
7508 }\forest@marshal
7509 }

```

8.4 Drawing the tree

```

7510 \newif\ifforest@drawtree@preservenodeboxes@
7511 \def\forest@node@drawtree{%
7512   \expandafter\ifstrequal\expandafter{\forest@drawtreebox}{\pgfkeysnovalue}{%
7513     \let\forest@drawtree@beginbox\relax
7514     \let\forest@drawtree@endbox\relax
7515   }{%
7516     \edef\forest@drawtree@beginbox{\global\setbox\forest@drawtreebox=\hbox\bgroup}%
7517     \let\forest@drawtree@endbox\egroup
7518   }%
7519   \ifforest@external@
7520     \ifforest@externalize@tree@
7521       \forest@temptrue
7522     \else
7523       \tikzifexternalizing{%
7524         \ifforest@was@tikzexternalwasenable
7525           \forest@temptrue
7526           \pgfkeys{/tikz/external/optimize=false}%
7527           \let\forest@drawtree@beginbox\relax
7528           \let\forest@drawtree@endbox\relax
7529         \else
7530           \forest@tempfalse
7531         \fi
7532       }{%
7533         \forest@tempfalse
7534       }%
7535     \fi
7536     \ifforest@temp
7537       \advance\forest@externalize@inner@n 1
7538       \edef\forest@externalize@filename{%
7539         \tikzexternalrealjob-forest-\forest@externalize@outer@n
7540         \ifnum\forest@externalize@inner@n=0 \else.\the\forest@externalize@inner@n\fi}%
7541       \expandafter\tikzsetnextfilename\expandafter{\forest@externalize@filename}%
7542       \tikzexternalenable
7543       \pgfkeysalso{/tikz/external/remake next,/tikz/external/export next}%
7544     \fi
7545     \ifforest@externalize@tree@
7546       \typeout{forest: Invoking a recursive call to generate the external picture
7547         '\forest@externalize@filename' for the following context+code:
7548         '\expandafter\detokenize\expandafter{\forest@externalize@id}'}%
7549     \fi
7550   \fi
7551   %
7552   \ifforesttikzcshack
7553     \let\forest@original@tikz@parse@node\tikz@parse@node
7554     \let\tikz@parse@node\forest@tikz@parse@node
7555   \fi
7556   \pgfkeysgetvalue{/forest/begin draw/.@cmd}\forest@temp@begindraw

```

```

7557 \pgfkeysgetvalue{/forest/end draw/.@cmd}\forest@temp@enddraw
7558 \edef\forest@marshal{%
7559   \noexpand\forest@drawtree@beginbox
7560   \expandonce{\forest@temp@begindraw\pgfkeysnovalue\pgfeov}%
7561   \noexpand\forest@node@drawtree@
7562   \expandonce{\forest@temp@enddraw\pgfkeysnovalue\pgfeov}%
7563   \noexpand\forest@drawtree@endbox
7564 } \forest@marshal
7565 \ifforesttikzcshack
7566   \let\tikz@parse@node\forest@original@tikz@parse@node
7567 \fi
7568 %
7569 \ifforest@external@
7570   \ifforest@externalize@tree@
7571     \tikzexternaldisable
7572     \eappto\forest@externalize@checkimages{%
7573       \noexpand\forest@includeexternal@check{\forest@externalize@filename}%
7574     }%
7575     \expandafter\ifstrequal\expandafter{\forest@drawtreebox}{\pgfkeysnovalue}{%
7576       \eappto\forest@externalize@loadimages{%
7577         \noexpand\forest@includeexternal{\forest@externalize@filename}%
7578       }%
7579     }{%
7580       \eappto\forest@externalize@loadimages{%
7581         \noexpand\forest@includeexternal@box\forest@drawtreebox{\forest@externalize@filename}%
7582       }%
7583     }%
7584   \fi
7585 \fi
7586 }
7587 \def\forest@drawtree@root{0}
7588 \def\forest@node@drawtree@{%
7589   \def\forest@clear@drawn{%
7590     \forest@forthis{%
7591       \forest@saveandrestoremacro\forest@drawtree@root{%
7592         \edef\forest@drawtree@root{\forest@cn}%
7593         \forestset{draw tree method}%
7594       }%
7595     }%
7596     \forest@node@ifnamedefined{forest@baseline@node}{%
7597       \edef\forest@baseline@id{\forest@node@Nametoid{forest@baseline@node}}%
7598       \ifnum\forest@baseline@id=0
7599         \else
7600           \ifcsdef{forest@drawn@\forest@baseline@id}{%
7601             \edef\forest@marshal{%
7602               \noexpand\pgfsetbaselinepointlater{%
7603                 \noexpand\pgfpointanchor
7604                   {\forest@ve{\forest@baseline@id}{name}}%
7605                   {\forest@ve{\forest@baseline@id}{anchor}}%
7606               }%
7607             } \forest@marshal
7608           }{%
7609             \PackageWarning{forest}{Baseline node (id=\forest@cn) was not drawn (most likely it's a phantom node)}
7610           }%
7611         \fi
7612       }%
7613     } \forest@clear@drawn
7614 }
7615 \def\forest@draw@node{%
7616   \ifnum\forest@ve{phantom}=0
7617     \forest@draw@node@

```



```

7618 \fi
7619 }
7620 \def\forest@draw@node@{%
7621 \forest@node@forest@position@node@later@restore
7622 \ifforest@draw@tree@preservenodeboxes@
7623 \pgfnodealias{forest@temp}{\forestove{later@name}}%
7624 \fi
7625 \pgfpositionnodenow{\pgfqpoint{\forestove{x}}{\forestove{y}}}%
7626 \ifforest@draw@tree@preservenodeboxes@
7627 \pgfnodealias{\forestove{later@name}}{forest@temp}%
7628 \fi
7629 \csdef{forest@drawn@\forest@cn}{}%
7630 \eappto\forest@clear@drawn{\noexpand\csundef{forest@drawn@\forest@cn}}%
7631 }
7632 \def\forest@draw@edge{%
7633 \ifcsdef{forest@drawn@\forest@cn}{% was the current node drawn?
7634 \ifnum\forestove{@parent}=0 % do we have a parent?
7635 \else
7636 \ifcsdef{forest@drawn@\forestove{@parent}}{% was the parent drawn?
7637 \forest@draw@edge@
7638 }{}%
7639 \fi
7640 }{}%
7641 }
7642 \def\forest@draw@edge@{%
7643 \edef\forest@temp{\forestove{edge path}}\forest@temp
7644 }
7645 \def\forest@draw@tikz{%
7646 \ifnum\forestove{phantom}=0
7647 \forest@draw@tikz@
7648 \fi
7649 }
7650 \def\forest@draw@tikz@{%
7651 \forestove{tikz}%
7652 }

```

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.

```

7653 \newdimen\forest@xg
7654 \newdimen\forest@yg
7655 \newdimen\forest@xs
7656 \newdimen\forest@ys
7657 \def\forest@setupgrowline#1{%
7658 \edef\forest@grow{#1}%
7659 \pgfqpointpolar{\forest@grow}{1pt}%
7660 \forest@xg=\pgf@x
7661 \forest@yg=\pgf@y
7662 \forest@xs=-\pgf@y
7663 \forest@ys=\pgf@x
7664 }

```

9.1 Projections

The following macro belongs to the $\backslash\text{pgfpoint}\dots$ family: it projects point #1 on the grow line. (The result is returned via $\backslash\text{pgf@x}$ and $\backslash\text{pgf@y}$.) The implementation is based on code from `tikzlibrarycalc`,

but optimized for projecting on grow lines, and split to optimize serial usage in `\forest@projectpath`.

```
7665 \def\forest@pgfpointprojectiontogrowline#1{%
7666   \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.
7667   \pgfutil@tempdima=\pgf@sys@tonumber{\pgf@x}\forest@xg%
7668   \advance\pgfutil@tempdima by\pgf@sys@tonumber{\pgf@y}\forest@yg%
    The projection is  $(x_g, y_g)$  scaled by the distance.
7669   \global\pgf@x=\pgf@sys@tonumber{\pgfutil@tempdima}\forest@xg%
7670   \global\pgf@y=\pgf@sys@tonumber{\pgfutil@tempdima}\forest@yg%
7671 }
```

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.

```
7672 \def\forest@distancetogrowline#1#2{%
7673   \pgf@process{#2}%
7674   #1=\pgf@sys@tonumber{\pgf@x}\forest@xs\relax
7675   \advance#1 by\pgf@sys@tonumber{\pgf@y}\forest@ys\relax
7676 }
```

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 simple² path #1 onto the grow line. The result is an array of tuples (x_o, y_o, x_p, y_p) , where x_o and y_o stand for the *original* point, and x_p and y_p stand for its *projection*. 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`.

```
7677 \let\forest@pp@n\relax
7678 \def\forest@projectpathtogrowline#1#2{%
7679   \edef\forest@pp@prefix{#2}%
7680   \forest@save@pgfsyssoftpath@tokendefs
7681   \let\pgfsyssoftpath@movetotoken\forest@projectpath@processpoint
7682   \let\pgfsyssoftpath@linetotoken\forest@projectpath@processpoint
7683   \c@pgf@counta=0
7684   #1%
7685   \csedef{#2n}{\the\c@pgf@counta}%
7686   \forest@restore@pgfsyssoftpath@tokendefs
7687 }
```

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

```
7688 \def\forest@projectpath@processpoint#1#2{%
7689   \pgfqpoint{#1}{#2}%
7690   \expandafter\edef\csname\forest@pp@prefix\the\c@pgf@counta x\endcsname{\the\pgf@x}%
7691   \expandafter\edef\csname\forest@pp@prefix\the\c@pgf@counta y\endcsname{\the\pgf@y}%
7692   \forest@pgfpointprojectiontogrowline{}%
7693   \expandafter\edef\csname\forest@pp@prefix\the\c@pgf@counta xp\endcsname{\the\pgf@x}%
7694   \expandafter\edef\csname\forest@pp@prefix\the\c@pgf@counta yp\endcsname{\the\pgf@y}%
7695   \advance\c@pgf@counta 1\relax
7696 }
```

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

```
7697 \def\forest@sortprojections#1{%
7698   % todo: optimize in cases when we know that the array is actually a
7699   % merger of sorted arrays; when does this happen? in
7700   % distance_between_paths, and when merging the edges of the parent
```

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

```

7701 % and its children in a uniform growth tree
7702 \edef\forest@ppi@inputprefix{#1}%
7703 \c@pgf@counta=\csname#1n\endcsname\relax
7704 \advance\c@pgf@counta -1
7705 \forest@sort\forest@ppiraw@cmp\forest@ppiraw@let\forest@sort@ascending{0}{\the\c@pgf@counta}%
7706 }

```

The following macro processes the data gathered by (possibly more than one invocation of) `\forest@projectpathtogroupline` 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.

