

Implementation of FOREST, a PGF/TikZ-based package for drawing linguistic trees

v2.1

Sašo Živanović*

December 5, 2016

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	33
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	52
7.1	.process	54
7.1.1	Registers	63
7.1.2	Declaring options	69
7.1.3	Option propagation	75
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	111
8.2	Packing	113
8.2.1	Tiers	125
8.2.2	Node boundary	129
8.3	Compute absolute positions	134
8.4	Drawing the tree	134
9	Geometry	137
9.1	Projections	137
9.2	Break path	140
9.3	Get tight edge of path	142
9.4	Get rectangle/band edge	148
9.5	Distance between paths	149
9.6	Utilities	151
10	The outer UI	153
10.1	Externalization	153
10.2	The forest environment	155
10.3	Standard node	158
10.4	ls coordinate system	159
10.5	Relative node names in <i>TikZ</i>	161
10.6	Anchors	161
11	Compatibility with previous versions	168

1 Identification

```

1 \ProvidesPackage{forest}[2016/12/05 v2.1 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{pgfopts}
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\ifforesttikzcschack
87 \newif\ifforest@install@keys@to@tikz@path@
88 \newif\ifforestdebugnodewalks

```

```

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\curname #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\curname #1repeat#2\endcsname{%
184         \noexpand\csdef{#1iterate#2}{####1\relax\noexpand\expandafter\expandafter\noexpand\curname#1iterate#
185         \expandafter\noexpand\curname#1iterate#2\endcsname
186         \let\expandafter\noexpand\curname#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     \edef\safeloop@marshal{%
196       \noexpand\csdef{safeiterate@#1}{%
197         \unexpanded{##1}\relax
198         \noexpand\expandafter
199         \expandonce{\csname safeiterate@#1\endcsname}%
200         \noexpand\fi
201       }%
202     }\safeloop@marshal
203     \csuse{safeiterate@#1}%
204     \advance\noexpand\safeloop@depth-1\relax
205     \cslet{safeiterate@#1}\relax
206   }%
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   \csdef{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     \edef{safeRKloop@marshal}%
221       \noexpand\csdef{safeRKiterate@#1}{%
222         \unexpanded{##1}\relax
223         \noexpand\expandafter
224         \expandonce{\csname safeRKiterate@#1\endcsname}%
225         \noexpand\fi
226       }%
227     }\safeRKloop@marshal
228     \csuse{safeRKiterate@#1}%
229     \advance\noexpand\safeRKloop@depth-1\relax
230     \cslet{safeRKiterate@#1}\relax
231   }%
232   \expandafter\newif\csname ifsafeRKbreak@the\safeRKloop@depth\endcsname
233 }%
234 \newcount\safeRKloop@depth
235 \def\safeRKloop{%
236   \advance\safeRKloop@depth1
237   \ifcsdef{safeRKloop@the\safeRKloop@depth}{-}\expandafter\nnewsafeRKloop\expandafter{\the\safeRKloop@depth}}%
238   \csdef{safeRKloopn@the\safeRKloop@depth}{0}%
239   \csuse{safeRKbreak@the\safeRKloop@depth false}%
240   \csuse{safeRKloop@the\safeRKloop@depth}%
241   \csdef{safeRKloopn@the\safeRKloop@depth}{\number\numexpr\csuse{safeRKloopn@the\safeRKloop@depth}+1}%
242 }
243 \let\safeRKrepeat\fi
244 \def\safeRKloopn{\csuse{safeRKloopn@the\safeRKloop@depth}}%

```

Additional loops (for embedding).

```

245 \newloop\forest@loop

```



```

300 \escapeif\@gobble
301 \else
302 \escapeif\forest@cotu@nospaceB
303 \fi
304 }
305 \def\forest@cotu@nospaceB#1{%
306 \ifcat#1a%
307 \let\forest@cotu@next\forest@cotu@have@alpha
308 \else
309 \if!\ifnum9<1#1!\fi
310 \let\forest@cotu@next\forest@cotu@have@num
311 \else
312 \let\forest@cotu@next\forest@cotu@have@other
313 \fi
314 \fi
315 \forest@cotu@next#1%
316 }
317 \def\forest@cotu@have@alpha#1{%
318 \appto\forest@cotu@result{#1}%
319 \forest@cotu
320 }
321 \def\forest@cotu@have@other#1{%
322 \appto\forest@cotu@result{#1}%
323 \forest@cotu
324 }

```

Additional list macros.

```

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

```

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

```

347 \def\forest@strip@braces#1{%
348 \forest@strip@braces@A#1\forest@strip@braces@preend\forest@strip@braces@end
349 }
350 \def\forest@strip@braces@A#1#2\forest@strip@braces@end{%
351 #1\ifx\forest@strip@braces@preend#2\else\escapeif{\forest@strip@braces@A#2\forest@strip@braces@end}\fi
352 }

```

Utilities dealing with pgfkeys.

```

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

```



```

356 }%
357 \pgfkeysgetvalue{#1/.@cmd}\forest@temp
358 \pgfkeyslet{#2/.@cmd}\forest@temp
359 \pgfkeysifdefined{#1/.@args}{%
360   \pgfkeysgetvalue{#1/.@args}\forest@temp
361   \pgfkeyslet{#2/.@args}\forest@temp
362 }{}%
363 \pgfkeysifdefined{#1/.@body}{%
364   \pgfkeysgetvalue{#1/.@body}\forest@temp
365   \pgfkeyslet{#2/.@body}\forest@temp
366 }{}%
367 \pgfkeysifdefined{#1/.@body}{%
368   \pgfkeysgetvalue{#1/.@body}\forest@temp
369   \pgfkeyslet{#2/.@body}\forest@temp
370 }{}%
371 \pgfkeysifdefined{#1/.@def}{%
372   \pgfkeysgetvalue{#1/.@def}\forest@temp
373   \pgfkeyslet{#2/.@def}\forest@temp
374 }{}%
375 }
376 \forestset{
377   copy command key/.code 2 args={\forest@copycommandkey{#1}{#2}},
378   autoforward/.code 2 args={\forest@autoforward{#1}{#2={#1={##1}}}{true}},
379   autoforward'/.code 2 args={\forest@autoforward{#1}{#2=#1,#2={#1={##1}}}{true}},
380   Autoforward/.code 2 args={\forest@autoforward{#1}{#2}{true}},
381   autoforward register/.code 2 args={\forest@autoforward{#1}{#2={#1={##1}}}{false}},
382   autoforward register'/.code 2 args={\forest@autoforward{#1}{#2=#1,#2={#1={##1}}}{false}},
383   Autoforward register/.code 2 args={\forest@autoforward{#1}{#2}{false}},
384   copy command key@if it exists/.code 2 args={%
385     \pgfkeysifdefined{#1/.@cmd}{%
386       \forest@copycommandkey{#1}{#2}%
387     }{}%
388   },
389   unautoforward/.style={
390     typeout={unautoforwarding #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     copy command key@if it exists={/forest/autoforwarded +#1}{/forest/+#1},
402   },
403   /handlers/.undef/.code={\csundef{pgfk@pgfkeyscurrentpath}},
404   undef option/.style={
405     /forest/#1/.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/+#1/.@cmd/.undef,

```

```

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

```

```

478 \ifx\forest@temp\forestnovalue
479 \expandafter\forest@remove@key@installer@defwithvalue\expandafter{\pgfkeyscurrentname}{\pgfkeysnovalu
480 \else
481 \expandafter\forest@remove@key@installer@defwithvalue\expandafter{\pgfkeyscurrentname}{#1}%
482 \fi
483 \fi
484 },
485 remove@key/.unknown/.code={% #1 = (inner) value
486 \expandafter\apptotoks\expandafter\forest@temp@toks\expandafter{\pgfkeyscurrentname={#1}},}%
487 },
488 remove@key@uninstaller/.unknown/.code={%
489 \pgfkeyslet{/forest/remove@key/\pgfkeyscurrentname/.@cmd}\@undefined},
490 }
491 \def\forest@remove@key@installer@defwithvalue#1#2{% #1=key name, #2 = outer value
492 \pgfkeysdef{/forest/remove@key/#1}{% ##1 = inner value
493 \def\forest@temp@outer{#2}%
494 \def\forest@temp@inner{##1}%
495 \ifx\forest@temp@outer\forest@temp@inner
496 \else
497 \apptotoks\forest@temp@toks{#1={##1}},}%
498 \fi
499 }%
500 }
501 \forestset{
502 show register/.code={%
503 \forestrget{#1}\foresttemp
504 \typeout{Forest register "#1"=\expandafter\detokenize\expandafter{\foresttemp}}%
505 },
506 }

```

4.1 Arrays

```

507 \def\forest@newarray#1{%
508 \forest@tempfalse % non-global
509 {%
510 \escapechar=-1
511 \expandafter\escapechar\expandafter\count@\expandafter
512 }%
513 \expandafter\forest@newarray@\expandafter{\string#1}%
514 }
515 \def\forest@newglobalarray#1{%
516 \forest@temptrue % global
517 {%
518 \escapechar=-1
519 \expandafter\escapechar\expandafter\count@\expandafter
520 }%
521 \expandafter\forest@newarray@\expandafter{\string#1}%
522 }
523 \def\forest@array@empty@error#1{%
524 \PackageError{forest}{Cannot pop from empty array "#1".}{}}%
525 \def\forest@array@oub@error#1#2{%
526 \PackageError{forest}{#2 is out of bounds of array "#1"
527 (\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.

```

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

```

```

533 \ifforest@temp\global\fi\expandonce{\csname#1M\endcsname}0
534 \ifforest@temp\global\fi\expandonce{\csname#1N\endcsname}0
535 }%
536 \csedef{#1lifempty}{%
537 \noexpand\ifnum\expandonce{\csname#1M\endcsname}<\expandonce{\csname#1N\endcsname}\relax
538 \unexpanded{\expandafter\@secondoftwo
539 \else
540 \expandafter\@firstoftwo
541 \fi}%
542 }%
543 \csedef{#1length}{% a numexpr
544 \noexpand\numexpr\expandonce{\csname#1N\endcsname}--\expandonce{\csname#1M\endcsname}\relax
545 }%
546 \csedef{#1checkrange}##1##2{% args can be \numexprs
547 \noexpand\forest@tempfalse
548 \noexpand\ifnum\numexpr##1<\expandonce{\csname#1M\endcsname}\relax
549 \noexpand\forest@temptrue
550 \noexpand\fi
551 \noexpand\ifnum\numexpr##2>\expandonce{\csname#1N\endcsname}\relax
552 \noexpand\forest@temptrue
553 \noexpand\fi
554 \noexpand\ifforest@temp
555 \noexpand\forest@array@oub@error{#1}{Range "\noexpand\number\noexpand\numexpr##1\relax--\noexpand\number\noexpand\numexpr##2\relax"}%
556 \noexpand\fi
557 }%
558 \csedef{#1checkindex}##1{% arg can be a \numexpr
559 \noexpand\forest@tempfalse
560 \noexpand\ifnum\numexpr##1<\expandonce{\csname#1M\endcsname}\relax
561 \noexpand\forest@temptrue
562 \noexpand\fi
563 \noexpand\ifnum\numexpr##1<\expandonce{\csname#1N\endcsname}\relax
564 \noexpand\else
565 \noexpand\forest@temptrue
566 \noexpand\fi
567 \noexpand\ifforest@temp
568 \noexpand\forest@array@oub@error{#1}{Index "\noexpand\number\noexpand\numexpr##1\relax"}%
569 \noexpand\fi
570 }%
571 \csedef{#1get}##1##2{% ##1 = index, ##2 = receiving cs
572 \expandonce{\csname#1checkindex\endcsname}{##1}%
573 \noexpand\letcs##2{##1}%
574 }%
575 \csedef{#1get0}##1##2{% ##1 = index, ##2 = receiving cs (don't check bounds)
576 \noexpand\letcs##2{##1}%
577 }%
578 \csedef{#1toppop}##1{% ##1 = receiving cs
579 \expandonce{\csname#1lifempty\endcsname}{%
580 \noexpand\forest@array@empty@error{#1}%
581 }{%
582 \ifforest@temp\global\fi\advance\expandonce{\csname#1N\endcsname}-1
583 \noexpand\letcs\noexpand##1{#1\noexpand\the\expandonce{\csname#1N\endcsname}}%
584 }%
585 }%
586 \InlineNoDef{\csdef{#1bottompop}##1{% ##1 = receiving cs
587 \Expand{\csname#1lifempty\endcsname}{%
588 \forest@array@empty@error{#1}%
589 }{%
590 \letcs##1{#1\the\Expand{\csname#1M\endcsname}}%
591 \ExpandIfT{forest@temp}\global\advance\Expand{\csname#1M\endcsname}1}%
592 }%
593 }}%

```

```

594 % \csdef{#1bottompop}##1{% we need this as \Inline chokes on \let\macro=\relax
595 % \expandafter\Inline\expandafter\def\csname#1bottompop\endcsname##1{% ##1 = receiving cs
596 %   \Expand{\csname#1ifempty\endcsname}{%
597 %     \forest@array@empty@error{#1}%
598 %   }{%
599 %     \letcs##1{#1\the\Expand{\csname#1M\endcsname}}}%
600 %   \ExpandIfT{\forest@temp}\global\advance\Expand{\csname#1M\endcsname 1}%
601 %   }%
602 % }%
603 % \csedef{#1bottompop}##1{% ##1 = receiving cs
604 %   \expandonce{\csname#1ifempty\endcsname}{%
605 %     \noexpand\forest@array@empty@error{#1}%
606 %   }{%
607 %     \noexpand\letcs\noexpand##1{#1\noexpand\the\expandonce{\csname#1M\endcsname}}}%
608 %     \ifforest@temp\global\fi\advance\expandonce{\csname#1M\endcsname}1
609 %   }%
610 % }%
611 \csedef{#1setappend}##1{% ##1 = definition
612   \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi
613   {#1\noexpand\the\expandonce{\csname#1N\endcsname}}}%
614   {\noexpand\unexpanded{##1}}%
615   \ifforest@temp\global\fi\advance\expandonce{\csname#1N\endcsname}1
616 }%
617 \csedef{#1setappend@}##1##2{% ##1 = continue by, ##2 = definition
618   \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi
619   {#1\noexpand\the\expandonce{\csname#1N\endcsname}}}%
620   {\noexpand\unexpanded{##2}}%
621   \ifforest@temp\global\fi\advance\expandonce{\csname#1N\endcsname}1
622   ##1%
623 }%
624 \csedef{#1setprepend}##1{% ##1 = definition
625   \ifforest@temp\global\fi\advance\expandonce{\csname#1M\endcsname}-1
626   \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi
627   {#1\noexpand\the\expandonce{\csname#1M\endcsname}}}%
628   {\noexpand\unexpanded{##1}}%
629 }%
630 \csedef{#1esetappend}##1{% ##1 = definition
631   \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi{#1\noexpand\the\expandonce{\csname#1N\endcsname}}}%
632   \ifforest@temp\global\fi\advance\expandonce{\csname#1N\endcsname}1
633 }%
634 \csedef{#1esetprepend}##1{% ##1 = definition
635   \ifforest@temp\global\fi\advance\expandonce{\csname#1M\endcsname}-1
636   \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi{#1\noexpand\the\expandonce{\csname#1M\endcsname}}}%
637 }%
638 \csedef{#1letappend}##1{% ##1 = cs
639   \ifforest@temp\noexpand\expandafter\noexpand\global\fi\noexpand\expandafter\noexpand\let
640   \noexpand\csname#1\noexpand\the\expandonce{\csname#1N\endcsname}\noexpand\endcsname
641   ##1%
642   \ifforest@temp\global\fi\advance\expandonce{\csname#1N\endcsname}1
643 }%
644 \csedef{#1letprepend}##1{% ##1 = cs
645   \ifforest@temp\global\fi\advance\expandonce{\csname#1M\endcsname}-1
646   \ifforest@temp\noexpand\expandafter\noexpand\global\fi\noexpand\expandafter\noexpand\let
647   \noexpand\csname#1\noexpand\the\expandonce{\csname#1M\endcsname}\noexpand\endcsname
648   ##1%
649 }%

```

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#1

```

```

\expandafter\expandafter\expandafter{%
  \expandafter\the
  \expandafter\arrayM %\arrayM <-- \expandonce{\csname#1M\endcsname}%
  \expandafter}%
\expandafter{%
  \the\arrayN %\arrayN <-- \expandonce{\csname#1N\endcsname}%
  }%
}%

650 \csedef{#1values}{%
651   \noexpand\expandafter\noexpand\expandafter\noexpand\expandafter\expandonce{\csname#1valuesfromrange\endcsname}%
652   \noexpand\expandafter\noexpand\expandafter\noexpand\expandafter{%
653     \noexpand\expandafter\noexpand\the
654     \noexpand\expandafter\expandonce{\csname#1M\endcsname}%
655     \noexpand\expandafter}%
656   \noexpand\expandafter{\noexpand\the\expandonce{\csname#1N\endcsname}}}%
657 }%

\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#1valuesf
  \expandafter{\number\numexpr##1+1}{##2}}}%
\fi
}%

```

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

```

658 \csedef{#1valuesfromrange}##1##2{% ##1/##2 = lower/upper bounds (we receive them expanded)
659   \noexpand\ifnum##1<##2
660     {\noexpand\expandafter\noexpand\expandonce\noexpand\expandafter{\noexpand\csname#1##1\noexpand\endcsname}%
661     \noexpand\expandafter\noexpand\@escapeif\noexpand\expandafter{\noexpand\expandafter\expandonce{\csname#1
662     \noexpand\expandafter{\noexpand\number\noexpand\numexpr##1+1}{##2}}}%
663   \noexpand\fi
664 }%

```

Puts all items until \forest@eov into the array. After that is done, execute \forest@topextend@next.

This code is difficult and not run often, so it doesn't need specialized control sequences.

```

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

```

```

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

```

4.2 Testing for numbers and dimensions

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

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

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

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

```

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

```

Expand in advance, like `pgfmath` does.

```

707   \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.

```

708   \expandafter\forest@isnum@a\forest@isnum@temp\forest@eon\forest@eon\forest@END
709   \ifforest@isnum
710     \expandafter\@firstoftwo
711   \else
712     \expandafter\@secondoftwo
713   \fi
714 }

```

```

715 \def\forest@isnum@a{\futurelet\forest@isnum@token\forest@isnum@b}

```

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

```

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

```

```

733 \forest@next
734 }

```

Eat + and -.

```

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

```

Eat the space!

```

736 \def\forest@isnum@s#1{\forest@isnum@a#1}%
737 \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.

```

738 \def\forest@isnum@p{%
739   \ifx0\forest@isnum@token
740     \let\forest@next\forest@isnum@next
741   \else
742     \let\forest@next\forest@isnum@nz@
743   \fi
744   \forest@isnumtrue
745   \afterassignment\forest@isnum@q\forest@isnum@count\ifforest@isnum@minus-\fi0%
746 }
747 \def\forest@isnum@q{%
748   \futurelet\forest@isnum@token\forest@next
749 }
750 \def\forest@isnum@nz@{%
751   \ifnum\forest@isnum@count=0
752     \forest@isnumfalse
753   \fi
754   \forest@isnum@next
755 }

```

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

```

756 \def\forest@isnum@finish#1\forest@END{%
757   \ifx\forest@isnum@token\forest@eon
758   \else
759     \forest@isnumfalse
760   \fi
761 }

```

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.

```

762 \newcount\forest@isdim@nonintpart
763 \newif\ifforest@isdim
764 \def\forest@isdim#1{%
765   \forest@isdimfalse
766   \forest@isnum@minusfalse
767   \def\forest@isdim@leadingzeros{}%
768   \forest@isdim@nonintpart=0
769   \def\forest@isdim@unit{pt}%
770   \let\forest@isnum@next\forest@isdim@checkfordot
771   \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.

```

772   \expandafter\forest@isnum@a\forest@isnum@temp\forest@eon\forest@eon\forest@eon\forest@eon\forest@END
773   \ifforest@isdim
774     \expandafter\@firstoftwo
775   \else
776     \expandafter\@secondoftwo
777   \fi

```



```

778 }
779 \def\forest@isdim@checkfordot{%
780   \ifx.\forest@isnum@token
781     \expandafter\forest@isdim@dot
782   \else
783     \ifx,\forest@isnum@token
784       \expandafter\expandafter\expandafter\forest@isdim@dot
785     \else
786       \expandafter\expandafter\expandafter\forest@isdim@nodot
787     \fi
788   \fi
789 }
790 \def\forest@isdim@nodot{%
791   \ifforest@isnum

```

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.

```

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

```

```

836 \fi
837 \fi
838 \forest@isdim@next
839 }
840 \def\forest@isdim@finish@dim{%
841 \futurelet\forest@isnum@token\forest@isdim@finish@dim@a
842 }
843 \def\forest@isdim@finish@dim@a{%
844 \expandafter\ifx\space\forest@isnum@token
845 \expandafter\forest@isdim@finish@dim@b
846 \else
847 \expandafter\forest@isdim@finish@dim@c
848 \fi
849 }
850 \expandafter\def\expandafter\forest@isdim@finish@dim@b\space{% eat one space
851 \futurelet\forest@isnum@token\forest@isdim@finish@dim@c
852 }
853 \def\forest@isdim@finish@dim@c#1\forest@END{%
854 \ifx\forest@isnum@token\forest@eon
855 \forest@isdim>true
856 \forest@isdim@dimen\the\forest@isnum@count.\forest@isdim@leadingzeros\the\forest@isdim@nonintpart\forest@
857 \else
858 \forest@isdim>false
859 \fi
860 }
861 \def\forest@isdim@finish@nodim#1\forest@END{%
862 \forest@isdim>false
863 }
864 \newdimen\forest@isdim@dimen
865 % \long\def\@firstofthree#1#2#3{#3} % defined by LaTeX
866 \long\def\@firstofthree#1#2#3{#1}
867 \long\def\@secondofthree#1#2#3{#2}
868 \def\forest@isnumdim#1{%
869 \forest@isdim{#1}{%
870 \forest@isnumdim@
871 }{%
872 \@thirdofthree
873 }%
874 }
875 \def\forest@isnumdim@{%
876 \ifforest@isnum
877 \expandafter\@firstofthree
878 \else
879 \expandafter\@secondofthree
880 \fi
881 }

```

4.3 forestmath

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

```

882 \def\forestmathtype@generic{_{}} % generic (token list)
883 \def\forestmathtype@count{n} % integer
884 \def\forestmathtype@dimen{d} % a dimension: <decimal> pt
885 \def\forestmathtype@unitless{P} % <decimal> (a unitless dimension) (P because pgfmath returns such numbers)
886 \def\forestmathtype@textasc{t} % text (ascending)
887 \def\forestmathtype@textdesc{T} % text (descending)
888 \def\forestmathtype@array{a} % array - internal
889 \def\forestmathtype@none{} % internal (for requests - means whatever)
890 \def\forestmathresult{}

```

```
891 \let\forestmathresulttype\forestmathtype@generic
```

\forest@tryprocess takes three “arguments”. The first 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 second argument; if it doesn’t, we execute the third argument.

```
892 \def\forest@tryprocess{%
893   \expandafter\forest@tryprocess@a\romannumeral-‘0}
894 \def\forest@tryprocess@a{\futurelet\forest@temp@token\forest@tryprocess@b}
895 \def\forest@tryprocess@b{%
896   \ifx>\forest@temp@token
897     \expandafter\forest@tryprocess@yes
898   \else
899     \expandafter\forest@tryprocess@no
900   \fi
901 }
902 \def\forest@spacegen{ \forest@spacegen}
903 \def\forest@tryprocess@yes#1#2\forest@spacegen{%
904   \forest@process{false}#2\forest@eov
905   \@firstoftwo
906 }
907 \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.)

```
908 \def\forestmathsetcount#1#2{%
909   \forest@tryprocess#2\forest@spacegen{%
910     #1=\forest@process@result\relax
911   }{%
912     \forestmathsetcount@#1{#2}%
913   }%
914 }
915 \def\forestmathsetcount@#1#2{%
916   \forest@isnum{#2}{%
917     #1=\forest@isnum@count
918   }{%
919     \pgfmathsetcount#1{#2}%
920   }%
921 }
922 \def\forestmathsetlength#1#2{%
923   \forest@tryprocess#2\forest@spacegen{%
924     #1=\forest@process@result\relax
925   }{%
926     \forestmathsetlength@#1{#2}%
927   }%
928 }
929 \def\forestmathsetlength@#1#2{%
930   \forest@isdim{#2}{%
931     #1=\forest@isdim@dimen
932   }{%
933     \pgfmathsetlength#1{#2}%
934   }%
935 }
936 \def\forestmathtruncatemacro#1#2{%
937   \forest@tryprocess#2\forest@spacegen{%
938     \forest@temp@count=\forest@process@result\relax
939     \edef#1{\the\forest@temp@count}%
940   }{%
941     \forestmathtruncatemacro@#1{#2}%
942   }%
943 }
```

```

944 \def\forestmathtruncatemacro#1#2{%
945   \forest@isnum{#2}{%
946     \edef#1{\the\forest@isnum@count}%
947   }{%
948     \pgfmathtruncatemacro#1{#2}%
949   }%
950 }
951 \def\forestmathsetlengthmacro#1#2{%
952   \forest@tryprocess#2\forest@spacegen{%
953     \forest@temp@dimen=\forest@process@result\relax
954     \edef#1{\the\forest@temp@dimen}%
955   }{%
956     \forestmathsetlengthmacro#1{#2}%
957   }%
958 }
959 \def\forestmathsetlengthmacro@#1#2{%
960   \forest@isdim{#2}{%
961     \edef#1{\the\forest@isdim@dimen}%
962   }{%
963     \pgfmathsetlengthmacro#1{#2}%
964   }%
965 }
966 \def\forestmathsetmacro#1#2{%
967   \forest@tryprocess#2\forest@spacegen{%
968     \let#1\forest@process@result
969     \let\forestmathresulttype\forest@process@result@type
970   }{%
971     \forestmathsetmacro@#1{#2}%
972     \def\forestmathresulttype{P}%
973   }%
974 }
975 \def\forestmathsetmacro@#1#2{%
976   \forest@isdim{#2}{%
977     \edef#1{\expandafter\Pgf@geT\the\forest@isdim@dimen}%
978   }{%
979     \pgfmathsetmacro#1{#2}%
980   }%
981 }
982 \def\forestmathparse#1{%
983   \forest@tryprocess#1\forest@spacegen{%
984     \let\forestmathresult\forest@process@result
985     \let\forestmathresulttype\forest@process@result@type
986   }{%
987     \forestmathparse@{#1}%
988     \def\forestmathresulttype{P}%
989   }%
990 }
991 \def\forestmathparse@#1{%
992   \forest@isdim{#1}{%
993     \edef\forestmathresult{\expandafter\Pgf@geT\the\forest@isdim@dimen}%
994   }{%
995     \pgfmathsetmacro\forestmathresult{#1}%
996   }%
997 }

```

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

```

998 \csdef{forest@bh@0}{0}
999 \csdef{forest@bh@false}{0}
1000 \csdef{forest@bh@1}{1}
1001 \csdef{forest@bh@true}{1}

```

```

1002 \def\forestmath@if#1{%
1003   \ifcsdef{forest@bh@\detokenize{#1}}{%
1004     \let\forest@next\forestmath@if@fast
1005   }{%
1006     \let\forest@next\forestmath@if@slow
1007   }%
1008   \forest@next{#1}%
1009   \ifnum\forest@temp=0
1010     \expandafter\@secondoftwo
1011   \else
1012     \expandafter\@firstoftwo
1013   \fi
1014 }
1015 \def\forestmath@if@fast#1{\letcs\forest@temp{forest@bh@\detokenize{#1}}}
1016 \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).

```

1017 \def\forestmath@convert@fromto#1#2#3{%
1018   \edef\forestmathresult{\csname forestmath@convert@from@#1@to@#2\endcsname{#3}}
1019 \def\forestmath@convert@from#1{\forestmath@convert@fromto{#1}{\forestmathresulttype}}
1020 \def\forestmath@convert@to{\forestmath@convert@fromto{\forestmathresulttype}}
1021 \def\forestmath@convert@from@n@to@n#1{#1}
1022 \def\forestmath@convert@from@n@to@d#1{#1\pgfm@th@pt}
1023 \def\forestmath@convert@from@n@to@P#1{#1}
1024 \def\forestmath@convert@from@d@to@n#1{%
1025   \expandafter\forestmath@convert@uptodot\pgf@geT#1.\forest@eov}
1026 \def\forestmath@convert@from@d@to@d#1{#1}
1027 \def\forestmath@convert@from@d@to@P#1{\pgf@geT#1}
1028 \def\forestmath@convert@from@P@to@n#1{%
1029   \forestmath@convert@uptodot#1.\forest@eov}
1030 \def\forestmath@convert@from@P@to@d#1{#1\pgfm@th@pt}
1031 \def\forestmath@convert@from@P@to@P#1{#1}
1032 \def\forestmath@convert@uptodot#1.#2\forest@eov{#1}
1033 \def\forestmathzero{\forestmath@convert@from\forestmath@type@count{0}}

```

These defer conversion (see aggregates).

```

1034 \csdef{forestmath@convert@from@n@to@_}#1{\unexpanded{#1}}
1035 \csdef{forestmath@convert@from@d@to@_}#1{\unexpanded{#1}}
1036 \csdef{forestmath@convert@from@P@to@_}#1{\unexpanded{#1}}

```

Sets `\pgfm@thresulttype` to the type of #1.

```

1037 \def\forestmathsettypefrom#1{%
1038   \forest@isnumdim{%
1039     \let\forestmathresulttype\forest@result@type@numexpr
1040   }{%
1041     \let\forestmathresulttype\forest@result@type@dimexpr
1042   }{%
1043     \let\forestmathresulttype\forest@result@type@pgfm@th
1044   }%
1045 }

```

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.

```

1046 \def\forestmathadd#1#2{\edef\forestmathresult{%
1047   \csname forestmathadd\forestmathresulttype\endcsname{#1}{#2}}}
1048 \def\forestmathadd@#1#2#3{\edef\forestmathresult{%
1049   \csname forestmathadd@#1\endcsname{#2}{#3}}}
1050 \def\forestmathadd@n#1#2{\the\numexpr#1+#2\relax}
1051 \def\forestmathadd@d#1#2{\the\dimexpr#1+#2\relax}
1052 \def\forestmathadd@P#1#2{\expandafter\pgf@geT\the\dimexpr#1pt+#2pt\relax}

```

```

1053 \def\forestmathmultiply#1#2{%
1054   \csname forestmathmultiply@\forestmathresulttype\endcsname{#1}{#2}}
1055 \def\forestmathmultiply@#1#2#3{%
1056   \csname forestmathmultiply@#1\endcsname{#2}{#3}}
1057 \def\forestmathmultiply@n#1#2{\edef\forestmathresult{%
1058   \the\numexpr#1*#2\relax}}
1059 \def\forestmathmultiply@d#1#2{%
1060   \edef\forestmath@marshal{\forestmathmultiply@d@{#1}{#2}}\forestmath@marshal
1061 }
1062 \def\forestmathmultiply@d#1#2{%
1063   \edef\forestmath@marshal{%
1064     \noexpand\pgfmultiply@{\Pgfg@geT#1}{\Pgfg@geT#2}%
1065   }\forestmath@marshal
1066   \edef\forestmathresult{\pgfmultiplyresult\pgfmultiply@pt}%
1067 }
1068 \def\forestmathmultiply@P#1#2{%
1069   \pgfmultiply@{#1}{#2}%
1070   \let\forestmathresult\pgfmultiplyresult
1071 }

```

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.

```

1072 \def\forestmathdivide#1#2{%
1073   \csname forestmathdivide@\forestmathresulttype\endcsname{#1}{#2}}
1074 \def\forestmathdivide@#1#2#3{%
1075   \csname forestmathdivide@#1\endcsname{#2}{#3}}
1076 \def\forestmathdivide@n#1#2{\edef\forestmathresult{%
1077   \the\numexpr#1/#2\relax}}
1078 \def\forestmathdivide@d#1#2{\edef\forestmathresult{%
1079   \the\dimexpr#1/#2\relax}}
1080 \def\forestmathdivide@P#1#2{%
1081   \edef\forest@marshal{%
1082     \noexpand\pgfmathdivide{+ #1}{+ #2}%
1083   }\forest@marshal
1084   \let\forestmathresult\pgfmathresult
1085 }

```

Booleans.

```

1086 \def\forestmathtrue{%
1087   \def\forestmathresult{1}%
1088   \let\forestmathresulttype\forestmathtype@count}
1089 \def\forestmathfalse{%
1090   \def\forestmathresult{0}%
1091   \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 `<`.

```

1092 \def\forestmath@if@lt@n#1#2{\ifnum#1<#2\relax
1093   \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1094 \def\forestmath@if@lt@d#1#2{\ifdim#1<#2\relax
1095   \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1096 \def\forestmath@if@lt@P#1#2{\ifdim#1pt<#2pt
1097   \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1098 \def\forestmath@if@lt@t#1#2{\ifnum\pdfstrcmp{#1}{#2}<0
1099   \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1100 \def\forestmath@if@lt@T#1#2{\ifnum\pdfstrcmp{#1}{#2}>0
1101   \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1102 \def\forest@cmp@error#1#2{\PackageError{forest}{Comparison
1103   ("<" or ">") of generic type arguments "#1" and "#2"
1104   makes no sense}{Use one of argument processor instructions

```

1105 "n", "d", "P" or "t" to change the type. Use package option
 1106 "debug=process" to see what's happening here.}}
 1107 \cslet{forestmath@if@lt@_}\forest@cmp@error

Low level =.

```
1108 \def\forestmath@if@eq@n#1#2{\ifnum#1=#2\relax
1109 \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1110 \def\forestmath@if@eq@d#1#2{\ifdim#1=#2\relax
1111 \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1112 \def\forestmath@if@eq@P#1#2{\ifdim#1pt=#2pt
1113 \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1114 \def\forestmath@if@eq@t#1#2{\ifnum\pdfstrcmp{#1}{#2}=0
1115 \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1116 \let\forestmath@if@eq@T\forestmath@if@eq@t
1117 \csdef{forestmath@if@eq@_}#1#2{%
1118 \def\forestmath@tempa{#1}%
1119 \def\forestmath@tempb{#2}%
1120 \ifx\forestmath@tempa\forestmath@tempb
1121 \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
```

High level <, > and =.

```
1122 \def\forestmathlt#1#2{%
1123 \csname forestmath@if@lt@\forestmathresulttype\endcsname{#1}{#2}%
1124 \forestmathtrue
1125 \forestmathfalse}
1126 \def\forestmathlt@#1#2#3{%
1127 \csname forestmath@if@lt@#1\endcsname{#2}{#3}%
1128 \forestmathtrue
1129 \forestmathfalse}
1130 \def\forestmathgt#1#2{%
1131 \csname forestmath@if@lt@\forestmathresulttype\endcsname{#2}{#1}%
1132 \forestmathtrue
1133 \forestmathfalse}
1134 \def\forestmathgt@#1#2#3{%
1135 \csname forestmath@if@lt@#1\endcsname{#3}{#2}%
1136 \forestmathtrue
1137 \forestmathfalse}
1138 \def\forestmatheq#1#2{%
1139 \csname forestmath@if@eq@\forestmathresulttype\endcsname{#1}{#2}%
1140 \forestmathtrue
1141 \forestmathfalse}
1142 \def\forestmatheq@#1#2#3{%
1143 \csname forestmath@if@eq@#1\endcsname{#2}{#3}%
1144 \forestmathtrue
1145 \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.

```
1146 \def\forestmathmin{\forestmath@minmax{min}{\forestmathresulttype}}
1147 \def\forestmathmax{\forestmath@minmax{max}{\forestmathresulttype}}
1148 \def\forestmathmin@{\forestmath@minmax{min}}
1149 \def\forestmathmax@{\forestmath@minmax{max}}
1150 \def\forestmath@minmax#1#2#3#4{% #1=min/max, #2=type, #3,#4=args
1151 \edef\forestmath@tempa{#3}%
1152 \edef\forestmath@tempb{#4}%
1153 \if\relax\detokenize\expandafter{\forestmath@tempa}\relax
1154 \forestmath@minmax@one{#1}{#2}\forestmath@tempb
1155 \else
1156 \if\relax\detokenize\expandafter{\forestmath@tempb}\relax
1157 \forestmath@minmax@one{#1}{#2}\forestmath@tempa
1158 \else
```

```

1159     \csname forestmath@#1\endcsname{#2}%
1160     \fi
1161     \fi
1162 }
1163 \def\forestmath@minmax@one#1#2#3{% #1=min/max, #2=type, #3 = the (possibly) non-empty arg
1164     \ifcsname forestmath@#1@one@#2\endcsname
1165         \csname forestmath@#1@one@#2\endcsname#3%
1166     \else
1167         \let\forestmathresult#3%
1168     \fi
1169 }
1170 \def\forestmath@min@one@t#1{\let\forestmathresult\forest@empty}
1171 \def\forestmath@max@one@t#1{\let\forestmathresult#1}
1172 \def\forestmath@min@one@T#1{\let\forestmathresult#1}
1173 \def\forestmath@max@one@T#1{\let\forestmathresult\forest@empty}
1174
1175 \def\forestmath@min#1{% #1 = type
1176     \csname forestmath@if@lt@#1\endcsname\forestmath@tempa\forestmath@tempb
1177     {\let\forestmathresult\forestmath@tempa}%
1178     {\let\forestmathresult\forestmath@tempb}%
1179 }
1180 \def\forestmath@max#1{% #1 = type
1181     \csname forestmath@if@lt@#1\endcsname\forestmath@tempa\forestmath@tempb
1182     {\let\forestmathresult\forestmath@tempb}%
1183     {\let\forestmathresult\forestmath@tempa}%
1184 }

```

4.4 Sorting

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

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

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

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

```

1185 \def\forest@sort#1#2#3#4#5{%
1186     \let\forest@sort@cmp#1\relax
1187     \let\forest@sort@let#2\relax
1188     \let\forest@sort@direction#3\relax
1189     \forest@@sort{#4}{#5}%
1190 }

```

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.

```

1191 \def\forest@quicksort@minarraylength{10000}
1192 \def\forest@@sort#1#2{%
1193     \ifnum#1<#2\relax\escapeif{%
1194         \forest@sort@m=#2
1195         \advance\forest@sort@m -#1

```

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`.


```

1196 \ifnum\forest@sort@m>\forest@quicksort@minarraylength\relax\@escapeif{%
1197 \forest@quicksort{#1}{#2}%
1198 }\else\@escapeif{%
1199 \forest@insertionsort{#1}{#2}%
1200 }\fi
1201 }\fi
1202 }

```

Various counters and macros needed by the sorting algorithms.

```

1203 \newcount\forest@sort@m\newcount\forest@sort@k\newcount\forest@sort@p
1204 \def\forest@sort@ascending{>}
1205 \def\forest@sort@descending{<}
1206 \def\forest@sort@cmp{%
1207 \PackageError{sort}{You must define forest@sort@cmp function before calling
1208 sort}{The macro must take two arguments, indices of the array
1209 elements to be compared, and return '=' if the elements are equal
1210 and '>'/'<' if the first is greater /less than the second element.}%
1211 }
1212 \def\forest@sort@cmp@gt{\def\forest@sort@cmp@result{>}}
1213 \def\forest@sort@cmp@lt{\def\forest@sort@cmp@result{<}}
1214 \def\forest@sort@cmp@eq{\def\forest@sort@cmp@result{=}}
1215 \def\forest@sort@let{%
1216 \PackageError{sort}{You must define forest@sort@let function before calling
1217 sort}{The macro must take two arguments, indices of the array:
1218 element 2 must be copied onto element 1.}%
1219 }

```

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

```

1220 \newloop\forest@sort@loop
1221 \newloop\forest@sort@loopA
1222 \def\forest@quicksort#1#2{%

```

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

```

1223 \forest@sort@m=#2
1224 \advance\forest@sort@m -#1
1225 \ifodd\forest@sort@m\relax\advance\forest@sort@m1 \fi
1226 \divide\forest@sort@m 2
1227 \advance\forest@sort@m #1

```

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

```

1228 \forest@sort@cmp{#1}{#2}%
1229 \if\forest@sort@cmp@result=%
1230 \forest@sort@p=#1
1231 \else
1232 \if\forest@sort@cmp@result>%
1233 \forest@sort@p=#1\relax
1234 \else
1235 \forest@sort@p=#2\relax
1236 \fi
1237 \forest@sort@cmp{\the\forest@sort@p}{\the\forest@sort@m}%
1238 \if\forest@sort@cmp@result<%
1239 \else
1240 \forest@sort@p=\the\forest@sort@m
1241 \fi
1242 \fi

```

Exchange the pivot and the first element.

```

1243 \forest@sort@xch{#1}{\the\forest@sort@p}%

```

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

```

1244 \forest@sort@m=#1\relax

```

Loop through the list.

```

1245 \forest@sort@k=#1\relax
1246 \forest@sort@loop
1247 \ifnum\forest@sort@k<#2\relax
1248 \advance\forest@sort@k 1

Compare the pivot and the current element.
1249 \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.
1250 \ifx\forest@sort@direction\forest@sort@cmp@result
1251 \advance\forest@sort@m 1
1252 \forest@sort@xch{\the\forest@sort@m}{\the\forest@sort@k}
1253 \fi
1254 \forest@sort@repeat

Move the pivot element into its final position.
1255 \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.
1256 \forest@sort@k=\forest@sort@m
1257 \advance\forest@sort@k -1
1258 \advance\forest@sort@m 1
1259 \edef\forest@sort@marshal{%
1260 \noexpand\forest@sort@{#1}{\the\forest@sort@k}%
1261 \noexpand\forest@sort@{\the\forest@sort@m}{#2}%
1262 }%
1263 \forest@sort@marshal
1264 }
1265 % We defines the item-exchange macro in terms of the (user-provided)
1266 % array let macro.
1267 % \begin{macrocode}
1268 \def\forest@sort@aux{aux}
1269 \def\forest@sort@xch#1#2{%
1270 \forest@sort@let{\forest@sort@aux}{#1}%
1271 \forest@sort@let{#1}{#2}%
1272 \forest@sort@let{#2}{\forest@sort@aux}%
1273 }

Insertion sort.
1274 \def\forest@insertionsort#1#2{%
1275 \forest@sort@m=#1
1276 \edef\forest@insertionsort@low{#1}%
1277 \forest@sort@loopA
1278 \ifnum\forest@sort@m<#2
1279 \advance\forest@sort@m 1
1280 \forest@insertionsort@Qbody
1281 \forest@sort@repeatA
1282 }
1283 \newif\ifforest@insertionsort@loop
1284 \def\forest@insertionsort@Qbody{%
1285 \forest@sort@let{\forest@sort@aux}{\the\forest@sort@m}%
1286 \forest@sort@k\forest@sort@m
1287 \advance\forest@sort@k -1
1288 \forest@insertionsort@looptrue
1289 \forest@sort@loop
1290 \ifforest@insertionsort@loop
1291 \forest@insertionsort@Qbody
1292 \forest@sort@repeat
1293 \advance\forest@sort@k 1

```

```

1294 \forest@sort@let{\the\forest@sort@k}{\forest@sort@aux}%
1295 }
1296 \def\forest@insertionsort@qbody{%
1297 \forest@sort@cmp{\the\forest@sort@k}{\forest@sort@aux}%
1298 \ifx\forest@sort@direction\forest@sort@cmp@result\relax
1299 \forest@sort@p=\forest@sort@k
1300 \advance\forest@sort@p 1
1301 \forest@sort@let{\the\forest@sort@p}{\the\forest@sort@k}%
1302 \advance\forest@sort@k -1
1303 \ifnum\forest@sort@k<\forest@insertionsort@low\relax
1304 \forest@insertionsort@loopfalse
1305 \fi
1306 \else
1307 \forest@insertionsort@loopfalse
1308 \fi
1309 }

```

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

Compare numbers.

```

1310 \def\forest@sort@cmpnumcs#1#2{%
1311 \ifnum\csname#1\endcsname>\csname#2\endcsname\relax
1312 \forest@sort@cmp@gt
1313 \else
1314 \ifnum\csname#1\endcsname<\csname#2\endcsname\relax
1315 \forest@sort@cmp@lt
1316 \else
1317 \forest@sort@cmp@eq
1318 \fi
1319 \fi
1320 }

```

Compare dimensions.

```

1321 \def\forest@sort@cmpdimcs#1#2{%
1322 \ifdim\csname#1\endcsname>\csname#2\endcsname\relax
1323 \forest@sort@cmp@gt
1324 \else
1325 \ifdim\csname#1\endcsname<\csname#2\endcsname\relax
1326 \forest@sort@cmp@lt
1327 \else
1328 \forest@sort@cmp@eq
1329 \fi
1330 \fi
1331 }

```

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

```

1332 \def\forest@sort@cmptwodimcs#1#2#3#4{%
1333 \ifdim\csname#1\endcsname>\csname#3\endcsname\relax
1334 \forest@sort@cmp@gt
1335 \else
1336 \ifdim\csname#1\endcsname<\csname#3\endcsname\relax
1337 \forest@sort@cmp@lt
1338 \else
1339 \ifdim\csname#2\endcsname>\csname#4\endcsname\relax
1340 \forest@sort@cmp@gt
1341 \else
1342 \ifdim\csname#2\endcsname<\csname#4\endcsname\relax
1343 \forest@sort@cmp@lt
1344 \else
1345 \forest@sort@cmp@eq
1346 \fi
1347 \fi

```

```

1348 \fi
1349 \fi
1350 }

```

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).

```

1351 \def\forest@reversearray#1#2#3{%
1352 \let\forest@sort@let#1%
1353 \c@pgf@countc=#2
1354 \c@pgf@countd=#3
1355 \advance\c@pgf@countd -1
1356 \safeloop
1357 \ifnum\c@pgf@countc<\c@pgf@countd\relax
1358 \forest@sort@exch{\the\c@pgf@countc}{\the\c@pgf@countd}%
1359 \advance\c@pgf@countc 1
1360 \advance\c@pgf@countd -1
1361 \saferepeat
1362 }

```

5 The bracket representation parser

5.1 The user interface macros

Settings.

```

1363 \def\bracketset#1{\pgfqkeys{/bracket}{#1}}%
1364 \bracketset{%
1365 /bracket/.is family,
1366 /handlers/.let/.style={\pgfkeyscurrentpath/.code={\let#1##1}},
1367 opening bracket/.let=\bracket@openingBracket,
1368 closing bracket/.let=\bracket@closingBracket,
1369 action character/.let=\bracket@actionCharacter,
1370 opening bracket=[,
1371 closing bracket=],
1372 action character,
1373 new node/.code n args={3}{% #1=preamble, #2=node spec, #3=cs receiving the id
1374 \forest@node@new#3%
1375 \forest@set{#3}{given options}{\forest@contentto=\unexpanded{#2}}%
1376 \ifblank{#1}{}%
1377 \forest@rset{preamble}{#1}%
1378 }%
1379 },
1380 set afterthought/.code 2 args={% #1=node id, #2=afterthought
1381 \ifblank{#2}{\forest@appto{#1}{given options}{,afterthought={#2}}}%
1382 }
1383 }

```

`\bracketParse` is the macro that should be called to parse a balanced bracket representation. It takes five 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.)

```

1384 \newtoks\bracket@content
1385 \newtoks\bracket@afterthought
1386 \def\bracketParse#1#2={%
1387 \def\bracketEndParsingHook{#1}%
1388 \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.)

```

1389 \bracket@content={}%
1390 \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.

```
1391 \let\bracket@state\bracket@state@starting
1392 \bracket@ignorepacestrue
```

By default, don't expand anything.

```
1393 \bracket@expandtokensfalse
```

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

```
1394 \def\bracket@parentNode{0}%
1395 \def\bracket@rootNode{0}%
1396 \def\bracket@newNode{0}%
1397 \def\bracket@afterthoughtNode{0}%
```

Finally, we start the parser.

```
1398 \bracket@Parse
1399 }
```

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.

```
1400 \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.

```
1401 \def\bracket@Parse{%
1402   \futurelet\bracket@next@token\bracket@Parse@checkForSpace
1403 }
1404 \def\bracket@Parse@checkForSpace{%
1405   \expandafter\ifx\space\bracket@next@token\@escapeif{%
1406     \ifbracket@ignorepaces\else
1407       \bracket@haveSpacetrue
1408     \fi
1409     \expandafter\bracket@Parse\romannumeral-‘0%
1410   }\else\@escapeif{%
1411     \bracket@Parse@maybeexpand
1412   }\fi
1413 }
```

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`.

```
1414 \newif\ifbracket@expandtokens
1415 \def\bracket@Parse@maybeexpand{%
1416   \ifbracket@expandtokens\@escapeif{%
1417     \expandafter\bracket@Parse@peekAhead\romannumeral-‘0%
1418   }\else\@escapeif{%
1419     \bracket@Parse@peekAhead
1420   }\fi
1421 }
```

We then look ahead to see what's coming.

```
1422 \def\bracket@Parse@peekAhead{%
1423   \futurelet\bracket@next@token\bracket@Parse@checkForTeXGroup
1424 }
```

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

```
1425 \def\bracket@Parse@checkForTeXGroup{%
1426   \ifx\bracket@next@token\bgroup%
1427     \@escapeif{\bracket@Parse@appendGroup}%
```

```

1428 \else
1429 \escapeif{\bracket@Parse@token}%
1430 \fi
1431 }

```

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.

```

1432 \long\def\bracket@Parse@token#1{%
1433 \ifx#1\bracket@openingBracket
1434 \escapeif{\bracket@Parse@openingBracketFound}%
1435 \else
1436 \escapeif{%
1437 \ifx#1\bracket@closingBracket
1438 \escapeif{\bracket@Parse@closingBracketFound}%
1439 \else
1440 \escapeif{%
1441 \ifx#1\bracket@actionCharacter
1442 \escapeif{\futurelet\bracket@next@token\bracket@Parse@actionCharacterFound}%
1443 \else
1444 \escapeif{\bracket@Parse@appendToken#1}%
1445 \fi
1446 }%
1447 \fi
1448 }%
1449 \fi
1450 }

```

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

```

1451 \newif\ifbracket@haveSpace
1452 \newif\ifbracket@ignorespaces
1453 \def\bracket@Parse@appendSpace{%
1454 \ifbracket@haveSpace
1455 \ifcase\bracket@state\relax
1456 \eapptotoks\bracket@content\space
1457 \or
1458 \eapptotoks\bracket@afterthought\space
1459 \or
1460 \eapptotoks\bracket@afterthought\space
1461 \fi
1462 \bracket@haveSpacefalse
1463 \fi
1464 }
1465 \long\def\bracket@Parse@appendToken#1{%
1466 \bracket@Parse@appendSpace
1467 \ifcase\bracket@state\relax
1468 \lapptotoks\bracket@content{#1}%
1469 \or
1470 \lapptotoks\bracket@afterthought{#1}%
1471 \or
1472 \lapptotoks\bracket@afterthought{#1}%
1473 \fi
1474 \bracket@ignorespacesfalse
1475 \bracket@Parse
1476 }
1477 \def\bracket@Parse@appendGroup#1{%
1478 \bracket@Parse@appendSpace
1479 \ifcase\bracket@state\relax
1480 \apptotoks\bracket@content{#1}%
1481 \or
1482 \apptotoks\bracket@afterthought{#1}%

```

```

1483 \or
1484 \apptotoks\bracket@afterthought{{#1}}%
1485 \fi
1486 \bracket@ignorespacesfalse
1487 \bracket@Parse
1488 }

```

Declare states.

```

1489 \def\bracket@state@inContent{0}
1490 \def\bracket@state@inAfterthought{1}
1491 \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.)

```

1492 \def\bracket@Parse@openingBracketFound{%
1493 \bracket@haveSpacefalse
1494 \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.

```

1495 \escapeif{%
1496 \bracket@createNode
1497 \ifnum\bracket@parentNode=0 \else
1498 \forest@node@Append{\bracket@parentNode}{\bracket@newNode}%
1499 \fi
1500 \let\bracket@parentNode\bracket@newNode
1501 \bracket@Parse
1502 }%
1503 \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.

```

1504 \escapeif{%
1505 \bracket@addAfterthought
1506 \let\bracket@state\bracket@state@inContent
1507 \bracket@Parse
1508 }%
1509 \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.

```

1510 \escapeif{%
1511 \let\bracket@state\bracket@state@inContent
1512 \bracket@Parse
1513 }%
1514 \fi
1515 }
1516 \def\bracket@Parse@closingBracketFound{%
1517 \bracket@haveSpacefalse
1518 \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.

```

1519 \escapeif{%
1520 \bracket@createNode

```

```

1521 \ifnum\bracket@parentNode=0
1522 \escapeif\bracketEndParsingHook
1523 \else
1524 \escapeif{%
1525 \let\bracket@afterthoughtNode\bracket@newNode
1526 \let\bracket@state\bracket@state@inAfterthought
1527 \forest@node@Append{\bracket@parentNode}{\bracket@newNode}%
1528 \bracket@Parse
1529 }%
1530 \fi
1531 }%
1532 \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.

```

1533 \escapeif{%
1534 \bracket@addAfterthought
1535 \let\bracket@afterthoughtNode\bracket@parentNode
1536 \edef\bracket@parentNode{\forestOve{\bracket@parentNode}{@parent}}%
1537 \ifnum\bracket@parentNode=0
1538 \expandafter\bracketEndParsingHook
1539 \else
1540 \expandafter\bracket@Parse
1541 \fi
1542 }%
1543 \else % starting

```

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

```

1544 \PackageError{forest}{You're attempting to start a bracket representation
1545 with a closing bracket}{}%
1546 \fi
1547 }

```

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

```

1548 \def\bracket@Parse@actionCharacterFound{%
  If a braced expression follows, its contents will be fully expanded.
1549 \ifx\bracket@next@token\bgroup\escapeif{%
1550 \bracket@Parse@action@expandgroup
1551 }\else\escapeif{%
1552 \bracket@Parse@action@notagroup
1553 }\fi
1554 }
1555 \def\bracket@Parse@action@expandgroup#1{%
1556 \edef\bracket@Parse@action@expandgroup@macro{#1}%
1557 \expandafter\bracket@Parse\bracket@Parse@action@expandgroup@macro
1558 }
1559 \let\bracket@action@fullyexpandCharacter+
1560 \let\bracket@action@dontexpandCharacter-
1561 \let\bracket@action@executeCharacter!
1562 \def\bracket@Parse@action@notagroup#1{%

```

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

```

1563 \ifx#1\bracket@action@fullyexpandCharacter\escapeif{%
1564 \bracket@expandtokenstrue\bracket@Parse
1565 }\else\escapeif{%

```

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

```

1566 \ifx#1\bracket@action@dontexpandCharacter\escapeif{%
1567 \bracket@expandtokensfalse\bracket@Parse
1568 }\else\escapeif{%

```


Inhibit expansion of the next token.

```
1569 \ifx#10\@escapeif{%
1570 \bracket@Parse@appendToken
1571 }\else\@escapeif{%
```

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

```
1572 \ifx#1\bracket@actionCharacter
1573 \else\@escapeif{%
```

Anything else will be expanded once.

```
1574 \expandafter\bracket@Parse#1%
1575 }\fi
1576 }\fi
1577 }\fi
1578 }\fi
1579 }
```

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.

```
1580 \def\bracket@createNode{%
1581 \ifnum\bracket@rootNode=0
1582 % root node
1583 \bracketset{new node/.expanded=%
1584 {\the\bracket@afterthought}}%
1585 {\the\bracket@content}}%
1586 \noexpand\bracket@newNode
1587 }%
1588 \bracket@afterthought={}%
1589 \let\bracket@rootNode\bracket@newNode
1590 \expandafter\let\bracket@saveRootNodeTo\bracket@newNode
1591 \else
1592 % other nodes
1593 \bracketset{new node/.expanded=%
1594 {}%
1595 {\the\bracket@content}}%
1596 \noexpand\bracket@newNode
1597 }%
1598 \fi
1599 \bracket@content={}%
1600 }
```

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

```
1601 \def\bracket@addAfterthought{%
1602 \bracketset{%
1603 set afterthought/.expanded={\bracket@afterthoughtNode}{\the\bracket@afterthought}}%
1604 }%
1605 \bracket@afterthought={}%
1606 }
```

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.

```

1607 % full expansion expands precisely to the value
1608 \def\forestov#1{\expandafter\expandonce\csname fRST\forest@cn/#1\endcsname}
1609 % full expansion expands all the way
1610 \def\forestove#1{\csname fRST\forest@cn/#1\endcsname}
1611 % full expansion expands to the cs holding the value
1612 \def\forestom#1{\expandonce{\csname fRST\forest@cn/#1\endcsname}}
1613 \def\forestoget#1#2{\expandafter\let\expandafter#2\csname fRST\forest@cn/#1\endcsname}
1614 \def\forestolet#1#2{\expandafter\let\csname fRST\forest@cn/#1\endcsname#2}
1615 % \def\forestocslet#1#2{%
1616 %   \edef\forest@marshal{%
1617 %     \noexpand\pgfkeyslet{/forest/@node/\forest@cn/#1}{\expandonce{\csname#2\endcsname}}%
1618 %   }\forest@marshal
1619 % }
1620 \def\forestoset#1#2{\expandafter\edef\csname fRST\forest@cn/#1\endcsname{\unexpanded{#2}}}
1621 \def\forestoeset#1#2
1622   {\expandafter\edef\csname fRST\forest@cn/#1\endcsname
1623     % {#2}
1624   }
1625 \def\forestooppto#1#2{%
1626   \forestoeset{#1}{\forestov{#1}\unexpanded{#2}}%
1627 }
1628 \def\forestoiifdefined#1#2#3
1629 {%
1630   \ifcsdef{fRST\forest@cn/#1}%{#2}{#3}%
1631 }

```

User macros for retrieving node options of the current node.

```

1632 \let\forestoption\forestov
1633 \let\foresteooption\forestove

```

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

```

1634 \def\forestOv#1#2{\expandafter\expandonce\csname fRST#1/#2\endcsname}
1635 \def\forestOve#1#2{\csname fRST#1/#2\endcsname}
1636 % full expansion expands to the cs holding the value
1637 \def\forestOm#1#2{\expandonce{\csname fRST#1/#2\endcsname}}
1638 \def\forestOget#1#2#3{\expandafter\let\expandafter#3\csname fRST#1/#2\endcsname}
1639 \def\forestOlet#1#2#3{\expandafter\let\csname fRST#1/#2\endcsname#3}
1640 % \def\forestOcslet#1#2#3{%
1641 %   \edef\forest@marshal{%
1642 %     \noexpand\pgfkeyslet{/forest/@node/#1/#2}{\expandonce{\csname#3\endcsname}}%
1643 %   }\forest@marshal
1644 % }
1645 \def\forestOset#1#2#3{\expandafter\edef\csname fRST#1/#2\endcsname{\unexpanded{#3}}}
1646 \def\forestOeset#1#2#3
1647   {\expandafter\edef\csname fRST#1/#2\endcsname
1648     % {#3}
1649   }
1650 \def\forestOappto#1#2#3{%
1651   \forestOeset{#1}{#2}{\forestOv{#1}{#2}\unexpanded{#3}}%
1652 }
1653 \def\forestOeappto#1#2#3{%
1654   \forestOeset{#1}{#2}{\forestOve{#1}{#2}#3}%
1655 }
1656 \def\forestOpreto#1#2#3{%
1657   \forestOeset{#1}{#2}{\unexpanded{#3}\forestOv{#1}{#2}}%
1658 }
1659 \def\forestOepreto#1#2#3{%
1660   \forestOeset{#1}{#2}{#3\forestOv{#1}{#2}}%
1661 }

```

```

1662 \def\forestOifdefined#1#2#3#4
1663 {%
1664   \ifcsdef{fRsT#1/#2}%{#3}{#4}%
1665 }
1666 \def\forestOlet0#1#2#3#4{% option #2 of node #1 <-- option #4 of node #3
1667   \forestOget{#3}{#4}\forestoption@temp
1668   \forestOlet{#1}{#2}\forestoption@temp}
1669 \def\forestOleto#1#2#3{%
1670   \forestOget{#3}\forestoption@temp
1671   \forestOlet{#1}{#2}\forestoption@temp}
1672 \def\forestOlet0#1#2#3{%
1673   \forestOget{#2}{#3}\forestoption@temp
1674   \forestOlet{#1}\forestoption@temp}
1675 \def\forestOleto#1#2{%
1676   \forestOget{#2}\forestoption@temp
1677   \forestOlet{#1}\forestoption@temp}

```

Macros for retrieving node options given by *<relative node name>.<option>*.

```

1678 \def\forestRNOget#1#2{% #1=rn!option, #2 = receiving cs
1679   \pgfutil@in@{.}{#1}%
1680   \ifpgfutil@in@
1681     \forestRNOget@rn#2#1\forest@END
1682   \else
1683     \forestOget{#1}#2%
1684   \fi
1685 }
1686 \def\forestRNOget@rn#1#2.#3\forest@END{%
1687   \forest@forthis{%
1688     \forest@nameandgo{#2}%
1689     \forestOget{#3}#1%
1690   }%
1691 }
1692 \def\forestRNOgetvalueandtype#1#2#3{% #1=rn.option, #2,#3 = receiving css
1693   \pgfutil@in@{.}{#1}%
1694   \ifpgfutil@in@
1695     \forestRNOgetvalueandtype@rn#2#3#1\forest@END
1696   \else
1697     \forestOget{#1}#2%
1698     \pgfkeysgetvalue{/forest/#1/@type}#3%
1699   \fi
1700 }
1701 \def\forestRNOgetvalueandtype@rn#1#2#3.#4\forest@END{%
1702   % #1,#2=receiving css, #3=relative node name, #4=option name
1703   \forest@forthis{%
1704     \forest@nameandgo{#3}%
1705     \forestOget{#4}#1%
1706   }%
1707   \pgfkeysgetvalue{/forest/#4/@type}#2%
1708 }

```

Macros for retrieving/setting registers.

```

1709 % full expansion expands precisely to the value
1710 \def\forestrv#1{\expandafter\expandonce\csname fRsT/#1\endcsname}
1711 % full expansion expands all the way
1712 \def\forestrv#1{\csname fRsT/#1\endcsname}
1713 % full expansion expands to the cs holding the value
1714 \def\forestrm#1{\expandonce{\csname fRsT/#1\endcsname}}
1715 \def\forestrget#1#2{\expandafter\let\expandafter#2\csname fRsT/#1\endcsname}
1716 \def\forestrlet#1#2{\expandafter\let\csname fRsT/#1\endcsname#2}
1717 % \def\forestrcslet#1#2{%
1718 %   \edef\forest@marshal{%
1719 %     \noexpand\pgfkeyslet{/forest/@node/register/#1}{\expandonce{\csname#2\endcsname}}%

```

```

1720 %   }\forest@marshal
1721 % }
1722 \def\forestreset#1#2{\expandafter\edef\csname fRsT/#1\endcsname{\unexpanded{#2}}}
1723 \def\forestreset#1#2
1724   {\expandafter\edef\csname fRsT/#1\endcsname}{%#2}
1725 \def\forestrappto#1#2{%
1726   \forestreset{#1}{\forestrv{#1}\unexpanded{#2}}%
1727 }
1728 \def\forestrpreto#1#2{%
1729   \forestreset{#1}{\unexpanded{#2}\forestrv{#1}}%
1730 }
1731 \def\forestrifdefined#1#2#3
1732 {%
1733   \ifcsdef{fRsT/#1}{%#2}{#3}%
1734 }

```

User macros for retrieving node options of the current node.

```

1735 \def\forestregister#1{\forestrv{#1}}
1736 \def\foresteregister#1{\forestrv{#1}}

```

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

```

1737 \def\forest@node@init{%
1738   \forestoset{@parent}{0}%
1739   \forestoset{@previous}{0}% previous sibling
1740   \forestoset{@next}{0}%      next sibling
1741   \forestoset{@first}{0}%     primary child
1742   \forestoset{@last}{0}%      last child
1743 }
1744 \def\forest@node@init#1{%
1745   \pgfkeysgetvalue{/forest/#1}\forest@node@temp
1746   \forest@node@init#1\forest@node@temp
1747 }
1748 \newcount\forest@node@maxid
1749 \def\forest@node@new#1{% #1 = cs receiving the new node id
1750   \advance\forest@node@maxid1
1751   \forest@node@init#1\forest@node@maxid}%
1752   \forest@node@init
1753   \forest@node@setname{id}\forest@node@cn}%
1754   \forest@node@setname{node@\forest@node@cn}%
1755   \forest@node@initializefromstandardnode
1756   \edef#1{\forest@node@cn}%
1757 }%
1758 }
1759 \let\forest@node@init\forest@node@init
1760 \def\forest@node@copy#1#2{% #1=from node id, cs receiving the new node id
1761   \advance\forest@node@maxid1
1762   \def\forest@node@init##1{\ifstrequal{##1}{name}{\forestoset{name}{node@\forest@node@cn}}{\forest@node@init}}
1763   \forest@node@init#1\forest@node@maxid}%
1764   \forest@node@init
1765   \forest@node@setname{id}\forest@node@cn}%
1766   \forest@node@setname{forest@copy@name@template{\forest@node@name{#1}{name}}}%
1767   \edef#2{\forest@node@cn}%
1768 }%
1769 \let\forest@node@init\forest@node@init
1770 }
1771 \forestset{
1772   copy name template/.code={\def\forest@copy@name@template##1{#1}},
1773   copy name template/.default={node@\forest@node@maxid},
1774   copy name template
1775 }
1776 \def\forest@tree@copy#1#2{% #1=from node id, #2=cs receiving the new node id
1777   \forest@node@copy#1\forest@node@copy@temp@id

```

```

1778 \forest@fornode{\forest@node@copy@temp@id}{%
1779   \expandafter\forest@tree@copy\expandafter{\forest@node@copy@temp@id}{#1}%
1780   \edef#2{\forest@cn}%
1781 }%
1782 }
1783 \def\forest@tree@copy@#1#2{%
1784   \forest@node@Foreachchild{#2}{%
1785     \expandafter\forest@tree@copy\expandafter{\forest@cn}\forest@node@copy@temp@childid
1786     \forest@node@Append{#1}{\forest@node@copy@temp@childid}%
1787   }%
1788 }

```

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.

```

1789 \def\forest@cn{0}
1790 \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.

```

1791 \def\forest@node@append#1{\expandtwonumberargs\forest@node@Append{\forest@cn}{#1}}
1792 \def\forest@node@prepend#1{\expandtwonumberargs\forest@node@Insertafter{\forest@cn}{#1}{0}}
1793 \def\forest@node@insertafter#1#2{%
1794   \expandthreenumberargs\forest@node@Insertafter{\forest@cn}{#1}{#2}}
1795 \def\forest@node@insertbefore#1#2{%
1796   \expandthreenumberargs\forest@node@Insertafter{\forest@cn}{#1}{\forestOve{#2}{@previous}}}%
1797 }
1798 \def\forest@node@remove{\expandnumberarg\forest@node@Remove{\forest@cn}}
1799 \def\forest@node@Append#1#2{\expandtwonumberargs\forest@node@Append@{#1}{#2}}
1800 \def\forest@node@Prepend#1#2{\expandtwonumberargs\forest@node@Insertafter{#1}{#2}{0}}
1801 \def\forest@node@Insertafter#1#2#3{% #2 is inserted after #3
1802   \expandthreenumberargs\forest@node@Insertafter@{#1}{#2}{#3}%
1803 }
1804 \def\forest@node@Insertbefore#1#2#3{% #2 is inserted before #3
1805   \expandthreenumberargs\forest@node@Insertafter{#1}{#2}{\forestOve{#3}{@previous}}}%
1806 }
1807 \def\forest@node@Remove#1{\expandnumberarg\forest@node@Remove@{#1}}
1808 \def\forest@node@Insertafter@#1#2#3{%
1809   \ifnum\forestOve{#2}{@parent}=0
1810   \else
1811     \PackageError{forest}{Insertafter(#1,#2,#3):
1812       node #2 already has a parent (\forestOve{#2}{@parent})}{}%
1813   \fi
1814   \ifnum#3=0
1815   \else
1816     \ifnum#1=\forestOve{#3}{@parent}
1817     \else
1818       \PackageError{forest}{Insertafter(#1,#2,#3): node #1 is not the parent of the
1819         intended sibling #3 (with parent \forestOve{#3}{@parent})}{}%
1820     \fi
1821   \fi
1822   \forestOset{#2}{@parent}{#1}%
1823   \forestOset{#2}{@previous}{#3}%
1824   \ifnum#3=0
1825     \forestOget{#1}{@first}\forest@node@temp
1826     \forestOset{#1}{@first}{#2}%
1827   \else

```

```

1828 \forestOget{#3}{@next}\forest@node@temp
1829 \forestOset{#3}{@next}{#2}%
1830 \fi
1831 \forestOset{#2}{@next}\forest@node@temp}%
1832 \ifnum\forest@node@temp=0
1833 \forestOset{#1}{@last}{#2}%
1834 \else
1835 \forestOset{\forest@node@temp}{@previous}{#2}%
1836 \fi
1837 }
1838 \def\forest@node@Append@#1#2{%
1839 \ifnum\forestOve{#2}{@parent}=0
1840 \else
1841 \PackageError{forest}{Append(#1,#2):
1842 node #2 already has a parent (\forestOve{#2}{@parent})}{}%
1843 \fi
1844 \forestOset{#2}{@parent}{#1}%
1845 \forestOget{#1}{@last}\forest@node@temp
1846 \forestOset{#1}{@last}{#2}%
1847 \forestOset{#2}{@previous}\forest@node@temp}%
1848 \ifnum\forest@node@temp=0
1849 \forestOset{#1}{@first}{#2}%
1850 \else
1851 \forestOset{\forest@node@temp}{@next}{#2}%
1852 \fi
1853 }
1854 \def\forest@node@Remove@#1{%
1855 \forestOget{#1}{@parent}\forest@node@temp@parent
1856 \ifnum\forest@node@temp@parent=0
1857 \else
1858 \forestOget{#1}{@previous}\forest@node@temp@previous
1859 \forestOget{#1}{@next}\forest@node@temp@next
1860 \ifnum\forest@node@temp@previous=0
1861 \forestOset{\forest@node@temp@parent}{@first}\forest@node@temp@next}%
1862 \else
1863 \forestOset{\forest@node@temp@previous}{@next}\forest@node@temp@next}%
1864 \fi
1865 \ifnum\forest@node@temp@next=0
1866 \forestOset{\forest@node@temp@parent}{@last}\forest@node@temp@previous}%
1867 \else
1868 \forestOset{\forest@node@temp@next}{@previous}\forest@node@temp@previous}%
1869 \fi
1870 \forestOset{#1}{@parent}{0}%
1871 \forestOset{#1}{@previous}{0}%
1872 \forestOset{#1}{@next}{0}%
1873 \fi
1874 }

```

Do some stuff and return to the current node.