```

7707 \def\forest@processprojectioninfo#1#2#3{%
7708 \edef\forest@ppi@inputprefix{#1}%

```

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

```

7709 \c@pgf@counta=0
7710 \c@pgf@countb=-1
7711 \safeloop
7712 \ifnum\c@pgf@counta<\csname#1n\endcsname\relax

```

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

```

7713 \letcs\forest@xo{#1\the\c@pgf@counta xo}%
7714 \letcs\forest@yo{#1\the\c@pgf@counta yo}%
7715 \letcs\forest@xp{#1\the\c@pgf@counta xp}%
7716 \letcs\forest@yp{#1\the\c@pgf@counta yp}%
7717 \ifnum\c@pgf@countb<0
7718 \forest@equaltotolerancefalse
7719 \else
7720 \forest@equaltotolerance
7721 {\pgfqpoint\forest@xp\forest@yp}%
7722 {\pgfqpoint
7723 {\csname#2\the\c@pgf@countb x\endcsname}%
7724 {\csname#2\the\c@pgf@countb y\endcsname}%
7725 }%
7726 \fi
7727 \ifforest@equaltotolerance\else

```

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

```

7728 \advance\c@pgf@countb 1
7729 \cslet{#2\the\c@pgf@countb x}\forest@xp
7730 \cslet{#2\the\c@pgf@countb y}\forest@yp
7731 \csdef{#2\the\c@pgf@countb @n}{0}%
7732 \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`):

```

7733 % todo: this is ugly!
7734 \ifdefined\forest@xo\ifx\forest@xo\relax\else
7735 \ifdefined\forest@yo\ifx\forest@yo\relax\else

```

³At first sight, this information could be cached “at the source”: by `forest@pgfqpointprojectiontogroupline`. 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.

```
7736 \forest@append@point@to@inner@array
7737 \forest@xo\forest@yo
7738 {#2\the\c@pgf@countb @}%
```

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

```
7739 \eapptotoks#3{(\forest@xo,\forest@yo){\the\c@pgf@countb}}%
7740 \fi\fi
7741 \fi\fi
```

Clean-up the raw array item.

```
7742 % todo: is this really necessary? (yes: see the "ugly" thing above)
7743 \cslet{#1\the\c@pgf@counta xo}\relax
7744 \cslet{#1\the\c@pgf@counta yo}\relax
7745 \cslet{#1\the\c@pgf@counta xp}\relax
7746 \cslet{#1\the\c@pgf@counta yp}\relax
7747 \advance\c@pgf@counta 1
7748 \saferepeat
```

Clean up the raw array length.

```
7749 % todo: is this really necessary?
7750 \cslet{#1n}\relax
```

Store the length of the outer result array.

```
7751 \advance\c@pgf@countb 1
7752 \csedef{#2n}{\the\c@pgf@countb}%
7753 }
```

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

```
7754 \def\forest@ppiraw@let#1#2{%
7755 \csletcs{\forest@ppi@inputprefix#1xo}{\forest@ppi@inputprefix#2xo}%
7756 \csletcs{\forest@ppi@inputprefix#1yo}{\forest@ppi@inputprefix#2yo}%
7757 \csletcs{\forest@ppi@inputprefix#1xp}{\forest@ppi@inputprefix#2xp}%
7758 \csletcs{\forest@ppi@inputprefix#1yp}{\forest@ppi@inputprefix#2yp}%
7759 }
```

Item comparison macro for sorting the raw projection data.

```
7760 \def\forest@ppiraw@cmp#1#2{%
7761 \forest@sort@cmptwodimcs
7762 {\forest@ppi@inputprefix#1xp}{\forest@ppi@inputprefix#1yp}%
7763 {\forest@ppi@inputprefix#2xp}{\forest@ppi@inputprefix#2yp}%
7764 }
```

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

```
7765 \def\forest@append@point@to@inner@array#1#2#3{%
7766 \c@pgf@countc=\csname#3n\endcsname\relax
7767 \csedef{#3\the\c@pgf@countc x}{#1}%
7768 \csedef{#3\the\c@pgf@countc y}{#2}%
7769 \forest@distancetogrowline\pgfutil@tempdima{\pgfqqpoint#1#2}%
7770 \csedef{#3\the\c@pgf@countc d}{\the\pgfutil@tempdima}%
7771 \advance\c@pgf@countc 1
7772 \csedef{#3n}{\the\c@pgf@countc}%
7773 }
```

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.)

```
7774 \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.

```
7775 \edef\forest@bp@prefix{#2}%
7776 \let\forest@breakpath@toks#3%
7777 \forest@save@pgfsyssoftpath@tokendefs
7778 \let\pgfsyssoftpath@movetotoken\forest@breakpath@processfirstpoint
7779 \let\pgfsyssoftpath@linetotoken\forest@breakpath@processfirstpoint
7780 %\pgfusepath{}% empty the current path. ok?
7781 #1%
7782 \forest@restore@pgfsyssoftpath@tokendefs
7783 \pgfsyssoftpath@getcurrentpath#4%
7784 }
```

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

```
7785 \def\forest@breakpath@processfirstpoint#1#2{%
7786 \forest@breakpath@processmoveto{#1}{#2}%
7787 \let\pgfsyssoftpath@movetotoken\forest@breakpath@processmoveto
7788 \let\pgfsyssoftpath@linetotoken\forest@breakpath@processlineto
7789 }
```

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.

```
7790 \def\forest@breakpath@processmoveto#1#2{%
7791 \pgfsyssoftpath@moveto{#1}{#2}%
7792 \def\forest@previous@x{#1}%
7793 \def\forest@previous@y{#2}%
7794 \forest@breakpath@getfromtoks\forest@breakpath@toks\forest@previous@i{#1}{#2}%
7795 \expandafter\let\expandafter\forest@previous@px
7796 \csname\forest@bp@prefix\forest@previous@i x\endcsname
7797 \expandafter\let\expandafter\forest@previous@py
7798 \csname\forest@bp@prefix\forest@previous@i y\endcsname
7799 }
7800 \def\forest@breakpath@getfromtoks#1#2#3#4{%
7801 % #1=cache toks register, #2=receiving cs, (#3,#4)=point;
7802 % we rely on the fact that the point we’re looking up should always be present
7803 \def\forest@breakpath@getfromtoks@##1(#3,#4)##2##3\forest@END{##2}%
7804 \edef#2{\expandafter\forest@breakpath@getfromtoks@the#1\forest@END}%
7805 }
```

This is the heart of the path-breaking procedure.

```
7806 \def\forest@breakpath@processlineto#1#2{%
```

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

```
7807 \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.)

```
7808 \forest@breakpath@getfromtoks\forest@breakpath@toks\forest@i{#1}{#2}%
7809 \expandafter\let\expandafter\forest@px
7810 \csname\forest@bp@prefix\forest@i x\endcsname
7811 \expandafter\let\expandafter\forest@py
7812 \csname\forest@bp@prefix\forest@i y\endcsname
```

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

```
7813 \forest@equaltotolerance
7814 {\pgfqpoint{\forest@previous@px}{\forest@previous@py}}%
```

```

7815   {\pgfqpoint{\forest@px}{\forest@py}}%
7816   \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.

```

7817   \let\forest@breakpath@op\pgfsyssoftpath@moveto
7818   \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.

```

7819   \forest@temp@count=\forest@previous@i\relax
7820   \ifnum\forest@previous@i<\forest@i\relax
7821     \def\forest@breakpath@step{1}%
7822     \def\forest@breakpath@test{\forest@temp@count<\forest@i\relax}%
7823   \else
7824     \def\forest@breakpath@step{-1}%
7825     \def\forest@breakpath@test{\forest@temp@count>\forest@i\relax}%
7826   \fi

```

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

```

7827   \safeloop
7828     \advance\forest@temp@count\forest@breakpath@step\relax
7829   \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.)

```

7830   \pgfpointintersectionoflines
7831     {\pgfqpoint
7832       {\csname\forest@bp@prefix\the\forest@temp@count x\endcsname}%
7833       {\csname\forest@bp@prefix\the\forest@temp@count y\endcsname}%
7834     }%
7835   {\pgfpointadd
7836     {\pgfqpoint
7837       {\csname\forest@bp@prefix\the\forest@temp@count x\endcsname}%
7838       {\csname\forest@bp@prefix\the\forest@temp@count y\endcsname}%
7839     }%
7840     {\pgfqpoint{\forest@xs}{\forest@ys}}%
7841   }%
7842   {\pgfqpoint{\forest@previous@x}{\forest@previous@y}}%
7843   {\pgfqpoint{#1}{#2}}%

```

Break the segment at the intersection.

```

7844   \pgfgetlastxy\forest@last@x\forest@last@y
7845   \pgfsyssoftpath@lineto\forest@last@x\forest@last@y

```

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

```

7846   \forest@append@point@to@inner@array
7847     \forest@last@x\forest@last@y
7848     {\forest@bp@prefix\the\forest@temp@count @}%
7849   \saferepeat
7850   \fi

```

Add the current point.

```

7851   \forest@breakpath@op{#1}{#2}%

```

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

```

7852   \def\forest@previous@x{#1}%
7853   \def\forest@previous@y{#2}%
7854   \let\forest@previous@i\forest@i
7855   \let\forest@previous@px\forest@px
7856   \let\forest@previous@py\forest@py
7857 }

```

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

```
7858 \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.

```
7859 \def\forest@getnegativetightedgeofpath#1#2{%
7860   \forest@get@onetightedgeofpath#1\forest@sort@ascending#2}
7861 \def\forest@getpositivetightedgeofpath#1#2{%
7862   \forest@get@onetightedgeofpath#1\forest@sort@descending#2}
7863 \def\forest@get@onetightedgeofpath#1#2#3{%
7864   {%
7865     \forest@get@one@tightedgeofpath#1#2\forest@gep@edge
7866     \global\let\forest@gep@global@edge\forest@gep@edge
7867   }%
7868   \let#3\forest@gep@global@edge
7869 }
7870 \newtoks\forest@pi@toks
7871 \newtoks\forest@segment@toks
7872 \def\forest@get@one@tightedgeofpath#1#2#3{%
```

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

```
7873 \forest@projectpathtogrowline#1{\forest@pp@}%
7874 \forest@sortprojections{\forest@pp@}%
7875 \forest@processprojectioninfo{\forest@pp@}{\forest@pi@}\forest@pi@toks
```

Break the path.

```
7876 \forest@breakpath#1{\forest@pi@}\forest@pi@toks\forest@brokenpath
```

Compile some more useful information.

```
7877 \forest@sort@inner@arrays{\forest@pi@}#2%
7878 \forest@pathtodict\forest@brokenpath\forest@segment@toks
```

The auxiliary data is set up: do the work!

```
7879 \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).

```
7880 \forest@simplifypath\forest@edge#3%
7881 }
```

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

```
7882 \def\forest@getbothtightedgesofpath#1#2#3{%
7883   {%
7884     \forest@get@one@tightedgeofpath#1\forest@sort@ascending\forest@gep@firstedge
```

Reverse the order of items in the inner arrays.

```
7885   \c@pgf@counta=0
7886   \forest@loop
```

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

```

7887 \ifnum\c@pgf@counta<\forest@pi@n\relax
7888 \forest@ppi@deflet{\forest@pi@the\c@pgf@counta @}%
7889 \forest@reversearray\forest@ppi@let
7890 {0}%
7891 {\csname forest@pi@the\c@pgf@counta @n\endcsname}%
7892 \advance\c@pgf@counta 1
7893 \forest@repeat

```

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

```

7894 \forest@gettightedgeofpath@getedge\forest@edge
7895 \forest@simplifypath\forest@edge\forest@gep@secondedge

```

Smuggle the results out of the enclosing \TeX group.