```

1875 \def\forest@forthis#1{%
1876 \edef\forest@node@marshal{\unexpanded{#1}\def\noexpand\forest@cn}%
1877 \expandafter\forest@node@marshal\expandafter{\forest@cn}%
1878 }
1879 \def\forest@fornode#1#2{%
1880 \edef\forest@node@marshal{\edef\noexpand\forest@cn{#1}\unexpanded{#2}\def\noexpand\forest@cn}%
1881 \expandafter\forest@node@marshal\expandafter{\forest@cn}%
1882 }

```

Looping methods: children.

```

1883 \def\forest@node@foreachchild#1{\forest@node@Foreachchild{\forest@cn}{#1}}
1884 \def\forest@node@Foreachchild#1#2{%
1885 \forest@fornode{\forestOve{#1}{@first}}{\forest@node@@forselfandfollowingsiblings{#2}}%

```

```

1886 }
1887 \def\forest@node@@forselfandfollowingsiblings#1{%
1888   \ifnum\forest@cn=0
1889   \else
1890     \forest@forthis{#1}%
1891     \@escapeif{%
1892       \edef\forest@cn{\forestove{@next}}%
1893       \forest@node@@forselfandfollowingsiblings{#1}%
1894     }%
1895   \fi
1896 }
1897 \def\forest@node@@forselfandfollowingsiblings@reversed#1{%
1898   \ifnum\forest@cn=0
1899   \else
1900     \@escapeif{%
1901       \edef\forest@marshal{%
1902         \noexpand\def\noexpand\forest@cn{\forestove{@next}}%
1903         \noexpand\forest@node@@forselfandfollowingsiblings@reversed{\unexpanded{#1}}%
1904         \noexpand\forest@fornode{\forest@cn}{\unexpanded{#1}}%
1905       }\forest@marshal
1906     }%
1907   \fi
1908 }
1909 \def\forest@node@foreachchild@reversed#1{\forest@node@Foreachchild@reversed{\forest@cn}{#1}}
1910 \def\forest@node@Foreachchild@reversed#1#2{%
1911   \forest@fornode{\forestOve{#1}{@last}}{\forest@node@@forselfandprecedingsiblings@reversed{#2}}%
1912 }
1913 \def\forest@node@@forselfandprecedingsiblings@reversed#1{%
1914   \ifnum\forest@cn=0
1915   \else
1916     \forest@forthis{#1}%
1917     \@escapeif{%
1918       \edef\forest@cn{\forestove{@previous}}%
1919       \forest@node@@forselfandprecedingsiblings@reversed{#1}%
1920     }%
1921   \fi
1922 }
1923 \def\forest@node@@forselfandprecedingsiblings#1{%
1924   \ifnum\forest@cn=0
1925   \else
1926     \@escapeif{%
1927       \edef\forest@marshal{%
1928         \noexpand\def\noexpand\forest@cn{\forestove{@previous}}%
1929         \noexpand\forest@node@@forselfandprecedingsiblings{\unexpanded{#1}}%
1930         \noexpand\forest@fornode{\forest@cn}{\unexpanded{#1}}%
1931       }\forest@marshal
1932     }%
1933   \fi
1934 }

```

Looping methods: (sub)tree and descendants.

```

1935 \def\forest@node@@foreach#1#2#3#4{%
1936   % #1 = do what
1937   % #2 = do that -1=before,1=after processing children
1938   % #3 & #4: normal or reversed order of children?
1939   %   #3 = @first/@last
1940   %   #4 = \forest@node@@forselfandfollowingsiblings / \forest@node@@forselfandprecedingsiblings@reversed
1941   \ifnum#2<0 \forest@forthis{#1}\fi
1942   \ifnum\forestove{#3}=0
1943   \else\@escapeif{%
1944     \forest@forthis{%

```

```

1945 \edef\forest@cn{\forest@ve{#3}}%
1946 #4{\forest@node@@foreach{#1}{#2}{#3}{#4}}%
1947 }%
1948 }\fi
1949 \ifnum#2>0 \forest@forthis{#1}\fi
1950 }
1951 \def\forest@node@foreach#1{%
1952 \forest@node@@foreach{#1}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1953 \def\forest@node@Foreach#1#2{%
1954 \forest@fornode{#1}{\forest@node@@foreach{#2}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}}
1955 \def\forest@node@foreach@reversed#1{%
1956 \forest@node@@foreach{#1}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1957 \def\forest@node@Foreach@reversed#1#2{%
1958 \forest@fornode{#1}{\forest@node@@foreach{#2}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}}
1959 \def\forest@node@foreach@childrenfirst#1{%
1960 \forest@node@@foreach{#1}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1961 \def\forest@node@Foreach@childrenfirst#1#2{%
1962 \forest@fornode{#1}{\forest@node@@foreach{#2}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}}
1963 \def\forest@node@foreach@childrenfirst@reversed#1{%
1964 \forest@node@@foreach{#1}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1965 \def\forest@node@Foreach@childrenfirst@reversed#1#2{%
1966 \forest@fornode{#1}{\forest@node@@foreach{#2}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}}
1967 \def\forest@node@foreach@descendant#1{%
1968 \forest@node@foreachchild{\forest@node@@foreach{#1}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}}
1969 \def\forest@node@Foreach@descendant#1#2{%
1970 \forest@node@Foreachchild{#1}{\forest@node@@foreach{#2}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}}
1971 \def\forest@node@foreach@descendant@reversed#1{%
1972 \forest@node@foreachchild@reversed{\forest@node@@foreach{#1}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}}
1973 \def\forest@node@Foreach@descendant@reversed#1#2{%
1974 \forest@node@Foreachchild@reversed{#1}{\forest@node@@foreach{#2}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}}
1975 \def\forest@node@foreach@descendant@childrenfirst#1{%
1976 \forest@node@foreachchild{\forest@node@@foreach{#1}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}}
1977 \def\forest@node@Foreach@descendant@childrenfirst#1#2{%
1978 \forest@node@Foreachchild{#1}{\forest@node@@foreach{#2}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}}
1979 \def\forest@node@foreach@descendant@childrenfirst@reversed#1{%
1980 \forest@node@foreachchild@reversed{\forest@node@@foreach{#1}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}}
1981 \def\forest@node@Foreach@descendant@childrenfirst@reversed#1#2{%
1982 \forest@node@Foreachchild@reversed{#1}{\forest@node@@foreach{#2}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}}

```

Looping methods: breadth-first.

```

1983 \def\forest@node@foreach@breadthfirst#1#2{% #1 = max level, #2 = code
1984 \forest@node@Foreach@breadthfirst@{\forest@cn}{@first}{@next}{#1}{#2}}
1985 \def\forest@node@foreach@breadthfirst@reversed#1#2{% #1 = max level, #2 = code
1986 \forest@node@Foreach@breadthfirst@{\forest@cn}{@last}{@previous}{#1}{#2}}
1987 \def\forest@node@Foreach@breadthfirst#1#2#3{% #1 = node id, #2 = max level, #3 = code
1988 \forest@node@Foreach@breadthfirst@{#1}{@first}{@next}{#2}{#3}}
1989 \def\forest@node@Foreach@breadthfirst@reversed#1#2#3{% #1 = node id, #2 = max level, #3 = code
1990 \forest@node@Foreach@breadthfirst@{#1}{@last}{@previous}{#2}{#3}}
1991 \def\forest@node@Foreach@breadthfirst@#1#2#3#4#5{%
1992 % #1 = root node,
1993 % #2 = @first/@last, #3 = @next/@previous (must be in sync with #2),
1994 % #4 = max level (< 0 means infinite)
1995 % #5 = code to execute at each node
1996 \forest@node@Foreach@breadthfirst@processqueue{#1}{#2}{#3}{#4}{#5}%
1997 }
1998 \def\forest@node@Foreach@breadthfirst@processqueue#1#2#3#4#5{%
1999 % #1 = queue,
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 \ifstrempy{#1}{-}{%

```



```

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

Compute n, n', n children and level.
2057 \def\forest@node@Compute@numeric@ts@info@#1{%
2058 \forest@node@Foreach{#1}{\forest@node@@compute@numeric@ts@info}%
2059 \ifnum\forest@ve{#1}{@parent}=0
2060 \else
2061 \forest@for@node{#1}{\forest@node@@compute@numeric@ts@info@nbar}%
2062 % hack: the parent of the node we called the update for gets +1 for n_children

```

```

2063 \edef\forest@node@temp{\forestOve{#1}{@parent}}%
2064 \forestOset{\forest@node@temp}{n children}{%
2065 \number\numexpr\forestOve{\forest@node@temp}{n children}-1%
2066 }%
2067 \fi
2068 \forest@node@Foreachdescendant{#1}{\forest@node@@compute@numeric@ts@info@nbar}%
2069 }
2070 \def\forest@node@@compute@numeric@ts@info{%
2071 \forestoset{n children}{0}%
2072 %
2073 \edef\forest@node@temp{\forestove{@previous}}%
2074 \ifnum\forest@node@temp=0
2075 \forestoset{n}{1}%
2076 \else
2077 \forestoeset{n}{\number\numexpr\forestOve{\forest@node@temp}{n}+1}%
2078 \fi
2079 %
2080 \edef\forest@node@temp{\forestove{@parent}}%
2081 \ifnum\forest@node@temp=0
2082 \forestoset{n}{0}%
2083 \forestoset{n'}{0}%
2084 \forestoset{level}{0}%
2085 \else
2086 \forestOset{\forest@node@temp}{n children}{%
2087 \number\numexpr\forestOve{\forest@node@temp}{n children}+1%
2088 }%
2089 \forestoeset{level}{%
2090 \number\numexpr\forestOve{\forest@node@temp}{level}+1%
2091 }%
2092 \fi
2093 }
2094 \def\forest@node@@compute@numeric@ts@info@nbar{%
2095 \forestoeset{n'}{\number\numexpr\forestOve{\forestove{@parent}}{n children}-\forestove{n}+1}%
2096 }
2097 \def\forest@node@compute@numeric@ts@info#1{%
2098 \expandnumberarg\forest@node@Compute@numeric@ts@info@{\forest@cn}%
2099 }
2100 \def\forest@node@Compute@numeric@ts@info#1{%
2101 \expandnumberarg\forest@node@Compute@numeric@ts@info@{#1}%
2102 }

```

Tree structure queries.

```

2103 \def\forest@node@rootid{%
2104 \expandnumberarg\forest@node@Rootid{\forest@cn}%
2105 }
2106 \def\forest@node@Rootid#1{% #1=node
2107 \ifnum\forestOve{#1}{@parent}=0
2108 #1%
2109 \else
2110 \@escapeif{\expandnumberarg\forest@node@Rootid{\forestOve{#1}{@parent}}}%
2111 \fi
2112 }
2113 \def\forest@node@nthchildid#1{% #1=n
2114 \ifnum#1<1
2115 0%
2116 \else
2117 \expandnumberarg\forest@node@nthchildid@{\number\forestove{@first}}{#1}%
2118 \fi
2119 }
2120 \def\forest@node@nthchildid@#1#2{%
2121 \ifnum#1=0

```

```

2122 0%
2123 \else
2124 \ifnum#2>1
2125 \escapeifif{\expandtwonumberargs
2126 \forest@node@nthchildid@{\forestOve{#1}{@next}}{\numexpr#2-1}}%
2127 \else
2128 #1%
2129 \fi
2130 \fi
2131 }
2132 \def\forest@node@nbarthchildid#1{% #1=n
2133 \expandnumberarg\forest@node@nbarthchildid@{\number\forestove{@last}}{#1}%
2134 }
2135 \def\forest@node@nbarthchildid@#1#2{%
2136 \ifnum#1=0
2137 0%
2138 \else
2139 \ifnum#2>1
2140 \escapeifif{\expandtwonumberargs
2141 \forest@node@nbarthchildid@{\forestOve{#1}{@previous}}{\numexpr#2-1}}%
2142 \else
2143 #1%
2144 \fi
2145 \fi
2146 }
2147 \def\forest@node@nornbarthchildid#1{%
2148 \ifnum#1>0
2149 \forest@node@nthchildid{#1}%
2150 \else
2151 \ifnum#1<0
2152 \forest@node@nbarthchildid{-#1}%
2153 \else
2154 \forest@node@nornbarthchildid@error
2155 \fi
2156 \fi
2157 }
2158 \def\forest@node@nornbarthchildid@error{%
2159 \PackageError{forest}{In \string\forest@node@nornbarthchildid, n should !=0}{}%
2160 }
2161 \def\forest@node@previousleafid{%
2162 \expandnumberarg\forest@node@Previousleafid{\forest@cn}%
2163 }
2164 \def\forest@node@Previousleafid#1{%
2165 \ifnum\forestOve{#1}{@previous}=0
2166 \escapeif{\expandnumberarg\forest@node@previousleafid@Goup{#1}}%
2167 \else
2168 \expandnumberarg\forest@node@previousleafid@Godown{\forestOve{#1}{@previous}}%
2169 \fi
2170 }
2171 \def\forest@node@previousleafid@Goup#1{%
2172 \ifnum\forestOve{#1}{@parent}=0
2173 \PackageError{forest}{get previous leaf: this is the first leaf}{}%
2174 \else
2175 \escapeif{\expandnumberarg\forest@node@Previousleafid{\forestOve{#1}{@parent}}}%
2176 \fi
2177 }
2178 \def\forest@node@previousleafid@Godown#1{%
2179 \ifnum\forestOve{#1}{@last}=0
2180 #1%
2181 \else
2182 \escapeif{\expandnumberarg\forest@node@previousleafid@Godown{\forestOve{#1}{@last}}}%

```

```

2183 \fi
2184 }
2185 \def\forest@node@nextleafid{%
2186 \expandnumberarg\forest@node@Nextleafid{\forest@cn}%
2187 }
2188 \def\forest@node@Nextleafid#1{%
2189 \ifnum\forestOve{#1}{@next}=0
2190 \@escapeif{\expandnumberarg\forest@node@nextleafid@Goup{#1}}%
2191 \else
2192 \expandnumberarg\forest@node@nextleafid@Gdown{\forestOve{#1}{@next}}%
2193 \fi
2194 }
2195 \def\forest@node@nextleafid@Goup#1{%
2196 \ifnum\forestOve{#1}{@parent}=0
2197 \PackageError{forest}{get next leaf: this is the last leaf}{}%
2198 \else
2199 \@escapeif{\expandnumberarg\forest@node@Nextleafid{\forestOve{#1}{@parent}}}%
2200 \fi
2201 }
2202 \def\forest@node@nextleafid@Gdown#1{%
2203 \ifnum\forestOve{#1}{@first}=0
2204 #1%
2205 \else
2206 \@escapeif{\expandnumberarg\forest@node@nextleafid@Gdown{\forestOve{#1}{@first}}}%
2207 \fi
2208 }
2209
2210
2211
2212 \def\forest@node@linearnextid{%
2213 \ifnum\forestove{@first}=0
2214 \expandafter\forest@node@linearnextnotdescendantid
2215 \else
2216 \forestove{@first}%
2217 \fi
2218 }
2219 \def\forest@node@linearnextnotdescendantid{%
2220 \expandnumberarg\forest@node@Linearnextnotdescendantid{\forest@cn}%
2221 }
2222 \def\forest@node@Linearnextnotdescendantid#1{%
2223 \ifnum\forestOve{#1}{@next}=0
2224 \ifnum\forestOve{#1}{@parent}=0
2225 0%
2226 \else
2227 \@escapeif{\expandnumberarg\forest@node@Linearnextnotdescendantid{\forestOve{#1}{@parent}}}%
2228 \fi
2229 \else
2230 \forestOve{#1}{@next}%
2231 \fi
2232 }
2233 \def\forest@node@linearpreviousid{%
2234 \ifnum\forestove{@previous}=0
2235 \forestove{@parent}%
2236 \else
2237 \forest@node@previousleafid
2238 \fi
2239 }

```

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.

```

2240
2241 \def\forest@ifancestorof#1{% is the current node an ancestor of #1? Yes: #2, no: #3
2242   \begingroup
2243   \expandnumberarg\forest@ifancestorof@{\forestOve{#1}{@parent}}%
2244 }
2245 \def\forest@ifancestorof@#1{%
2246   \ifnum#1=0
2247     \def\forest@ifancestorof@next{\expandafter\endgroup\@secondoftwo}%
2248   \else
2249     \ifnum\forest@cn=#1
2250       \def\forest@ifancestorof@next{\expandafter\endgroup\@firstoftwo}%
2251     \else
2252       \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.

```

2253       \def\forest@ifancestorof@next{\expandafter\endgroup\@secondoftwo}%
2254     }{%
2255       \csdef{forest@circularity@used#1}{}%
2256       \def\forest@ifancestorof@next{\expandnumberarg\forest@ifancestorof@{\forestOve{#1}{@parent}}}%
2257     }%
2258   \fi
2259 \fi
2260 \forest@ifancestorof@next
2261 }

```

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

```

2262 \NewDocumentCommand\forestdebugtypeouttrees{o}{%
2263   \forestdebug@typeouttrees\forest@temp
2264   \typeout{%
2265     \forestdebugtypeouttreesprefix
2266     \IfValueTF{#1}{#1: }{}%
2267     \detokenize\expandafter{\forest@temp}%
2268     \forestdebugtypeouttreessuffix
2269   }%
2270 }
2271 \def\forestdebug@typeouttrees#1{% #1 = cs to store the result
2272   \begingroup
2273   \edef\forest@temp@message{}%
2274   \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.

```

2275   \loop
2276   \ifnum\forestdebug@typeouttrees@n<\forest@node@maxid
2277     \edef\forestdebug@typeouttrees@n{\number\numexpr\forestdebug@typeouttrees@n+1}%
2278     \ifcsdef{forestdebug@typeouttree@used@\forestdebug@typeouttrees@n}{}%
2279     \forest@for node{\forestdebug@typeouttrees@n}{%

```

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

```

2280     \begingroup
2281     \forestdebug@typeouttrees@findroot
2282     \expandafter\endgroup
2283     \expandafter\edef\expandafter\forest@cn\expandafter{\forest@cn}%
2284     \forestdebug@typeouttree@build
2285     \appto\forest@temp@message{ }%
2286   }%
2287 }%
2288 \repeat
2289 \expandafter\endgroup
2290 \expandafter\def\expandafter#1\expandafter{\forest@temp@message}%
2291 }

```

```

2292 \def\forestdebug@typeouttrees@findroot{%
2293   \let\forestdebug@typeouttrees@next\relax
2294   \edef\forestdebug@typeouttrees@parent{\forestOve{\forest@cn}{@parent}}%
2295   \ifnum\forestdebug@typeouttrees@parent=0
2296   \else
2297     \ifcsdef\forestdebug@typeouttree@used@\forest@cn}{-}{%
2298       \csdef\forestdebug@typeouttree@used@\forest@cn}{-}%
2299       \edef\forest@cn{\forestdebug@typeouttrees@parent}%
2300       \let\forestdebug@typeouttrees@next\forestdebug@typeouttrees@findroot
2301     }%
2302   \fi
2303   \forestdebug@typeouttrees@next
2304 }
2305 \def\forestdebug@typeouttree#1#2{% #1=root id, #2=cs to receive result
2306   \begingroup
2307   \edef\forest@temp@message{%
2308     \forest@for@node{#1}{\forestdebug@typeouttree@build}%
2309     \expandafter\endgroup
2310     \expandafter\edef\expandafter#2\expandafter{\forest@temp@message}%
2311   }
2312   \NewDocumentCommand\forestdebugtypeouttree{o m}{%
2313     \forestdebug@typeouttree{#1}\forest@temp
2314     \typeout{\IfValueTF{#1}{#1: }{\}\forest@temp}%
2315   }

```

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

```

2316 \def\forestdebug@typeouttree@build{%
2317   \eappto\forest@temp@message{\forestdebugtypeouttreenodeinfo%]
2318   \ifcsdef\forestdebug@typeouttree@used@\forest@cn}{*}{}%
2319   }%
2320   \ifcsdef\forestdebug@typeouttree@used@\forest@cn}{-}{%
2321     \csdef\forestdebug@typeouttree@used@\forest@cn}{-}%
2322     \forest@node@foreachchild{\forestdebug@typeouttree@build}%
2323   }%
2324   \eappto\forest@temp@message{[%
2325   ]}%
2326 }
2327 \def\forestdebugtypeouttreenodeinfo{\forest@cn}
2328 \def\forestdebugtypeouttreesprefix{}
2329 \def\forestdebugtypeouttreessuffix{}

```

6.3 Node options

6.3.1 Option-declaration mechanism

Common code for declaring options.

```

2330 \def\forest@declarehandler#1#2#3{% #1=handler for specific type,#2=option name,#3=default value
2331   \pgfkeyssetvalue{/forest/#2}{#3}%
2332   \appto\forest@node@init{\forest@init{#2}}%
2333   \pgfkeyssetvalue{/forest/#2/node@or@reg}{\forest@cn}%
2334   \forest@convert@others@to@underscores{#2}\forest@pgfmathoptionname
2335   \edef\forest@marshal{%
2336     \noexpand#1{/forest/#2}{/forest}{#2}{\forest@pgfmathoptionname}%
2337   }\forest@marshal
2338 }
2339 \def\forest@def@with@pgfeov#1#2{% \pgfeov mustn't occur in the arg of the .code handler!!!
2340   \long\def##1\pgfeov{#2}%
2341 }

```

Option-declaration handlers.

```

2342 \def\forest@declaretoks@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname

```

```

2343 \forest@declaretoks@handler@A{#1}{#2}{#3}{#4}{}%
2344 }
2345 \def\forest@declarekeylist@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2346 \forest@declaretoks@handler@A{#1}{#2}{#3}{#4}{,}%
2347 \forest@copycommandkey{#1}{#1'}%
2348 \pgfkeyssetvalue{#1'/option@name}{#3}%
2349 \forest@copycommandkey{#1+}{#1}%
2350 \pgfkeysalso{#1-/.code={%
2351 \forest@fornode{\forest@setter@node}{%
2352 \forest@node@removekeysfromkeylist{##1}{#3}%
2353 }}}%
2354 \pgfkeyssetvalue{#1-/.option@name}{#3}%
2355 }
2356 \def\forest@declaretoks@handler@A#1#2#3#4#5{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
2357 \pgfkeysalso{%
2358 #1/.code={\forest0set{\forest@setter@node}{#3}{##1}},
2359 #2/if #3/.code n args={3}{%
2360 \forest0get{#3}\forest@temp@option@value
2361 \edef\forest@temp@compared@value{\unexpanded{##1}}%
2362 \ifx\forest@temp@option@value\forest@temp@compared@value
2363 \pgfkeysalso{##2}%
2364 \else
2365 \pgfkeysalso{##3}%
2366 \fi
2367 },
2368 #2/if in #3/.code n args={3}{%
2369 \forest0get{#3}\forest@temp@option@value
2370 \edef\forest@temp@compared@value{\unexpanded{##1}}%
2371 \expandafter\expandafter\expandafter\pgfutil@in@\expandafter\expandafter\expandafter{\expandafter\forest@temp@option@value\forest@temp@compared@value}
2372 \ifpgfutil@in@
2373 \pgfkeysalso{##2}%
2374 \else
2375 \pgfkeysalso{##3}%
2376 \fi
2377 },
2378 #2/where #3/.style n args={3}{for tree={#2/if #3={##1}{##2}{##3}}},
2379 #2/where in #3/.style n args={3}{for tree={#2/if in #3={##1}{##2}{##3}}}%
2380 }%
2381 \ifstrempy{#5}{%
2382 \pgfkeysalso{%
2383 #1+/.code={\forest0appto{\forest@setter@node}{#3}{#5##1}},
2384 #2/+##3/.code={\forest0preto{\forest@setter@node}{#3}{##1#5}},
2385 }%
2386 }{%
2387 \pgfkeysalso{%
2388 #1+/.code={%
2389 \forest0get{\forest@setter@node}{#3}\forest@temp
2390 \ifdefempty{\forest@temp}{%
2391 \forest0set{\forest@setter@node}{#3}{##1}%
2392 }{%
2393 \forest0appto{\forest@setter@node}{#3}{#5##1}%
2394 }%
2395 },
2396 #2/+##3/.code={%
2397 \forest0get{\forest@setter@node}{#3}\forest@temp
2398 \ifdefempty{\forest@temp}{%
2399 \forest0set{\forest@setter@node}{#3}{##1}%
2400 }{%
2401 \forest0preto{\forest@setter@node}{#3}{##1#5}%
2402 }%
2403 }%

```

```

2404 }%
2405 }%
2406 \pgfkeyssetvalue{#1/option@name}{#3}%
2407 \pgfkeyssetvalue{#1+/option@name}{#3}%
2408 \pgfkeyssetvalue{#2/+ #3/option@name}{#3}%
2409 \pgfkeyslet{#1/@type}\forestmathtype@generic % for .process & co
2410 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@toks{##1}{#3}}%
2411 }
2412 \def\forest@declareautowrappedtoks@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
2413 \forest@declaretoks@handler{#1}{#2}{#3}{#4}%
2414 \forest@copycommandkey{#1}{#1'}%
2415 \pgfkeysalso{#1/.style={#1'/.wrap value={##1}}}%
2416 \pgfkeyssetvalue{#1'/option@name}{#3}%
2417 \forest@copycommandkey{#1+}{#1+'}%
2418 \pgfkeysalso{#1+/.style={#1+'/.wrap value={##1}}}%
2419 \pgfkeyssetvalue{#1+'/option@name}{#3}%
2420 \forest@copycommandkey{#2/+ #3}{#2/+ #3'}%
2421 \pgfkeysalso{#2/+ #3/.style={#2/+ #3'/.wrap value={##1}}}%
2422 \pgfkeyssetvalue{#2/+ #3'/option@name}{#3}%
2423 }
2424 \def\forest@declarereadonlydimen@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2425 % this is to have 'pt' with the correct category code
2426 \pgfutil@tempdima=\pgfkeysvalueof{/forest/#3}\relax
2427 \edef\forest@marshal{%
2428 \noexpand\pgfkeyssetvalue{/forest/#3}{\the\pgfutil@tempdima}%
2429 }\forest@marshal
2430 \pgfkeysalso{%
2431 #2/if #3/.code n args={3}{%
2432 \foresttoget{#3}\forest@temp@option@value
2433 \ifdim\forest@temp@option@value>##1\relax
2434 \pgfkeysalso{##2}%
2435 \else
2436 \pgfkeysalso{##3}%
2437 \fi
2438 },
2439 #2/if #3</.code n args={3}{%
2440 \foresttoget{#3}\forest@temp@option@value
2441 \ifdim\forest@temp@option@value>##1\relax
2442 \pgfkeysalso{##3}%
2443 \else
2444 \pgfkeysalso{##2}%
2445 \fi
2446 },
2447 #2/if #3>/.code n args={3}{%
2448 \foresttoget{#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/where #3/.style n args={3}{for tree={#2/if #3={##1}{##2}{##3}}},
2456 #2/where #3</.style n args={3}{for tree={#2/if #3<={##1}{##2}{##3}}},
2457 #2/where #3>/.style n args={3}{for tree={#2/if #3>={##1}{##2}{##3}}},
2458 }%
2459 \pgfkeyslet{#1/@type}\forestmathtype@dimen % for .process & co
2460 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@dimen{##1}{#3}}%
2461 }
2462 \def\forest@declaredimen@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2463 \forest@declarereadonlydimen@handler{#1}{#2}{#3}{#4}%
2464 \pgfkeysalso{%

```



```

2465 #1/.code={%
2466   \forestmathsetlengthmacro\forest@temp{##1}%
2467   \forest0let{\forest@setter@node}{#3}\forest@temp
2468 },
2469 #1+/.code={%
2470   \forestmathsetlengthmacro\forest@temp{##1}%
2471   \pgfutil@tempdima=\forestove{#3}
2472   \advance\pgfutil@tempdima\forest@temp\relax
2473   \forest0eset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2474 },
2475 #1-/.code={%
2476   \forestmathsetlengthmacro\forest@temp{##1}%
2477   \pgfutil@tempdima=\forestove{#3}
2478   \advance\pgfutil@tempdima-\forest@temp\relax
2479   \forest0eset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2480 },
2481 #1*/.style={%
2482   #1={#4()}*(##1)}%
2483 },
2484 #1:/ .style={%
2485   #1={#4()/ (##1)}%
2486 },
2487 #1'/.code={%
2488   \pgfutil@tempdima=##1\relax
2489   \forest0eset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2490 },
2491 #1'+/.code={%
2492   \pgfutil@tempdima=\forestove{#3}\relax
2493   \advance\pgfutil@tempdima##1\relax
2494   \forest0eset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2495 },
2496 #1'-/.code={%
2497   \pgfutil@tempdima=\forestove{#3}\relax
2498   \advance\pgfutil@tempdima-##1\relax
2499   \forest0eset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2500 },
2501 #1'*/.style={%
2502   \pgfutil@tempdima=\forestove{#3}\relax
2503   \multiply\pgfutil@tempdima##1\relax
2504   \forest0eset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2505 },
2506 #1':/.style={%
2507   \pgfutil@tempdima=\forestove{#3}\relax
2508   \divide\pgfutil@tempdima##1\relax
2509   \forest0eset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2510 },
2511 }%
2512 \pgfkeyssetvalue{#1/option@name}{#3}%
2513 \pgfkeyssetvalue{#1+/option@name}{#3}%
2514 \pgfkeyssetvalue{#1-/option@name}{#3}%
2515 \pgfkeyssetvalue{#1*/option@name}{#3}%
2516 \pgfkeyssetvalue{#1:/option@name}{#3}%
2517 \pgfkeyssetvalue{#1'/option@name}{#3}%
2518 \pgfkeyssetvalue{#1'+/option@name}{#3}%
2519 \pgfkeyssetvalue{#1'-/option@name}{#3}%
2520 \pgfkeyssetvalue{#1'*/option@name}{#3}%
2521 \pgfkeyssetvalue{#1':/option@name}{#3}%
2522 }
2523 \def\forest@declarereadonlycount@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2524   \pgfkeysalso{
2525     #2/if #3/.code n args={3}{%

```

```

2526 \foresttoget{#3}\forest@temp@option@value
2527 \ifnum\forest@temp@option@value=##1\relax
2528 \pgfkeysalso{##2}%
2529 \else
2530 \pgfkeysalso{##3}%
2531 \fi
2532 },
2533 #2/if #3</.code n args={3}{%
2534 \foresttoget{#3}\forest@temp@option@value
2535 \ifnum\forest@temp@option@value>##1\relax
2536 \pgfkeysalso{##3}%
2537 \else
2538 \pgfkeysalso{##2}%
2539 \fi
2540 },
2541 #2/if #3>/.code n args={3}{%
2542 \foresttoget{#3}\forest@temp@option@value
2543 \ifnum\forest@temp@option@value<##1\relax
2544 \pgfkeysalso{##3}%
2545 \else
2546 \pgfkeysalso{##2}%
2547 \fi
2548 },
2549 #2/where #3/.style n args={3}{for tree={#2/if #3={##1}{##2}{##3}}},
2550 #2/where #3</.style n args={3}{for tree={#2/if #3<={##1}{##2}{##3}}},
2551 #2/where #3>/.style n args={3}{for tree={#2/if #3>={##1}{##2}{##3}}},
2552 }%
2553 \pgfkeyslet{#1/@type}\forest@mathtype@count % for .process & co
2554 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@count{##1}{#3}}%
2555 }
2556 \def\forest@declarecount@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2557 \forest@declarereadonlycount@handler{#1}{#2}{#3}{#4}%
2558 \pgfkeysalso{
2559 #1/.code={%
2560 \forestmathtruncatemacro\forest@temp{##1}%
2561 \forest@let{\forest@setter@node}{#3}\forest@temp
2562 },
2563 #1+/.code={%
2564 \forestmathtruncatemacro\forest@temp{##1}%
2565 \c@pgf@counta=\forestove{#3}\relax
2566 \advance\c@pgf@counta\forest@temp\relax
2567 \forest@set{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2568 },
2569 #1-/.code={%
2570 \forestmathtruncatemacro\forest@temp{##1}%
2571 \c@pgf@counta=\forestove{#3}\relax
2572 \advance\c@pgf@counta-\forest@temp\relax
2573 \forest@set{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2574 },
2575 #1*/.code={%
2576 \forestmathtruncatemacro\forest@temp{##1}%
2577 \c@pgf@counta=\forestove{#3}\relax
2578 \multiply\c@pgf@counta\forest@temp\relax
2579 \forest@set{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2580 },
2581 #1:/.code={%
2582 \forestmathtruncatemacro\forest@temp{##1}%
2583 \c@pgf@counta=\forestove{#3}\relax
2584 \divide\c@pgf@counta\forest@temp\relax
2585 \forest@set{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2586 },

```

```

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

```

Nothing else should be defined in this namespace.