```

7896 \global\let\forest@gep@global@firstedge\forest@gep@firstedge
7897 \global\let\forest@gep@global@secondedge\forest@gep@secondedge
7898 }%
7899 \let#2\forest@gep@global@firstedge
7900 \let#3\forest@gep@global@secondedge
7901 }

```

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

```

7902 \def\forest@sort@inner@arrays#1#2{%
7903 \c@pgf@counta=0
7904 \safeloop
7905 \ifnum\c@pgf@counta<\csname#1n\endcsname
7906 \c@pgf@countb=\csname#1the\c@pgf@counta @n\endcsname\relax
7907 \ifnum\c@pgf@countb>1
7908 \advance\c@pgf@countb -1
7909 \forest@ppi@deflet{#1the\c@pgf@counta @}%
7910 \forest@ppi@defcmp{#1the\c@pgf@counta @}%
7911 \forest@sort\forest@ppi@cmp\forest@ppi@let#2{0}{\the\c@pgf@countb}%
7912 \fi
7913 \advance\c@pgf@counta 1
7914 \saferepeat
7915 }

```

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.

```

7916 \def\forest@ppi@deflet#1{%
7917 \edef\forest@ppi@let##1##2{%
7918 \noexpand\csletcs{#1##1x}{#1##2x}%
7919 \noexpand\csletcs{#1##1y}{#1##2y}%
7920 \noexpand\csletcs{#1##1d}{#1##2d}%
7921 }%
7922 }

```

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.

```

7923 \def\forest@ppi@defcmp#1{%
7924 \edef\forest@ppi@cmp##1##2{%
7925 \noexpand\forest@sort@cmpdimcs{#1##1d}{#1##2d}%
7926 }%
7927 }

```

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 $(x_1, y_1) \text{--} (x_2, y_2)$ into `toks #2`. (The “dictionary” was reimplemented in v2.1.4. It’s based on a `toks` register now, we search using `\pgfutil@in@.`)

```

7928 \def\forest@pathtodict#1#2{%
7929 \let\forest@pathtodict@toks#2%
7930 \forest@save@pgfsyssoftpath@token@defs
7931 \let\pgfsyssoftpath@movetotoken\forest@pathtodict@movetooop
7932 \let\pgfsyssoftpath@linetotoken\forest@pathtodict@linetooop
7933 \def\forest@pathtodict@subpathstart{}%

```



```

7934 #1%
7935 \forest@restore@pgfsyssoftpath@tokendef
7936 }

```

When a move-to operation is encountered:

```

7937 \def\forest@pathdict@movetooop#1#2{%
7938 \apptotoks\forest@pathdict@toks{(#1,#2)}%
7939 }

```

When a line-to operation is encountered:

```

7940 \def\forest@pathdict@linetooop#1#2{%
7941 \apptotoks\forest@pathdict@toks{--(#1,#2)}%
7942 }

```

In this macro, the edge is actually computed.

```

7943 \def\forest@gettightedgeofpath@getedge#1{% cs to store the edge into

```

Clear the path and the last projection.

```

7944 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
7945 \let\forest@last@x\relax
7946 \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.)

```

7947 \c@pgf@counta=0
7948 \forest@temp@count=\forest@pi@n\relax
7949 \advance\forest@temp@count -1
7950 \edef\forest@nminusone{\the\forest@temp@count}%
7951 \safeloop
7952 \ifnum\c@pgf@counta<\forest@nminusone\relax
7953 \forest@gettightedgeofpath@getedge@loopa
7954 \saferepeat

```

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

```

7955 \ifnum\forest@nminusone<\forest@n\relax\else
7956 \ifnum\csname forest@pi@\forest@nminusone @n\endcsname>0
7957 \forest@gettightedgeofpath@maybemoveto{\forest@nminusone}{0}%
7958 \fi
7959 \fi
7960 \pgfsyssoftpath@getcurrentpath#1%
7961 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
7962 }

```

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.

```

7963 \def\forest@gettightedgeofpath@getedge@loopa{%
7964 \ifnum\csname forest@pi@\the\c@pgf@counta @n\endcsname>0

```

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

```

7965 \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.

```

7966 \c@pgf@countb=0
7967 \safeloop
7968 \ifnum\c@pgf@countb<\csname forest@pi@\the\c@pgf@counta @n\endcsname\relax
7969 \forest@gettightedgeofpath@getedge@loopb
7970 \saferepeat
7971 \fi
7972 \advance\c@pgf@counta 1
7973 }

```

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

```

7974 \def\forest@gettightedgeofpath@getedge@loopb{%
7975     \c@pgf@countc=0
7976     \advance\c@pgf@counta 1
7977     \edef\forest@aplusone{\the\c@pgf@counta}%
7978     \advance\c@pgf@counta -1
7979     \safeloop
7980     \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!

```

7981     \edef\forest@temp{%
7982         (\csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb x\endcsname,%
7983         \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb y\endcsname)--(%
7984         \csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname,%
7985         \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname)%
7986     }%
7987     \expandafter\expandafter\expandafter\pgfutil@in@\expandafter\expandafter\expandafter
7988     {\expandafter\forest@temp\expandafter}\expandafter
7989     {\the\forest@segment@toks}%
7990     \ifpgfutil@in@
7991     \else
7992         \edef\forest@temp{%
7993             (\csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname,%
7994             \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname)--(%
7995             \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb x\endcsname,%
7996             \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb y\endcsname)%
7997         }%
7998         \expandafter\expandafter\expandafter\pgfutil@in@\expandafter\expandafter\expandafter
7999         {\expandafter\forest@temp\expandafter}\expandafter
8000         {\the\forest@segment@toks}%
8001     \fi
8002     \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.

```

8003     \forest@gettightedgeofpath@maybemoveto{\the\c@pgf@counta}{\the\c@pgf@countb}%

```

Second, create a line to the end point.

```

8004     \edef\forest@last@x{%
8005         \csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname}%
8006     \edef\forest@last@y{%
8007         \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname}%
8008     \pgfsyssoftpath@lineto\forest@last@x\forest@last@y

```

Finally, “break” out of the innermost two loops.

```

8009     \c@pgf@countc=\csname forest@pi@\forest@aplusone @n\endcsname
8010     \c@pgf@countb=\csname forest@pi@\the\c@pgf@counta @n\endcsname
8011     \fi
8012     \advance\c@pgf@countc 1

```

```

8013     \saferepeat
8014     \advance\c@pgf@countb 1
8015 }

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

8016 \def\forest@gettightedgeofpath@maybemoveto#1#2{%
8017   \forest@temptrue
8018   \ifx\forest@last@x\relax\else
8019     \ifdim\forest@last@x=\csname forest@pi@#1@#2x\endcsname\relax
8020     \ifdim\forest@last@y=\csname forest@pi@#1@#2y\endcsname\relax
8021     \forest@tempfalse
8022   \fi
8023   \fi
8024   \fi
8025   \ifforest@temp
8026     \edef\forest@last@x{\csname forest@pi@#1@#2x\endcsname}%
8027     \edef\forest@last@y{\csname forest@pi@#1@#2y\endcsname}%
8028     \pgfsyssoftpath@moveto\forest@last@x\forest@last@y
8029   \fi
8030 }

```

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

```

8031 \def\forest@simplifypath#1#2{%
8032   \pgfsyssoftpath@setcurrentpath\pgfutil@empty
8033   \forest@save\pgfsyssoftpath@tokendefs
8034   \let\pgfsyssoftpath@movetotoken\forest@simplifypath@moveto
8035   \let\pgfsyssoftpath@linetotoken\forest@simplifypath@lineto
8036   \let\forest@last@x\relax
8037   \let\forest@last@y\relax
8038   \let\forest@last@atan\relax
8039   #1%
8040   \ifx\forest@last@x\relax\else
8041     \ifx\forest@last@atan\relax\else
8042       \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
8043     \fi
8044   \fi
8045   \forest@restore\pgfsyssoftpath@tokendefs
8046   \pgfsyssoftpath@getcurrentpath#2%
8047   \pgfsyssoftpath@setcurrentpath\pgfutil@empty
8048 }

```

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.

```

8049 \def\forest@simplifypath@moveto#1#2{%
8050   \ifx\forest@last@x\relax\else
8051     \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
8052   \fi
8053   \pgfsyssoftpath@moveto{#1}{#2}%
8054   \def\forest@last@x{#1}%
8055   \def\forest@last@y{#2}%
8056   \let\forest@last@atan\relax
8057 }

```

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

```

8058 \def\forest@getedgeofpath@precision{1pt}

```

When a line-to is encountered...

```

8059 \def\forest@simplifypath@lineto#1#2{%
8060   \ifx\forest@last@x\relax

```

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

```

8061 \def\forest@last@x{#1}%
8062 \def\forest@last@y{#2}%
8063 \let\forest@last@atan\relax
8064 \else
    Otherwise, we calculate the slope of the current segment (i.e. the segment between the last and the
    current point), ...
8065 \pgfpointdiff{\pgfpoint{#1}{#2}}{\pgfpoint{\forest@last@x}{\forest@last@y}}%
8066 \ifdim\pgf@x<\pgfintersectiontolerance
8067 \ifdim-\pgf@x<\pgfintersectiontolerance
8068 \pgf@x=0pt
8069 \fi
8070 \fi
8071 \edef\forest@marshal{%
8072 \noexpand\pgfmathatan@
8073 {\expandafter\pgf@geT\the\pgf@x}%
8074 {\expandafter\pgf@geT\the\pgf@y}%
8075 }\forest@marshal
8076 \let\forest@current@atan\pgfmathresult
8077 \ifx\forest@last@atan\relax

```

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

```

8078 \def\forest@last@x{#1}%
8079 \def\forest@last@y{#2}%
8080 \let\forest@last@atan\forest@current@atan
8081 \else

```

Otherwise, compare the first and the current slope.

```

8082 \pgfutil@tempdima=\forest@current@atan pt
8083 \advance\pgfutil@tempdima -\forest@last@atan pt
8084 \ifdim\pgfutil@tempdima<0pt\relax
8085 \multiply\pgfutil@tempdima -1
8086 \fi
8087 \ifdim\pgfutil@tempdima<\forest@getedgeofpath@precision\relax
8088 \else

```

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

```

8089 \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
8090 \let\forest@last@atan\forest@current@atan
8091 \fi

```

In any event, update the last point.