```

2623 \def\forest@declareboolean@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2624   \pgfkeysalso{%
2625     #1/.code={%
2626       \forestmath@if{##1}{%
2627         \def\forest@temp{1}%
2628       }{%
2629         \def\forest@temp{0}%
2630       }%
2631       \forestOlet{\forest@setter@node}{#3}\forest@temp
2632     },
2633     #1/.default=1,
2634     #2/not #3/.code={\forestOset{\forest@setter@node}{#3}{0}},
2635     #2/if #3/.code 2 args={%
2636       \foresttoget{#3}\forest@temp@option@value
2637       \ifnum\forest@temp@option@value=0
2638         \pgfkeysalso{##2}%
2639       \else
2640         \pgfkeysalso{##1}%
2641       \fi
2642     },
2643     #2/where #3/.style 2 args={for tree={#2/if #3={##1}{##2}}}
2644   }%
2645   \pgfkeyssetvalue{#1/option@name}{#3}%

```

```

2646 \pgfkeyslet{#1/@type}\forestmathtype@count % for .process & co
2647 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@count{##1}{#3}}%
2648 }
2649 \forestset{
2650   declare toks/.code 2 args={%
2651     \forest@declarehandler\forest@declaretoks@handler{#1}{#2}%
2652   },
2653   declare autowrapped toks/.code 2 args={%
2654     \forest@declarehandler\forest@declareautowrappedtoks@handler{#1}{#2}%
2655   },
2656   declare keylist/.code 2 args={%
2657     \forest@declarehandler\forest@declarekeylist@handler{#1}{#2}%
2658   },
2659   declare readonly dimen/.code 2 args={%
2660     \forestmathsetlengthmacro\forest@temp{#2}%
2661     \edef\forest@marshal{%
2662       \unexpanded{\forest@declarehandler\forest@declarereadonlydimen@handler{#1}}{\forest@temp}%
2663     }\forest@marshal
2664   },
2665   declare dimen/.code 2 args={%
2666     \forestmathsetlengthmacro\forest@temp{#2}%
2667     \edef\forest@marshal{%
2668       \unexpanded{\forest@declarehandler\forest@declaredimen@handler{#1}}{\forest@temp}%
2669     }\forest@marshal
2670   },
2671   declare readonly count/.code 2 args={%
2672     \forestmathtruncatemacro\forest@temp{#2}%
2673     \edef\forest@marshal{%
2674       \unexpanded{\forest@declarehandler\forest@declarereadonlycount@handler{#1}}{\forest@temp}%
2675     }\forest@marshal
2676   },
2677   declare count/.code 2 args={%
2678     \forestmathtruncatemacro\forest@temp{#2}%
2679     \edef\forest@marshal{%
2680       \unexpanded{\forest@declarehandler\forest@declarecount@handler{#1}}{\forest@temp}%
2681     }\forest@marshal
2682   },
2683   declare boolean/.code 2 args={%
2684     \forestmath@if{#2}{%
2685       \def\forest@temp{1}%
2686     }{%
2687       \def\forest@temp{0}%
2688     }%
2689     \edef\forest@marshal{%
2690       \unexpanded{\forest@declarehandler\forest@declareboolean@handler{#1}}{\forest@temp}%
2691     }\forest@marshal
2692   },

```

7 Handlers

```

2693 /handlers/.restore default value/.code={%
2694   \edef\forest@handlers@currentpath{\pgfkeys@currentpath}%
2695   \pgfkeys@getvalue{\pgfkeys@currentpath/option@name}\forest@currentoptionname
2696   \pgfkeys@getvalue{/forest/\forest@currentoptionname}\forest@temp
2697   \expandafter\pgfkeysalso\expandafter{\forest@handlers@currentpath/.expand once=\forest@temp}%
2698 },
2699 /handlers/.pgfmath/.code={%
2700   \pgfmathparse{#1}%
2701   \pgfkeysalso{\pgfkeys@currentpath/.expand once=\pgfmathresult}%
2702 },

```

```

2703 /handlers/.wrap value/.code={%
2704 \edef\forest@handlers@wrap@currentpath{\pgfkeyscurrentpath}%
2705 \pgfkeysgetvalue{\forest@handlers@wrap@currentpath/option@name}\forest@currentoptionname
2706 \forest@get{\pgfkeysvalueof{/forest/\forest@currentoptionname/node@or@reg}}{\forest@currentoptionname}\forest@
2707 \forest@def@with@pgfeov\forest@wrap@code#1}%
2708 \expandafter\edef\expandafter\forest@wrapped@value\expandafter{\expandafter\expandafter\expandafter{\expandafter\forest@
2709 \pgfkeysalso{\forest@handlers@wrap@currentpath/.expand once=\forest@wrapped@value}%
2710 },
2711 /handlers/.option/.code={%
2712 \edef\forest@temp{\pgfkeyscurrentpath}%
2713 \expandafter\forest@handlers@option\expandafter{\forest@temp}{#1}%
2714 },
2715 }
2716 \def\forest@handlers@option#1#2{%#1=pgfkeyscurrentpath,#2=relative node name
2717 \forestRN@get{#2}\forest@temp
2718 \pgfkeysalso{#1/.expand once={\forest@temp}}%
2719 }%
2720 \forestset{
2721 /handlers/.register/.code={%
2722 \edef\forest@marshal{%
2723 \noexpand\pgfkeysalso{\pgfkeyscurrentpath={\forestregister{#1}}}%
2724 }\forest@marshal
2725 },
2726 /handlers/.wrap pgfmath arg/.code 2 args={%
2727 \forestmathparse{#2}\let\forest@wrap@arg@i\forestmathresult
2728 \edef\forest@wrap@args{{\expandafter\forest@wrap@arg@i}}%
2729 \def\forest@wrap@code##1{#1}%
2730 % here we don't call \forest@wrap@pgfmath@args@@@wrapandpasson, as compat-2.0.2-wrappgfmathargs changes t
2731 \expandafter\expandafter\expandafter\forest@temp@toks\expandafter\expandafter\expandafter{\expandafter\forest@
2732 \expandafter\pgfkeysalso\expandafter{\expandafter\pgfkeyscurrentpath\expandafter=\expandafter{\the\forest@
2733 },
2734 /handlers/.wrap 2 pgfmath args/.code n args={3}{%
2735 \forestmathparse{#2}\let\forest@wrap@arg@i\forestmathresult
2736 \forestmathparse{#3}\let\forest@wrap@arg@ii\forestmathresult
2737 \edef\forest@wrap@args{{\expandafter\forest@wrap@arg@i}{\expandafter\forest@wrap@arg@ii}}%
2738 \def\forest@wrap@code##1##2{#1}%
2739 \forest@wrap@pgfmath@args@@@wrapandpasson
2740 },
2741 /handlers/.wrap 3 pgfmath args/.code n args={4}{%
2742 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{-}{-}{-}{-}{3}%
2743 \forest@wrap@n@pgfmath@do{#1}{3}},
2744 /handlers/.wrap 4 pgfmath args/.code n args={5}{%
2745 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{-}{-}{-}{-}{4}%
2746 \forest@wrap@n@pgfmath@do{#1}{4}},
2747 /handlers/.wrap 5 pgfmath args/.code n args={6}{%
2748 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{-}{-}{-}{5}%
2749 \forest@wrap@n@pgfmath@do{#1}{5}},
2750 /handlers/.wrap 6 pgfmath args/.code n args={7}{%
2751 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{#7}{-}{-}{6}%
2752 \forest@wrap@n@pgfmath@do{#1}{6}},
2753 /handlers/.wrap 7 pgfmath args/.code n args={8}{%
2754 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{#7}{#8}{-}{7}%
2755 \forest@wrap@n@pgfmath@do{#1}{7}},
2756 /handlers/.wrap 8 pgfmath args/.code n args={9}{%
2757 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}{8}%
2758 \forest@wrap@n@pgfmath@do{#1}{8}},
2759 }
2760 \def\forest@wrap@n@pgfmath@args#1#2#3#4#5#6#7#8#9{%
2761 \forestmathparse{#1}\let\forest@wrap@arg@i\forestmathresult
2762 \ifnum#9>1 \forestmathparse{#2}\let\forest@wrap@arg@ii\forestmathresult\fi
2763 \ifnum#9>2 \forestmathparse{#3}\let\forest@wrap@arg@iii\forestmathresult\fi

```

```

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

```

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

```

2793 \def\forest@wrap@pgfmath@args@@@wrapandpasson{%
2794   \expandafter\expandafter\expandafter\forest@temp@toks
2795     \expandafter\expandafter\expandafter{%
2796       \expandafter\forest@wrap@code\forest@wrap@args}%
2797   \expandafter\pgfkeysalso\expandafter{%
2798     \expandafter\pgfkeyscurrentpath\expandafter=\expandafter{%
2799       \the\forest@temp@toks}}%
2800 }

```

7.1 .process

```

2801 \def\forest@process@catregime{} % filled by processor defs
2802 \forest@newarray\forest@process@left@ % processed args
2803 \forest@newarray\forest@process@right@ % unprocessed args
2804 \forest@newarray\forest@process@savet@ % used by instructions |S| and |U|
2805 \let\forest@process@savetype\forestmathtype@none
2806 \forest@newglobalarray\forest@process@result@
2807 \newif\ifforest@process@returnarray@

```

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

```

2808 \def\forest@process#1#2{% #1 = true/false (should we return an array?)
2809   % #2 = processing instructions (if non-empty),
2810   % (initial) args follow
2811   \ifblank{#2}{\forest@process@a{#1}}{\forest@process@a{#1}{#2}}%
2812 }
2813 \Inline\def\forest@process@a#1#2{%
2814   \begingroup
2815   \forest@process@left@clear
2816   \forest@process@right@clear
2817   \forest@process@savet@clear
2818   \let\forest@process@savetype\forestmathtype@generic
2819   \ExpandIf{\forestdebug}{%

```

```

2820 \edef\forest@process@debug@args{\unexpanded{#2}}%
2821 \typeout{[forest .process] Start "\unexpanded{#2}"}%
2822 }%
2823 \csname forest@process@returnarray@#1\endcsname
2824 \def\forest@topextend@next{%
2825 \forest@process@catregime
2826 \endlinechar=-1
2827 \scantokens{#2}%
2828 \forest@process@finish
2829 }%
2830 \forest@process@right@topextend
2831 }
2832 \pgfkeys{%
2833 /handlers/.process/.code={%
2834 \forest@process{true}#1\forest@eov
2835 \edef\forest@marshal{%
2836 \noexpand\pgfkeysalso{\noexpand\pgfkeyscurrentpath=\forest@process@result@values}%
2837 }\forest@marshal
2838 },
2839 /forest/copy command key={/handlers/.process}{/handlers/.process args},
2840 }
2841 \def\forest@process@finish{%
2842 \ifforest@process@returnarray@
2843 \forest@process@finish@array
2844 \global\let\forest@process@result@type\forest@result@type@array
2845 \else
2846 \forest@process@finish@single
2847 \fi
2848 \ifforestdebugprocess\forest@process@debug@end\fi
2849 \endgroup
2850 }
2851 \def\forest@process@finish@single{%
2852 \edef\forest@temp{\forest@process@finish@single@%
2853 \the\numexpr\forest@process@left@N-\forest@process@left@M\relax
2854 \the\numexpr\forest@process@right@N-\forest@process@right@M\relax
2855 }%
2856 \ifcsname\forest@temp\endcsname
2857 \csname\forest@temp\endcsname
2858 \global\let\forest@process@result\forest@temp
2859 \else
2860 \forest@process@lengtherror
2861 \fi
2862 \global\let\forest@process@result@type\forestmathresulttype
2863 }
2864 \csdef{forest@process@finish@single@10}{\forest@process@left@toppop\forest@temp}
2865 \csdef{forest@process@finish@single@01}{\forest@process@right@toppop\forest@temp}
2866 \def\forest@process@finish@array{%
2867 \forest@process@result@clear
2868 \forest@temp@count\forest@process@left@M\relax
2869 \forest@loop
2870 \ifnum\forest@temp@count<\forest@process@left@N\relax
2871 \forest@process@left@get@{\the\forest@temp@count}\forest@temp
2872 \forest@process@result@letappend\forest@temp
2873 \advance\forest@temp@count1
2874 \forest@repeat
2875 \forest@temp@count\forest@process@right@M\relax
2876 \forest@loop
2877 \ifnum\forest@temp@count<\forest@process@right@N\relax
2878 \forest@process@right@get@{\the\forest@temp@count}\forest@temp
2879 \forest@process@result@letappend\forest@temp
2880 \advance\forest@temp@count1

```

```

2881 \forest@repeat
2882 }

Debugging and error messages.
2883 \ifforestdebug
2884 \let\forest@process@d\forest@process@b
2885 \def\forest@process@b#1\forest@eov{% save and print initial arguments
2886 \edef\forest@process@debug@args{\unexpanded{#1}}%
2887 \typeout{[forest .process] Start "\unexpanded{#1}"}%
2888 \forest@process@d#1\forest@eov
2889 }
2890 \fi
2891 \def\forest@process@debug@end{%
2892 \typeout{[forest .process] \space\space\space End "\expandonce{\forest@process@debug@args}" -> "\forest@pro
2893 }
2894 \def\forest@process@lengtherror{%
2895 \PackageError{forest}{%
2896 The ".process" expression was expected to evaluate to a single argument,
2897 but the result is \the\forest@process@result@N
2898 \space items long.}{}%
2899 }

```

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

```

2900 \def\forest@def@processor#1{%
2901 {%
2902 \def\forest@dp@double##1{%
2903 \gdef\forest@global@temp{\forest@def@processor@{#1}{##1}}%
2904 }%
2905 \let\\forest@dp@double
2906 \catcode'#1=13
2907 \scantokens{\\#1}%
2908 }%
2909 \forest@global@temp
2910 }
2911 \def\forest@def@processor@#1#2{%
2912 % #1 = instruction char (normal catcode), #2 = instruction char (active)
2913 % #3 = default n (optional numeric arg, which precedes any other args;
2914 % if the default is empty, this means no optional n)
2915 % #4 = args spec,
2916 % #5 = code
2917 \eappto\forest@process@catregime{%
2918 \unexpanded{\let#2}\expandonce{\cename forest@processor@#1\endcsname}%
2919 \unexpanded{\catcode'#1=13 }%
2920 }%
2921 \def\forest@def@processor@inschar{#1}%
2922 \forest@def@processor@@
2923 }

```

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

```

2924 \def\forest@def@processor@@#1{%
2925 \ifstrepty{#1}{%
2926 \forest@def@processor@@non
2927 }{%
2928 \def\forest@def@processor@@defaultn{#1}%
2929 \forest@def@processor@@n
2930 }%
2931 }

```

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:

```

2932 \def\forest@def@processor@@non#1#2{% #1=args spec, #2=code
2933 \csedef{forest@processor@\forest@def@processor@inschar}{#1{%

```



```

2934 \relax %% we need this (see above)
2935 \unexpanded{#2}%
2936 \expandafter\forest@def@processor@debuginfo\expandafter{%
2937 \expandafter"\forest@def@processor@inschar"\ifstrempy{#1}{#{1}}}%
2938 \ignorespaces
2939 }%
2940 }

Optional n: * after the given default means that the operation should be repeated n times.
2941 \def\forest@def@processor@n{%
2942 \@ifnextchar*%
2943 {\forest@temptrue\forest@def@processor@n}%
2944 {\forest@tempfalse\forest@def@processor@n@}%
2945 }
2946 \def\forest@def@processor@n@*{\forest@def@processor@n@}
2947 \def\forest@def@processor@n@#1#2{% #1=args spec, #2=code
2948 \csedef{forest@processor@\forest@def@processor@inschar}{%
2949 \relax %% we need this (see above)
2950 \noexpand\forestprocess@get@n
2951 {\forest@def@processor@n@default@n}%
2952 \expandonce{\csname forest@processor@\forest@def@processor@inschar @\endcsname}%
2953 }%
2954 \ifforest@temp
2955 \csedef{forest@processor@\forest@def@processor@inschar @}{%
2956 \noexpand\forest@repeat@n@times{\forest@process@n}{%
2957 \expandonce{\csname forest@processor@\forest@def@processor@inschar @rep\endcsname}%
2958 }%
2959 }%
2960 \fi
2961 \edef\forest@temp{%
2962 \forest@def@processor@inschar
2963 \ifforest@temp\else\noexpand\the\forest@process@n\fi
2964 "%}
2965 \csedef{forest@processor@\forest@def@processor@inschar @\ifforest@temp rep\fi}#1{%
2966 \unexpanded{#2}%
2967 \expandafter\forest@def@processor@debuginfo\expandafter{%
2968 \forest@temp
2969 \ifstrempy{#1}{#{1}}}%
2970 }%
2971 }
2972 \def\forest@def@processor@debuginfo#1{% #1 = instruction call
2973 \ifforest@debug
2974 \expandonce{\forest@processor@debuginfo@template{#1}}%
2975 \fi
2976 }
2977 \def\forest@processor@debuginfo@template#1{%
2978 \ifforest@debugprocess
2979 \edef\forest@temp@left{\forest@process@left@values}%
2980 \edef\forest@temp@right{\forest@process@right@values}%
2981 \edef\forest@temp@savd{\forest@process@savd@values}%
2982 \typeout{[forest .process]\space\space\space\space After #1: left="\expandonce{\forest@temp@left}", right
2983 \fi
2984 }

A helper macro which puts the optional numeric argument into count \forest@process@n (default being
#1) and then executes control sequence #2.
2985 \newcount\forest@process@n
2986 \def\forestprocess@get@n#1#2{%
2987 \def\forestprocess@default@n{#1}%
2988 \let\forestprocess@after@get@n@#2%
2989 \afterassignment\forestprocess@get@n@\forest@process@n=0%
2990 }

```

```

2991 \def\forestprocess@get@n@{%
2992   \ifnum\forest@process@n=0
2993     \forest@process@n\forestprocess@default@n\relax
2994   \fi
2995   \forestprocess@after@get@n@
2996 }

```

Definitions of processing instructions. Processors should be defined using `\forest@def@processor`.

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

```

2997 \forest@def@processor@_@{1}*{}{% no processing, no type
2998   \forest@process@right@bottompop\forest@temp
2999   \forest@process@left@letappend\forest@temp
3000 }
3001 \forest@def@processor@n@{1}*{}{% numexpr
3002   \forest@process@right@bottompop\forest@temp
3003   \forest@process@left@esetappend{\number\numexpr\forest@temp}%
3004   \let\forestmathresulttype\forestmathtype@count
3005 }
3006 \forest@def@processor@d@{1}*{}{% dimexpr
3007   \forest@process@right@bottompop\forest@temp
3008   \forest@process@left@esetappend{\the\dimexpr\forest@temp}%
3009   \let\forestmathresulttype\forestmathtype@dimen
3010 }
3011 \forest@def@processor@P@{1}*{}{% pgfmath expression
3012   \forest@process@right@bottompop\forest@temp
3013   \pgfmathparse{\forest@temp}%
3014   \forest@process@left@letappend\pgfmathresult
3015   \let\forestmathresulttype\forestmathtype@unitless
3016 }
3017 \forest@def@processor@t@{1}*{}{% text
3018   \forest@process@right@bottompop\forest@temp
3019   \forest@process@left@letappend\forest@temp
3020   \let\forestmathresulttype\forestmathtype@textasc
3021 }
3022 \forest@def@processor@-@{}{}{% toggle ascending/descending
3023   \forest@process@left@toppop\forestmathresult
3024   \csname forest@processor@-@\forestmathresulttype\endcsname
3025   \forest@process@left@letappend\forestmathresult
3026 }
3027 \cslet{forest@processor@-@\forestmathtype@generic}\relax
3028 \csdef{forest@processor@-@\forestmathtype@count}{%
3029   \forestmathadd{\forestmathzero}{-\forestmathresult}}
3030 \csletcs{forest@processor@-@\forestmathtype@dimen}
3031   {forest@processor@-@\forestmathtype@count}
3032 \csletcs{forest@processor@-@\forestmathtype@unitless}
3033   {forest@processor@-@\forestmathtype@count}
3034 \csdef{forest@processor@-@\forestmathtype@textasc}{%
3035   \let\forestmathresulttype\forestmathtype@textdesc}
3036 \csdef{forest@processor@-@\forestmathtype@textdesc}{%
3037   \let\forestmathresulttype\forestmathtype@textasc}
3038
3039 \forest@def@processor@c@{}{}{% to lowercase
3040   \forest@process@right@bottompop\forest@temp
3041   \expandafter\lowercase\expandafter{\expandafter\def\expandafter\forest@temp\expandafter{\forest@temp}}%
3042   \forest@process@left@letappend\forest@temp
3043 }
3044 \forest@def@processor@C@{}{}{% to uppercase
3045   \forest@process@right@bottompop\forest@temp
3046   \expandafter\uppercase\expandafter{\expandafter\def\expandafter\forest@temp\expandafter{\forest@temp}}%

```

```

3047 \forest@process@left@letappend\forest@temp
3048 }

Expansions:
3049 \forest@def@processor{x}{-}{-}{% expand
3050 \forest@process@right@bottompop\forest@temp
3051 \forest@process@left@setappend{\forest@temp}%
3052 \let\forestmathresulttype\forestmathtype@generic
3053 }
3054 \forest@def@processor{o}{1}{-}{% expand once (actually, \forest@count@n times)
3055 \forest@process@right@bottompop\forest@temp
3056 \forest@repeat@n@times{\forest@process@n}{%
3057 \expandafter\expandafter\expandafter\def
3058 \expandafter\expandafter\expandafter\forest@temp
3059 \expandafter\expandafter\expandafter{\forest@temp}%
3060 }%
3061 \expandafter\forest@process@left@setappend\expandafter{\forest@temp}%
3062 \let\forestmathresulttype\forestmathtype@generic
3063 }

Access to FOREST data.
3064 \forest@def@processor{0}{1}*{-}{% option
3065 \forest@process@right@bottompop\forest@temp
3066 \expandafter\forestRNO@getvalueandtype\expandafter{\forest@temp}\forest@tempvalue\forest@temp@type
3067 \let\forestmathresulttype\forest@temp@type
3068 \forest@process@left@letappend\forest@tempvalue
3069 }
3070 \forest@def@processor{R}{1}*{-}{% register
3071 \forest@process@right@bottompop\forest@temp
3072 \forestRget{\forest@temp}\forest@tempvalue
3073 \forest@process@left@letappend\forest@tempvalue
3074 \pgfkeysgetvalue{/forest/\forest@temp/@type}\forest@temp@type
3075 \let\forestmathresulttype\forest@temp@type
3076 }

The following processors muck about with the argument / result list.
3077 \forest@def@processor{+}{1}*{-}{% join processors = pop one from result
3078 \forest@process@left@toppop\forest@temp
3079 \forest@process@right@letprepend\forest@temp
3080 }
3081 \forest@def@processor{u}{-}{-}{% ungroup: remove braces and leave in the argument list
3082 \forest@process@right@bottompop\forest@temp
3083 \forest@temparray@clear
3084 \let\forestmathresulttype\forestmathtype@generic
3085 \let\forest@topextend@next\forest@processor@u@
3086 \expandafter\forest@temparray@topextend\forest@temp\forest@eov
3087 }
3088 \def\forest@processor@u@{%
3089 \forest@loop
3090 \ifnum\forest@temparray@N>0
3091 \forest@temparray@toppop\forest@temp
3092 \expandafter\forest@process@right@setprepend\expandafter{\expandafter{\forest@temp}}%
3093 \forest@repeat
3094 }
3095 \def\forest@process@check@mn#1#2#3#4{%
3096 % #1 = processor, #2 = given n, #3/#4 = lower/upper bound (inclusive)
3097 \ifnum#3>#2\relax
3098 \forest@process@check@n@error{#1}{#2}{#3<=}{<=#4}%
3099 \else
3100 \ifnum#4<#2\relax
3101 \forest@process@check@n@error{#1}{#2}{#3<=}{<=#4}%
3102 \fi
3103 \fi

```

```

3104 }
3105 \def\forest@process@check@m#1#2#3{%
3106   % #1 = processor, #2 = given n, #3 = lower bound (inclusive)
3107   \ifnum#2<#3\relax
3108     \forest@process@check@n@error{#1}{#2}{#3<=}{}%
3109   \fi
3110 }
3111 \def\forest@process@check@n@error#1#2#3#4{%
3112   \PackageError{forest}{'.process' instruction '#1' requires a numeric modifier #3n#4, but n="#2" was given.}
3113 }
3114 \newif\ifforest@process@W
3115 \forest@def@processor{w}{1}{1}{% consuming wrap: first test 1<=#1<=9
3116   \forest@process@Wtrue
3117   \forest@process@check@mn{w}{0\the\forest@process@n}{1}{9}%
3118   \expandafter\forest@processor@wW@{\the\forest@process@n}%
3119 }
3120 \forest@def@processor{W}{1}{1}{% nonconsuming wrap: first test 1<=#1<=9
3121   \forest@process@Wfalse
3122   \forest@process@check@mn{W}{0\the\forest@process@n}{1}{9}%
3123   \expandafter\forest@processor@wW@{\the\forest@process@n}%
3124 }
3125 \def\forest@processor@wW@#1{%
3126   \forest@process@left@checkindex{\forest@process@left@N-#1}%
3127   \edef\forest@marshal{%
3128     \edef\noexpand\forest@temp@args{%
3129       \noexpand\forest@process@left@valuesfromrange
3130       {\number\numexpr\forest@process@left@N-#1}%
3131       {\the\forest@process@left@N}%
3132     }%
3133   }\forest@marshal
3134   \ifforest@process@W
3135     \advance\forest@process@left@N-#1\relax
3136   \fi
3137   \forest@process@right@bottompop\forest@temp@macrobody
3138   \expandafter\forest@def@n\expandafter\forest@process@temp@macro\expandafter{\expandafter#1\expandafter}\exp
3139   \expandafter\expandafter\expandafter\forest@process@left@setappend\expandafter\expandafter\expandafter{\exp
3140   \let\forestmathresulttype\forestmathtype@generic
3141 }
3142 \def\forest@def@n#1#2{\csname forest@def@n@#2\endcsname#1}
3143 \csdef{forest@def@n@1}{#1}{\def#1##1}
3144 \csdef{forest@def@n@2}{#1}{\def#1##1##2}
3145 \csdef{forest@def@n@3}{#1}{\def#1##1##2##3}
3146 \csdef{forest@def@n@4}{#1}{\def#1##1##2##3##4}
3147 \csdef{forest@def@n@5}{#1}{\def#1##1##2##3##4##5}
3148 \csdef{forest@def@n@6}{#1}{\def#1##1##2##3##4##5##6}
3149 \csdef{forest@def@n@7}{#1}{\def#1##1##2##3##4##5##6##7}
3150 \csdef{forest@def@n@8}{#1}{\def#1##1##2##3##4##5##6##7##8}
3151 \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.

```

3152 \forest@def@processor{s}{1}{1}{%
3153   \forest@temptrue   % delete the originals
3154   \expandafter\forest@processor@save\expandafter{%
3155     \the\numexpr\forest@process@left@N-\forest@process@n}}
3156 \forest@def@processor{S}{1}{1}{%
3157   \forest@tempfalse  % keep the originals
3158   \expandafter\forest@processor@save\expandafter{%
3159     \the\numexpr\forest@process@left@N-\forest@process@n}}
3160 \def\forest@processor@save#1{%
3161   \forest@process@left@checkindex{#1}%

```

```

3162 \forest@temp@count#1
3163 \forest@loop
3164 \ifnum\forest@temp@count<\forest@process@left@N\relax
3165   \forest@process@left@get@{\the\forest@temp@count}\forest@temp
3166   \forest@process@savetappend\forest@temp
3167   \advance\forest@temp@count+1
3168 \forest@repeat
3169 \let\forest@process@savetype\forestmathresulttype
3170 \ifforest@temp
3171   \forest@process@left@N=#1
3172 \fi
3173 }

```

Load n arguments from the end of the special place to the left side. If $n = 0$, load the entire special place. 1 deletes the args from the special place, L keeps them there as well.

```

3174 \forest@def@processor{1}{0}{}%
3175   \forest@temptrue
3176   \forest@processor@U@@
3177 }
3178 \forest@def@processor{L}{0}{}%
3179   \forest@tempfalse
3180   \forest@processor@U@@
3181 }
3182
3183 \def\forest@processor@U@@{%
3184   \ifnum\forest@process@n=0
3185     \forest@process@n\forest@process@savet@N\relax
3186   \fi
3187   \expandafter\forest@processor@U@@@%
3188   \the\numexpr\forest@process@savet@N-\forest@temp@count}%
3189 }
3190 \def\forest@processor@U@@@#1{%
3191   \forest@temp@count#1
3192   \forest@loop
3193   \ifnum\forest@temp@count<\forest@process@savet@N\relax
3194     \forest@process@savet@get@{\the\forest@temp@count}\forest@temp
3195     \forest@process@left@letappend\forest@temp
3196     \advance\forest@temp@count1
3197   \forest@repeat
3198   \let\forestmathresulttype\forest@process@savetype
3199   \ifforest@temp
3200     \let\forest@process@savetype\forest@result@type@any
3201     \forest@process@savet@N#1
3202   \fi
3203 }

```

Boolean operations:

```

3204 \forest@def@processor{&}{2}{}% and
3205   \def\forest@tempa{1}%
3206   \forest@repeat@n@times{\forest@process@n}{%
3207     \forest@process@left@bottompop\forest@tempb
3208     \edef\forest@tempa{\ifnum10<\forest@tempa\forest@tempb\space 1\else0\fi}%
3209   }%
3210   \forest@process@left@setappend{\forest@tempa}%
3211   \let\forestmathresulttype\forestmathtype@count
3212 }
3213 \forest@def@processor{||}{2}{}% or
3214   \def\forest@tempa{0}%
3215   \forest@repeat@n@times{\forest@process@n}{%
3216     \forest@process@left@bottompop\forest@tempb
3217     \edef\forest@tempa{\ifnum0=\forest@tempa\forest@tempb\space 0\else1\fi}%
3218   }%

```

```

3219 \forest@process@left@esetappend{\forest@tempa}%
3220 \let\forestmathresulttype\forestmathtype@count
3221 }
3222 \forest@def@processor{!}{-}{-}{% not
3223 \forest@process@left@toppop\forest@temp
3224 \forest@process@left@esetappend{\ifnum0=\forest@temp\space 1\else0\fi}%
3225 \let\forestmathresulttype\forestmathtype@count
3226 }
3227 \forest@def@processor{?}{-}{-}{%
3228 \forest@process@left@toppop\forest@temp
3229 \forest@process@right@bottompop\forest@tempa
3230 \forest@process@right@bottompop\forest@tempb
3231 \ifnum\forest@temp=0
3232 \forest@process@right@letprepend\forest@tempb
3233 \else
3234 \forest@process@right@letprepend\forest@tempa
3235 \fi
3236 \let\forestmathresulttype\forestmathtype@generic
3237 }

```

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

```

3238 \forest@def@processor{=}{-}{-}{%
3239 \forest@process@left@toppop\forest@tempa
3240 \forest@process@left@toppop\forest@tempb
3241 \forest@process@left@esetappend{\ifx\forest@tempa\forest@tempb 1\else0\fi}%
3242 \let\forestmathresulttype\forestmathtype@count
3243 }
3244 \forest@def@processor{<}{-}{-}{%
3245 \forest@process@left@toppop\forest@tempb
3246 \forest@process@left@toppop\forest@tempa
3247 \ifx\forestmathresulttype\forestmathtype@generic
3248 \forest@cmp@error\forest@tempa\forest@tempb
3249 \else
3250 \forestmathlt{\forest@tempa}{\forest@tempb}%
3251 \forest@process@left@esetappend{\forestmathresult}%
3252 \fi
3253 }
3254 \forest@def@processor{>}{-}{-}{%
3255 \forest@process@left@toppop\forest@tempb
3256 \forest@process@left@toppop\forest@tempa
3257 \ifx\forestmathresulttype\forestmathtype@generic
3258 \forest@cmp@error\forest@tempa\forest@tempb
3259 \else
3260 \forestmathgt{\forest@tempa}{\forest@tempb}%
3261 \forest@process@left@esetappend{\forestmathresult}%
3262 \fi
3263 }

```

Various.

```

3264 \forest@def@processor{r}{-}{-}{% reverse keylist
3265 \forest@process@right@bottompop\forest@temp
3266 \expandafter\forest@processor@r@\expandafter{\forest@temp}%
3267 }
3268 \def\forest@processor@r@#1{%
3269 \forest@temp@toks{}%
3270 \def\forest@tempcomma{}%
3271 \pgfqkeys{/forest}{split={#1}{,}{process@rk}}%
3272 \forest@process@left@esetappend{\the\forest@temp@toks}%
3273 \let\forestmathresulttype\forestmathtype@generic
3274 }
3275 \forestset{%

```

```

3276 process@rk/.code={%
3277   \epretotoks\forest@temp@toks{#1\forest@tempcomma}%
3278   \def\forest@tempcomma{,}%
3279 }%
3280 }

```

7.1.1 Registers

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

```

3281 \def\forest@pgfmathhelper@register@toks#1#2{% #1 is discarded: it is present only for analogy with options
3282   \forestrget{#2}\pgfmathresult
3283 }
3284 \def\forest@pgfmathhelper@register@dimen#1#2{%
3285   \forestrget{#2}\forest@temp
3286   \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
3287 }
3288 \def\forest@pgfmathhelper@register@count#1#2{%
3289   \forestrget{#2}\pgfmathresult
3290 }
3291 \def\forest@declareregisterhandler#1#2{%#1=handler for specific type,#2=option name
3292   \pgfkeyssetvalue{/forest/#2/node@or@reg}{}% empty = register (node id=node)
3293   \forest@convert@others@to@underscores{#2}\forest@pgfmathoptionname
3294   \edef\forest@marshal{%
3295     \noexpand#1{/forest/#2}{/forest}{#2}{\forest@pgfmathoptionname}%
3296   }\forest@marshal
3297 }
3298 \def\forest@declaretoksregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
3299   \forest@declaretoksregister@handler@A{#1}{#2}{#3}{#4}{}%
3300 }
3301 \def\forest@declarekeylistregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
3302   \forest@declaretoksregister@handler@A{#1}{#2}{#3}{#4}{,}%
3303   \forest@copycommandkey{#1}{#1'}%
3304   \pgfkeyssetvalue{#1'/option@name}{#3}%
3305   \forest@copycommandkey{#1+}{#1}%
3306   \pgfkeysalso{#1-/.code={%
3307     \forest@fornode{register}{%
3308       \forest@node@removekeysfromkeylist{##1}{#3}%
3309     }}%
3310   \pgfkeyssetvalue{#1-/.option@name}{#3}%
3311 }
3312 \def\forest@declaretoksregister@handler@A#1#2#3#4#5{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
3313   \pgfkeysalso{%
3314     #1/.code={\forestset{#3}{##1}},
3315     #2/if #3/.code n args={3}{%
3316       \forestrget{#3}\forest@temp@option@value
3317       \edef\forest@temp@compared@value{\unexpanded{##1}}%
3318       \ifx\forest@temp@option@value\forest@temp@compared@value
3319         \pgfkeysalso{##2}%
3320       \else
3321         \pgfkeysalso{##3}%
3322       \fi
3323     },
3324     #2/if in #3/.code n args={3}{%
3325       \forestrget{#3}\forest@temp@option@value
3326       \edef\forest@temp@compared@value{\unexpanded{##1}}%
3327       \expandafter\expandafter\expandafter\pgfutil@in@\expandafter\expandafter\expandafter{\expandafter\forest@
3328       \ifpgfutil@in@
3329         \pgfkeysalso{##2}%
3330       \else
3331         \pgfkeysalso{##3}%
3332       \fi