```

8092 \def\forest@last@x{#1}%
8093 \def\forest@last@y{#2}%
8094 \fi
8095 \fi
8096 }

```

9.4 Get rectangle/band edge

```

8097 \def\forest@getnegativerectangleedgeofpath#1#2{%
8098 \forest@getnegativerectangleorbandededgeofpath{#1}{#2}{\the\pgf@xb}}
8099 \def\forest@getpositiverectangleedgeofpath#1#2{%
8100 \forest@getpositiverectangleorbandededgeofpath{#1}{#2}{\the\pgf@xb}}
8101 \def\forest@getbothrectangleedgesofpath#1#2#3{%
8102 \forest@getbothrectangleorbandededgesofpath{#1}{#2}{#3}{\the\pgf@xb}}
8103 \def\forest@bandlength{5000pt} % something large (ca. 180cm), but still manageable for TeX without producing
8104 \def\forest@getnegativebandededgeofpath#1#2{%
8105 \forest@getnegativerectangleorbandededgeofpath{#1}{#2}{\forest@bandlength}}
8106 \def\forest@getpositivebandededgeofpath#1#2{%
8107 \forest@getpositiverectangleorbandededgeofpath{#1}{#2}{\forest@bandlength}}

```

```

8108 \def\forest@getbothbandedgesofpath#1#2#3{%
8109   \forest@getbothrectangleorbandedgesofpath{#1}{#2}{#3}{\forest@bandlength}}
8110 \def\forest@getnegativerectangleorbandedgeofpath#1#2#3{%
8111   \forest@path@getboundingrectangle@ls#1{\forest@grow}%
8112   \edef\forest@gre@path{%
8113     \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@ya}%
8114     \noexpand\pgfsyssoftpath@linetotoken{#3}{\the\pgf@ya}%
8115   }%
8116   {%
8117     \pgftransformreset
8118     \forest@pgfqtransformrotate{\forest@grow}%
8119     \forest@pgfpathtransformed\forest@gre@path
8120   }%
8121   \pgfsyssoftpath@getcurrentpath#2%
8122 }
8123 \def\forest@getpositiverectangleorbandedgeofpath#1#2#3{%
8124   \forest@path@getboundingrectangle@ls#1{\forest@grow}%
8125   \edef\forest@gre@path{%
8126     \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@yb}%
8127     \noexpand\pgfsyssoftpath@linetotoken{#3}{\the\pgf@yb}%
8128   }%
8129   {%
8130     \pgftransformreset
8131     \forest@pgfqtransformrotate{\forest@grow}%
8132     \forest@pgfpathtransformed\forest@gre@path
8133   }%
8134   \pgfsyssoftpath@getcurrentpath#2%
8135 }
8136 \def\forest@getbothrectangleorbandedgesofpath#1#2#3#4{%
8137   \forest@path@getboundingrectangle@ls#1{\forest@grow}%
8138   \edef\forest@gre@negpath{%
8139     \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@ya}%
8140     \noexpand\pgfsyssoftpath@linetotoken{#4}{\the\pgf@ya}%
8141   }%
8142   \edef\forest@gre@pospath{%
8143     \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@yb}%
8144     \noexpand\pgfsyssoftpath@linetotoken{#4}{\the\pgf@yb}%
8145   }%
8146   {%
8147     \pgftransformreset
8148     \forest@pgfqtransformrotate{\forest@grow}%
8149     \forest@pgfpathtransformed\forest@gre@negpath
8150   }%
8151   \pgfsyssoftpath@getcurrentpath#2%
8152   {%
8153     \pgftransformreset
8154     \forest@pgfqtransformrotate{\forest@grow}%
8155     \forest@pgfpathtransformed\forest@gre@pospath
8156   }%
8157   \pgfsyssoftpath@getcurrentpath#3%
8158 }

```

9.5 Distance between paths

Another crucial part of the package.

```

8159 \newtoks\forest@PIi@toks
8160 \newtoks\forest@PIii@toks
8161 \def\forest@distance@between@edge@paths#1#2#3{%
8162   \begingroup
8163   % #1, #2 = (edge) paths
8164   %

```

```

8165 % project paths
8166 \forest@projectpathtogrowline#1{forest@p1@}%
8167 \forest@projectpathtogrowline#2{forest@p2@}%
8168 % merge projections (the lists are sorted already, because edge
8169 % paths are |sorted|)
8170 \forest@dbep@mergeprojections
8171   {forest@p1@}{forest@p2@}%
8172   {forest@P1@}{forest@P2@}%
8173 % process projections
8174 \forest@processprojectioninfo{forest@P1@}{forest@PI1@}\forest@PIi@toks
8175 \forest@processprojectioninfo{forest@P2@}{forest@PI2@}\forest@PIii@toks
8176 % break paths
8177 \forest@breakpath#1{forest@PI1@}\forest@PIi@toks\forest@broken@one
8178 \forest@breakpath#2{forest@PI2@}\forest@PIii@toks\forest@broken@two
8179 % sort inner arrays ---optimize: it's enough to find max and min
8180 \forest@sort@inner@arrays{forest@PI1@}\forest@sort@descending
8181 \forest@sort@inner@arrays{forest@PI2@}\forest@sort@ascending
8182 % compute the distance
8183 \let\forest@distance\relax
8184 \c@pgf@countc=0
8185 \forest@loop
8186 \ifnum\c@pgf@countc<\csname forest@PI1@n\endcsname\relax
8187   \ifnum\csname forest@PI1@the\c@pgf@countc @n\endcsname=0 \else
8188     \ifnum\csname forest@PI2@the\c@pgf@countc @n\endcsname=0 \else
8189       \pgfutil@tempdima=\csname forest@PI2@the\c@pgf@countc @0d\endcsname\relax
8190       \advance\pgfutil@tempdima -\csname forest@PI1@the\c@pgf@countc @0d\endcsname\relax
8191       \ifx\forest@distance\relax
8192         \edef\forest@distance{\the\pgfutil@tempdima}%
8193       \else
8194         \ifdim\pgfutil@tempdima<\forest@distance\relax
8195           \edef\forest@distance{\the\pgfutil@tempdima}%
8196         \fi
8197       \fi
8198     \fi
8199   \fi
8200   \advance\c@pgf@countc 1
8201 \forest@repeat
8202 \global\let\forest@global@temp\forest@distance
8203 \endgroup
8204 \let#3\forest@global@temp
8205 }
8206 % merge projections: we need two projection arrays, both containing
8207 % projection points from both paths, but each with the original
8208 % points from only one path
8209 \def\forest@dbep@mergeprojections#1#2#3#4{%
8210 % TODO: optimize: v bistvu ni treba sortirat, ker je edge path e sortiran
8211 \forest@sortprojections{#1}%
8212 \forest@sortprojections{#2}%
8213 \c@pgf@counta=0
8214 \c@pgf@countb=0
8215 \c@pgf@countc=0
8216 \edef\forest@input@prefix@one{#1}%
8217 \edef\forest@input@prefix@two{#2}%
8218 \edef\forest@output@prefix@one{#3}%
8219 \edef\forest@output@prefix@two{#4}%
8220 \forest@dbep@mp@iterate
8221 \csedef{#3n}{\the\c@pgf@countc}%
8222 \csedef{#4n}{\the\c@pgf@countc}%
8223 }
8224 \def\forest@dbep@mp@iterate{%
8225 \let\forest@dbep@mp@next\forest@dbep@mp@iterate

```

```

8226 \ifnum\c@pgf@counta<\csname\forest@input@prefix@one n\endcsname\relax
8227 \ifnum\c@pgf@countb<\csname\forest@input@prefix@two n\endcsname\relax
8228 \let\forest@dbep@mp@next\forest@dbep@mp@do
8229 \else
8230 \let\forest@dbep@mp@next\forest@dbep@mp@iteratefirst
8231 \fi
8232 \else
8233 \ifnum\c@pgf@countb<\csname\forest@input@prefix@two n\endcsname\relax
8234 \let\forest@dbep@mp@next\forest@dbep@mp@iteratesecond
8235 \else
8236 \let\forest@dbep@mp@next\relax
8237 \fi
8238 \fi
8239 \forest@dbep@mp@next
8240 }
8241 \def\forest@dbep@mp@do{%
8242 \forest@sort@cmptwodimcs%
8243 {\forest@input@prefix@one\the\c@pgf@counta xp}%
8244 {\forest@input@prefix@one\the\c@pgf@counta yp}%
8245 {\forest@input@prefix@two\the\c@pgf@countb xp}%
8246 {\forest@input@prefix@two\the\c@pgf@countb yp}%
8247 \if\forest@sort@cmp@result=%
8248 \forest@dbep@mp@@store@p\forest@input@prefix@one\c@pgf@counta
8249 \forest@dbep@mp@@store@o\forest@input@prefix@one
8250 \c@pgf@counta\forest@output@prefix@one
8251 \forest@dbep@mp@@store@o\forest@input@prefix@two
8252 \c@pgf@countb\forest@output@prefix@two
8253 \advance\c@pgf@counta 1
8254 \advance\c@pgf@countb 1
8255 \else
8256 \if\forest@sort@cmp@result>%
8257 \forest@dbep@mp@@store@p\forest@input@prefix@two\c@pgf@countb
8258 \forest@dbep@mp@@store@o\forest@input@prefix@two
8259 \c@pgf@countb\forest@output@prefix@two
8260 \advance\c@pgf@countb 1
8261 \else%<
8262 \forest@dbep@mp@@store@p\forest@input@prefix@one\c@pgf@counta
8263 \forest@dbep@mp@@store@o\forest@input@prefix@one
8264 \c@pgf@counta\forest@output@prefix@one
8265 \advance\c@pgf@counta 1
8266 \fi
8267 \fi
8268 \advance\c@pgf@countc 1
8269 \forest@dbep@mp@iterate
8270 }
8271 \def\forest@dbep@mp@@store@p#1#2{%
8272 \csletcs
8273 {\forest@output@prefix@one\the\c@pgf@countc xp}%
8274 {#1\the#2xp}%
8275 \csletcs
8276 {\forest@output@prefix@one\the\c@pgf@countc yp}%
8277 {#1\the#2yp}%
8278 \csletcs
8279 {\forest@output@prefix@two\the\c@pgf@countc xp}%
8280 {#1\the#2xp}%
8281 \csletcs
8282 {\forest@output@prefix@two\the\c@pgf@countc yp}%
8283 {#1\the#2yp}%
8284 }
8285 \def\forest@dbep@mp@@store@o#1#2#3{%
8286 \csletcs{#3\the\c@pgf@countc xo}{#1\the#2xo}%

```

```

8287 \csletcs{#3\the\c@pgf@countc yo}{#1\the#2yo}%
8288 }
8289 \def\forest@dbep@mp@iteratefirst{%
8290 \forest@dbep@mp@iterateone\forest@input@prefix@one\c@pgf@counta\forest@output@prefix@one
8291 }
8292 \def\forest@dbep@mp@iteratesecond{%
8293 \forest@dbep@mp@iterateone\forest@input@prefix@two\c@pgf@countb\forest@output@prefix@two
8294 }
8295 \def\forest@dbep@mp@iterateone#1#2#3{%
8296 \forest@loop
8297 \ifnum#2<\csname#1n\endcsname\relax
8298 \forest@dbep@mp@@store@p#1#2%
8299 \forest@dbep@mp@@store@o#1#2#3%
8300 \advance\c@pgf@countc 1
8301 \advance#21
8302 \forest@repeat
8303 }

```

9.6 Utilities

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

```

8304 \newif\ifforest@equaltotolerance
8305 \def\forest@equaltotolerance#1#2{%
8306 \pgfpointdiff{#1}{#2}%
8307 \ifdim\pgf@x<Opt \multiply\pgf@x -1 \fi
8308 \ifdim\pgf@y<Opt \multiply\pgf@y -1 \fi
8309 \global\forest@equaltotolerancefalse
8310 \ifdim\pgf@x<\pgfintersectiontolerance\relax
8311 \ifdim\pgf@y<\pgfintersectiontolerance\relax
8312 \global\forest@equaltotolerancetrue
8313 \fi
8314 \fi
8315 }}

```

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

```

8316 \def\forest@save@pgfsyssoftpath@tokendefs{%
8317 \let\forest@origmovetotoken\pgfsyssoftpath@movetotoken
8318 \let\forest@origlinetotoken\pgfsyssoftpath@linetotoken
8319 \let\forest@origcurvetosupportatoken\pgfsyssoftpath@curvetosupportatoken
8320 \let\forest@origcurvetosupportbtoken\pgfsyssoftpath@curvetosupportbtoken
8321 \let\forest@origcurvetotoken\pgfsyssoftpath@curvetototoken
8322 \let\forest@origrectcornertoken\pgfsyssoftpath@rectcornertoken
8323 \let\forest@origrectsizetoken\pgfsyssoftpath@rectsizetoken
8324 \let\forest@origclosepathtoken\pgfsyssoftpath@closepathtoken
8325 \let\pgfsyssoftpath@movetotoken\forest@badtoken
8326 \let\pgfsyssoftpath@linetotoken\forest@badtoken
8327 \let\pgfsyssoftpath@curvetosupportatoken\forest@badtoken
8328 \let\pgfsyssoftpath@curvetosupportbtoken\forest@badtoken
8329 \let\pgfsyssoftpath@curvetototoken\forest@badtoken
8330 \let\pgfsyssoftpath@rectcornertoken\forest@badtoken
8331 \let\pgfsyssoftpath@rectsizetoken\forest@badtoken
8332 \let\pgfsyssoftpath@closepathtoken\forest@badtoken
8333 }
8334 \def\forest@badtoken{%
8335 \PackageError{forest}{This token should not be in this path}{}%
8336 }
8337 \def\forest@restore@pgfsyssoftpath@tokendefs{%
8338 \let\pgfsyssoftpath@movetotoken\forest@origmovetotoken
8339 \let\pgfsyssoftpath@linetotoken\forest@origlinetotoken
8340 \let\pgfsyssoftpath@curvetosupportatoken\forest@origcurvetosupportatoken

```



```

8341 \let\pgfsyssoftpath@curvetosupportbtoken\forest@origcurvetosupportbtoken
8342 \let\pgfsyssoftpath@curvetotoken\forest@origcurvetotoken
8343 \let\pgfsyssoftpath@rectcornertoken\forest@origrectcornertoken
8344 \let\pgfsyssoftpath@rectsizetoken\forest@origrectsizetoken
8345 \let\pgfsyssoftpath@closepathtoken\forest@origclosepathtoken
8346 }

```

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

```

8347 \def\forest@extendpath#1#2#3{%
8348   \pgf@process{#3}%
8349   \pgfsyssoftpath@setcurrentpath#1%
8350   \forest@save@pgfsyssoftpath@tokendefs
8351   \let\pgfsyssoftpath@movetotoken\forest@extendpath@moveto
8352   \let\pgfsyssoftpath@linetotoken\forest@extendpath@lineto
8353   #2%
8354   \forest@restore@pgfsyssoftpath@tokendefs
8355   \pgfsyssoftpath@getcurrentpath#1%
8356 }
8357 \def\forest@extendpath@moveto#1#2{%
8358   \forest@extendpath@do{#1}{#2}\pgfsyssoftpath@moveto
8359 }
8360 \def\forest@extendpath@lineto#1#2{%
8361   \forest@extendpath@do{#1}{#2}\pgfsyssoftpath@lineto
8362 }
8363 \def\forest@extendpath@do#1#2#3{%
8364   {%
8365     \advance\pgf@x #1
8366     \advance\pgf@y #2
8367     #3{\the\pgf@x}{\the\pgf@y}%
8368   }%
8369 }

```

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.)

```

8370 \def\forest@path@getboundingrectangle@ls#1#2{%
8371   {%
8372     \pgftransformreset
8373     \forest@pgfqtransformrotate{-#2}%
8374     \forest@pgfpathtransformed#1%
8375   }%
8376   \pgfsyssoftpath@getcurrentpath\forest@gbr@rotatedpath
8377   \forest@path@getboundingrectangle@xy\forest@gbr@rotatedpath
8378 }
8379 \def\forest@path@getboundingrectangle@xy#1{%
8380   \forest@save@pgfsyssoftpath@tokendefs
8381   \let\pgfsyssoftpath@movetotoken\forest@gbr@firstpoint
8382   \let\pgfsyssoftpath@linetotoken\forest@gbr@firstpoint
8383   #1%
8384   \forest@restore@pgfsyssoftpath@tokendefs
8385 }
8386 \def\forest@gbr@firstpoint#1#2{%
8387   \pgf@xa=#1 \pgf@xb=#1 \pgf@ya=#2 \pgf@yb=#2
8388   \let\pgfsyssoftpath@movetotoken\forest@gbr@point
8389   \let\pgfsyssoftpath@linetotoken\forest@gbr@point
8390 }
8391 \def\forest@gbr@point#1#2{%
8392   \ifdim#1<\pgf@xa\relax\pgf@xa=#1 \fi
8393   \ifdim#1>\pgf@xb\relax\pgf@xb=#1 \fi
8394   \ifdim#2<\pgf@ya\relax\pgf@ya=#2 \fi
8395   \ifdim#2>\pgf@yb\relax\pgf@yb=#2 \fi
8396 }

```

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.

```

8397 \let\forest@pgfqtransformrotate\pgftransformrotate
8398 \let\forest@pgftransformcm\pgftransformcm
8399 \let\forest@pgf@transformcm\pgf@transformcm
8400 \patchcmd{\forest@pgfqtransformrotate}{\pgfmathparse{#1}}{\edef\pgfmathresult{\number\numexpr#1}}{}{}
8401 \patchcmd{\forest@pgfqtransformrotate}{\pgftransformcm}{\forest@pgftransformcm}{}{}
8402 \patchcmd{\forest@pgftransformcm}{\pgf@transformcm}{\forest@pgf@transformcm}{}{}
8403 \patchcmd{\forest@pgf@transformcm}{\pgfmathsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x
8404 \patchcmd{\forest@pgf@transformcm}{\pgfmathsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x
8405 \patchcmd{\forest@pgf@transformcm}{\pgfmathsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x
8406 \patchcmd{\forest@pgf@transformcm}{\pgfmathsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x
8407 \def\forest@pgf@transformcm@setlength#1#2{#1=#2pt}

```

10 The outer UI

10.1 Externalization

```

8408 \pgfkeys{/forest/external/.cd,
8409   %copy command/.initial={cp "\source" "\target"},
8410   copy command/.initial={},
8411   optimize/.is if=forest@external@optimize@,
8412   context/.initial={%
8413     \forestOve{\csname forest@id@of@standard node\endcsname}{environment@formula}},
8414   depends on macro/.style={context/.append/.expanded={%
8415     \expandafter\detokenize\expandafter{#1}}},
8416 }
8417 \def\forest@file@copy#1#2{%
8418   \IfFileExists{#1}{%
8419     \pgfkeysgetvalue{/forest/external/copy command}\forest@copy@command
8420     \ifdefempty\forest@copy@command{%
8421       \forest@file@copy@{#1}{#2}%
8422     }{ % copy by external command
8423       \def\source{#1}%
8424       \def\target{#2}%
8425       \immediate\write18{\forest@copy@command}%
8426     }%
8427   }{}%
8428 }
8429 \newread\forest@copy@in
8430 \newwrite\forest@copy@out
8431 \def\forest@file@copy@#1#2{%
8432   \begingroup
8433   \openin\forest@copy@in=#1
8434   \immediate\openout\forest@copy@out=#2
8435   \endlinechar-1
8436   \loop
8437   \unless\ifeof\forest@copy@in
8438     \readline\forest@copy@in to\forest@temp
8439     \immediate\write\forest@copy@out{\forest@temp}%
8440   \repeat
8441   \immediate\closeout\forest@copy@out
8442   \closein\forest@copy@in
8443   \endgroup
8444 }
8445 \newif\ifforest@external@optimize@
8446 \forest@external@optimize@true
8447 \ifforest@install@keys@to@tikz@path@
8448 \tikzset{
8449   fit to/.style={

```

```

8450 /forest/for nodewalk=%
8451 {TeX={\def\forest@fitto{}},#1}%
8452 {TeX={\eappto\forest@fitto{(\forestove{name})}}},
8453 fit/.expanded={\forest@fitto}
8454 },
8455 }
8456 \fi
8457 \ifforest@external@
8458 \ifdefined\tikzexternal@tikz@replacement\else
8459 \usetikzlibrary{external}%
8460 \fi
8461 \pgfkeys{%
8462 /tikz/external/failed ref warnings for={},
8463 /pgf/images/aux in dpth=false,
8464 }%
8465 \tikzifexternalizing{}{%
8466 \forest@file@copy{\jobname.aux}{\jobname.aux.copy}%
8467 }%
8468 \AtBeginDocument{%
8469 \tikzifexternalizing{%
8470 \IfFileExists{\tikzexternalrealjob.aux.copy}{%
8471 \makeatletter
8472 \input\tikzexternalrealjob.aux.copy\relax
8473 \makeatother
8474 }{}%
8475 }{%
8476 \newwrite\forest@auxout
8477 \immediate\openout\forest@auxout=\tikzexternalrealjob.for.tmp
8478 }%
8479 \IfFileExists{\tikzexternalrealjob.for}{%
8480 {%
8481 \makehashother\makeatletter
8482 \input\tikzexternalrealjob.for\relax
8483 }%
8484 }{}%
8485 }%
8486 \AtEndDocument{%
8487 \tikzifexternalizing{}{%
8488 \immediate\closeout\forest@auxout
8489 \forest@file@copy{\jobname.for.tmp}{\jobname.for}%
8490 }%
8491 }%
8492 \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.

```

8493 \NewDocumentEnvironment{forest}{D()}{}{%
8494 \forest@config{#1}%
8495 \Collect@Body
8496 \forest@env
8497 }{}
8498 \NewDocumentCommand{\Forest}{s D(){} m}{}%
8499 \forest@config{#2}%
8500 \IfBooleanTF{#1}{\let\forest@next\forest@env}{\let\forest@next\forest@group@env}%
8501 \forest@next{#3}%
8502 }
8503 \def\forest@config#1{%
8504 \forest@defstages{stages}%

```

```

8505 \forestset{@config/.cd,#1}%
8506 }
8507 \def\forest@defstages#1{%
8508 \def\forest@stages{#1}%
8509 }
8510 \forestset{@config/.cd,
8511 %stages/.store in=\forest@stages,
8512 stages/.code={\forest@defstages{#1}},
8513 .unknown/.code={\PackageError{forest}{Unknown config option for forest environment/command.}{In Forest v2.0}}
8514 }
8515 \def\forest@group@env#1{{\forest@env{#1}}}
8516 \newif\ifforest@externalize@tree@
8517 \newif\ifforest@was@tikzexternalwasenable
8518 \newcommand\forest@env[1]{%
8519 \let\forest@external@next\forest@begin
8520 \forest@was@tikzexternalwasenablefalse
8521 \ifdefined\tikzexternal@tikz@replacement
8522 \ifx\tikz\tikzexternal@tikz@replacement
8523 \forest@was@tikzexternalwasenabletrue
8524 \tikzexternaldisable
8525 \fi
8526 \fi
8527 \forest@externalize@tree@false
8528 \ifforest@external@
8529 \ifforest@was@tikzexternalwasenable
8530 \forest@env@
8531 \fi
8532 \fi
8533 \forest@standardnode@calibrate
8534 \forest@external@next{#1}%
8535 }
8536 \def\forest@env@{%
8537 \iftikzexternalexportnext
8538 \tikzifexternalizing{%
8539 \let\forest@external@next\forest@begin@externalizing
8540 }{%
8541 \let\forest@external@next\forest@begin@externalize
8542 }%
8543 \else
8544 \tikzexternalexportnexttrue
8545 \fi
8546 }

```

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

```

8547 \long\def\forest@begin@externalizing#1{%
8548 \forest@external@setup{#1}%
8549 \let\forest@external@next\forest@begin
8550 \forest@externalize@inner@n=-1
8551 \ifforest@external@optimize@\forest@externalizing@maybeoptimize\fi
8552 \forest@external@next{#1}%
8553 \tikzexternalenable
8554 }
8555 \def\forest@externalizing@maybeoptimize{%
8556 \edef\forest@temp{\tikzexternalrealjob-forest-\forest@externalize@outer@n}%
8557 \edef\forest@marshal{%
8558 \noexpand\pgfutil@in@
8559 {\expandafter\detokenize\expandafter{\forest@temp}.}
8560 {\expandafter\detokenize\expandafter{\pgfactualjobname}.}%
8561 }\forest@marshal
8562 \ifpgfutil@in@
8563 \else