```

```

3333 },
3334 }%
3335 \ifstrempy{#5}{%
3336   \pgfkeysalso{%
3337     #1+/.code={\forestrappto{#3}{#5##1}},
3338     #2/+ #3/.code={\forestrpreto{#3}{##1#5}},
3339   }%
3340 }{%
3341   \pgfkeysalso{%
3342     #1+/.code={%
3343       \forestrget{#3}\forest@temp
3344       \ifdefempty{\forest@temp}{%
3345         \forestrset{#3}{##1}%
3346       }{%
3347         \forestrappto{#3}{#5##1}%
3348       }%
3349     },
3350     #2/+ #3/.code={%
3351       \forestrget{#3}\forest@temp
3352       \ifdefempty{\forest@temp}{%
3353         \forestrset{#3}{##1}%
3354       }{%
3355         \forestrpreto{#3}{##1#5}%
3356       }%
3357     }%
3358   }%
3359 }%
3360 \pgfkeyssetvalue{#1/option@name}{#3}%
3361 \pgfkeyssetvalue{#1+/option@name}{#3}%
3362 \pgfkeyssetvalue{#2/+ #3/option@name}{#3}%
3363 \pgfkeyslet{#1/@type}\forestmathtype@generic % for .process & co
3364 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@toks{##1}{#3}}%
3365 }
3366 \def\forest@declareautowrappedtoksregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
3367   \forest@declaretoksregister@handler{#1}{#2}{#3}{#4}%
3368   \forest@copycommandkey{#1}{#1'}%
3369   \pgfkeysalso{#1/.style={#1'/.wrap value={##1}}}%
3370   \pgfkeyssetvalue{#1'/option@name}{#3}%
3371   \forest@copycommandkey{#1+}{#1+'}%
3372   \pgfkeysalso{#1+/.style={#1+'/.wrap value={##1}}}%
3373   \pgfkeyssetvalue{#1+'/option@name}{#3}%
3374   \forest@copycommandkey{#2/+ #3}{#2/+ #3'}%
3375   \pgfkeysalso{#2/+ #3/.style={#2/+ #3'/.wrap value={##1}}}%
3376   \pgfkeyssetvalue{#2/+ #3'/option@name}{#3}%
3377 }
3378 \def\forest@declarereadonlydimenregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
3379   \pgfkeysalso{%
3380     #2/if #3/.code n args={3}{%
3381       \forestrget{#3}\forest@temp@option@value
3382       \ifdim\forest@temp@option@value>##1\relax
3383         \pgfkeysalso{##2}%
3384       \else
3385         \pgfkeysalso{##3}%
3386       \fi
3387     },
3388     #2/if #3</.code n args={3}{%
3389       \forestrget{#3}\forest@temp@option@value
3390       \ifdim\forest@temp@option@value>##1\relax
3391         \pgfkeysalso{##3}%
3392       \else
3393         \pgfkeysalso{##2}%

```



```

3394     \fi
3395 },
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{##3}%
3400     \else
3401         \pgfkeysalso{##2}%
3402     \fi
3403 },
3404 }%
3405 \pgfkeyslet{#1/@type}\forestmathtype@dimen    % for .process & co
3406 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@dimen{##1}{#3}}%
3407 }
3408 \def\forest@declaredimenregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
3409     \forest@declaredimenregister@handler{#1}{#2}{#3}{#4}%
3410     \pgfkeysalso{%
3411         #1/.code={%
3412             \forestmathsetlengthmacro\forest@temp{##1}%
3413             \forestrlet{#3}\forest@temp
3414         },
3415         #1+/.code={%
3416             \forestmathsetlengthmacro\forest@temp{##1}%
3417             \pgfutil@tempdima=\forestrve{#3}
3418             \advance\pgfutil@tempdima\forest@temp\relax
3419             \forestreset{#3}{\the\pgfutil@tempdima}%
3420         },
3421         #1-/.code={%
3422             \forestmathsetlengthmacro\forest@temp{##1}%
3423             \pgfutil@tempdima=\forestrve{#3}
3424             \advance\pgfutil@tempdima-\forest@temp\relax
3425             \forestreset{#3}{\the\pgfutil@tempdima}%
3426         },
3427         #1*/.style={%
3428             #1={#4()}*(##1)}%
3429         },
3430         #1:/ .style={%
3431             #1={#4()}/(##1)}%
3432         },
3433         #1'/.code={%
3434             \pgfutil@tempdima=##1\relax
3435             \forestreset{#3}{\the\pgfutil@tempdima}%
3436         },
3437         #1'+/.code={%
3438             \pgfutil@tempdima=\forestrve{#3}\relax
3439             \advance\pgfutil@tempdima##1\relax
3440             \forestreset{#3}{\the\pgfutil@tempdima}%
3441         },
3442         #1'-/.code={%
3443             \pgfutil@tempdima=\forestrve{#3}\relax
3444             \advance\pgfutil@tempdima-##1\relax
3445             \forestreset{#3}{\the\pgfutil@tempdima}%
3446         },
3447         #1'*/.style={%
3448             \pgfutil@tempdima=\forestrve{#3}\relax
3449             \multiply\pgfutil@tempdima##1\relax
3450             \forestreset{#3}{\the\pgfutil@tempdima}%
3451         },
3452         #1':/.style={%
3453             \pgfutil@tempdima=\forestrve{#3}\relax
3454             \divide\pgfutil@tempdima##1\relax

```

```

3455     \forestreset{#3}{\the\pgfutil@tempdima}%
3456   },
3457 }%
3458 \pgfkeyssetvalue{#1/option@name}{#3}%
3459 \pgfkeyssetvalue{#1+/option@name}{#3}%
3460 \pgfkeyssetvalue{#1-/option@name}{#3}%
3461 \pgfkeyssetvalue{#1*/option@name}{#3}%
3462 \pgfkeyssetvalue{#1:/option@name}{#3}%
3463 \pgfkeyssetvalue{#1'/option@name}{#3}%
3464 \pgfkeyssetvalue{#1'+/option@name}{#3}%
3465 \pgfkeyssetvalue{#1'-/option@name}{#3}%
3466 \pgfkeyssetvalue{#1*/option@name}{#3}%
3467 \pgfkeyssetvalue{#1':/option@name}{#3}%
3468 }
3469 \def\forest@declarereadonlycountregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
3470   \pgfkeysalso{
3471     #2/if #3/.code n args={3}{%
3472       \foreststrget{#3}\forest@temp@option@value
3473       \ifnum\forest@temp@option@value=##1\relax
3474         \pgfkeysalso{##2}%
3475       \else
3476         \pgfkeysalso{##3}%
3477       \fi
3478     },
3479     #2/if #3</.code n args={3}{%
3480       \foreststrget{#3}\forest@temp@option@value
3481       \ifnum\forest@temp@option@value>##1\relax
3482         \pgfkeysalso{##3}%
3483       \else
3484         \pgfkeysalso{##2}%
3485       \fi
3486     },
3487     #2/if #3>/.code n args={3}{%
3488       \foreststrget{#3}\forest@temp@option@value
3489       \ifnum\forest@temp@option@value<##1\relax
3490         \pgfkeysalso{##3}%
3491       \else
3492         \pgfkeysalso{##2}%
3493       \fi
3494     },
3495   }%
3496   \pgfkeyslet{#1/@type}\forestmathtype@count % for .process & co
3497   \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@count{##1}{#3}}%
3498 }
3499 \def\forest@declarecountregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
3500   \forest@declarereadonlycountregister@handler{#1}{#2}{#3}{#4}%
3501   \pgfkeysalso{
3502     #1/.code={%
3503       \forestmathtruncatemacro\forest@temp{##1}%
3504       \forestrllet{#3}\forest@temp
3505     },
3506     #1+/.code={%
3507       \forestmathtruncatemacro\forest@temp{##1}%
3508       \c@pgf@counta=\forestrive{#3}\relax
3509       \advance\c@pgf@counta\forest@temp\relax
3510       \forestreset{#3}{\the\c@pgf@counta}%
3511     },
3512     #1-/.code={%
3513       \forestmathtruncatemacro\forest@temp{##1}%
3514       \c@pgf@counta=\forestrive{#3}\relax
3515       \advance\c@pgf@counta-\forest@temp\relax

```

```

3516     \forestreset{#3}{\the\c@pgf@counta}%
3517 },
3518 #1*/.code={%
3519     \forestmathtruncatemacro\forest@temp{##1}%
3520     \c@pgf@counta=\forestrv{#3}\relax
3521     \multiply\c@pgf@counta\forest@temp\relax
3522     \forestreset{#3}{\the\c@pgf@counta}%
3523 },
3524 #1:/.code={%
3525     \forestmathtruncatemacro\forest@temp{##1}%
3526     \c@pgf@counta=\forestrv{#3}\relax
3527     \divide\c@pgf@counta\forest@temp\relax
3528     \forestreset{#3}{\the\c@pgf@counta}%
3529 },
3530 #1'/.code={%
3531     \c@pgf@counta=##1\relax
3532     \forestreset{#3}{\the\c@pgf@counta}%
3533 },
3534 #1'+/.code={%
3535     \c@pgf@counta=\forestrv{#3}\relax
3536     \advance\c@pgf@counta##1\relax
3537     \forestreset{#3}{\the\c@pgf@counta}%
3538 },
3539 #1'-/.code={%
3540     \c@pgf@counta=\forestrv{#3}\relax
3541     \advance\c@pgf@counta-##1\relax
3542     \forestreset{#3}{\the\c@pgf@counta}%
3543 },
3544 #1'*/.style={%
3545     \c@pgf@counta=\forestrv{#3}\relax
3546     \multiply\c@pgf@counta##1\relax
3547     \forestreset{#3}{\the\c@pgf@counta}%
3548 },
3549 #1':/.style={%
3550     \c@pgf@counta=\forestrv{#3}\relax
3551     \divide\c@pgf@counta##1\relax
3552     \forestreset{#3}{\the\c@pgf@counta}%
3553 },
3554 }%
3555 \pgfkeyssetvalue{#1/option@name}{#3}%
3556 \pgfkeyssetvalue{#1+/option@name}{#3}%
3557 \pgfkeyssetvalue{#1-/option@name}{#3}%
3558 \pgfkeyssetvalue{#1*/option@name}{#3}%
3559 \pgfkeyssetvalue{#1:/option@name}{#3}%
3560 \pgfkeyssetvalue{#1'/option@name}{#3}%
3561 \pgfkeyssetvalue{#1'+/option@name}{#3}%
3562 \pgfkeyssetvalue{#1'-/option@name}{#3}%
3563 \pgfkeyssetvalue{#1'*/option@name}{#3}%
3564 \pgfkeyssetvalue{#1':/option@name}{#3}%
3565 }
3566 \def\forest@declarebooleanregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
3567     \pgfkeysalso{%
3568         #1/.code={%
3569             \ifcsdef{forest@bh@detokenize{##1}}{%
3570                 \letcs\forest@temp{forest@bh@detokenize{##1}}%
3571             }{%
3572                 \forestmathtruncatemacro\forest@temp{##1}%
3573                 \ifx\forest@temp0\else\def\forest@temp{1}\fi
3574             }%
3575             \forestrl{#3}\forest@temp
3576         },

```

```

3577 #1/.default=1,
3578 #2/not #3/.code={\forestset{#3}{0}},
3579 #2/if #3/.code 2 args={%
3580   \forestget{#3}\forest@temp@option@value
3581   \ifnum\forest@temp@option@value=1
3582     \pgfkeysalso{##1}%
3583   \else
3584     \pgfkeysalso{##2}%
3585   \fi
3586 },
3587 }%
3588 \pgfkeyssetvalue{#1/option@name}{#3}%
3589 \pgfkeyslet{#1/@type}\forestmathtype@count % for .process & co
3590 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@count{##1}{#3}}%
3591 }
3592 \forestset{
3593   declare toks register/.code={%
3594     \forest@declareregisterhandler\forest@declaretoksregister@handler{#1}%
3595     \forestset{#1={}}%
3596   },
3597   declare autowrapped toks register/.code={%
3598     \forest@declareregisterhandler\forest@declareautowrappedtoksregister@handler{#1}%
3599     \forestset{#1={}}%
3600   },
3601   declare keylist register/.code={%
3602     \forest@declareregisterhandler\forest@declarekeylistregister@handler{#1}%
3603     \forestset{#1'={}}%
3604   },
3605   declare dimen register/.code={%
3606     \forest@declareregisterhandler\forest@declaredimenregister@handler{#1}%
3607     \forestset{#1'={0pt}}%
3608   },
3609   declare count register/.code={%
3610     \forest@declareregisterhandler\forest@declarecountregister@handler{#1}%
3611     \forestset{#1'={0}}%
3612   },
3613   declare boolean register/.code={%
3614     \forest@declareregisterhandler\forest@declarebooleanregister@handler{#1}%
3615     \forestset{#1={0}}%
3616   },
3617 }

```

Declare some temporary registers.

```

3618 \forestset{
3619   declare toks register=temptoksa,temptoksa={},
3620   declare toks register=temptoksb,temptoksb={},
3621   declare toks register=temptoksc,temptoksc={},
3622   declare toks register=temptoksd,temptoksd={},
3623   declare keylist register=tempkeylista,tempkeylista'={},
3624   declare keylist register=tempkeylistb,tempkeylistb'={},
3625   declare keylist register=tempkeylistc,tempkeylistc'={},
3626   declare keylist register=tempkeylistd,tempkeylistd'={},
3627   declare dimen register=tempdima,tempdima'={0pt},
3628   declare dimen register=tempdimb,tempdimb'={0pt},
3629   declare dimen register=tempdimc,tempdimc'={0pt},
3630   declare dimen register=tempdimd,tempdimd'={0pt},
3631   declare dimen register=tempdimx,tempdimx'={0pt},
3632   declare dimen register=tempdimxa,tempdimxa'={0pt},
3633   declare dimen register=tempdimxb,tempdimxb'={0pt},
3634   declare dimen register=tempdimy,tempdimy'={0pt},
3635   declare dimen register=tempdimya,tempdimya'={0pt},

```

```

3636 declare dimen register=tempdimyb,tempdimyb'={Opt},
3637 declare dimen register=tempdiml,tempdiml'={Opt},
3638 declare dimen register=tempdimla,tempdimla'={Opt},
3639 declare dimen register=tempdimlb,tempdimlb'={Opt},
3640 declare dimen register=tempdims,tempdims'={Opt},
3641 declare dimen register=tempdimsa,tempdimsa'={Opt},
3642 declare dimen register=tempdimsb,tempdimsb'={Opt},
3643 declare count register=tempcounta,tempcounta'={0},
3644 declare count register=tempcountb,tempcountb'={0},
3645 declare count register=tempcountc,tempcountc'={0},
3646 declare count register=tempcountd,tempcountd'={0},
3647 declare boolean register=tempboola,tempboola={0},
3648 declare boolean register=tempboolb,tempboolb={0},
3649 declare boolean register=tempboolc,tempboolc={0},
3650 declare boolean register=tempboold,tempboold={0},
3651 }

```

7.1.2 Declaring options

```

3652 \def\forest@node@Nametoid#1{% #1 = name
3653   \csname forest@id@of@#1\endcsname
3654 }
3655 \def\forest@node@ifnamedefined#1#2#3{% #1 = name, #2=true,#3=false
3656   \ifcsvoid{forest@id@of@#1}{#3}{#2}%
3657 }
3658 \def\forest@node@setname#1{%
3659   \def\forest@temp@setname{y}%
3660   \def\forest@temp@silent{n}%
3661   \def\forest@temp@propagating{n}%
3662   \forest@node@setnameoralias{#1}%
3663 }
3664 \def\forest@node@setname@silent#1{%
3665   \def\forest@temp@setname{y}%
3666   \def\forest@temp@silent{y}%
3667   \def\forest@temp@propagating{n}%
3668   \forest@node@setnameoralias{#1}%
3669 }
3670 \def\forest@node@setalias#1{%
3671   \def\forest@temp@setname{n}%
3672   \def\forest@temp@silent{n}%
3673   \def\forest@temp@propagating{n}%
3674   \forest@node@setnameoralias{#1}%
3675 }
3676 \def\forest@node@setalias@silent#1{%
3677   \def\forest@temp@setname{n}%
3678   \def\forest@temp@silent{y}%
3679   \def\forest@temp@propagating{n}%
3680   \forest@node@setnameoralias{#1}%
3681 }
3682 \def\forest@node@setnameoralias#1{%
3683   \ifstrepty{#1}{%
3684     \forest@node@setnameoralias{node@\forest@cn}%
3685   }{%
3686     \forest@node@ifnamedefined{#1}{%
3687       \if y\forest@temp@propagating
3688         % this will find a unique name, eventually:
3689         \@escapeif{\forest@node@setnameoralias{#1@\forest@cn}}%
3690       \else\@escapeif{%
3691         \if y\forest@temp@setname
3692           \edef\forest@marshal{%
3693             \ifstrequal{\forestove{name}}{#1}%

```

```

3694     }\forest@marshal{%
3695     % same name, no problem
3696     }{%
3697     \@escapeif{\forest@node@setnameoralias@nameclash{#1}}%
3698     }%
3699     \else\@escapeif{% setting an alias: clashing with alias is not a problem
3700     \forest@get{\forest@node@Nametoid{#1}}{name}\forest@temp
3701     \expandafter\ifstrequal\expandafter{\forest@temp}{#1}{%
3702     \forest@node@setnameoralias@nameclash{#1}%
3703     }{%
3704     \forest@node@setnameoralias@do{#1}%
3705     }%
3706     }\fi
3707     }\fi
3708     }{%
3709     \forest@node@setnameoralias@do{#1}%
3710     }%
3711     }%
3712 }
3713 \def\forest@node@setnameoralias@nameclash#1{%
3714     \if y\forest@temp@silent
3715     \forest@fornode{\forest@node@Nametoid{#1}}{%
3716     \def\forest@temp@propagating{y}%
3717     \forest@node@setnameoralias{}%
3718     }%
3719     \forest@node@setnameoralias@do{#1}%
3720     \else
3721     \PackageError{forest}{Node name "#1" is already used}{}%
3722     \fi
3723 }
3724 \def\forest@node@setnameoralias@do#1{%
3725     \if y\forest@temp@setname
3726     \csdef{forest@id@of@\forest@ve{name}}{}%
3727     \forest@oeset{name}{#1}%
3728     \fi
3729     \csedef{forest@id@of@#1}{\forest@cn}%
3730 }
3731 \forestset{
3732     TeX/.code={#1},
3733     TeX'/.code={\appto\forest@externalize@loadimages{#1}#1},
3734     TeX''/.code={\appto\forest@externalize@loadimages{#1}},
3735     options/.code={\forestset{#1}},
3736     typeout/.style={TeX={\typeout{#1}}},
3737     declare toks={name}{},
3738     name/.code={% override the default setter
3739     \forest@fornode{\forest@setter@node}{\forest@node@setname{#1}}%
3740     },
3741     name/.default={},
3742     name'/.code={% override the default setter
3743     \forest@fornode{\forest@setter@node}{\forest@node@setname@silent{#1}}%
3744     },
3745     name'/.default={},
3746     alias/.code={\forest@fornode{\forest@setter@node}{\forest@node@setalias{#1}}},
3747     alias'/.code={\forest@fornode{\forest@setter@node}{\forest@node@setalias@silent{#1}}},
3748     begin draw/.code={\begin{tikzpicture}},
3749     end draw/.code={\end{tikzpicture}},
3750     declare keylist register=default preamble,
3751     default preamble'={},
3752     declare keylist register=preamble,
3753     preamble'={},
3754     declare autowrapped toks={content}{},

```

```

3755 % #1 = which option to split, #2 = separator (one char!), #3 = receiving options
3756 split option/.code n args=3{%
3757   \forestRNOget{#1}\forest@temp
3758   \edef\forest@marshal{%
3759     \noexpand\pgfkeysalso{split={\expandonce{\forest@temp}}\unexpanded{{#2}{#3}}}%
3760   }\forest@marshal
3761 },
3762 split register/.code n args=3{% #1 = which register to split, #2 = separator (one char!), #3 = receiving options
3763   \forestRget{#1}\forest@temp
3764   \edef\forest@marshal{%
3765     \noexpand\pgfkeysalso{split={\expandonce{\forest@temp}}\unexpanded{{#2}{#3}}}%
3766   }\forest@marshal
3767 },
3768 TeX={%
3769   \def\forest@split@sourcevalues{%
3770     \def\forest@split@sourcevalue{%
3771       \def\forest@split@receivingoptions{%
3772         \def\forest@split@receivingoption{%
3773           },
3774         split/.code n args=3{% #1 = string to split, #2 = separator (one char!), #3 = receiving options
3775           \forest@saveandrestoremacro\forest@split@sourcevalues{%
3776             \forest@saveandrestoremacro\forest@split@sourcevalue{%
3777               \forest@saveandrestoremacro\forest@split@receivingoptions{%
3778                 \forest@saveandrestoremacro\forest@split@receivingoption{%
3779                   \def\forest@split@sourcevalues{#1#2}%
3780                   \edef\forest@split@receivingoptions{#3,%
3781                     \def\forest@split@receivingoption{%
3782                       \safeloop
3783                         \expandafter\forest@split\expandafter{\forest@split@sourcevalues}{#2}\forest@split@sourcevalue\
3784                         \ifdefempty\forest@split@receivingoptions{}%
3785                         \expandafter\forest@split\expandafter{\forest@split@receivingoptions}{,}\forest@temp\forest@s
3786                         \ifdefempty\forest@temp{ }\let\forest@split@receivingoption\forest@temp\def\forest@temp{ }%
3787                       }%
3788                       \edef\forest@marshal{%
3789                         \noexpand\pgfkeysalso{\forest@split@receivingoption={\expandonce{\forest@split@sourcevalue}}%
3790                       }\forest@marshal
3791                       \ifdefempty\forest@split@sourcevalues{\forest@tempfalse}{\forest@temptrue}%
3792                     \ifforest@temp
3793                     \saferepeat
3794                   }}}}%
3795           },
3796           declare count={grow}{270},
3797           TeX={% a hack for grow-reversed connection, and compass-based grow specification
3798             \forest@copycommandkey{/forest/grow}{/forest/grow@}%
3799             %\pgfkeysgetvalue{/forest/grow/.@cmd}\forest@temp
3800             %\pgfkeyslet{/forest/grow@/.@cmd}\forest@temp
3801           },
3802           grow/.style={grow@={#1},reversed=0},
3803           grow'/.style={grow@={#1},reversed=1},
3804           grow''/.style={grow@={#1}},
3805           grow@/.is choice,
3806           grow@/east/.style={/forest/grow@=0},
3807           grow@/north east/.style={/forest/grow@=45},
3808           grow@/north/.style={/forest/grow@=90},
3809           grow@/north west/.style={/forest/grow@=135},
3810           grow@/west/.style={/forest/grow@=180},
3811           grow@/south west/.style={/forest/grow@=225},
3812           grow@/south/.style={/forest/grow@=270},
3813           grow@/south east/.style={/forest/grow@=315},
3814           grow@/.unknown/.code={\let\forest@temp@grow\pgfkeyscurrentname
3815             \pgfkeysalso{/forest/grow@/.expand once=\forest@temp@grow}},

```

```

3816 declare boolean={reversed}{0},
3817 declare toks={parent anchor}{},
3818 declare toks={child anchor}{},
3819 declare toks={anchor}{base},
3820 Autoforward={anchor}{
3821     node options--anchor,
3822     node options+={anchor={##1}}
3823 },
3824 anchor'/.style={anchor@no@compass=true,anchor=#1},
3825 anchor+'/.style={anchor@no@compass=true,anchor+=#1},
3826 anchor-'/.style={anchor@no@compass=true,anchor-=#1},
3827 anchor*'//.style={anchor@no@compass=true,anchor*=#1},
3828 anchor: '/.style={anchor@no@compass=true,anchor:=#1},
3829 anchor+'/.style={anchor@no@compass=true,anchor'+=#1},
3830 anchor-'/.style={anchor@no@compass=true,anchor'-=#1},
3831 anchor*'/.style={anchor@no@compass=true,anchor'*=#1},
3832 anchor:'/.style={anchor@no@compass=true,anchor':=#1},
3833 % /tikz/forest anchor/.style={
3834 %     /forest/TeX={\forestanchortotikzanchor{#1}\forest@temp@anchor},
3835 %     anchor/.expand once=\forest@temp@anchor
3836 % },
3837 declare toks={calign}{midpoint},
3838 TeX={%
3839     \forest@copycommandkey{/forest/calign}{/forest/calign'}%
3840 },
3841 calign/.is choice,
3842 calign/child/.style={calign'=child},
3843 calign/first/.style={calign'=child,calign primary child=1},
3844 calign/last/.style={calign'=child,calign primary child=-1},
3845 calign with current/.style={for parent/.wrap pgfmath arg={calign=child,calign primary child=##1}{n}},
3846 calign with current edge/.style={for parent/.wrap pgfmath arg={calign=child edge,calign primary child=##1}{-}},
3847 calign/child edge/.style={calign'=child edge},
3848 calign/midpoint/.style={calign'=midpoint},
3849 calign/center/.style={calign'=midpoint,calign primary child=1,calign secondary child=-1},
3850 calign/edge midpoint/.style={calign'=edge midpoint},
3851 calign/fixed angles/.style={calign'=fixed angles},
3852 calign/fixed edge angles/.style={calign'=fixed edge angles},
3853 calign/.unknown/.code={\PackageError{forest}{unknown calign '\pgfkeyscurrentname'}{}}},
3854 declare count={calign primary child}{1},
3855 declare count={calign secondary child}{-1},
3856 declare count={calign primary angle}{-35},
3857 declare count={calign secondary angle}{35},
3858 calign child/.style={calign primary child={#1}},
3859 calign angle/.style={calign primary angle=-#1,calign secondary angle={#1}},
3860 declare toks={tier}{},
3861 declare toks={fit}{tight},
3862 declare boolean={ignore}{0},
3863 declare boolean={ignore edge}{0},
3864 no edge/.style={edge'={},ignore edge},
3865 declare keylist={edge}{draw},
3866 declare toks={edge path}{%
3867     \noexpand\path[\forestoption{edge}]%
3868     (\forestOve{\forestove{@parent}}{name}.parent anchor)--(\forestove{name}.child anchor)
3869     % =
3870     % (!u.parent anchor)--(.child anchor)\forestoption{edge label};
3871     \forestoption{edge label};%
3872 },
3873 edge path'/.style={
3874     edge path={%
3875         \noexpand\path[\forestoption{edge}]%
3876         #1%

```



```

3877     \forestoption{edge label};
3878 }
3879 },
3880 declare toks={edge label}{},
3881 declare boolean={phantom}{0},
3882 baseline/.style={alias={forest@baseline@node}},
3883 declare readonly count={id}{0},
3884 declare readonly count={n}{0},
3885 declare readonly count={n'}{0},
3886 declare readonly count={n children}{-1},
3887 declare readonly count={level}{-1},
3888 declare dimen=x{0pt},
3889 declare dimen=y{0pt},
3890 declare dimen={s}{0pt},
3891 declare dimen={l}{6ex}, % just in case: should be set by the calibration
3892 declare dimen={s sep}{0.6666em},
3893 declare dimen={l sep}{1ex}, % just in case: calibration!
3894 declare keylist={node options}{anchor=base},
3895 declare toks={tikz}{},
3896 afterthought/.style={tikz+=#{1}},
3897 label/.style={tikz+={\path[late options={%
3898     name=\forestoption{name},label=#{1}]]}}},
3899 pin/.style={tikz+={\path[late options={%
3900     name=\forestoption{name},pin=#{1}]]}}},
3901 declare toks={content format}{\forestoption{content}},
3902 plain content/.style={content format={\forestoption{content}}},
3903 math content/.style={content format={\noexpand\ensuremath{\forestoption{content}}}},
3904 declare toks={node format}{%
3905     \noexpand\node
3906     (\forestoption{name})%
3907     [\forestoption{node options}]%
3908     {\forestoption{content format}};%
3909 },
3910 node format'/.style={
3911     node format={\noexpand\node(\forestoption{name})#1;}
3912 },
3913 tabular@environment/.style={content format={%
3914     \noexpand\begin{tabular}[\forestoption{base}]{\forestoption{align}}%
3915     \forestoption{content}%
3916     \noexpand\end{tabular}%
3917 }},
3918 declare toks={align}{},
3919 TeX={%
3920     \forest@copycommandkey{/forest/align}{/forest/align'}%
3921     %\pgfkeysgetvalue{/forest/align/.@cmd}\forest@temp
3922     %\pgfkeyslet{/forest/align'/.@cmd}\forest@temp
3923 },
3924 align/.is choice,
3925 align/.unknown/.code={%
3926     \edef\forest@marshal{%
3927         \noexpand\pgfkeysalso{%
3928             align'={\pgfkeyscurrentname},%
3929             tabular@environment
3930         }%
3931     }\forest@marshal
3932 },
3933 align/center/.style={align'={@{}c@{}}},tabular@environment},
3934 align/left/.style={align'={@{}l@{}}},tabular@environment},
3935 align/right/.style={align'={@{}r@{}}},tabular@environment},
3936 declare toks={base}{t},
3937 TeX={%

```

```

3938 \forest@copycommandkey{/forest/base}{/forest/base'}%
3939 %\pgfkeysgetvalue{/forest/base/.@cmd}\forest@temp
3940 %\pgfkeyslet{/forest/base'/.@cmd}\forest@temp
3941 },
3942 base/.is choice,
3943 base/top/.style={base'=t},
3944 base/bottom/.style={base'=b},
3945 base/.unknown/.style={base'/.expand once=\pgfkeyscurrentname},
3946 unknown to/.store in=\forest@unknownto,
3947 unknown to=node options,
3948 unknown key error/.code={\PackageError{forest}{Unknown keyval: \detokenize{#1}}{}}%,
3949 content to/.store in=\forest@contentto,
3950 content to=content,
3951 .unknown/.code={%
3952 \expandafter\pgfutil@in@\expandafter.\expandafter{\pgfkeyscurrentname}%
3953 \ifpgfutil@in@
3954 \expandafter\forest@relatednode@option@setter\pgfkeyscurrentname=#1\forest@END
3955 \else
3956 \edef\forest@marshal{%
3957 \noexpand\pgfkeysalso{\forest@unknownto={\pgfkeyscurrentname=\unexpanded{#1}}}%
3958 }\forest@marshal
3959 \fi
3960 },
3961 get node boundary/.code={%
3962 \foresttoget{@boundary}\forest@node@boundary
3963 \def#1{%
3964 \forest@extendpath#1\forest@node@boundary{\pgfqpoint{\forestove{x}}{\forestove{y}}}%
3965 },
3966 % get min l tree boundary/.code={%
3967 % \forest@get@tree@boundary{negative}{\the\numexpr\forestove{grow}-90\relax}#1},
3968 % get max l tree boundary/.code={%
3969 % \forest@get@tree@boundary{positive}{\the\numexpr\forestove{grow}-90\relax}#1},
3970 get min s tree boundary/.code={%
3971 \forest@get@tree@boundary{negative}{\forestove{grow}}#1},
3972 get max s tree boundary/.code={%
3973 \forest@get@tree@boundary{positive}{\forestove{grow}}#1},
3974 use as bounding box/.style={%
3975 before drawing tree={
3976 tikz+/.expanded={%
3977 \noexpand\pgfresetboundingbox
3978 \noexpand\useasboundingbox
3979 ($(.anchor)+(\forestoption{min x},\forestoption{min y})$)
3980 rectangle
3981 ($(.anchor)+(\forestoption{max x},\forestoption{max y})$)
3982 ;
3983 }
3984 }
3985 },
3986 use as bounding box'/.style={%
3987 before drawing tree={
3988 tikz+/.expanded={%
3989 \noexpand\pgfresetboundingbox
3990 \noexpand\useasboundingbox
3991 ($(.anchor)+(\forestoption{min x}+\pgfkeysvalueof{/pgf/outer xsep}/2+\pgfkeysvalueof{/pgf/inner xsep})
3992 rectangle
3993 ($(.anchor)+(\forestoption{max x}-\pgfkeysvalueof{/pgf/outer xsep}/2-\pgfkeysvalueof{/pgf/inner xsep})
3994 ;
3995 }
3996 }
3997 },
3998 }%

```

```

3999 \def\forest@iftikzkey#1#2#3{% #1 = key name, #2 = true code, #3 = false code
4000   \forest@temptrue
4001   \pgfkeysifdefined{/tikz/\pgfkeyscurrentname}{-}{%
4002     \pgfkeysifdefined{/tikz/\pgfkeyscurrentname/.@cmd}{-}{%
4003       \pgfkeysifdefined{/pgf/\pgfkeyscurrentname}{-}{%
4004         \pgfkeysifdefined{/pgf/\pgfkeyscurrentname/.@cmd}{-}{%
4005           \forest@tempfalse
4006         }}}}
4007   \ifforest@temp\@escapeif{#2}\else\@escapeif{#3}\fi
4008 }
4009 \def\forest@ifoptionortikzkey#1#2#3{% #1 = key name, #2 = true code, #3 = false code
4010   \forest@temptrue
4011   \pgfkeysifdefined{/forest/\pgfkeyscurrentname}{-}{%
4012     \pgfkeysifdefined{/forest/\pgfkeyscurrentname/.@cmd}{-}{%
4013       \forest@iftikzkey{#1}{-}{-}%
4014     }-}%
4015   \ifforest@temp\@escapeif{#2}\else\@escapeif{#3}\fi
4016 }
4017 \def\forest@get@tree@boundary#1#2#3{%#1=pos/neg,#2=grow,#3=receiving cs
4018   \def#3{}%
4019   \forest@node@getedge{#1}{#2}\forest@temp@boundary
4020   \forest@extendpath#3\forest@temp@boundary{\pgfqpoint{\forestove{x}}{\forestove{y}}}%
4021 }
4022 \def\forest@setter@node{\forest@cn}%
4023 \def\forest@relatednode@option@compat@ignoreinvalidsteps#1{#1}
4024 \def\forest@relatednode@option@setter#1.#2=#3\forest@END{%
4025   \forest@forthis{%
4026     \forest@relatednode@option@compat@ignoreinvalidsteps{%
4027       \forest@nameandgo{#1}%
4028       \let\forest@setter@node\forest@cn
4029     }%
4030   }%
4031   \ifnum\forest@setter@node=0
4032   \else
4033     \forestset{#2={#3}}%
4034   \fi
4035   \def\forest@setter@node{\forest@cn}%
4036 }%
4037 \def\forest@split#1#2#3#4{% #1=list (assuming that the list is nonempty and finishes with the separator), #2
4038   \def\forest@split@@##1#2##2\forest@split@@##3##4{\def##3{##1}\def##4{##2}}%
4039   \forest@split@@#1\forest@split@@{#3}{#4}}

```

7.1.3 Option propagation

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

```

4040 \forestset{
4041   for tree'/.style 2 args={#1,for children={for tree'={#1}{#2}},#2},
4042   if/.code n args={3}{%
4043     \forestmathtruncatemacro\forest@temp{#1}%
4044     \ifnum\forest@temp=0
4045       \@escapeif{\pgfkeysalso{#3}}%
4046     \else
4047       \@escapeif{\pgfkeysalso{#2}}%
4048     \fi
4049   },
4050   %LaTeX if/.code n args={3}{#1{\pgfkeysalso{#2}}{\pgfkeysalso{#3}}},
4051   if nodewalk valid/.code n args={3}{%
4052     \forest@forthis{%
4053       \forest@configured@nodewalk{independent}{inherited}{fake}{%
4054         #1,
4055         TeX={\global\let\forest@global@temp\forest@cn}