```

```

8564 \let\forest@external@next@gobble
8565 \fi
8566 }

```

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

```

8567 \long\def\forest@begin@externalize#1{%
8568 \forest@external@setup{#1}%
8569 \iftikzexternal@file@isuptodate
8570 \setbox0=\hbox{%
8571 \csname forest@externalcheck@\forest@externalize@outer@n\endcsname
8572 }%
8573 \fi
8574 \iftikzexternal@file@isuptodate
8575 \csname forest@externalload@\forest@externalize@outer@n\endcsname
8576 \else
8577 \forest@externalize@tree@true
8578 \forest@externalize@inner@n=-1
8579 \forest@begin{#1}%
8580 \ifcsdef{forest@externalize@@\forest@externalize@id}{-}{%
8581 \immediate\write\forest@auxout{%
8582 \noexpand\forest@external
8583 {\forest@externalize@outer@n}%
8584 {\expandafter\detokenize\expandafter{\forest@externalize@id}}%
8585 {\expandonce\forest@externalize@checkimages}%
8586 {\expandonce\forest@externalize@loadimages}%
8587 }%
8588 }%
8589 \fi
8590 \tikzexternalenable
8591 }
8592 \def\forest@includeexternal@check#1{%
8593 \tikzsetnextfilename{#1}%
8594 \IfFileExists{\tikzexternal@filenameprefix/#1}{\tikzexternal@file@isuptodatetrue}{\tikzexternal@file@isuptodatefalse}
8595 }
8596 \def\makehashother{\catcode'\#=12}%
8597 \long\def\forest@external@setup#1{%
8598 % set up \forest@externalize@id and \forest@externalize@outer@n
8599 % we need to deal with #s correctly (\write doubles them)
8600 \setbox0=\hbox{\makehashother\makeatletter
8601 \scantokens{\forest@temp@toks{#1}}\expandafter
8602 }%
8603 \expandafter\forest@temp@toks\expandafter{\the\forest@temp@toks}%
8604 \edef\forest@temp{\pgfkeysvalueof{/forest/external/context}}%
8605 \edef\forest@externalize@id{%
8606 \expandafter\detokenize\expandafter{\forest@temp}%
8607 @@%
8608 \expandafter\detokenize\expandafter{\the\forest@temp@toks}%
8609 }%
8610 \letcs\forest@externalize@outer@n{forest@externalize@@\forest@externalize@id}%
8611 \ifdefined\forest@externalize@outer@n
8612 \global\tikzexternal@file@isuptodatetrue
8613 \else
8614 \global\advance\forest@externalize@max@outer@n 1
8615 \edef\forest@externalize@outer@n{\the\forest@externalize@max@outer@n}%
8616 \global\tikzexternal@file@isuptodatefalse
8617 \fi
8618 \def\forest@externalize@loadimages{}%
8619 \def\forest@externalize@checkimages{}%
8620 }
8621 \newcount\forest@externalize@max@outer@n
8622 \global\forest@externalize@max@outer@n=0

```

8623 \newcount\forest@externalize@inner@n

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

```
8624 \long\def\forest@external#1#2#3#4{% #1=n,#2=context+source code,#3=update check code, #4=load code
8625 \ifnum\forest@externalize@max@outer@n<#1
8626   \global\forest@externalize@max@outer@n=#1
8627 \fi
8628 \global\csdef{forest@externalize@@\detokenize{#2}}{#1}%
8629 \global\csdef{forest@externalcheck@#1}{#3}%
8630 \global\csdef{forest@externalload@#1}{#4}%
8631 \tikzifexternalizing{}{%
8632   \immediate\write\forest@auxout{%
8633     \noexpand\forest@external{#1}%
8634     {\expandafter\detokenize\expandafter{#2}}%
8635     {\unexpanded{#3}}%
8636     {\unexpanded{#4}}%
8637   }%
8638 }%
8639 }
```

These two macros include the external picture.

```
8640 \def\forest@includeexternal#1{%
8641   \edef\forest@temp{\pgfkeysvalueof{/forest/external/context}}%
8642   %\typeout{forest: Including external picture '#1' for forest context+code: '\expandafter\detokenize\expand
8643   {%
8644     %\def\pgf@declaredraftimage##1##2{\def\pgf@image{\hbox{}}}%
8645     \tikzsetnextfilename{#1}%
8646     \tikzexternalenable
8647     \tikz{}%
8648   }%
8649 }
8650 \def\forest@includeexternal@box#1#2{%
8651   \global\setbox#1=\hbox{\forest@includeexternal{#2}}%
8652 }
```

This code runs the bracket parser and stage processing.

```
8653 \long\def\forest@begin#1{%
8654   \iffalse{\fi\forest@parsebracket#1}%
8655 }
8656 \def\forest@parsebracket{%
8657   \bracketParse{\forest@get@root@afterthought}\forest@root=%
8658 }
8659 \def\forest@get@root@afterthought{%
8660   \expandafter\forest@get@root@afterthought@\expandafter{\iffalse}\fi
8661 }
8662 \long\def\forest@get@root@afterthought@#1{%
8663   \ifblank{#1}{-}{%
8664     \forest@eappto{\forest@root}{given options}{,afterthought={\unexpanded{#1}}}%
8665   }%
8666   \forest@do
8667 }
8668 \def\forest@do{%
8669   \forest@node@Compute@numeric@ts@info{\forest@root}%
8670   \expandafter\forestset\expandafter{\forest@stages}%
8671   \ifforest@was@tikzexternalwasenable
8672     \tikzexternalenable
8673   \fi
8674 }
```

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!

```

8675 \def\forest@standardnode@new{%
8676   \advance\forest@node@maxid1
8677   \forest@fornode{\the\forest@node@maxid}{%
8678     \forest@node@init
8679     \forestoaset{id}{\forest@cn}%
8680     \forest@node@setname@silent{standard node}%
8681   }%
8682 }
8683 \def\forest@standardnode@calibrate{%
8684   \forest@fornode{\forest@node@Nametoid{standard node}}{%
8685     \edef\forest@environment{\forestove{environment@formula}}%
8686     \forest@get{previous@environment}\forest@previous@environment
8687     \ifx\forest@environment\forest@previous@environment\else
8688       \forestolet{previous@environment}\forest@environment
8689       \forest@node@typeset
8690       \forest@get{calibration@procedure}\forest@temp
8691       \expandafter\forestset\expandafter{\forest@temp}%
8692     \fi
8693   }%
8694 }

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.)

8695 \def\forestStandardNode[#1]#2#3#4{%
8696   \let\forest@standardnode@restoretikzexternal\relax
8697   \ifdefined\tikzexternaldisable
8698     \ifx\tikz\tikzexternal@tikz@replacement
8699       \tikzexternaldisable
8700       \let\forest@standardnode@restoretikzexternal\tikzexternalenable
8701     \fi
8702   \fi
8703   \forest@standardnode@new
8704   \forest@fornode{\forest@node@Nametoid{standard node}}{%
8705     \forestset{content=#1}%
8706     \forest@set{environment@formula}{#2}%
8707     \edef\forest@temp{\unexpanded{#3}}%
8708     \forestolet{calibration@procedure}\forest@temp
8709     \def\forest@calibration@initializing@code{%
8710       \pgfqkeys{/forest/initializing@code}{#4}%
8711       \forestolet{initializing@code}\forest@calibration@initializing@code
8712     \forest@standardnode@restoretikzexternal
8713   }
8714 }
8715 \forestset{initializing@code/.unknown/.code={%
8716   \eappto\forest@calibration@initializing@code{%
8717     \noexpand\forest@get{\forest@node@Nametoid{standard node}}{\pgfkeyscurrentname}\noexpand\forest@temp
8718     \noexpand\forestolet{\pgfkeyscurrentname}\noexpand\forest@temp
8719   }%
8720 }
8721 }

```

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

```

8722 \def\forest@initializefromstandardnode{%
8723   \forestOve{\forest@node@Nametoid{standard node}}{initializing@code}%
8724 }

```

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`.)

```

8725 \forestStandardNode[dj]
8726   {%
8727     \forestOve{\forest@node@Nametoid{standard node}}{content},%
8728     \the\ht\strutbox,\the\pgflinewidth,%
8729     \pgfkeysvalueof{/pgf/inner ysep},\pgfkeysvalueof{/pgf/outer ysep},%
8730     \pgfkeysvalueof{/pgf/inner xsep},\pgfkeysvalueof{/pgf/outer xsep}%
8731   }
8732   {
8733     l sep'/.expanded={\the\dimexpr\the\ht\strutbox+\pgfkeysvalueof{/pgf/inner ysep}},
8734     l={l_sep()+abs(max_y()-min_y())+2*\pgfkeysvalueof{/pgf/outer ysep}},
8735     s sep'/.expanded={\the\dimexpr \pgfkeysvalueof{/pgf/inner xsep}*2}
8736   }
8737   {l sep,l,s sep}

```

10.4 ls coordinate system

```

8738 \pgfqkeys{/forest/@cs}{%
8739   name/.code={%
8740     \edef\forest@cn{\forest@node@Nametoid{#1}}%
8741     \forest@forestcs@resetxy},
8742   id/.code={%
8743     \edef\forest@cn{#1}%
8744     \forest@forestcs@resetxy},
8745   go/.code={%
8746     \forest@go{#1}%
8747     \forest@forestcs@resetxy},
8748   anchor/.code={\forest@forestcs@anchor{#1}},
8749   l/.code={%
8750     \forestmathsetlengthmacro\forest@forestcs@l{#1}%
8751     \forest@forestcs@ls
8752   },
8753   s/.code={%
8754     \forestmathsetlengthmacro\forest@forestcs@s{#1}%
8755     \forest@forestcs@ls
8756   },
8757   .unknown/.code={%
8758     \expandafter\pgfutil@in@\expandafter.\expandafter{\pgfkeyscurrentname}%
8759     \ifpgfutil@in@
8760       \expandafter\forest@forestcs@namegoanchor\pgfkeyscurrentname\forest@end
8761     \else
8762       \expandafter\forest@nameandgo\expandafter{\pgfkeyscurrentname}%
8763       \forest@forestcs@resetxy
8764     \fi
8765   }
8766 }
8767 \def\forest@forestcs@resetxy{%
8768   \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi

```



```

8769 \global\pgf@x\forestove{x}\relax
8770 \global\pgf@y\forestove{y}\relax
8771 }
8772 \def\forest@forestcs@ls{%
8773 \ifdefined\forest@forestcs@l
8774 \ifdefined\forest@forestcs@s
8775 {%
8776 \pgftransformreset
8777 \forest@pgfqtransformrotate{\forestove{grow}}%
8778 \pgfpointransformed{\pgfqpoint{\forest@forestcs@l}{\forest@forestcs@s}}%
8779 }%
8780 \global\advance\pgf@x\forestove{x}%
8781 \global\advance\pgf@y\forestove{y}%
8782 \fi
8783 \fi
8784 }
8785 \def\forest@forestcs@anchor#1{%
8786 \edef\forest@marshal{%
8787 \noexpand\forest@original@tikz@parse@node\relax
8788 (\forestove{name}\ifx\relax#1\relax\else.\fi#1)%
8789 }\forest@marshal
8790 }
8791 \def\forest@forestcs@namegoanchor#1.#2\forest@end{%
8792 \forest@nameandgo{#1}%
8793 \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
8794 \forest@forestcs@anchor{#2}%
8795 }
8796 \def\forest@cs@invalidnodeerror{%
8797 \PackageError{forest}{Attempt to refer to the invalid node by "forest cs"}{}%
8798 }
8799 \tikzdeclarecoordinatesystem{forest}{%
8800 \forest@forthis{%
8801 \forest@forestcs@resetxy
8802 \ifdefined\forest@forestcs@l\undef\forest@forestcs@l\fi
8803 \ifdefined\forest@forestcs@s\undef\forest@forestcs@s\fi
8804 \pgfqkeys{/forest/@cs}{#1}%
8805 }%
8806 }