```

```

4056     }{}%
4057 }%
4058 \ifnum\forest@global@temp=0
4059   \@escapeif{\pgfkeysalso{#3}}%
4060 \else
4061   \@escapeif{\pgfkeysalso{#2}}%
4062 \fi
4063 },
4064 if nodewalk empty/.code n args={3}{%
4065   \forest@forthis{%
4066     \forest@configured@nodewalk{independent}{independent}{fake}{%
4067       #1,
4068       TeX={\global\let\forest@global@temp\forest@nodewalk@n},
4069     }{}%
4070   }%
4071   \ifnum\forest@global@temp=0
4072     \@escapeif{\pgfkeysalso{#2}}%
4073   \else
4074     \@escapeif{\pgfkeysalso{#3}}%
4075   \fi
4076 },
4077 if current nodewalk empty/.code 2 args={%
4078   \ifnum\forest@nodewalk@n=0
4079     \@escapeif{\pgfkeysalso{#1}}%
4080   \else
4081     \@escapeif{\pgfkeysalso{#2}}%
4082   \fi
4083 },
4084 where/.style n args={3}{for tree={if={#1}{#2}{#3}}},
4085 where nodewalk valid/.style n args={3}{for tree={if nodewalk valid={#1}{#2}{#3}}},
4086 where nodewalk empty/.style n args={3}{for tree={if nodewalk empty={#1}{#2}{#3}}},
4087 repeat/.code 2 args={%
4088   \forestmathtruncatemacro\forest@temp{#1}%
4089   \expandafter\forest@repeatkey\expandafter{\forest@temp}{#2}%
4090 },
4091 until/.code 2 args={%
4092   \ifstrempy{#1}{%
4093     \forest@untilkey{\ifnum\forest@cn=0\else\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
4094   }{%
4095     \forest@untilkey{\forestmath@if{#1}{\forestloopbreak}}{#2}%
4096   }%
4097 },
4098 while/.code 2 args={%
4099   \ifstrempy{#1}{%
4100     \forest@untilkey{\ifnum\forest@cn=0\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
4101   }{%
4102     \forest@untilkey{\forestmath@if{#1}{\forestloopbreak}}{#2}%
4103   }%
4104 },
4105 do until/.code 2 args={%
4106   \ifstrempy{#1}{%
4107     \forest@dountilkey{\ifnum\forest@cn=0\else\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
4108   }{%
4109     \forest@dountilkey{\forestmath@if{#1}{\forestloopbreak}}{#2}%
4110   }%
4111 },
4112 do while/.code 2 args={%
4113   \ifstrempy{#1}{%
4114     \forest@dountilkey{\ifnum\forest@cn=0\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
4115   }{%
4116     \forest@dountilkey{\forestmath@if{#1}{\forestloopbreak}}{#2}%

```

```

4117 }%
4118 },
4119 until nodewalk valid/.code 2 args={%
4120   \forest@untilkey{\forest@forthis{%
4121     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\else\forestloopbreak\fi}}{}}{#2}%
4122   },
4123   while nodewalk valid/.code 2 args={%
4124     \forest@untilkey{\forest@forthis{%
4125       \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\forestloopbreak\fi}}{}}{#2}%
4126     },
4127     do until nodewalk valid/.code 2 args={%
4128       \forest@dountilkey{\forest@forthis{%
4129         \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\else\forestloopbreak\fi}}{}}{#2}%
4130       },
4131       do while nodewalk valid/.code 2 args={%
4132         \forest@dountilkey{\forest@forthis{%
4133           \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\forestloopbreak\fi}}{}}{#2}%
4134         },
4135         until nodewalk empty/.code 2 args={%
4136           \forest@untilkey{\forest@forthis{%
4137             \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\forestloopbreak\fi}}{}}{}}
4138           },
4139           while nodewalk empty/.code 2 args={%
4140             \forest@untilkey{\forest@forthis{%
4141               \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\else\forestloopbreak\fi}}
4142             },
4143             do until nodewalk empty/.code 2 args={%
4144               \forest@dountilkey{\forest@forthis{%
4145                 \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\forestloopbreak\fi}}{}}{}}
4146               },
4147               do while nodewalk empty/.code 2 args={%
4148                 \forest@dountilkey{\forest@forthis{%
4149                   \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\else\forestloopbreak\fi}}
4150                 },
4151                 break/.code={\forestloopBreak{#1}},
4152                 break/.default=0,
4153               }
4154 \def\forest@repeatkey#1#2{%
4155   \safeRKloop
4156   \ifnum\safeRKloopn>#1\relax
4157   \csuse{safeRKbreak@\the\safeRKloop@depth true}%
4158   \fi
4159   \expandafter\unless\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
4160   \pgfkeysalso{#2}%
4161   \safeRKrepeat
4162 }
4163 \def\forest@untilkey#1#2{% #1 = condition, #2 = keys
4164   \safeRKloop
4165   #1%
4166   \expandafter\unless\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
4167   \pgfkeysalso{#2}%
4168   \safeRKrepeat
4169 }
4170 \def\forest@dountilkey#1#2{% #1 = condition, #2 = keys
4171   \safeRKloop
4172   \pgfkeysalso{#2}%
4173   #1%
4174   \expandafter\unless\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
4175   \safeRKrepeat
4176 }
4177 \def\forestloopbreak{%

```

```

4178 \csname safeRKbreak@\the\safeRKloop@depth true\endcsname
4179 }
4180 \def\forestloopBreak#1{%
4181 \csname safeRKbreak@\number\numexpr\the\safeRKloop@depth-#1\relax true\endcsname
4182 }
4183 \def\forestloopcount{%
4184 \csname safeRKloopn@\number\numexpr\the\safeRKloop@depth\endcsname
4185 }
4186 \def\forestloopCount#1{%
4187 \csname safeRKloopn@\number\numexpr\the\safeRKloop@depth-#1\endcsname
4188 }
4189 \pgfmathdeclarefunction{forestloopcount}{1}{%
4190 \edef\pgfmathresult{\forestloopCount{\ifstrempty{#1}{0}{#1}}}%
4191 }
4192 \forest@copycommandkey{/forest/repeat}{/forest/nodewalk/repeat}
4193 \forest@copycommandkey{/forest/while}{/forest/nodewalk/while}
4194 \forest@copycommandkey{/forest/do while}{/forest/nodewalk/do while}
4195 \forest@copycommandkey{/forest/until}{/forest/nodewalk/until}
4196 \forest@copycommandkey{/forest/do until}{/forest/nodewalk/do until}
4197 \forest@copycommandkey{/forest/if}{/forest/nodewalk/if}
4198 \forest@copycommandkey{/forest/if nodewalk valid}{/forest/nodewalk/if nodewalk valid}
4199 %

```

7.2 Aggregate functions

```

4200 \forestset{
4201   aggregate postparse/.is choice,
4202   aggregate postparse/int/.code={%
4203     \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@toint},
4204   aggregate postparse/none/.code={%
4205     \let\forest@aggregate@pgfmathpostparse\relax},
4206   aggregate postparse/print/.code={%
4207     \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@print},
4208   aggregate postparse/macro/.code={%
4209     \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@usemacro},
4210   aggregate postparse macro/.store in=\forest@aggregate@pgfmathpostparse@macro,
4211 }
4212 \def\forest@aggregate@pgfmathpostparse@print{%
4213 \pgfmathprintnumberto{\pgfmathresult}{\pgfmathresult}%
4214 }
4215 \def\forest@aggregate@pgfmathpostparse@toint{%
4216 \expandafter\forest@split\expandafter{\pgfmathresult.}{.}\pgfmathresult\forest@temp
4217 }
4218 \def\forest@aggregate@pgfmathpostparse@usemacro{%
4219 \forest@aggregate@pgfmathpostparse@macro
4220 }
4221 \let\forest@aggregate@pgfmathpostparse\relax
4222 \forestset{
4223 /handlers/.aggregate/.code n args=4{%
4224   % #1 = start value (forestmath)
4225   % #2 = forestmath expression that calculates "aggregate result" at each step
4226   % #3 = forestmath expression that calculates "aggregate result" at the end of the nodewalk
4227   % #4 = nodewalk
4228   \forest@aggregate@handler{\forest@aggregate@generic{#1}{#2}{#3}{#4}}%
4229 },
4230 /handlers/.sum/.code 2 args={% #1=forestmath, #2=nodewalk
4231   \forest@aggregate@handler{\forest@aggregate@sum{#1}{#2}}%
4232 },
4233 /handlers/.count/.code={% #1=nodewalk
4234   \forest@aggregate@handler{\forest@aggregate@count{#1}}%

```

```

4235 },
4236 /handlers/.average/.code 2 args={% #1=forestmath, #2=nodewalk
4237   \forest@aggregate@handler{\forest@aggregate@average{#1}{#2}}%
4238 },
4239 /handlers/.product/.code 2 args={% #1=forestmath, #2=nodewalk
4240   \forest@aggregate@handler{\forest@aggregate@product{#1}{#2}}%
4241 },
4242 /handlers/.min/.code 2 args={% #1=forestmath, #2=nodewalk
4243   \forest@aggregate@handler{\forest@aggregate@min{#1}{#2}}%
4244 },
4245 /handlers/.max/.code 2 args={% #1=forestmath, #2=nodewalk
4246   \forest@aggregate@handler{\forest@aggregate@max{#1}{#2}}%
4247 },
4248 declare count register={aggregate n},
4249 declare toks register={aggregate value},
4250 declare toks register={aggregate result},
4251 aggregate result={},
4252 }
4253 \def\forest@aggregate@handler#1{%
4254   \edef\forest@marshal{%
4255     \unexpanded{%
4256       #1%
4257     }{%
4258       \noexpand\pgfkeysalso{\pgfkeyscurrentpath/.register=aggregate result}%
4259     }%
4260   }\forest@marshal
4261 }
4262 \def\forest@aggregate@pgfmathfunction@finish{%
4263   \forest@get{aggregate result}\pgfmathresult
4264 }
4265 \pgfmathdeclarefunction{aggregate}{4}{%
4266   \forest@aggregate@generic{#1}{#2}{#3}{#4}%
4267   \forest@aggregate@pgfmathfunction@finish
4268 }
4269 \pgfmathdeclarefunction{aggregate_count}{1}{%
4270   \forest@aggregate@sum{#1}%
4271   \forest@aggregate@pgfmathfunction@finish
4272 }
4273 \pgfmathdeclarefunction{aggregate_sum}{2}{%
4274   \forest@aggregate@sum{#1}{#2}%
4275   \forest@aggregate@pgfmathfunction@finish
4276 }
4277 \pgfmathdeclarefunction{aggregate_product}{2}{%
4278   \forest@aggregate@product{#1}{#2}%
4279   \forest@aggregate@pgfmathfunction@finish
4280 }
4281 \pgfmathdeclarefunction{aggregate_average}{2}{%
4282   \forest@aggregate@average{#1}{#2}%
4283   \forest@aggregate@pgfmathfunction@finish
4284 }
4285 \pgfmathdeclarefunction{aggregate_min}{2}{%
4286   \forest@aggregate@min{#1}{#2}%
4287   \forest@aggregate@pgfmathfunction@finish
4288 }
4289 \pgfmathdeclarefunction{aggregate_max}{2}{%
4290   \forest@aggregate@max{#1}{#2}%
4291   \forest@aggregate@pgfmathfunction@finish
4292 }

```

Define particular aggregate functions.

```

4293 \def\forest@aggregate#1#2#3#4#5#6{% #1...#5=real args,

```

```

4294                                     % #6=what to do with |aggregate result| register
4295     % #1 = start value (forestmath)
4296     % #2 = forestmath expression that calculates "aggregate current" at each step
4297     % #3 = forestmath expression that calculates "aggregate result" at each step
4298     % #4 = forestmath expression that calculates "aggregate result" at the end of the nodewalk
4299     % #5 = nodewalk
4300     \forest@saveandrestoreregister{aggregate result}{%
4301         \forest@saveandrestoreregister{aggregate n}{%
4302             \forest@aggregate@{#1}{#2}{#3}{#4}{#5}%
4303             #6%
4304         }%
4305     }%
4306 }
4307 \def\forest@aggregate@generic#1#2#3#4{\forest@aggregate
4308     {\forestmathparse{#1}}}%
4309     {}%
4310     {\forestmathparse{#2}}}%
4311     {\forestmathparse{#3}}}%
4312     {#4}}%
4313 }
4314 \def\forest@aggregate@sum#1#2{\forest@aggregate
4315     {\forestmath@convert@fromto\forestmathtype@count\forestmathtype@generic{0}}}%
4316     {\forestmathparse{#1}}}%
4317     {\forestmathadd{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
4318     {\forestregister{aggregate result}\forestmathresult}%
4319     {#2}}%
4320 }
4321 \def\forest@aggregate@count#1{\forest@aggregate
4322     {\def\forestmathresult{1}}}%
4323     {\def\forestmathresult{1}}}%
4324     {\edef\forestmathresult{\the\numexpr\forestregister{aggregate result}+1}}}%
4325     {\forestregister{aggregate result}\forestmathresult}%
4326     {#1}}%
4327 }
4328 \def\forest@aggregate@average#1#2{\forest@aggregate
4329     {\forestmath@convert@fromto\forestmathtype@count\forestmathtype@generic{0}}}%
4330     {\forestmathparse{#1}}}%
4331     {\forestmathadd{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
4332     {\forestmathdivideP{\forestregister{aggregate result}}{\forestregister{aggregate n}}}%
4333     {#2}}%
4334 }
4335 \def\forest@aggregate@product#1#2{\forest@aggregate
4336     {\forestmath@convert@fromto\forestmathtype@count\forestmathtype@generic{1}}}%
4337     {\forestmathparse{#1}}}%
4338     {\forestmathmultiply{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
4339     {\forestregister{aggregate result}\forestmathresult}%
4340     {#2}}%
4341 }
4342 \def\forest@aggregate@min#1#2{\forest@aggregate
4343     {\def\forestmathresult{}}}%
4344     {\forestmathparse{#1}}}%
4345     {\forestmathmin{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
4346     {\forestregister{aggregate result}\forestmathresult}%
4347     {#2}}%
4348 }
4349 \def\forest@aggregate@max#1#2{\forest@aggregate
4350     {\def\forestmathresult{}}}%
4351     {\forestmathparse{#1}}}%
4352     {\forestmathmax{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
4353     {\forestregister{aggregate result}\forestmathresult}%
4354     {#2}}%

```


4355 }

Actual computation.

```

4356 \def\forest@aggregate@#1#2#3#4#5{%
4357   % #1 = start value (forestmath)
4358   % #2 = forestmath expression that calculates "aggregate current" at each step
4359   % #3 = forestmath expression that calculates "aggregate result" at each step
4360   % #4 = forestmath expression that calculates "aggregate result" at the end of the nodewalk
4361   % #5 = nodewalk
4362   #1%
4363   \forestrlset{aggregate result}\forestmathresult
4364   \forestrset{aggregate value}{}%
4365   \forestrset{aggregate n}{0}%
4366   \forest@forthis{%
4367     \forest@nodewalk{#5}{%
4368       TeX={%
4369         \forestreset{aggregate n}{\number\numexpr\forestrv{aggregate n}+1}%
4370         #2%
4371         \forestrlset{aggregate value}\forestmathresult
4372         #3%
4373         \forestrlset{aggregate result}\forestmathresult
4374       }%
4375     }{}%
4376   }%
4377   #4%
4378   \let\forest@temp@pgfmaphpostparse\pgfmaphpostparse
4379   \let\pgfmaphpostparse\forest@aggregate@pgfmaphpostparse
4380   \forestmath@convert@to\forestmathtype@dimen{\forestmathresult}
4381   \pgfmaphqparse{\forestmathresult}%
4382   \let\pgfmaphpostparse\forest@temp@pgfmaphpostparse
4383   \forestrlset{aggregate result}\pgfmaphresult
4384 }
```

7.2.1 pgfmaph extensions

```

4385 \pgfmaphdeclarefunction{strequal}{2}{%
4386   \ifstrequal{#1}{#2}{\def\pgfmaphresult{1}}{\def\pgfmaphresult{0}}%
4387 }
4388 \pgfmaphdeclarefunction{instr}{2}{%
4389   \pgfutil@in@{#1}{#2}%
4390   \ifpgfutil@in@\def\pgfmaphresult{1}\else\def\pgfmaphresult{0}\fi
4391 }
4392 \pgfmaphdeclarefunction{strcat}{...}{%
4393   \edef\pgfmaphresult{\forest@strip@braces{#1}}%
4394 }
4395 \pgfmaphdeclarefunction{min_s}{2}{% #1 = node, #2 = context node (for growth rotation)
4396   \forest@forthis{%
4397     \forest@nameandgo{#1}%
4398     \forest@compute@minmax@ls{#2}%
4399     \edef\forest@temp{\forestove{min@s}}%
4400     \edef\pgfmaphresult{\expandafter\Pgf@geT\forest@temp}%
4401   }%
4402 }
4403 \pgfmaphdeclarefunction{min_l}{2}{% #1 = node, #2 = context node (for growth rotation)
4404   \forest@forthis{%
4405     \forest@nameandgo{#1}%
4406     \forest@compute@minmax@ls{#2}%
4407     \edef\forest@temp{\forestove{min@l}}%
4408     \edef\pgfmaphresult{\expandafter\Pgf@geT\forest@temp}%
4409   }%
4410 }
4411 \pgfmaphdeclarefunction{max_s}{2}{% #1 = node, #2 = context node (for growth rotation)
```

```

4412 \forest@forthis{%
4413   \forest@nameandgo{#1}%
4414   \forest@compute@minmax@ls{#2}%
4415   \edef\forest@temp{\forestove{max@s}}%
4416   \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
4417 }%
4418 }
4419 \pgfmathdeclarefunction{max_l}{2}{% #1 = node, #2 = context node (for growth rotation)
4420   \forest@forthis{%
4421     \forest@nameandgo{#1}%
4422     \forest@compute@minmax@ls{#2}%
4423     \edef\forest@temp{\forestove{max@l}}%
4424     \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
4425   }%
4426 }
4427 \def\forest@compute@minmax@ls#1{% #1 = nodewalk; in the context of which node?
4428   {%
4429     \pgftransformreset
4430     \forest@forthis{%
4431       \forest@nameandgo{#1}%
4432       \forest@pgfqtransformrotate{-\forestove{grow}}%
4433     }%
4434     \forestoget{min x}\forest@temp@minx
4435     \forestoget{min y}\forest@temp@miny
4436     \forestoget{max x}\forest@temp@maxx
4437     \forestoget{max y}\forest@temp@maxy
4438     \pgfpointrtransformed{\pgfqpoint{\forest@temp@minx}{\forest@temp@miny}}%
4439     \forestoeset{min@l}{\the\pgf@x}%
4440     \forestoeset{min@s}{\the\pgf@y}%
4441     \forestoeset{max@l}{\the\pgf@x}%
4442     \forestoeset{max@s}{\the\pgf@y}%
4443     \pgfpointrtransformed{\pgfqpoint{\forest@temp@minx}{\forest@temp@maxy}}%
4444     \ifdim\pgf@x<\forestove{min@l}\relax\forestoeset{min@l}{\the\pgf@x}\fi
4445     \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
4446     \ifdim\pgf@x>\forestove{max@l}\relax\forestoeset{max@l}{\the\pgf@x}\fi
4447     \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
4448     \pgfpointrtransformed{\pgfqpoint{\forest@temp@maxx}{\forest@temp@miny}}%
4449     \ifdim\pgf@x<\forestove{min@l}\relax\forestoeset{min@l}{\the\pgf@x}\fi
4450     \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
4451     \ifdim\pgf@x>\forestove{max@l}\relax\forestoeset{max@l}{\the\pgf@x}\fi
4452     \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
4453     \pgfpointrtransformed{\pgfqpoint{\forest@temp@maxx}{\forest@temp@maxy}}%
4454     \ifdim\pgf@x<\forestove{min@l}\relax\forestoeset{min@l}{\the\pgf@x}\fi
4455     \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
4456     \ifdim\pgf@x>\forestove{max@l}\relax\forestoeset{max@l}{\the\pgf@x}\fi
4457     \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
4458     % smuggle out
4459     \edef\forest@marshal{%
4460       \noexpand\forestoeset{min@l}{\forestove{min@l}}%
4461       \noexpand\forestoeset{min@s}{\forestove{min@s}}%
4462       \noexpand\forestoeset{max@l}{\forestove{max@l}}%
4463       \noexpand\forestoeset{max@s}{\forestove{max@s}}%
4464     }\expandafter
4465   }\forest@marshal
4466 }
4467 \def\forest@pgfmathhelper@attribute@toks#1#2{%
4468   \forest@forthis{%
4469     \forest@nameandgo{#1}%
4470     \ifnum\forest@cn=0
4471       \def\pgfmathresult{}%
4472     \else

```

```

4473     \foresttoget{#2}\pgfmathresult
4474   \fi
4475 }%
4476 }
4477 \def\forest@pgfmathhelper@attribute@dimen#1#2{%
4478   \forest@forthis{%
4479     \forest@nameandgo{#1}%
4480     \ifnum\forest@cn=0
4481       \def\pgfmathresult{0}%
4482     \else
4483       \foresttoget{#2}\forest@temp
4484       \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
4485     \fi
4486   }%
4487 }
4488 \def\forest@pgfmathhelper@attribute@count#1#2{%
4489   \forest@forthis{%
4490     \forest@nameandgo{#1}%
4491     \ifnum\forest@cn=0
4492       \def\pgfmathresult{0}%
4493     \else
4494       \foresttoget{#2}\pgfmathresult
4495     \fi
4496   }%
4497 }
4498 \pgfmathdeclarefunction*{id}{1}{%
4499   \forest@forthis{%
4500     \forest@nameandgo{#1}%
4501     \let\pgfmathresult\forest@cn
4502   }%
4503 }

```

7.3 Nodewalk

Setup machinery.

```

4504 \def\forest@nodewalk@n{0}
4505 \def\forest@nodewalk@historyback{0,}
4506 \def\forest@nodewalk@historyforward{0,}
4507 \def\forest@nodewalk@origin{0}
4508 \def\forest@nodewalk@config@everystep@independent@before#1{% #1 = every step keylist
4509   \foreststrset{every step}{#1}%
4510 }
4511 \def\forest@nodewalk@config@everystep@independent@after{%
4512   \noexpand\foreststrset{every step}{\forestrv{every step}}%
4513 }
4514 \def\forest@nodewalk@config@history@independent@before{%
4515   \def\forest@nodewalk@n{0}%
4516   \edef\forest@nodewalk@origin{\forest@cn}%
4517   \def\forest@nodewalk@historyback{0,}%
4518   \def\forest@nodewalk@historyforward{0,}%
4519 }
4520 \def\forest@nodewalk@config@history@independent@after{%
4521   \edef\noexpand\forest@nodewalk@n{\expandonce{\forest@nodewalk@n}}%
4522   \edef\noexpand\forest@nodewalk@origin{\expandonce{\forest@nodewalk@origin}}%
4523   \edef\noexpand\forest@nodewalk@historyback{\expandonce{\forest@nodewalk@historyback}}%
4524   \edef\noexpand\forest@nodewalk@historyforward{\expandonce{\forest@nodewalk@historyforward}}%
4525 }
4526 \def\forest@nodewalk@config@everystep@shared@before#1{% #1 = every step keylist
4527 \def\forest@nodewalk@config@everystep@shared@after{}
4528 \def\forest@nodewalk@config@history@shared@before{}
4529 \def\forest@nodewalk@config@history@shared@after{}

```

```

4530 \def\forest@nodewalk@config@everystep@inherited@before#1{}% #1 = every step keylist
4531 \let\forest@nodewalk@config@everystep@inherited@after\forest@nodewalk@config@everystep@independent@after
4532 \def\forest@nodewalk@config@history@inherited@before{}
4533 \let\forest@nodewalk@config@history@inherited@after\forest@nodewalk@config@history@independent@after
4534 \def\forest@nodewalk#1#2{% #1 = nodewalk, #2 = every step keylist
4535   \forest@configured@nodewalk{independent}{independent}{inherited}{#1}{#2}%
4536 }
4537 \def\forest@configured@nodewalk#1#2#3#4#5{%
4538   % #1 = every step method, #2 = history method, #3 = on invalid
4539   % #4 = nodewalk, #5 = every step keylist
4540   \def\forest@nodewalk@config@everystep@method{#1}%
4541   \def\forest@nodewalk@config@history@method{#2}%
4542   \def\forest@nodewalk@config@oninvalid{#3}%
4543   \forest@Nodewalk{#4}{#5}%
4544 }
4545 \def\forest@nodewalk@oninvalid@inherited@text{inherited}
4546 \def\forest@Nodewalk#1#2{% #1 = nodewalk, #2 = every step keylist
4547   \ifx\forest@nodewalk@config@oninvalid\forest@nodewalk@oninvalid@inherited@text
4548     \edef\forest@nodewalk@config@oninvalid{\forest@nodewalk@oninvalid}%
4549   \fi
4550   \edef\forest@marshal{%
4551     \noexpand\pgfqkeys{/forest/nodewalk}{\unexpanded{#1}}%
4552     \csname forest@nodewalk@config@everystep\forest@nodewalk@config@everystep@method @after\endcsname
4553     \csname forest@nodewalk@config@history\forest@nodewalk@config@history@method @after\endcsname
4554     \edef\noexpand\forest@nodewalk@oninvalid{\forest@nodewalk@oninvalid}%
4555   }%
4556   \csname forest@nodewalk@config@everystep\forest@nodewalk@config@everystep@method @before\endcsname{#2}%
4557   \csname forest@nodewalk@config@history\forest@nodewalk@config@history@method @before\endcsname
4558   \edef\forest@nodewalk@oninvalid{\forest@nodewalk@config@oninvalid}%
4559   \forest@nodewalk@fakefalse
4560   \forest@marshal
4561 }
4562 \pgfmathdeclarefunction{valid}{1}{%
4563   \forest@forthis{%
4564     \forest@nameandgo{#1}%
4565     \edef\pgfmathresult{\ifnum\forest@cn=0 0\else 1\fi}%
4566   }%
4567 }
4568 \pgfmathdeclarefunction{invalid}{1}{%
4569   \forest@forthis{%
4570     \forest@nameandgo{#1}%
4571     \edef\pgfmathresult{\ifnum\forest@cn=0 1\else 0\fi}%
4572   }%
4573 }
4574 \newif\ifforest@nodewalk@fake
4575 \def\forest@nodewalk@oninvalid{error}
4576 \def\forest@nodewalk@makestep{%
4577   \ifnum\forest@cn=0
4578     \csname forest@nodewalk@makestep@oninvalid\forest@nodewalk@oninvalid\endcsname
4579   \else
4580     \forest@nodewalk@makestep@
4581   \fi
4582 }
4583 \csdef{forest@nodewalk@makestep@oninvalid@error if real}{\ifforest@nodewalk@fake\expandafter\forest@nodewalk@
4584 \csdef{forest@nodewalk@makestep@oninvalid@last valid}{%
4585   \forest@nodewalk@tolastvalid
4586   \ifforestdebugnodewalks\forest@nodewalk@makestep@invalidtolastvalid@debug\fi}%
4587 \def\forest@nodewalk@makestep@oninvalid@error{\PackageError{forest}{nodewalk stepped to the invalid node\Mess
4588 \let\forest@nodewalk@makestep@oninvalid@fake\relax
4589 \def\forest@nodewalk@makestep@oninvalid@compatfake{%
4590   \forest@deprecated{last step in stack "\forest@nodewalk@currentstepname", which stepped on an invalid node;

```

```

4591 }%
4592 \def\forest@nodewalk@makestep@{%
4593   \ifforestdebugnodewalks\forest@nodewalk@makestep@debug\fi
4594   \ifforest@nodewalk@fake
4595   \else
4596     \edef\forest@nodewalk@n{\number\numexpr\forest@nodewalk@n+1}%
4597     \epreto\forest@nodewalk@historyback{\forest@cn,}%
4598     \def\forest@nodewalk@historyforward{0,}%
4599     \forest@process@keylist@register{every step}%
4600   \fi
4601 }
4602 \def\forest@nodewalk@makestep@debug{%
4603   \edef\forest@marshal{%
4604     \noexpand\typeout{\ifforest@nodewalk@fake fake \fi "\forest@nodewalk@currentstepname" step to node id=\fo
4605   }\forest@marshal
4606 }%
4607 \def\forest@nodewalk@makestep@invalidtolastvalid@debug{%
4608   \edef\forest@marshal{%
4609     \noexpand\typeout{\ifforest@nodewalk@fake fake \fi "\forest@nodewalk@currentstepname" step to invalid nod
4610   }\forest@marshal
4611 }%
4612 \def\forest@handlers@savecurrentpath{%
4613   \edef\pgfkeyscurrentkey{\pgfkeyscurrentpath}%
4614   \let\forest@currentkey\pgfkeyscurrentkey
4615   \pgfkeys@split@path
4616   \edef\forest@currentpath{\pgfkeyscurrentpath}%
4617   \let\forest@currentname\pgfkeyscurrentname
4618 }
4619 \pgfkeys{/handlers/save current path/.code={\forest@handlers@savecurrentpath}}
4620 \newif\ifforest@nodewalkstephandler@style
4621 \newif\ifforest@nodewalkstephandler@autostep
4622 \newif\ifforest@nodewalkstephandler@stripfakesteps
4623 \newif\ifforest@nodewalkstephandler@muststartatvalidnode
4624 \newif\ifforest@nodewalkstephandler@makefor
4625 \let\forest@nodewalkstephandler@styletrueorfalse\forest@nodewalkstephandler@stylefalse
4626 \def\forest@nodewalk@currentstepname{}
4627 \forestset{
4628   /forest/define@step/style/.is if=forest@nodewalkstephandler@style,
4629   /forest/define@step/autostep/.is if=forest@nodewalkstephandler@autostep,
4630   % the following is useful because some macros use grouping (by \forest@forthis or similar) and therefore, a
4631   /forest/define@step/strip fake steps/.is if=forest@nodewalkstephandler@stripfakesteps,
4632   % this can never happen with autosteps ...
4633   /forest/define@step/autostep/.append code={%
4634     \ifforest@nodewalkstephandler@autostep
4635       \forest@nodewalkstephandler@stripfakestepsfalse
4636     \fi
4637   },
4638   /forest/define@step/must start at valid node/.is if=forest@nodewalkstephandler@muststartatvalidnode,
4639   /forest/define@step/n args/.store in=\forest@nodewalkstephandler@nargs,
4640   /forest/define@step/make for/.is if=forest@nodewalkstephandler@makefor,
4641   /forest/define@step/@bare/.style={strip fake steps=false,must start at valid node=false,make for=false},
4642   define long step/.code n args=3{%
4643     \forest@nodewalkstephandler@styletrueorfalse % true for end users; but in the package, most of steps are
4644     \forest@nodewalkstephandler@autostepfalse
4645     \forest@nodewalkstephandler@stripfakesteptrue
4646     \forest@nodewalkstephandler@muststartatvalidnodetrue % most steps can only start at a valid node
4647     \forest@nodewalkstephandler@makefortrue % make for prefix?
4648     \def\forest@nodewalkstephandler@nargs{0}%
4649     \pgfqkeys{/forest/define@step}{#2}%
4650     \forest@temp@toks{#3}% handler code
4651     \ifforest@nodewalkstephandler@style

```

```

4652     \expandafter\forest@temp@toks\expandafter{%
4653     \expandafter\pgfkeysalso\expandafter{\the\forest@temp@toks}%
4654     }%
4655 \fi
4656 \ifforest@nodewalkstephandler@autostep
4657     \apptotoks\forest@temp@toks{\forest@nodewalk@makestep}%
4658 \fi
4659 \ifforest@nodewalkstephandler@stripfakesteps
4660     \expandafter\forest@temp@toks\expandafter{\expandafter\forest@nodewalk@stripfakesteps\expandafter{\the\
4661 \fi
4662 \ifforest@nodewalkstephandler@muststartatvalidnode
4663     \edef\forest@marshal{%
4664     \noexpand\forest@temp@toks{%
4665     \unexpanded{%
4666     \ifnum\forest@cn=0
4667     \csname forest@nodewalk@start@oninvalid@\forest@nodewalk@oninvalid\endcsname{#1}%
4668     \else
4669     }%
4670     \noexpand\@escapeif{\the\forest@temp@toks}%
4671     \noexpand\fi
4672     }%
4673     }\forest@marshal
4674 \fi
4675 \pretotoks\forest@temp@toks{\appto\forest@nodewalk@currentstepname{, #1}}%
4676 \expandafter\forest@temp@toks\expandafter{\expandafter\forest@saveandrestoremacro\expandafter\forest@node
4677 \ifforestdebugnodewalks
4678     \epretotoks\forest@temp@toks{\noexpand\typeout{Starting step "#1" from id=\noexpand\forest@cn
4679     \ifnum\forest@nodewalkstephandler@nargs>0 \space with args ####1\fi
4680     \ifnum\forest@nodewalkstephandler@nargs>1 ,####2\fi
4681     \ifnum\forest@nodewalkstephandler@nargs>2 ,####3\fi
4682     \ifnum\forest@nodewalkstephandler@nargs>3 ,####4\fi
4683     \ifnum\forest@nodewalkstephandler@nargs>4 ,####5\fi
4684     \ifnum\forest@nodewalkstephandler@nargs>5 ,####6\fi
4685     \ifnum\forest@nodewalkstephandler@nargs>6 ,####7\fi
4686     \ifnum\forest@nodewalkstephandler@nargs>7 ,####8\fi
4687     \ifnum\forest@nodewalkstephandler@nargs>8 ,####9\fi
4688     }}%
4689 \fi
4690 \def\forest@temp{/forest/nodewalk/#1/.code}%
4691 \ifnum\forest@nodewalkstephandler@nargs<2
4692     \eappto\forest@temp{=}%
4693 \else\ifnum\forest@nodewalkstephandler@nargs=2
4694     \eappto\forest@temp{ 2 args=}%
4695 \else
4696     \eappto\forest@temp{ n args={\forest@nodewalkstephandler@nargs}}%
4697 \fi\fi
4698 \eappto\forest@temp{{\the\forest@temp@toks}}%
4699 \expandafter\pgfkeysalso\expandafter{\forest@temp}%
4700 \ifforest@nodewalkstephandler@makefor
4701     \ifnum\forest@nodewalkstephandler@nargs=0
4702     \forestset{%
4703     for #1/.code={\forest@forstepwrapper{#1}{##1}},
4704     }%
4705 \else\ifnum\forest@nodewalkstephandler@nargs=1
4706     \forestset{%
4707     for #1/.code 2 args={\forest@forstepwrapper{#1}{##1}}{##2}},
4708     }%
4709 \else
4710     \forestset{%
4711     for #1/.code n args/.expanded=%
4712     {\number\numexpr\forest@nodewalkstephandler@nargs+1}%

```

```

4713         {\noexpand\forest@forstepwrapper{#1\ifnum\forest@nodewalkstephandler@nargs>0=\fi\forest@util@narg
4714         }%
4715         \fi\fi
4716         % Uncomment this to collect a list of for-based spatial propagators
4717         %\appto\bigbadforlist{[#1]}%
4718         \fi
4719     },
4720 }
4721 \pgfqkeys{/handlers}{
4722   .nodewalk style/.code={\forest@handlers@savecurrentpath\pgfkeysalso{%
4723     \forest@currentpath/nodewalk/\forest@currentname/.style={#1}%
4724   }},
4725 }

```

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

```

4726 \def\forest@forstepwrapper#1#2{\forest@forthis{\forest@nodewalk{#1}{#2}}}
4727 \def\forest@util@nargs#1#2#3{% #1 = prefix (#, ##, ...), #2 = n args, #3=start; returns {#start+1}...{#start+
4728   \ifnum#2>0 {#1\number\numexpr#3+1}\fi
4729   \ifnum#2>1 {#1\number\numexpr#3+2}\fi
4730   \ifnum#2>2 {#1\number\numexpr#3+3}\fi
4731   \ifnum#2>3 {#1\number\numexpr#3+4}\fi
4732   \ifnum#2>4 {#1\number\numexpr#3+5}\fi
4733   \ifnum#2>5 {#1\number\numexpr#3+6}\fi
4734   \ifnum#2>6 {#1\number\numexpr#3+7}\fi
4735   \ifnum#2>7 {#1\number\numexpr#3+8}\fi
4736   \ifnum#2>8 {#1\number\numexpr#3+9}\fi
4737 }
4738 \def\forest@nodewalk@start@oninvalid@fake#1{}
4739 \def\forest@nodewalk@start@oninvalid@compatfake#1{%
4740   \forest@deprecated{last step in stack "\forest@nodewalk@currentstepname", which started from an invalid nod
4741 }%
4742 \let\forest@nodewalk@start@oninvalid@errorifreal\forest@nodewalk@start@oninvalid@fake % the step will be to a
4743 \let\forest@nodewalk@start@oninvalid@lastvalid\forest@nodewalk@start@oninvalid@fake
4744 \def\forest@nodewalk@start@oninvalid@error#1{\PackageError{forest}{nodewalk step "#1" cannot start at the inv

```

Define long-form single-step walks.

```

4745 \forestset{
4746   define long step={current}{autostep}{},
4747   define long step={next}{autostep}{\edef\forest@cn{\forestove{@next}}},
4748   define long step={previous}{autostep}{\edef\forest@cn{\forestove{@previous}}},
4749   define long step={parent}{autostep}{\edef\forest@cn{\forestove{@parent}}},
4750   define long step={first}{autostep}{\edef\forest@cn{\forestove{@first}}},
4751   define long step={last}{autostep}{\edef\forest@cn{\forestove{@last}}},
4752   define long step={sibling}{autostep}{%
4753     \edef\forest@cn{%
4754       \ifnum\forestove{@previous}=0
4755         \forestove{@next}%
4756       \else
4757         \forestove{@previous}%
4758       \fi
4759     }%
4760   },
4761   define long step={next node}{autostep}{\edef\forest@cn{\forest@node@linearnextid}},
4762   define long step={previous node}{autostep}{\edef\forest@cn{\forest@node@linearpreviousid}},
4763   define long step={first leaf}{autostep}{%
4764     \safeloop
4765     \edef\forest@cn{\forestove{@first}}%
4766     \unless\ifnum\forestove{@first}=0
4767       \saferepeat
4768   },

```

```

4769 define long step={first leaf'}{autostep}{%
4770   \safeloop
4771   \unless\ifnum\forestove{@first}=0
4772     \edef\forest@cn{\forestove{@first}}%
4773   \saferepeat
4774 },
4775 define long step={last leaf}{autostep}{%
4776   \safeloop
4777     \edef\forest@cn{\forestove{@last}}%
4778   \unless\ifnum\forestove{@last}=0
4779   \saferepeat
4780 },
4781 define long step={last leaf'}{autostep}{%
4782   \safeloop
4783   \unless\ifnum\forestove{@last}=0
4784     \edef\forest@cn{\forestove{@last}}%
4785   \saferepeat
4786 },
4787 define long step={next leaf}{style,strip fake steps=false}{group={do until={n_children()==0}{next node}}},
4788 define long step={previous leaf}{style,strip fake steps=false}{group={do until={n_children()==0}{previous n
4789 define long step={next on tier}{autostep,n args=1}{%
4790   \def\forest@temp{#1}%
4791   \ifx\forest@temp\pgfkeysnovalue@text
4792     \foresttoget{tier}\forest@nodewalk@giventier
4793   \else
4794     \def\forest@nodewalk@giventier{#1}%
4795   \fi
4796   \edef\forest@cn{\forest@node@linearnextid}%
4797   \safeloop
4798     \forest@nodewalk@gettier
4799   \ifforest@temp
4800     \edef\forest@cn{\forest@node@linearnextid}%
4801   \saferepeat
4802 },
4803 define long step={previous on tier}{autostep,n args=1}{%
4804   \def\forest@temp{#1}%
4805   \ifx\forest@temp\pgfkeysnovalue@text
4806     \foresttoget{tier}\forest@nodewalk@giventier
4807   \else
4808     \def\forest@nodewalk@giventier{#1}%
4809   \fi
4810   \safeloop
4811     \edef\forest@cn{\forest@node@linearpreviousid}%
4812     \forest@nodewalk@gettier
4813   \ifforest@temp
4814   \saferepeat
4815 },
4816 TeX={%
4817   \def\forest@nodewalk@gettier{%
4818     \ifnum\forest@cn=0
4819       \forest@tempfalse
4820     \else
4821       \foresttoget{tier}\forest@temp
4822       \ifx\forest@temp\forest@nodewalk@giventier
4823         \forest@tempfalse
4824       \else
4825         \forest@temptrue
4826       \fi
4827     \fi
4828   }%
4829 },

```



```

4830 %
4831 define long step={root}{autostep,must start at valid node=false}{%
4832   \edef\forest@cn{\forest@node@rootid}},
4833 define long step={root'}{autostep,must start at valid node=false}{%
4834   \forest0ifdefined{\forest@root}{@parent}{\edef\forest@cn{\forest@root}}{\edef\forest@cn{0}}%
4835 },
4836 define long step={origin}{autostep,must start at valid node=false}{\edef\forest@cn{\forest@nodewalk@origin}}
4837 %
4838 define long step={n}{autostep,n args=1}{%
4839   \forestmathtruncatemacro\forest@temp@n{#1}%
4840   \edef\forest@cn{\forest@node@nthchildid{\forest@temp@n}}%
4841 },
4842 define long step={n}{autostep,make for=false,n args=1}{%
4843   % Yes, twice. ;- )
4844   % n=1 and n(ext)
4845   \def\forest@nodewalk@temp{#1}%
4846   \ifx\forest@nodewalk@temp\pgfkeysnovalue@text
4847     \edef\forest@cn{\forest@tove{@next}}%
4848   \else
4849     \forestmathtruncatemacro\forest@temp@n{#1}%
4850     \edef\forest@cn{\forest@node@nthchildid{\forest@temp@n}}%
4851   \fi
4852 },
4853 define long step={n'}{autostep,n args=1}{%
4854   \forestmathtruncatemacro\forest@temp@n{#1}%
4855   \edef\forest@cn{\forest@node@nbarthchildid{\forest@temp@n}}%
4856 },
4857 define long step={to tier}{autostep,n args=1}{%
4858   \def\forest@nodewalk@giventier{#1}%
4859   \safeloop
4860     \forest@nodewalk@gettier
4861   \ifforest@temp
4862     \forest@toget{@parent}\forest@cn
4863   \saferepeat
4864 },
4865 %
4866 define long step={name}{autostep,n args=1,must start at valid node=false}{%
4867   \edef\forest@cn{%
4868     \forest@node@ifnamedefined{#1}{\forest@node@Nametoid{#1}}{0}}%
4869   }%
4870 },
4871 define long step={id}{autostep,n args=1,must start at valid node=false}{%
4872   \forest0ifdefined{#1}{@parent}{\edef\forest@cn{#1}}{\edef\forest@cn{0}}%
4873 },
4874 define long step={Nodewalk}{n args=3,@bare}{% #1 = config, #2 = nodewalk
4875   \def\forest@nodewalk@config@everystep@method{independent}%
4876   \def\forest@nodewalk@config@history@method{shared}%
4877   \def\forest@nodewalk@config@oninvalid{inherited}%
4878   \pgfqkeys{/forest/nodewalk@config}{#1}%
4879   \forest@Nodewalk{#2}{#3}%
4880 },
4881 define long step={nodewalk}{n args=2,@bare}{% #1 = nodewalk, #2 = every step
4882   \forest@nodewalk{#1}{#2}%
4883 },
4884 define long step={nodewalk'}{n args=1,@bare}{% #1 = nodewalk
4885   \forest@configured@nodewalk{inherited}{inherited}{#1}{}%
4886 },
4887 % these "for ..." keys must be defined explicitly
4888 % (and copied into node keyspace manually),
4889 % as prefix "for" normally introduces the every-step keylist
4890 define long step={for nodewalk}{n args=2,@bare}{% #1 = nodewalk, #2 = every step

```

```

4891 \forest@forthis{\forest@nodewalk{#1}{#2}},
4892 define long step={for nodewalk'}{n args=1,@bare}{% #1 = nodewalk
4893 \forest@forthis{%
4894 \forest@configured@nodewalk{inherited}{independent}{inherited}{#1}{}%
4895 }%
4896 },
4897 define long step={for Nodewalk}{n args=3,@bare}{% #1 = config, #2 = nodewalk, #3 = every-step
4898 \def\forest@nodewalk@config@everystep@method{independent}%
4899 \def\forest@nodewalk@config@history@method{shared}%
4900 \def\forest@nodewalk@config@oninvalid{inherited}%
4901 \pgfqkeys{/forest/nodewalk@config}{#1}%
4902 \forest@forthis{\forest@Nodewalk{#2}{#3}}%
4903 },
4904 copy command key={/forest/nodewalk/Nodewalk}{/forest/Nodewalk},
4905 copy command key={/forest/nodewalk/for nodewalk}{/forest/for nodewalk},
4906 copy command key={/forest/nodewalk/for Nodewalk}{/forest/for Nodewalk},
4907 declare keylist register=every step,
4908 every step'={},
4909 %%% begin nodewalk config
4910 nodewalk@config/.cd,
4911 every@step/.is choice,
4912 every@step/independent/.code={},
4913 every@step/inherited/.code={},
4914 every@step/shared/.code={},
4915 every step/.store in=\forest@nodewalk@config@everystep@method,
4916 every step/.prefix style={every@step=#1},
4917 @history/.is choice,
4918 @history/independent/.code={},
4919 @history/inherited/.code={},
4920 @history/shared/.code={},
4921 history/.store in=\forest@nodewalk@config@history@method,
4922 history/.prefix style={@history=#1},
4923 on@invalid/.is choice,
4924 on@invalid/error/.code={},
4925 on@invalid/fake/.code={},
4926 on@invalid/error if real/.code={},
4927 on@invalid/last valid/.code={},
4928 on@invalid/inherited/.code={},
4929 on invalid/.store in=\forest@nodewalk@config@oninvalid,
4930 on invalid/.prefix style={on@invalid=#1},
4931 %%% end nodewalk config
4932 }
4933 \newtoks\forest@nodewalk@branch@toks
4934 \forestset{
4935 declare toks register=branch@temp@toks,
4936 branch@temp@toks={},
4937 declare keylist register=branched@nodewalk,
4938 branched@nodewalk={},
4939 define long step={branch}{n args=1,@bare,make for,style}{@branch={#1}{branch@build@realstep,branch@build@fa
4940 define long step={branch'}{n args=1,@bare,make for,style}{@branch={#1}{branch@build@realstep}},
4941 @branch/.style 2 args={%
4942 save and restore register={branched@nodewalk}{
4943 branch@temp@toks={},
4944 split/.process={r}{#1}{,}{#2},
4945 branch@temp@style/.style/.register=branch@temp@toks,
4946 branch@temp@style,
4947 branch@temp@style/.style/.register=branched@nodewalk,
4948 branch@temp@style,
4949 }
4950 },
4951 nodewalk/branch@build@realstep/.style={% #1 = nodewalk for this branch

```

```

4952     branch@temp@toks/.expanded={for nodewalk={\unexpanded{#1}}{
4953         branched@nodewalk+/.expanded={id=\noexpand\forestoption{id}},
4954         \forestregister{branch@temp@toks}}},
4955 },
4956 nodewalk/branch@build@fakestep/.style={% #1 = nodewalk for this branch
4957     branch@temp@toks/.expanded={for nodewalk={\unexpanded{#1}}{
4958         \forestregister{branch@temp@toks}}},
4959 },
4960 define long step={group}{autostep,n args=1}{\forest@go{#1}},
4961 nodewalk/fake/.code={%
4962     \forest@saveandrestoreifcs{forest@nodewalk@fake}{%
4963         \forest@nodewalk@fakefalse
4964         \pgfkeysalso{#1}%
4965     }%
4966 },
4967 nodewalk/real/.code={%
4968     \forest@saveandrestoreifcs{forest@nodewalk@fake}{%
4969         \forest@nodewalk@fakefalse
4970         \pgfkeysalso{#1}%
4971     }%
4972 },
4973 declare keylist register=filtered@nodewalk,
4974 filtered@nodewalk={},
4975 define long step={filter}{n args=2,@bare,make for,style}{% #1 = nodewalk, #2 = condition
4976     save and restore register={filtered@nodewalk}{
4977         filtered@nodewalk'={},
4978         Nodewalk=%
4979         {history=inherited}%
4980         {#1}%
4981         {if={#2}{filtered@nodewalk+/.expanded={id=\forestoption{id}}}{}},
4982         filtered@nodewalk@style/.style/.register=filtered@nodewalk,
4983         filtered@nodewalk@style
4984     },
4985 },
4986 on@invalid/.is choice,
4987 on@invalid/error/.code={},
4988 on@invalid/fake/.code={},
4989 on@invalid/error if real/.code={},
4990 on@invalid/last valid/.code={},
4991 on invalid/.code 2 args={%
4992     \pgfkeysalso{/forest/on@invalid={#1}}%
4993     \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
4994         \def\forest@nodewalk@oninvalid{#1}%
4995         \pgfkeysalso{#2}%
4996     }%
4997 },
4998 define long step={strip fake steps}{n args=1,@bare}{%
4999     \forest@nodewalk@stripfakesteps{\pgfkeysalso{#1}}},
5000 define long step={unique}{n args=1}{%
5001     \begin{group}
5002     \def\forest@nodewalk@unique@temp{}%
5003     \forest@nodewalk{#1}{%
5004         TeX={%
5005             \foresttoget{unique@visited}\forest@temp
5006             \ifx\forest@temp\relax
5007                 \forestoset{unique@visited}{1}%
5008                 \eappto\forest@nodewalk@unique@temp{,id=\forest@cn}%
5009             \fi
5010         }%
5011     }%
5012     \global\let\forest@global@temp\forest@nodewalk@unique@temp

```

```

5013 \endgroup
5014 \pgfkeysalsofrom{\forest@global@temp}%
5015 },
5016 define long step={walk back}{n args=1,@bare}{%
5017 \forestmathtruncatemacro\forest@temp@n{#1}%
5018 \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\ifnum\forest@cn=0\fi}%
5019 \forest@nodewalk@back@updatehistory
5020 },
5021 nodewalk/walk back/.default=1,
5022 define long step={jump back}{n args=1,@bare}{%
5023 \forestmathtruncatemacro\forest@temp@n{(#1)+\ifnum\forest@cn=0 0\else1\fi}%
5024 \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\forest@temp@n-1}%
5025 \forest@nodewalk@back@updatehistory
5026 },
5027 nodewalk/jump back/.default=1,
5028 define long step={back}{n args=1,@bare}{%
5029 \forestmathtruncatemacro\forest@temp@n{#1}%
5030 \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\ifnum\forest@cn=0\fi}%
5031 \forest@nodewalk@back@updatehistory
5032 },
5033 nodewalk/back/.default=1,
5034 define long step={walk forward}{n args=1,@bare}{%
5035 \forestmathtruncatemacro\forest@temp@n{#1}%
5036 \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{0}{\forest@temp@n-1}%
5037 \forest@nodewalk@forward@updatehistory
5038 },
5039 nodewalk/walk forward/.default=1,
5040 define long step={jump forward}{n args=1,@bare}{%
5041 \forestmathtruncatemacro\forest@temp@n{#1}%
5042 \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{\forest@temp@n-1}%
5043 \forest@nodewalk@forward@updatehistory
5044 },
5045 nodewalk/jump forward/.default=1,
5046 define long step={forward}{n args=1,@bare}{%
5047 \forestmathtruncatemacro\forest@temp@n{#1}%
5048 \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{0}{\forest@temp@n-1}%
5049 \forest@nodewalk@forward@updatehistory
5050 },
5051 nodewalk/forward/.default=1,
5052 define long step={last valid'}{@bare}{%
5053 \ifnum\forest@cn=0
5054 \forest@nodewalk@tolastvalid
5055 \forest@nodewalk@makestep
5056 \fi
5057 },
5058 define long step={last valid}{@bare}{%
5059 \forest@nodewalk@tolastvalid
5060 },
5061 define long step={reverse}{n args=1,@bare,make for}{%
5062 \forest@nodewalk{#1,TeX={%
5063 \global\let\forest@global@temp\forest@nodewalk@historyback
5064 \global\let\forest@global@tempn\forest@nodewalk@n
5065 }}{}}%
5066 \forest@nodewalk@walklist{}{\forest@global@temp}{0}{\forest@global@tempn}{\let\forest@cn\forest@nodewalk@n}%
5067 },
5068 define long step={walk and reverse}{n args=1,@bare,make for}{%
5069 \edef\forest@marshal{%
5070 \noexpand\pgfkeysalso{\unexpanded{#1}}%
5071 \noexpand\forest@nodewalk@walklist{}{\noexpand\forest@nodewalk@historyback}{0}{\noexpand\forest@nodewalk@n}%
5072 }\forest@marshal
5073 },

```

```

5074 define long step={sort}{n args=1,@bare,make for}{%
5075   \forest@nodewalk{#1,TeX={%
5076     \global\let\forest@global@temp\forest@nodewalk@historyback
5077     \global\let\forest@global@tempn\forest@nodewalk@n
5078   }}{%
5079     \forest@nodewalk@sortlist{\forest@global@temp}{\forest@global@tempn}\forest@sort@ascending
5080   },
5081 define long step={sort'}{n args=1,@bare,make for}{%
5082   \forest@nodewalk{#1,TeX={%
5083     \global\let\forest@global@temp\forest@nodewalk@historyback
5084     \global\let\forest@global@tempn\forest@nodewalk@n
5085   }}{%
5086     \forest@nodewalk@sortlist{\forest@global@temp}{\forest@global@tempn}\forest@sort@descending
5087   },
5088 define long step={walk and sort}{n args=1,@bare,make for}{% walk as given, then walk sorted
5089   \edef\forest@marshal{%
5090     \noexpand\pgfkeysalso{\unexpanded{#1}}%
5091     \noexpand\forest@nodewalk@sortlist{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-}
5092   }\forest@marshal
5093 },
5094 define long step={walk and sort'}{n args=1,@bare,make for}{%
5095   \edef\forest@marshal{%
5096     \noexpand\pgfkeysalso{\unexpanded{#1}}%
5097     \noexpand\forest@nodewalk@sortlist{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-}
5098   }\forest@marshal
5099 },
5100 declare keylist register=sort by,
5101 copy command key={/forest/sort by'}{/forest/sort by},
5102 sort by={},
5103 define long step={save}{n args=2,@bare,make for}{% #1 = name, #2 = nodewalk
5104   \forest@forthis{%
5105     \forest@nodewalk{#2,TeX={%
5106       \global\let\forest@global@temp\forest@nodewalk@historyback
5107       \global\let\forest@global@tempn\forest@nodewalk@n
5108     }}{%
5109       %
5110       \forest@nodewalk@walklist{}{\forest@global@temp}{0}{\forest@global@tempn}\relax
5111       \csedef\forest@nodewalk@saved@#1{\forest@nodewalk@walklist@walked}%
5112     },
5113 define long step={walk and save}{n args=2,@bare,make for}{% #1 = name, #2 = nodewalk
5114   \edef\forest@marshal{%
5115     \noexpand\pgfkeysalso{\unexpanded{#2}}%
5116     \noexpand\forest@nodewalk@walklist{}{\noexpand\forest@nodewalk@historyback}{0}{\noexpand\forest@nodewalk@n-}
5117   }\forest@marshal
5118   \csedef\forest@nodewalk@saved@#1{\forest@nodewalk@walklist@walked}%
5119 },
5120 define long step={save append}{style,n args=2,@bare,make for}{% #1 = nodewalk name, #2 = nodewalk
5121   save@append@prepend={#1}{#2}{save}{\cseappto}},
5122 define long step={save prepend}{style,n args=2,@bare,make for}{% #1 = nodewalk name, #2 = nodewalk
5123   save@append@prepend={#1}{#2}{save}{\csepreto}},
5124 define long step={walk and save append}{style,n args=2,@bare,make for}{% #1 = nodewalk name, #2 = nodewalk
5125   save@append@prepend={#1}{#2}{walk and save}{\cseappto}},
5126 define long step={walk and save prepend}{style,n args=2,@bare,make for}{% #1 = nodewalk name, #2 = nodewalk
5127   save@append@prepend={#1}{#2}{walk and save}{\csepreto}},
5128 nodewalk/save@append@prepend/.code n args=4{%
5129   % #1 = nodewalk name, #2 = nodewalk
5130   % #3 = "(walk and) save" #4 = \cseappto/\csepreto
5131   \pgfkeysalso{#3={@temp}{#2}}%
5132   \letcs\forest@temp{\forest@nodewalk@saved@@temp}%
5133   #4{\forest@nodewalk@saved@#1}{\expandonce{\forest@temp}}%
5134 },

```

```

5135 nodewalk/save history/.code 2 args={% #1 = back, forward
5136   \csedef{forest@nodewalk@saved@#1}{\forest@nodewalk@historyback}%
5137   \csedef{forest@nodewalk@saved@#2}{\forest@nodewalk@historyforward}%
5138 },
5139 define long step={load}{n args=1,@bare,make for}{%
5140   \forest@nodewalk@walklist}{\csuse{forest@nodewalk@saved@#1}0,}{0}{-1}{\ifnum\forest@nodewalk@cn=0 \else\
5141 },
5142 if in saved nodewalk/.code n args=4{% is node #1 in nodewalk #2; yes: #3, no: #4
5143   \forest@forthis{%
5144     \forest@go{#1}%
5145     \edef\forest@marshal{%
5146       \noexpand\pgfutil@in@{,\forest@cn,}{,\csuse{forest@nodewalk@saved@#2},}%
5147     }\forest@marshal
5148   }%
5149   \ifpgfutil@in@
5150     \@escapeif{\pgfkeysalso{#3}}%
5151   \else
5152     \@escapeif{\pgfkeysalso{#4}}%
5153   \fi
5154 },
5155 where in saved nodewalk/.style n args=4{
5156   for tree={if in saved nodewalk={#1}{#2}{#3}{#4}}
5157 },
5158 nodewalk/options/.code={\forestset{#1}},
5159 nodewalk/TeX/.code={#1},
5160 nodewalk/TeX'/.code={\appto\forest@externalize@loadimages{#1}#1},
5161 nodewalk/TeX''/.code={\appto\forest@externalize@loadimages{#1}},
5162 nodewalk/typeout/.style={TeX={\typeout{#1}}},
5163 % repeat is taken later from /forest/repeat
5164 }
5165 \def\forest@nodewalk@walklist#1#2#3#4#5{%
5166   % #1 = list of preceding, #2 = list to walk
5167   % #3 = from, #4 = to
5168   % #5 = every step code
5169   \let\forest@nodewalk@cn\forest@cn
5170   \edef\forest@marshal{%
5171     \noexpand\forest@nodewalk@walklist@{#1}{#2}{\number\numexpr#3}{\number\numexpr#4}{1}{0}{\unexpanded{#5}}%
5172   }\forest@marshal
5173 }
5174 \def\forest@nodewalk@walklist@#1#2#3#4#5#6#7{%
5175   % #1 = list of walked, #2 = list to walk
5176   % #3 = from, #4 = to
5177   % #5 = current step n, #6 = steps made
5178   % #7 = every step code
5179   \def\forest@nodewalk@walklist@walked{#1}%
5180   \def\forest@nodewalk@walklist@rest{#2}%
5181   \edef\forest@nodewalk@walklist@stepsmade{#6}%
5182   \ifnum#4<0
5183     \forest@temptrue
5184   \else
5185     \ifnum#5>#4\relax
5186       \forest@tempfalse
5187     \else
5188       \forest@temptrue
5189     \fi
5190   \fi
5191   \ifforest@temp
5192     \edef\forest@nodewalk@cn{\forest@csvlist@getfirst@{#2}}%
5193     \ifnum\forest@nodewalk@cn=0
5194       #7%
5195     \else

```

```

5196 \ifnum#5>#3\relax
5197 #7%
5198 \edef\forest@nodewalk@walklist@stepsmade{\number\numexpr#6+1}%
5199 \fi
5200 \forest@csvlist@getfirstrest@{#2}\forest@nodewalk@cn\forest@nodewalk@walklist@rest
5201 \@escapeifif{%
5202 \edef\forest@marshal{%
5203 \noexpand\forest@nodewalk@walklist@
5204 {\forest@nodewalk@cn,#1}\forest@nodewalk@walklist@rest}{#3}{#4}{\number\numexpr#5+1}\forest@nod
5205 }\forest@marshal
5206 }%
5207 \fi
5208 \fi
5209 }
5210
5211 \def\forest@nodewalk@back@updatehistory{%
5212 \ifnum\forest@cn=0
5213 \let\forest@nodewalk@historyback\forest@nodewalk@walklist@rest
5214 \let\forest@nodewalk@historyforward\forest@nodewalk@walklist@walked
5215 \else
5216 \expandafter\forest@csvlist@getfirstrest\expandafter{\forest@nodewalk@walklist@walked}\forest@temp\forest@
5217 \edef\forest@nodewalk@historyback{\forest@temp,\forest@nodewalk@walklist@rest}%
5218 \fi
5219 }
5220 \def\forest@nodewalk@forward@updatehistory{%
5221 \let\forest@nodewalk@historyforward\forest@nodewalk@walklist@rest
5222 \let\forest@nodewalk@historyback\forest@nodewalk@walklist@walked
5223 }
5224 \def\forest@go#1{%
5225 \forest@configured@nodewalk{independent}{inherited}{inherited}{#1}{}%
5226 }
5227 \def\forest@csvlist@getfirst@#1{% assuming that the list is nonempty and finishes with a comma
5228 \forest@csvlist@getfirst@@#1\forest@csvlist@getfirst@@}
5229 \def\forest@csvlist@getfirst@@#1,#2\forest@csvlist@getfirst@@{#1}
5230 \def\forest@csvlist@getrest@#1{% assuming that the list is nonempty and finishes with a comma
5231 \forest@csvlist@getrest@@#1\forest@csvlist@getrest@@}
5232 \def\forest@csvlist@getrest@@#1,#2\forest@csvlist@getrest@@{#2}
5233 \def\forest@csvlist@getfirstrest@#1#2#3{% assuming that the list is nonempty and finishes with a comma
5234 % #1 = list, #2 = cs receiving first, #3 = cs receiving rest
5235 \forest@csvlist@getfirstrest@@#1\forest@csvlist@getfirstrest@@{#2}{#3}}
5236 \def\forest@csvlist@getfirstrest@@#1,#2\forest@csvlist@getfirstrest@@#3#4{%
5237 \def#3{#1}%
5238 \def#4{#2}%
5239 }
5240 \def\forest@nodewalk@stripfakesteps#1{%
5241 % go to the last valid node if the walk contained any nodes, otherwise restore the current node
5242 \edef\forest@marshal{%
5243 \unexpanded{#1}%
5244 \noexpand\ifnum\noexpand\forest@nodewalk@n=\forest@nodewalk@n\relax
5245 \def\noexpand\forest@cn{\forest@cn}%
5246 \noexpand\else
5247 \unexpanded{%
5248 \edef\forest@cn{%
5249 \expandafter\forest@csvlist@getfirst\expandafter{\forest@nodewalk@historyback}%
5250 }%
5251 }%
5252 \noexpand\fi
5253 }\forest@marshal
5254 }
5255 \def\forest@nodewalk@tolastvalid{%
5256 \ifnum\forest@cn=0

```

```

5257 \edef\forest@cn{\expandafter\forest@csvlist@getfirst@\expandafter{\forest@nodewalk@historyback}}%
5258 \ifnum\forest@cn=0
5259 \let\forest@cn\forest@nodewalk@origin
5260 \fi
5261 \fi
5262 }
5263 \def\forest@nodewalk@sortlist#1#2#3{%#1=list,#2=to,#3=asc/desc
5264 \edef\forest@nodewalksort@list{#1}%
5265 \expandafter\forest@nodewalk@sortlist@\expandafter{\number\numexpr#2-#3}%
5266 }
5267 \def\forest@nodewalk@sortlist@#1#2{%#1=to,#2=asc/desc
5268 \safeloop
5269 \unless\ifnum\safeloopn>#1\relax
5270 \expandafter\forest@csvlist@getfirstrest@\expandafter{\forest@nodewalksort@list}\forest@nodewalksort@cn\
5271 \csedef\forest@nodesort@\safeloopn{\forest@nodewalksort@cn}%
5272 \saferepeat
5273 \foreststrget{sort by}\forest@nodesort@sortkey
5274 \forest@sort\forest@nodesort@cmpnodes\forest@nodesort@let#2{1}{#1}%
5275 \def\forest@nodewalksort@sorted{}%
5276 \safeloop
5277 \unless\ifnum\safeloopn>#1\relax
5278 \edef\forest@cn{\csname forest@nodesort@\safeloopn\endcsname}%
5279 \forest@nodewalk@makestep
5280 \saferepeat
5281 }

```

Find minimal/maximal node in a walk.

```

5282 \forestset{
5283   define long step={min}{n args=1,@bare,make for}{% the first min in the argument nodewalk
5284     \forest@nodewalk{#1,TeX={%
5285       \global\let\forest@global@temp\forest@nodewalk@historyback
5286     }}{}%
5287     \forest@nodewalk@minmax{\forest@global@temp}{-1}{<}{\forest@nodewalk@minmax@node,}%
5288   },
5289   define long step={mins}{n args=1,@bare,make for}{% all mins in the argument nodewalk
5290     \forest@nodewalk{#1,TeX={%
5291       \global\let\forest@global@temp\forest@nodewalk@historyback
5292     }}{}%
5293     \forest@nodewalk@minmax{\forest@global@temp}{-1}{<}{\forest@nodewalk@minmax@nodes}%
5294   },
5295   define long step={walk and min}{n args=1,@bare}{%
5296     \edef\forest@marshal{%
5297       \noexpand\pgfkeysalso{\unexpanded{#1}}%
5298       \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
5299     }\forest@marshal
5300   },
5301   define long step={walk and mins}{n args=1,@bare}{%
5302     \edef\forest@marshal{%
5303       \noexpand\pgfkeysalso{\unexpanded{#1}}%
5304       \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
5305     }\forest@marshal
5306   },
5307   define long step={min in nodewalk}{@bare}{% find the first min in the preceding nodewalk, step to it
5308     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{\forest@nodewalk@minmax@node,}%
5309   },
5310   define long step={mins in nodewalk}{@bare}{% find mins in the preceding nodewalk, step to mins
5311     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{\forest@nodewalk@minmax@nodes}%
5312   },
5313   define long step={min in nodewalk'}{@bare}{% find the first min in the preceding nodewalk, step to min in h
5314     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{}%
5315   },

```



```

5316 %
5317 define long step={max}{n args=1,@bare,make for}{% the first max in the argument nodewalk
5318   \forest@nodewalk{#1,TeX={%
5319     \global\let\forest@global@temp\forest@nodewalk@historyback
5320   }}{%
5321     \forest@nodewalk@minmax{\forest@global@temp}{-1}{>}{\forest@nodewalk@minmax@node,}%
5322   },
5323 define long step={maxs}{n args=1,@bare,make for}{% all maxs in the argument nodewalk
5324   \forest@nodewalk{#1,TeX={%
5325     \global\let\forest@global@temp\forest@nodewalk@historyback
5326   }}{%
5327     \forest@nodewalk@minmax{\forest@global@temp}{-1}{>}{\forest@nodewalk@minmax@nodes}%
5328   },
5329 define long step={walk and max}{n args=1,@bare}{%
5330   \edef\forest@marshal{%
5331     \noexpand\pgfkeysalso{\unexpanded{#1}}%
5332     \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\forest@nodewalk@n}%
5333   }\forest@marshal
5334 },
5335 define long step={walk and maxs}{n args=1,@bare}{%
5336   \edef\forest@marshal{%
5337     \noexpand\pgfkeysalso{\unexpanded{#1}}%
5338     \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\forest@nodewalk@n}%
5339   }\forest@marshal
5340 },
5341 define long step={max in nodewalk}{@bare}{% find the first max in the preceding nodewalk, step to it
5342   \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{\forest@nodewalk@minmax@node,}%
5343 },
5344 define long step={maxs in nodewalk}{@bare}{% find maxs in the preceding nodewalk, step to maxs
5345   \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{\forest@nodewalk@minmax@nodes}%
5346 },
5347 define long step={max in nodewalk'}{@bare}{% find the first max in the preceding nodewalk, step to max in h
5348   \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{}%
5349 },
5350 }
5351
5352 \def\forest@nodewalk@minmax#1#2#3#4{%
5353   % #1 = list of nodes
5354   % #2 = max index in list (start with 1)
5355   % #3 = min/max = ascending/descending = </>
5356   % #4 = how many results? 1 = {\forest@nodewalk@minmax@node,}, all={\forest@nodewalk@minmax@nodes}, walk in
5357   \forest@get{sort by}\forest@nodesort@sortkey
5358   \edef\forest@nodewalk@minmax@N{\number\numexpr#2}%
5359   \edef\forest@nodewalk@minmax@n{%
5360     \edef\forest@nodewalk@minmax@list{#1}%
5361     \def\forest@nodewalk@minmax@nodes{%
5362       \def\forest@nodewalk@minmax@node{%
5363         \ifempty{\forest@nodewalk@minmax@list}{%
5364         }{%
5365           \safeloop
5366           \expandafter\forest@csvlist@getfirstrest@\expandafter{\forest@nodewalk@minmax@list}\forest@nodewalk@minmax@node
5367           \ifnum\forest@nodewalk@minmax@cn=0 \else
5368             \ifempty{\forest@nodewalk@minmax@node}{%
5369               \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
5370               \edef\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,}%
5371               \edef\forest@nodewalk@minmax@n{\safeloopn}%
5372             }{%
5373               \csedef{forest@nodesort@1}{\forest@nodewalk@minmax@node}%
5374               \csedef{forest@nodesort@2}{\forest@nodewalk@minmax@cn}%
5375               \forest@nodesort@cmpnodes{2}{1}%
5376               \if=\forest@sort@cmp@result

```

```

5377         \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
5378         \epreto\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,}%
5379         \edef\forest@nodewalk@minmax@n{\safeloopn}%
5380     \else
5381         \if#3\forest@sort@cmp@result
5382             \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
5383             \edef\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,}%
5384             \edef\forest@nodewalk@minmax@n{\safeloopn}%
5385         \fi
5386     \fi
5387 }%
5388 \fi
5389 \ifdefempty{\forest@nodewalk@minmax@list}{\forest@tempfalse}{\forest@temptrue}%
5390 \ifnum\safeloopn=\forest@nodewalk@minmax@N\relax\forest@temptrue\fi
5391 \ifforest@temp
5392 \saferrepeat
5393 \edef\forest@nodewalk@minmax@list{#4}%
5394 \ifdefempty\forest@nodewalk@minmax@list{%
5395     \forestset{nodewalk/jump back=\forest@nodewalk@minmax@n-1}% CHECK
5396 }{%
5397     \safeloop
5398     \expandafter\forest@csvglist@getfirstrest@\expandafter{\forest@nodewalk@minmax@list}\forest@cn\forest@
5399     \forest@nodewalk@makestep
5400     \ifdefempty{\forest@nodewalk@minmax@list}{\forest@tempfalse}{\forest@temptrue}%
5401     \ifforest@temp
5402     \saferrepeat
5403     }%
5404 }%
5405 }