```

10.5 Relative node names in TikZ

A hack into TikZ's coordinate parser: implements relative node names!

```

8807 \def\forest@tikz@parse@node#1(#2){%
8808 \pgfutil@in@.#2}%
8809 \ifpgfutil@in@
8810 \expandafter\forest@tikz@parse@node@checkiftikzname@withdot
8811 \else%
8812 \expandafter\forest@tikz@parse@node@checkiftikzname@withoutdot
8813 \fi%
8814 #1(#2)\forest@end
8815 }
8816 \def\forest@tikz@parse@node@checkiftikzname@withdot#1(#2.#3)\forest@end{%
8817 \forest@tikz@parse@node@checkiftikzname#1{#2}{.#3}}
8818 \def\forest@tikz@parse@node@checkiftikzname@withoutdot#1(#2)\forest@end{%
8819 \forest@tikz@parse@node@checkiftikzname#1{#2}{}}
8820 \def\forest@tikz@parse@node@checkiftikzname#1#2#3{%
8821 \expandafter\ifx\csname pgf@sh@ns@#2\endcsname\relax
8822 \forest@forthis{%
8823 \forest@nameandgo{#2}%
8824 \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
8825 \edef\forest@temp@relativenodename{\forestove{name}}%

```

```

8826 }%
8827 \else
8828 \def\forest@temp@relativenodename{#2}%
8829 \fi
8830 \expandafter\forest@original@tikz@parse@node\expandafter#1\expandafter(\forest@temp@relativenodename#3)%
8831 }
8832 \def\forest@nameandgo#1{%
8833 \pgfutil@in@!{#1}%
8834 \ifpgfutil@in@
8835 \forest@nameandgo@(#1)%
8836 \else
8837 \ifstrempy{#1}{-}{\edef\forest@cn{\forest@node@Nametoid{#1}}}%
8838 \fi
8839 }
8840 \def\forest@nameandgo@(#1!#2){%
8841 \ifstrempy{#1}{-}{\edef\forest@cn{\forest@node@Nametoid{#1}}}%
8842 \forest@go{#2}%
8843 }

```

10.6 Anchors

FOREST anchors are (child/parent)_anchor and growth anchors parent/children_first/last. The following code resolves them into TikZ 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.

```

8844 \forestset{
8845 declare count={rotate}{0},
8846 autoforward'={rotate}{node options},
8847 }

```

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

```

8848 \newif\ifforest@anchor@snapbordertocompass

```

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

```

8849 \newif\ifforest@anchor@forwardtotikz

```

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

```

8850 \newif\ifforest@anchor@isborder

```

The UI macro.

```

8851 \def\forestanchortotikzanchor#1#2{% #1 = forest anchor, #2 = macro to receive the tikz anchor
8852 \forest@anchor@forwardtotikzfalse
8853 \forest@anchor@do{#1}{\forest@cn}%
8854 \let#2\forest@temp@anchor
8855 }

```

Generic anchors.

```

8856 \pgfdeclaregenericanchor{child anchor}{%
8857 \forest@anchor@forwardtotikztrue
8858 \forest@anchor@do{#1}{child anchor}{\forest@referencednodeid}%
8859 }
8860 \pgfdeclaregenericanchor{parent anchor}{%
8861 \forest@anchor@forwardtotikztrue
8862 \forest@anchor@do{#1}{parent anchor}{\forest@referencednodeid}%
8863 }
8864 \pgfdeclaregenericanchor{anchor}{%
8865 \forest@anchor@forwardtotikztrue
8866 \forest@anchor@do{#1}{anchor}{\forest@referencednodeid}%
8867 }
8868 \pgfdeclaregenericanchor{children}{%

```

```

8869 \forest@anchor@forwardtotikztrue
8870 \forest@anchor@do{#1}{children}{\forest@referencednodeid}%
8871 }
8872 \pgfdeclaregenericanchor{-children}{%
8873 \forest@anchor@forwardtotikztrue
8874 \forest@anchor@do{#1}{-children}{\forest@referencednodeid}%
8875 }
8876 \pgfdeclaregenericanchor{children first}{%
8877 \forest@anchor@forwardtotikztrue
8878 \forest@anchor@do{#1}{children first}{\forest@referencednodeid}%
8879 }
8880 \pgfdeclaregenericanchor{-children first}{%
8881 \forest@anchor@forwardtotikztrue
8882 \forest@anchor@do{#1}{-children first}{\forest@referencednodeid}%
8883 }
8884 \pgfdeclaregenericanchor{first}{%
8885 \forest@anchor@forwardtotikztrue
8886 \forest@anchor@do{#1}{first}{\forest@referencednodeid}%
8887 }
8888 \pgfdeclaregenericanchor{parent first}{%
8889 \forest@anchor@forwardtotikztrue
8890 \forest@anchor@do{#1}{parent first}{\forest@referencednodeid}%
8891 }
8892 \pgfdeclaregenericanchor{-parent first}{%
8893 \forest@anchor@forwardtotikztrue
8894 \forest@anchor@do{#1}{-parent first}{\forest@referencednodeid}%
8895 }
8896 \pgfdeclaregenericanchor{parent}{%
8897 \forest@anchor@forwardtotikztrue
8898 \forest@anchor@do{#1}{parent}{\forest@referencednodeid}%
8899 }
8900 \pgfdeclaregenericanchor{-parent}{%
8901 \forest@anchor@forwardtotikztrue
8902 \forest@anchor@do{#1}{-parent}{\forest@referencednodeid}%
8903 }
8904 \pgfdeclaregenericanchor{parent last}{%
8905 \forest@anchor@forwardtotikztrue
8906 \forest@anchor@do{#1}{parent last}{\forest@referencednodeid}%
8907 }
8908 \pgfdeclaregenericanchor{-parent last}{%
8909 \forest@anchor@forwardtotikztrue
8910 \forest@anchor@do{#1}{-parent last}{\forest@referencednodeid}%
8911 }
8912 \pgfdeclaregenericanchor{last}{%
8913 \forest@anchor@forwardtotikztrue
8914 \forest@anchor@do{#1}{last}{\forest@referencednodeid}%
8915 }
8916 \pgfdeclaregenericanchor{children last}{%
8917 \forest@anchor@forwardtotikztrue
8918 \forest@anchor@do{#1}{children last}{\forest@referencednodeid}%
8919 }
8920 \pgfdeclaregenericanchor{-children last}{%
8921 \forest@anchor@forwardtotikztrue
8922 \forest@anchor@do{#1}{-children last}{\forest@referencednodeid}%
8923 }
8924 \pgfdeclaregenericanchor{children'}{%
8925 \forest@anchor@forwardtotikztrue
8926 \forest@anchor@do{#1}{children'}{\forest@referencednodeid}%
8927 }
8928 \pgfdeclaregenericanchor{-children'}{%
8929 \forest@anchor@forwardtotikztrue

```

```

8930 \forest@anchor@do{#1}{-children'}{\forest@referencednodeid}%
8931 }
8932 \pgfdeclaregenericanchor{children first'}{%
8933 \forest@anchor@forwardtotikztrue
8934 \forest@anchor@do{#1}{children first'}{\forest@referencednodeid}%
8935 }
8936 \pgfdeclaregenericanchor{-children first'}{%
8937 \forest@anchor@forwardtotikztrue
8938 \forest@anchor@do{#1}{-children first'}{\forest@referencednodeid}%
8939 }
8940 \pgfdeclaregenericanchor{first'}{%
8941 \forest@anchor@forwardtotikztrue
8942 \forest@anchor@do{#1}{first'}{\forest@referencednodeid}%
8943 }
8944 \pgfdeclaregenericanchor{parent first'}{%
8945 \forest@anchor@forwardtotikztrue
8946 \forest@anchor@do{#1}{parent first'}{\forest@referencednodeid}%
8947 }
8948 \pgfdeclaregenericanchor{-parent first'}{%
8949 \forest@anchor@forwardtotikztrue
8950 \forest@anchor@do{#1}{-parent first'}{\forest@referencednodeid}%
8951 }
8952 \pgfdeclaregenericanchor{parent'}{%
8953 \forest@anchor@forwardtotikztrue
8954 \forest@anchor@do{#1}{parent'}{\forest@referencednodeid}%
8955 }
8956 \pgfdeclaregenericanchor{-parent'}{%
8957 \forest@anchor@forwardtotikztrue
8958 \forest@anchor@do{#1}{-parent'}{\forest@referencednodeid}%
8959 }
8960 \pgfdeclaregenericanchor{parent last'}{%
8961 \forest@anchor@forwardtotikztrue
8962 \forest@anchor@do{#1}{parent last'}{\forest@referencednodeid}%
8963 }
8964 \pgfdeclaregenericanchor{-parent last'}{%
8965 \forest@anchor@forwardtotikztrue
8966 \forest@anchor@do{#1}{-parent last'}{\forest@referencednodeid}%
8967 }
8968 \pgfdeclaregenericanchor{last'}{%
8969 \forest@anchor@forwardtotikztrue
8970 \forest@anchor@do{#1}{last'}{\forest@referencednodeid}%
8971 }
8972 \pgfdeclaregenericanchor{children last'}{%
8973 \forest@anchor@forwardtotikztrue
8974 \forest@anchor@do{#1}{children last'}{\forest@referencednodeid}%
8975 }
8976 \pgfdeclaregenericanchor{-children last'}{%
8977 \forest@anchor@forwardtotikztrue
8978 \forest@anchor@do{#1}{-children last'}{\forest@referencednodeid}%
8979 }

```

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

```

8980 \def\forest@anchor@do#1#2#3{% #1 = shape name, #2 = (potentially) forest anchor, #3 = node id
8981 \forest@fornode{#3}{%
8982 \def\forest@temp@anchor{#2}%
8983 \forest@anchor@snappbordertocompassfalse
8984 \forest@anchor@isborderfalse
8985 \forest@anchor@to@tikz@anchor
8986 \forest@anchor@border@to@compass
8987 \ifforest@anchor@forwardtotikz
8988 \forest@anchor@forward{#1}%

```

```

8989   \else
8990   \fi
8991 }%
8992 }

```

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

```

8993 \def\forest@anchor@to@tikz@anchor{%
8994   \ifcsdef{forest@anchor@@\forest@temp@anchor}{%
8995     \csuse{forest@anchor@@\forest@temp@anchor}%
8996     \forest@anchor@to@tikz@anchor
8997   }{}%
8998 }

Actual computation.

8999 \csdef{forest@anchor@@parent anchor}{%
9000   \foresttoget{parent anchor}\forest@temp@anchor}
9001 \csdef{forest@anchor@@child anchor}{%
9002   \foresttoget{child anchor}\forest@temp@anchor}
9003 \csdef{forest@anchor@@anchor}{%
9004   \foresttoget{anchor}\forest@temp@anchor}
9005 \csdef{forest@anchor@@children'}{%
9006   \forest@anchor@isbordertrue
9007   \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}}%
9008 }
9009 \csdef{forest@anchor@@-children'}{%
9010   \forest@anchor@isbordertrue
9011   \edef\forest@temp@anchor{\number\numexpr 180+\forestove{grow}-\forestove{rotate}}%
9012 }
9013 \csdef{forest@anchor@@parent'}{%
9014   \forest@anchor@isbordertrue
9015   \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestove{\forestove{@parent}}{gro
9016   \edef\forest@temp@anchor{\number\numexpr\forest@temp@grow-\forestove{rotate}+180}%
9017 }
9018 \csdef{forest@anchor@@-parent'}{%
9019   \forest@anchor@isbordertrue
9020   \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestove{\forestove{@parent}}{gro
9021   \edef\forest@temp@anchor{\number\numexpr\forest@temp@grow-\forestove{rotate}}%
9022 }
9023 \csdef{forest@anchor@@first'}{%
9024   \forest@anchor@isbordertrue
9025   \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=0 -\e
9026 }
9027 \csdef{forest@anchor@@last'}{%
9028   \forest@anchor@isbordertrue
9029   \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=0 +\e
9030 }
9031 \csdef{forest@anchor@@parent first'}{%
9032   \forest@anchor@isbordertrue
9033   \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestove{\forestove{@parent}}{gro
9034   \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestove{\forestove{@pare
9035   \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}+180}%
9036   \edef\forest@temp@anchor@first{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@revers
9037   \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@first}\forest@temp@anchor
9038 }
9039 \csdef{forest@anchor@@-parent first'}{%
9040   \forest@anchor@isbordertrue
9041   \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestove{\forestove{@parent}}{gro
9042   \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestove{\forestove{@pare
9043   \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}}%
9044   \edef\forest@temp@anchor@first{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@revers
9045   \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@first}\forest@temp@anchor
9046 }

```

```

9047 \csdef{forest@anchor@@parent last'}{%
9048   \forest@anchor@isbordertrue
9049   \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestove{\forestove{@parent}}{gro
9050   \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestove{\forestove{@pare
9051   \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}+180}%
9052   \edef\forest@temp@anchor@last{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@reverse
9053   \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@last}\forest@temp@anchor
9054 }
9055 \csdef{forest@anchor@@-parent last'}{%
9056   \forest@anchor@isbordertrue
9057   \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestove{\forestove{@parent}}{gro
9058   \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestove{\forestove{@pare
9059   \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}}}%
9060   \edef\forest@temp@anchor@last{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@reverse
9061   \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@last}\forest@temp@anchor
9062 }
9063 \csdef{forest@anchor@@children first'}{%
9064   \forest@anchor@isbordertrue
9065   \edef\forest@temp@anchor@first{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}
9066   \forest@getaverageangle{\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@first}\forest@temp@anchor
9067 }
9068 \csdef{forest@anchor@@-children first'}{%
9069   \forest@anchor@isbordertrue
9070   \edef\forest@temp@anchor@first{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}
9071   \forest@getaverageangle{180+\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@first}\forest@temp@anc
9072 }
9073 \csdef{forest@anchor@@children last'}{%
9074   \forest@anchor@isbordertrue
9075   \edef\forest@temp@anchor@last{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=
9076   \forest@getaverageangle{\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@last}\forest@temp@anchor
9077 }
9078 \csdef{forest@anchor@@-children last'}{%
9079   \forest@anchor@isbordertrue
9080   \edef\forest@temp@anchor@last{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=
9081   \forest@getaverageangle{180+\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@last}\forest@temp@anch
9082 }
9083 \csdef{forest@anchor@@children}{%
9084   \forest@anchor@snapbordertocompasstrue
9085   \csuse{forest@anchor@@children'}%
9086 }
9087 \csdef{forest@anchor@@-children}{%
9088   \forest@anchor@snapbordertocompasstrue
9089   \csuse{forest@anchor@@-children'}%
9090 }
9091 \csdef{forest@anchor@@parent}{%
9092   \forest@anchor@snapbordertocompasstrue
9093   \csuse{forest@anchor@@parent'}%
9094 }
9095 \csdef{forest@anchor@@-parent}{%
9096   \forest@anchor@snapbordertocompasstrue
9097   \csuse{forest@anchor@@-parent'}%
9098 }
9099 \csdef{forest@anchor@@first}{%
9100   \forest@anchor@snapbordertocompasstrue
9101   \csuse{forest@anchor@@first'}%
9102 }
9103 \csdef{forest@anchor@@last}{%
9104   \forest@anchor@snapbordertocompasstrue
9105   \csuse{forest@anchor@@last'}%
9106 }
9107 \csdef{forest@anchor@@parent first}{%

```

```

9108 \forest@anchor@snapbordertocompasstrue
9109 \csuse{forest@anchor@@parent first'}%
9110 }
9111 \csdef{forest@anchor@@parent first}{%
9112 \forest@anchor@snapbordertocompasstrue
9113 \csuse{forest@anchor@@-parent first'}%
9114 }
9115 \csdef{forest@anchor@@parent last}{%
9116 \forest@anchor@snapbordertocompasstrue
9117 \csuse{forest@anchor@@parent last'}%
9118 }
9119 \csdef{forest@anchor@@-parent last}{%
9120 \forest@anchor@snapbordertocompasstrue
9121 \csuse{forest@anchor@@-parent last'}%
9122 }
9123 \csdef{forest@anchor@@children first}{%
9124 \forest@anchor@snapbordertocompasstrue
9125 \csuse{forest@anchor@@children first'}%
9126 }
9127 \csdef{forest@anchor@@-children first}{%
9128 \forest@anchor@snapbordertocompasstrue
9129 \csuse{forest@anchor@@-children first'}%
9130 }
9131 \csdef{forest@anchor@@children last}{%
9132 \forest@anchor@snapbordertocompasstrue
9133 \csuse{forest@anchor@@children last'}%
9134 }
9135 \csdef{forest@anchor@@-children last}{%
9136 \forest@anchor@snapbordertocompasstrue
9137 \csuse{forest@anchor@@-children last'}%
9138 }

```

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

```

9139 \def\forest@getaverageangle#1#2#3{%
9140 \edef\forest@temp{\number\numexpr #1-#2+540}%
9141 \expandafter\pgfmathMod@\expandafter{\forest@temp}{360}%
9142 \forest@truncatepgfmathresult
9143 \edef\forest@temp{\number\numexpr 360+#2+((\pgfmathresult-180)/2)}%
9144 \expandafter\pgfmathMod@\expandafter{\forest@temp}{360}%
9145 \forest@truncatepgfmathresult
9146 \let#3\pgfmathresult
9147 }
9148 \def\forest@truncatepgfmathresult{%
9149 \afterassignment\forest@gobbletoEND
9150 \forest@temp@count=\pgfmathresult\forest@END
9151 \def\pgfmathresult{\the\forest@temp@count}%
9152 }
9153 \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.

```

9154 \def\forest@anchor@border@to@compass{%
9155 \ifforest@anchor@isborder % snap to 45 deg, to range 0-360
9156 \ifforest@anchor@snapbordertocompass
9157 \forest@anchor@snap@border@to@compass
9158 \else % to range 0-360
9159 \pgfmathMod@{\forest@temp@anchor}{360}%
9160 \forest@truncatepgfmathresult
9161 \let\forest@temp@anchor\pgfmathresult

```

```

9162 \fi
9163 \ifforest@anchor@snapbordertocompass
9164 \ifforest@anchor@forwardtotikz
9165 \ifcsname pgf@anchor%
9166 @\csname pgf@sh@ns@\pgfreferencednodename\endcsname
9167 @\csname forest@compass@\forest@temp@anchor\endcsname
9168 \endcsname
9169 \letcs\forest@temp@anchor{forest@compass@\forest@temp@anchor}%
9170 \fi
9171 \else
9172 \letcs\forest@temp@anchor{forest@compass@\forest@temp@anchor}%
9173 \fi
9174 \fi
9175 \fi
9176 }
9177 \csdef{forest@compass@0}{east}
9178 \csdef{forest@compass@45}{north east}
9179 \csdef{forest@compass@90}{north}
9180 \csdef{forest@compass@135}{north west}
9181 \csdef{forest@compass@180}{west}
9182 \csdef{forest@compass@225}{south west}
9183 \csdef{forest@compass@270}{south}
9184 \csdef{forest@compass@315}{south east}
9185 \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).

```

9186 \def\forest@anchor@snap@border@to@compass{%
9187 \pgfmathMod@{\forest@temp@anchor}{360}%
9188 \pgfmathdivide@{\pgfmathresult}{45}%
9189 \pgfmathround@{\pgfmathresult}%
9190 \pgfmathmultiply@{\pgfmathresult}{45}%
9191 \forest@truncatepgfmathresult
9192 \let\forest@temp@anchor\pgfmathresult
9193 }

```

This macro forwards the resulting anchor to TikZ.

```

9194 \def\forest@anchor@forward#1{% #1 = shape name
9195 \ifdefempty\forest@temp@anchor{%
9196 \pgf@sh@reanchor{#1}{center}%
9197 \xdef\forest@hack@tikzshapeborder{%
9198 \noexpand\tikz@shapebordertrue
9199 \def\noexpand\tikz@shapeborder@name{\pgfreferencednodename}%
9200 }\aftergroup\forest@hack@tikzshapeborder
9201 }{%
9202 \pgf@sh@reanchor{#1}{\forest@temp@anchor}%
9203 }%
9204 }

```

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

```

9205 \def\forest@referencednodeid{\forest@node@Nametoid{\forest@referencednodename}}%
9206 \def\forest@referencednodename{%
9207 \expandafter\expandafter\expandafter\forest@referencednodename@expandafter\pgfreferencednodename\forest@pgf
9208 }
9209 \expandafter\def\expandafter\forest@referencednodename@expandafter#\expandafter1\forest@pgf@notyetpositioned
9210 \if\relax#1\relax\forest@referencednodename@stripafter#2\relax\fi
9211 \if\relax#2\relax#1\fi
9212 }
9213 \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.

```
9214 \def\forest@pointanchor#1{% #1 = anchor
9215   \forest@Pointanchor{\forest@cn}{#1}%
9216 }
9217 \def\forest@Pointanchor#1#2{% #1 = node id, #2 = anchor
9218   \def\forest@pa@temp@name{name}%
9219   \forest@ifdefined{#1}{@box}{%
9220     \forest@get{#1}{@box}\forest@temp
9221     \ifdefempty\forest@temp{}{%
9222       \def\forest@pa@temp@name{later@name}%
9223     }%
9224   }{}%
9225   \setbox0\hbox{%
9226     \begin{pgfpicture}%
9227       \if\relax\forest@ve{#1}{#2}\relax
9228         \pgfpointanchor{\forest@ve{#1}{\forest@pa@temp@name}}{center}%
9229       \else
9230         \pgfpointanchor{\forest@ve{#1}{\forest@pa@temp@name}}{\forest@ve{#1}{#2}}%
9231       \fi
9232       \xdef\forest@global@marshal{%
9233         \noexpand\global\noexpand\pgf@x=\the\pgf@x\relax
9234         \noexpand\global\noexpand\pgf@y=\the\pgf@y\relax\relax
9235       }%
9236     \end{pgfpicture}%
9237   }%
9238   \forest@global@marshal
9239 }
```

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

```
9240 \forest@node@init
```

11 Compatibility with previous versions

```
9241 \ifdefempty{\forest@compat}{}{%
9242   \RequirePackage{forest-compat}
9243 }
```