```

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.

```

5406 \newtoks\forest@nodewalk@shortsteps@resolution
5407 \newif\ifforest@nodewalk@areshortsteps
5408 \pgfqkeys{/forest/nodewalk}{
5409     .unknown/.code={%
5410         \forest@nodewalk@areshortstepsfalse
5411         \pgfkeysifdefined{/forest/\pgfkeyscurrentname/@.cmd}{%
5412             }{%
5413                 \ifx\pgfkeyscurrentvalue\pgfkeysnovalue@text % no value, so possibly short steps
5414                     \forest@nodewalk@shortsteps@resolution{}%
5415                     \forest@nodewalk@areshortstepstrue
5416                     \expandafter\forest@nodewalk@shortsteps\pgfkeyscurrentname=====,% "=" and "," cannot be short st
5417                 \fi
5418             }%
5419         \ifforest@nodewalk@areshortsteps
5420             \@escapeif{\expandafter\pgfkeysalso\expandafter{\the\forest@nodewalk@shortsteps@resolution}}%
5421         \else
5422             \@escapeif{\pgfkeysalso{/forest/\pgfkeyscurrentname={#1}}}%
5423         \fi
5424     },
5425 }
5426 \def\forest@nodewalk@shortsteps{%
5427     \futurelet\forest@nodewalk@nexttoken\forest@nodewalk@shortsteps@
5428 }
5429 \def\forest@nodewalk@shortsteps@{%
5430     \ifx\forest@nodewalk@nexttoken=%
5431         \let\forest@nodewalk@nextop\forest@nodewalk@shortsteps@end
5432     \else
5433         \ifx\forest@nodewalk@nexttoken\bgroup

```

```

5434 \letcs\forest@nodewalk@nextop{forest@shortstep@group}%
5435 \else
5436 \let\forest@nodewalk@nextop\forest@nodewalk@shortsteps@@
5437 \fi
5438 \fi
5439 \forest@nodewalk@nextop
5440 }
5441 \def\forest@nodewalk@shortsteps@@#1{%
5442 \ifcsdef{forest@shortstep@#1}{%
5443 \csname forest@shortstep@#1\endcsname
5444 }{%
5445 \forest@nodewalk@aeshortstepsfalse
5446 \forest@nodewalk@shortsteps@end
5447 }%
5448 }
5449 % in the following definitions:
5450 % #1 = short step
5451 % #2 = (long) step, or a style in /forest/nodewalk (taking n args)
5452 \csdef{forest@nodewalk@defshortstep@0@args}#1#2{%
5453 \csdef{forest@shortstep@#1}{%
5454 \apptotoks\forest@nodewalk@shortsteps@resolution{,#2}%
5455 \forest@nodewalk@shortsteps}}
5456 \csdef{forest@nodewalk@defshortstep@1@args}#1#2{%
5457 \csdef{forest@shortstep@#1}##1{%
5458 \edef\forest@marshal####1{#2}%
5459 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}}%
5460 \forest@nodewalk@shortsteps}}
5461 \csdef{forest@nodewalk@defshortstep@2@args}#1#2{%
5462 \csdef{forest@shortstep@#1}##1##2{%
5463 \edef\forest@marshal####1####2{#2}%
5464 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}}%
5465 \forest@nodewalk@shortsteps}}
5466 \csdef{forest@nodewalk@defshortstep@3@args}#1#2{%
5467 \csdef{forest@shortstep@#1}##1##2##3{%
5468 \edef\forest@marshal####1####2####3{#2}%
5469 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}}%
5470 \forest@nodewalk@shortsteps}}
5471 \csdef{forest@nodewalk@defshortstep@4@args}#1#2{%
5472 \csdef{forest@shortstep@#1}##1##2##3##4{%
5473 \edef\forest@marshal####1####2####3####4{#2}%
5474 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}}%
5475 \forest@nodewalk@shortsteps}}
5476 \csdef{forest@nodewalk@defshortstep@5@args}#1#2{%
5477 \csdef{forest@shortstep@#1}##1##2##3##4##5{%
5478 \edef\forest@marshal####1####2####3####4####5{#2}%
5479 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}}%
5480 \forest@nodewalk@shortsteps}}
5481 \csdef{forest@nodewalk@defshortstep@6@args}#1#2{%
5482 \csdef{forest@shortstep@#1}##1##2##3##4##5##6{%
5483 \edef\forest@marshal####1####2####3####4####5####6{#2}%
5484 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}}%
5485 \forest@nodewalk@shortsteps}}
5486 \csdef{forest@nodewalk@defshortstep@7@args}#1#2{%
5487 \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7{%
5488 \edef\forest@marshal####1####2####3####4####5####6####7{#2}%
5489 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}}%
5490 \forest@nodewalk@shortsteps}}
5491 \csdef{forest@nodewalk@defshortstep@8@args}#1#2{%
5492 \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7##8{%
5493 \edef\forest@marshal####1####2####3####4####5####6####7####8{#2}%
5494 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}{##8}}%

```

```

5495 \forest@nodewalk@shortsteps}}
5496 \csdef{forest@nodewalk@defshortstep@9@args}#1#2{%
5497 \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7##8##9{%
5498 \edef\forest@marshal#####1#####2#####3#####4#####5#####6#####7#####8#####9{#2}%
5499 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}{##8}
5500 \forest@nodewalk@shortsteps}}
5501 \forestset{
5502   define short step/.code n args=3{% #1 = short step, #2 = n args, #3 = long step
5503     \csname forest@nodewalk@defshortstep@#2@args\endcsname{#1}{#3}%
5504   },
5505 }
5506 \def\forest@nodewalk@shortsteps@end#1,{
  Define short-form steps.

5507 \forestset{
5508   define short step={group}{1}{group={#1}}, % {braces} are special
5509   define short step={p}{0}{previous},
5510   define short step={n}{0}{next},
5511   define short step={u}{0}{parent},
5512   define short step={s}{0}{sibling},
5513   define short step={c}{0}{current},
5514   define short step={o}{0}{origin},
5515   define short step={r}{0}{root},
5516   define short step={R}{0}{root'},
5517   define short step={P}{0}{previous leaf},
5518   define short step={N}{0}{next leaf},
5519   define short step={F}{0}{first leaf},
5520   define short step={L}{0}{last leaf},
5521   define short step={>}{0}{next on tier},
5522   define short step={<}{0}{previous on tier},
5523   define short step={1}{0}{n=1},
5524   define short step={2}{0}{n=2},
5525   define short step={3}{0}{n=3},
5526   define short step={4}{0}{n=4},
5527   define short step={5}{0}{n=5},
5528   define short step={6}{0}{n=6},
5529   define short step={7}{0}{n=7},
5530   define short step={8}{0}{n=8},
5531   define short step={9}{0}{n=9},
5532   define short step={l}{0}{last},
5533   define short step={b}{0}{back},
5534   define short step={f}{0}{forward},
5535   define short step={v}{0}{last valid},
5536   define short step={*}{2}{repeat={#1}{#2}},
5537   for 1/.style={for nodewalk={n=1}{#1}},
5538   for 2/.style={for nodewalk={n=2}{#1}},
5539   for 3/.style={for nodewalk={n=3}{#1}},
5540   for 4/.style={for nodewalk={n=4}{#1}},
5541   for 5/.style={for nodewalk={n=5}{#1}},
5542   for 6/.style={for nodewalk={n=6}{#1}},
5543   for 7/.style={for nodewalk={n=7}{#1}},
5544   for 8/.style={for nodewalk={n=8}{#1}},
5545   for 9/.style={for nodewalk={n=9}{#1}},
5546   for -1/.style={for nodewalk={n'=1}{#1}},
5547   for -2/.style={for nodewalk={n'=2}{#1}},
5548   for -3/.style={for nodewalk={n'=3}{#1}},
5549   for -4/.style={for nodewalk={n'=4}{#1}},
5550   for -5/.style={for nodewalk={n'=5}{#1}},
5551   for -6/.style={for nodewalk={n'=6}{#1}},
5552   for -7/.style={for nodewalk={n'=7}{#1}},
5553   for -8/.style={for nodewalk={n'=8}{#1}},

```

```

5554   for -9/.style={for nodewalk={n'=9}{#1}},
5555 }

```

Define multiple-step walks.

```

5556 \forestset{
5557   define long step={tree}{\forest@node@foreach{\forest@nodewalk@makestep}},
5558   define long step={tree reversed}{\forest@node@foreach@reversed{\forest@nodewalk@makestep}},
5559   define long step={tree children-first}{\forest@node@foreach@childrenfirst{\forest@nodewalk@makestep}},
5560   define long step={tree children-first reversed}{\forest@node@foreach@childrenfirst@reversed{\forest@nodewalk@makestep}},
5561   define long step={tree breadth-first}{\forest@node@foreach@breadthfirst{-1}{\forest@nodewalk@makestep}},
5562   define long step={tree breadth-first reversed}{\forest@node@foreach@breadthfirst@reversed{-1}{\forest@nodewalk@makestep}},
5563   define long step={descendants}{\forest@node@foreachdescendant{\forest@nodewalk@makestep}},
5564   define long step={descendants reversed}{\forest@node@foreachdescendant@reversed{\forest@nodewalk@makestep}},
5565   define long step={descendants children-first}{\forest@node@foreachdescendant@childrenfirst{\forest@nodewalk@makestep}},
5566   define long step={descendants children-first reversed}{\forest@node@foreachdescendant@childrenfirst@reversed{\forest@nodewalk@makestep}},
5567   define long step={descendants breadth-first}{\forest@node@foreach@breadthfirst{0}{\forest@nodewalk@makestep}},
5568   define long step={descendants breadth-first reversed}{\forest@node@foreach@breadthfirst@reversed{0}{\forest@nodewalk@makestep}},
5569   define long step={level}{n args=1}{%
5570     \forestmathtruncatemacro\forest@temp{#1}%
5571     \edef\forest@marshal{%
5572       \noexpand\forest@node@foreach@breadthfirst
5573       {\forest@temp}%
5574       {\noexpand\ifnum\noexpand\forest@temp{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexpand\forest@temp{level}<\forest@temp\relax\noexpand\forest@nodewalk@makestep}%
5575     }\forest@marshal
5576   },
5577   define long step={level>}{n args=1}{%
5578     \forestmathtruncatemacro\forest@temp{#1}%
5579     \edef\forest@marshal{%
5580       \noexpand\forest@node@foreach@breadthfirst
5581       {-1}%
5582       {\noexpand\ifnum\noexpand\forest@temp{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@makestep}%
5583     }\forest@marshal
5584   },
5585   define long step={level<}{n args=1}{%
5586     \forestmathtruncatemacro\forest@temp{(#1)-1}%
5587     \ifnum\forest@temp=-1
5588       % special case, as \forest@node@foreach@breadthfirst uses level<0 as a signal for unlimited max level
5589       \ifnum\forest@temp{level}=0
5590         \forest@nodewalk@makestep
5591       \fi
5592     \else
5593       \edef\forest@marshal{%
5594         \noexpand\forest@node@foreach@breadthfirst
5595         {\forest@temp}%
5596         {\noexpand\forest@nodewalk@makestep}%
5597       }\forest@marshal
5598     \fi
5599   },
5600   define long step={level reversed}{n args=1}{%
5601     \forestmathtruncatemacro\forest@temp{#1}%
5602     \edef\forest@marshal{%
5603       \noexpand\forest@node@foreach@breadthfirst@reversed
5604       {\forest@temp}%
5605       {\noexpand\ifnum\noexpand\forest@temp{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexpand\forest@temp{level}<\forest@temp\relax\noexpand\forest@nodewalk@makestep}%
5606     }\forest@marshal
5607   },
5608   define long step={level reversed>}{n args=1}{%
5609     \forestmathtruncatemacro\forest@temp{#1}%
5610     \edef\forest@marshal{%
5611       \noexpand\forest@node@foreach@breadthfirst@reversed
5612       {-1}%

```

```

5613     {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
5614     }\forest@marshal
5615 },
5616 define long step={level reversed<}{n args=1}{%
5617     \forestmathtruncatemacro\forest@temp{(#1)-1}%
5618     \edef\forest@marshal{%
5619         \noexpand\forest@node@foreach@breadthfirst@reversed
5620         {\forest@temp}%
5621         {\noexpand\forest@nodewalk@makestep}%
5622     }\forest@marshal
5623 },
5624 %
5625 define long step={relative level}{n args=1}{%
5626     \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
5627     \edef\forest@marshal{%
5628         \noexpand\forest@node@foreach@breadthfirst
5629         {\forest@temp}%
5630         {\noexpand\ifnum\noexpand\forestove{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexp
5631     }\forest@marshal
5632 },
5633 define long step={relative level>}{n args=1}{%
5634     \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
5635     \edef\forest@marshal{%
5636         \noexpand\forest@node@foreach@breadthfirst
5637         {-1}%
5638         {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
5639     }\forest@marshal
5640 },
5641 define long step={relative level<}{n args=1}{%
5642     \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}-1}%
5643     \edef\forest@marshal{%
5644         \noexpand\forest@node@foreach@breadthfirst
5645         {\forest@temp}%
5646         {\noexpand\forest@nodewalk@makestep}%
5647     }\forest@marshal
5648 },
5649 define long step={relative level reversed}{n args=1}{%
5650     \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
5651     \edef\forest@marshal{%
5652         \noexpand\forest@node@foreach@breadthfirst@reversed
5653         {\forest@temp}%
5654         {\noexpand\ifnum\noexpand\forestove{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexp
5655     }\forest@marshal
5656 },
5657 define long step={relative level reversed>}{n args=1}{%
5658     \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
5659     \edef\forest@marshal{%
5660         \noexpand\forest@node@foreach@breadthfirst@reversed
5661         {-1}%
5662         {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
5663     }\forest@marshal
5664 },
5665 define long step={relative level reversed<}{n args=1}{%
5666     \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}-1}%
5667     \edef\forest@marshal{%
5668         \noexpand\forest@node@foreach@breadthfirst@reversed
5669         {\forest@temp}%
5670         {\noexpand\forest@nodewalk@makestep}%
5671     }\forest@marshal
5672 },
5673 define long step={leaves}{}{%

```

```

5674 \forest@node@foreach{%
5675 \ifnum\forest@ve{n children}=0
5676 \forest@nodewalk@makestep
5677 \fi
5678 }%
5679 },
5680 define long step={-level}{n args=1,style}{%
5681 unique={branch={leaves,{group={repeat={#1}{parent}}}}}}
5682 },
5683 define long step={-level'}{n args=1,style}{%
5684 unique={on invalid={fake}{branch={leaves,{group={repeat={#1}{parent}}}}}}
5685 },
5686 define long step={children}{\forest@node@foreachchild{\forest@nodewalk@makestep}},
5687 define long step={children reversed}{\forest@node@foreachchild@reversed{\forest@nodewalk@makestep}},
5688 define long step={current and following siblings}{\forest@node@@forselfandfollowingsiblings{\forest@nodewalk@makestep}},
5689 define long step={following siblings}{style}{if nodewalk valid={next}{fake=next,current and following siblings}{\forest@node@@forselfandfollowingsiblings{\forest@nodewalk@makestep}}},
5690 define long step={current and preceding siblings}{\forest@node@@forselfandprecedingsiblings{\forest@nodewalk@makestep}},
5691 define long step={preceding siblings}{style}{if nodewalk valid={previous}{fake=previous,current and preceding siblings}{\forest@node@@forselfandprecedingsiblings{\forest@nodewalk@makestep}}},
5692 define long step={current and following siblings reversed}{\forest@node@@forselfandfollowingsiblings@reversed{\forest@nodewalk@makestep}},
5693 define long step={following siblings reversed}{style}{fake=next,current and following siblings reversed},
5694 define long step={current and preceding siblings reversed}{\forest@node@@forselfandprecedingsiblings@reversed{\forest@nodewalk@makestep}},
5695 define long step={preceding siblings reversed}{style}{fake=previous,current and preceding siblings reversed},
5696 define long step={siblings}{style}{for nodewalk'={preceding siblings},following siblings},
5697 define long step={siblings reversed}{style}{for nodewalk'={following siblings reversed},preceding siblings},
5698 define long step={current and siblings}{style}{for nodewalk'={preceding siblings},current and following siblings},
5699 define long step={current and siblings reversed}{style}{for nodewalk'={current and following siblings reversed},current and preceding siblings},
5700 define long step={ancestors}{style}{while={}{parent},last valid},
5701 define long step={current and ancestors}{style}{current,ancestors},
5702 define long step={following nodes}{style}{while={}{next node},last valid},
5703 define long step={preceding nodes}{style}{while={}{previous node},last valid},
5704 define long step={current and following nodes}{style}{current,following nodes},
5705 define long step={current and preceding nodes}{style}{current,preceding nodes},
5706 }
5707 \let\forest@nodewalkstephandler@styletrueorfalse\forest@nodewalkstephandler@styletrue

```

7.4 Dynamic tree

```

5708 \def\forest@last@node{0}
5709 \csdef{forest@nodewalk@saved@dynamic nodes}{}
5710 \def\forest@nodehandleby@name@nodewalk@or@bracket#1{%
5711 \ifx\pgfkeysnovalue#1%
5712 \edef\forest@last@node{\forest@node@Nametoid{forest@last@node}}%
5713 \else
5714 \forest@nodehandleby@nnb@checkfirst#1\forest@END
5715 \fi
5716 }
5717 \def\forest@nodehandleby@nnb@checkfirst#1#2\forest@END{%
5718 \ifx[#1%
5719 \forest@create@node{#1#2}%
5720 \cseappto{forest@nodewalk@saved@dynamic nodes}{\forest@last@node,}%
5721 \else
5722 \forest@forthis{%
5723 \forest@nameandgo{#1#2}%
5724 \ifnum\forest@cn=0
5725 \PackageError{forest}{Cannot use a dynamic key on the invalid node}{}%
5726 \fi
5727 \let\forest@last@node\forest@cn
5728 }%
5729 \fi
5730 }

```

```

5731 \def\forest@create@node#1{% #1=bracket representation
5732   \bracketParse{\forest@create@collectafterthought}%
5733   \forest@last@node=#1\forest@end@create@node
5734 }
5735 \def\forest@create@collectafterthought#1\forest@end@create@node{%
5736   \forest@node@Foreach{\forest@last@node}{%
5737     \forest@toletto{delay}{given options}%
5738     \forest@set{given options}{}%
5739   }%
5740   \forest@oappto{\forest@last@node}{delay}{,\unexpanded{#1}}%
5741   \forest@set{\forest@last@node}{given options}{delay={}}%
5742 }
5743 \def\forest@create@node@and@process@given@options#1{% #1=bracket representation
5744   \bracketParse{\forest@createandprocess@collectafterthought}%
5745   \forest@last@node=#1\forest@end@create@node
5746 }
5747 \def\forest@createandprocess@collectafterthought#1\forest@end@create@node{%
5748   \forest@node@Compute@numeric@ts@info{\forest@last@node}%
5749   \forest@saveandrestoremacro\forest@root{%
5750     \let\forest@root\forest@last@node
5751     \forest@set{process keylist=given options}%
5752   }%
5753 }
5754 \def\forest@saveandrestoremacro#1#2{% #1 = the (zero-arg) macro to save before and restore after processing c
5755   \edef\forest@marshal{%
5756     \unexpanded{#2}%
5757     \noexpand\def\noexpand#1{\expandonce{#1}}%
5758   }\forest@marshal
5759 }
5760 \def\forest@saveandrestoreifcs#1#2{% #1 = the if cs to save before and restore after processing code in #2
5761   \edef\forest@marshal{%
5762     \unexpanded{#2}%
5763     \ifbool{#1}{\noexpand\setbool{#1}{true}}{\noexpand\setbool{#1}{false}}%
5764   }\forest@marshal
5765 }
5766 \def\forest@globalsaveandrestoreifcs#1#2{% #1 = the if cs to save before and restore after processing code in
5767   \edef\forest@marshal{%
5768     \unexpanded{#2}%
5769     \ifbool{#1}{\global\noexpand\setbool{#1}{true}}{\global\noexpand\setbool{#1}{false}}%
5770   }\forest@marshal
5771 }
5772 \def\forest@saveandrestoretoks#1#2{% #1 = the toks to save before and restore after processing code in #2
5773   \edef\forest@marshal{%
5774     \unexpanded{#2}%
5775     \noexpand#1{\the#1}%
5776   }\forest@marshal
5777 }
5778 \def\forest@saveandrestoreregister#1#2{% #1 = the register to save before and restore after processing code i
5779   \edef\forest@marshal{%
5780     \unexpanded{#2}%
5781     \noexpand\forest@set{#1}{\forestregister{#1}}%
5782   }\forest@marshal
5783 }
5784 \forestset{
5785   save and restore register/.code 2 args={%
5786     \forest@saveandrestoreregister{#1}%
5787     \pgfkeysalso{#2}%
5788   }%
5789 },
5790 }
5791 \def\forest@remove@node#1{%

```



```

5792 \ifforestdebugdynamics\forestdebug@dynamics{before removing #1}\fi
5793 \forest@node@Remove{#1}%
5794 }
5795 \def\forest@append@node#1#2{%
5796 \ifforestdebugdynamics\forestdebug@dynamics{before appending #2 to #1}\fi
5797 \forest@dynamic@circularitytest{#2}{#1}{append}%
5798 \forest@node@Remove{#2}%
5799 \forest@node@Append{#1}{#2}%
5800 }
5801 \def\forest@prepend@node#1#2{%
5802 \ifforestdebugdynamics\forestdebug@dynamics{before prepending #2 to #1}\fi
5803 \forest@dynamic@circularitytest{#2}{#1}{prepend}%
5804 \forest@node@Remove{#2}%
5805 \forest@node@Prepend{#1}{#2}%
5806 }
5807 \def\forest@insertafter@node#1#2{%
5808 \ifforestdebugdynamics\forestdebug@dynamics{before inserting #2 after #1}\fi
5809 \forest@node@Remove{#2}%
5810 \forest@node@Insertafter{\forestOve{#1}{@parent}}{#2}{#1}%
5811 }
5812 \def\forest@insertbefore@node#1#2{%
5813 \ifforestdebugdynamics\forestdebug@dynamics{before inserting #2 before #1}\fi
5814 \forest@node@Remove{#2}%
5815 \forest@node@Insertbefore{\forestOve{#1}{@parent}}{#2}{#1}%
5816 }
5817 \def\forest@set@root#1#2{%
5818 \ifforestdebugdynamics\forestdebug@dynamics{before setting #1 as root}\fi
5819 \def\forest@root{#2}%
5820 }
5821 \def\forest@dynamic@circularitytest#1#2#3{%
5822 % #1=potential ancestor, #2=potential descendant, #3=message prefix
5823 \ifnum#1=#2
5824 \forest@circularityerror{#1}{#2}{#3}%
5825 \else
5826 \forest@for@node{#1}{%
5827 \forest@if@ancestorof{#2}{\forest@circularityerror{#1}{#2}{#3}}{}}%
5828 }%
5829 \fi
5830 }
5831 \def\forest@circularityerror#1#2#3{%
5832 \forestdebug@typeouttrees{\forest@temp}%
5833 \PackageError{forest}{#3ing node id=#1 to id=#2 would result in a circular tree\MessageBreak forest of ids:
5834 }%
5835 \def\forestdebug@dynamics#1{%
5836 \forestdebug@typeouttrees\forest@temp
5837 \typeout{#1: \forest@temp}%
5838 }
5839 \def\forest@appto@do@dynamics#1#2{%
5840 \forest@node@handleby@name@nodewalk@or@bracket{#2}%
5841 \ifcase\forest@dynamics@copyhow\relax\or
5842 \forest@tree@copy{\forest@last@node}\forest@last@node
5843 \or
5844 \forest@node@copy{\forest@last@node}\forest@last@node
5845 \fi
5846 \forest@node@if@named@defined{forest@last@node}{%
5847 \forest@depreto{\forest@last@node}{delay}
5848 {for id={\forest@node@Nametoid{forest@last@node}}{alias=forest@last@node}},}%
5849 }{}%
5850 \edef\forest@marshal{%
5851 \noexpand\apptotoks\noexpand\forest@do@dynamics{%
5852 \noexpand#1{\forest@cn}{\forest@last@node}}%

```

```

5853 } \forest@marshal
5854 }
5855 \forestset{%
5856   create/.code={%
5857     \forest@create@node{#1}%
5858     \forest@for@node{\forest@last@node}{%
5859       \forest@node@setalias{\forest@last@node}%
5860       \cseappto{\forest@nodewalk@saved@dynamic nodes}{\forest@last@node,}%
5861     }%
5862   },
5863   create'/.code={%
5864     \forest@create@node@and@process@given@options{#1}%
5865     \forest@for@node{\forest@last@node}{%
5866       \forest@node@setalias{\forest@last@node}%
5867       \cseappto{\forest@nodewalk@saved@dynamic nodes}{\forest@last@node,}%
5868     }%
5869   },
5870   append/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@append@node{#1}},
5871   prepend/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@prepend@node{#1}},
5872   insert after/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@insertafter@node{#1}},
5873   insert before/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@insertbefore@node{#1}},
5874   append'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@append@node{#1}},
5875   prepend'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@prepend@node{#1}},
5876   insert after'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@insertafter@node{#1}},
5877   insert before'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@insertbefore@node{#1}},
5878   append''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@append@node{#1}},
5879   prepend''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@prepend@node{#1}},
5880   insert after''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@insertafter@node{#1}},
5881   insert before''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@insertbefore@node{#1}},
5882   remove/.code={%
5883     \pgfkeysalso{alias=forest@last@node}%
5884     \cseappto{\forest@nodewalk@saved@dynamic nodes}{\forest@cn,}%
5885     \expandafter\apptotoks\expandafter\forest@do@dynamics\expandafter{%
5886       \expandafter\forest@remove@node\expandafter{\forest@cn}}%
5887   },
5888   set root/.code={%
5889     \def\forest@dynamics@copyhow{0}%
5890     \forest@appto@do@dynamics\forest@set@root{#1}%
5891   },
5892   replace by/.code={\forest@replaceby@code{#1}{insert after}},
5893   replace by'/.code={\forest@replaceby@code{#1}{insert after'}},
5894   replace by''/.code={\forest@replaceby@code{#1}{insert after''}},
5895   sort/.code={%
5896     \eapptotoks\forest@do@dynamics{%
5897       \def\noexpand\forest@nodesort@sortkey{\forestrv{sort by}}%
5898       \noexpand\forest@nodesort\noexpand\forest@sort@ascending{\forest@cn}
5899     }%
5900   },
5901   sort'/.code={%
5902     \eapptotoks\forest@do@dynamics{%
5903       \def\noexpand\forest@nodesort@sortkey{\forestrv{sort by}}%
5904       \noexpand\forest@nodesort\noexpand\forest@sort@descending{\forest@cn}
5905     }%
5906   },
5907 }
5908 \def\forest@replaceby@code#1#2{%#1=node spec,#2=insert after['']['']}
5909 \ifnum\forestove{@parent}=0
5910   \cseappto{\forest@nodewalk@saved@dynamic nodes}{\forest@cn,}%
5911   \pgfkeysalso{alias=forest@last@node,set root={#1}}%
5912 \else
5913   \cseappto{\forest@nodewalk@saved@dynamic nodes}{\forest@cn,}%

```

```

5914 \pgfkeysalso{alias=forest@last@node,#2={#1}}%
5915 \eapptotoks\forest@do@ynamics{%
5916 \noexpand\ifnum\noexpand\forest@ve{\forest@cn}{@parent}=\forest@ve{@parent}
5917 \noexpand\forest@remove@node{\forest@cn}%
5918 \noexpand\fi
5919 }%
5920 \fi
5921 }
5922 \def\forest@nodesort#1#2{% #1 = direction, #2 = parent node
5923 \ifforestdebugdynamics\forestdebug@ynamics{before sorting children of #2}\fi
5924 \forest@for@node{#2}{\forest@nodesort@#1}%
5925 \ifforestdebugdynamics\forestdebug@ynamics{after sorting children of #2}\fi
5926 }
5927 \def\forest@nodesort@#1{%
5928 % prepare the array of child ids
5929 \c@pgf@counta=0
5930 \forest@toget{@first}\forest@nodesort@id
5931 \forest@loop
5932 \ifnum\forest@nodesort@id>0
5933 \advance\c@pgf@counta 1
5934 \csedef\forest@nodesort@the\c@pgf@counta{\forest@nodesort@id}%
5935 \forest@get{\forest@nodesort@id}{@next}\forest@nodesort@id
5936 \forest@repeat
5937 % sort
5938 \forest@toget{n children}\forest@nodesort@n
5939 \forest@sort\forest@nodesort@cmpnodes\forest@nodesort@let#1{1}{\forest@nodesort@n}%
5940 % remove all children
5941 \forest@toget{@first}\forest@nodesort@id
5942 \forest@loop
5943 \ifnum\forest@nodesort@id>0
5944 \forest@node@Remove{\forest@nodesort@id}%
5945 \forest@toget{@first}\forest@nodesort@id
5946 \forest@repeat
5947 % insert the children in new order
5948 \c@pgf@counta=0
5949 \forest@loop
5950 \ifnum\c@pgf@counta<\forest@nodesort@n\relax
5951 \advance\c@pgf@counta 1
5952 \forest@node@append{\csname forest@nodesort@the\c@pgf@counta\endcsname}%
5953 \forest@repeat
5954 }
5955 \def\forest@nodesort@cmpnodes#1#2{%
5956 \expandafter\forest@nodesort@cmpnodes@\forest@nodesort@sortkey,\forest@END{#1}{#2}%
5957 }
5958 \def\forest@nodesort@cmpnodes@#1,#2\forest@END#3#4{%
5959 % #1=process ins+arg for this dimension, #2=for next dimensions
5960 % #3, #4 = node ids
5961 {%
5962 \forest@for@node{\csname forest@nodesort@#3\endcsname}{%
5963 \forestmathsetmacro\forest@nodesort@resulta{#1}%
5964 }%
5965 \forest@for@node{\csname forest@nodesort@#4\endcsname}{%
5966 \forestmathsetmacro\forest@nodesort@resultb{#1}%
5967 }%
5968 \ifx\forestmathresulttype\forestmath@type@generic
5969 \forest@cmp@error{\forest@nodesort@resulta}{\forest@nodesort@resultb}%
5970 \fi
5971 \edef\forest@temp{%
5972 \noexpand\forest@nodesort@cmp
5973 {\expandonce{\forest@nodesort@resulta}}%
5974 {\expandonce{\forest@nodesort@resultb}}%

```

```

5975 }%
5976 \xdef\forest@global@temp{\forest@temp}%
5977 }%
5978 \if=\forest@global@temp
5979 \let\forest@next\forest@nodesort@cmpnodes@
5980 \else
5981 \let\forest@next\forest@nodesort@cmpnodes@finish
5982 \fi
5983 \ifstrempy{#2}{\let\forest@next\forest@nodesort@cmpnodes@finish}{}%
5984 \forest@next#2\forest@END{#3}{#4}%
5985 }
5986 \def\forest@nodesort@cmpnodes@finish#1\forest@END#2#3{%
5987 \let\forest@sort@cmp@result\forest@global@temp
5988 }

```

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

```

5989 \def\forest@nodesort@cmp{\csname fRsT@nsc@\forestmathresulttype\endcsname}
5990 \def\forest@nodesort@cmp#1{\csname fRsT@nsc@#1\endcsname}
5991 \def\forest@nodesort@cmp#1#2{\ifnum#1<#2 <\else\ifnum#1=#2 =\else>\fi\fi}
5992 \def\forest@nodesort@cmp#1#2{\ifdim#1<#2 <\else\ifdim#1=#2 =\else>\fi\fi}
5993 \def\forest@nodesort@cmp#1#2{\ifdim#1pt<#2pt <\else\ifdim#1pt=#2pt =\else>\fi\fi}
5994 \def\forest@nodesort@cmp#1#2{\csname fRsT@nsc@\pdfstrcmp{#1}{#2}\endcsname}
5995 \def\forest@nodesort@cmp#1#2{\csname fRsT@nsc@\pdfstrcmp{#2}{#1}\endcsname}
5996 \csdef{fRsT@nsc@-1}{<}
5997 \csdef{fRsT@nsc@0}{=}
5998 \csdef{fRsT@nsc@1}{>}
5999 \def\forest@nodesort@let#1#2{%
6000 \csletcs{forest@nodesort@#1}{forest@nodesort@#2}%
6001 }
6002 \forestset{
6003   define long step={last dynamic node}{style,must start at valid node=false}{%
6004     name=forest@last@node
6005   }
6006 }

```

8 Stages

```

6007 \def\forest@root{0}
6008 %%% begin listing region: stages
6009 \forestset{
6010   stages/.style={
6011     for root'={
6012       process keylist register=default preamble,
6013       process keylist register=preamble
6014     },
6015     process keylist=given options,
6016     process keylist=before typesetting nodes,
6017     typeset nodes stage,
6018     process keylist=before packing,
6019     pack stage,
6020     process keylist=before computing xy,
6021     compute xy stage,
6022     process keylist=before drawing tree,
6023     draw tree stage
6024   },
6025   typeset nodes stage/.style={for root'=typeset nodes},
6026   pack stage/.style={for root'=pack},
6027   compute xy stage/.style={for root'=compute xy},
6028   draw tree stage/.style={for root'=draw tree},
6029 }

```

```

6030   %% end listing region: stages
6031 \forestset{
6032   process keylist/.code={%
6033     \forest@process@hook@keylist{#1}{#1 processing order/.try,processing order/.lastretry}},
6034   process keylist'/.code 2 args={\forest@process@hook@keylist@nodynamics{#1}{#2}},
6035   process keylist''/.code 2 args={\forest@process@hook@keylist@{#1}{#2}},
6036   process keylist register/.code={\forest@process@keylist@register{#1}},
6037   process delayed/.code={%
6038     \forest@havedelayedoptions{@delay}{#1}%
6039     \ifforest@havedelayedoptions
6040       \forest@process@hook@keylist@nodynamics{@delay}{#1}%
6041     \fi
6042   },
6043   do dynamics/.code={%
6044     \the\forest@do@dynamics
6045     \forest@do@dynamics{}%
6046     \forest@node@Compute@numeric@ts@info{\forest@root}%
6047   },
6048   declare keylist={given options}{},
6049   declare keylist={before typesetting nodes}{},
6050   declare keylist={before packing}{},
6051   declare keylist={before packing node}{},
6052   declare keylist={after packing node}{},
6053   declare keylist={before computing xy}{},
6054   declare keylist={before drawing tree}{},
6055   declare keylist={delay}{},
6056   delay n/.code 2 args={%
6057     \forestmathsetcount\forest@temp@count{#1}%
6058     \pgfkeysalso{delay n'={\forest@temp@count}{#2}}%
6059   },
6060   delay n'/.code 2 args={
6061     \ifnum#1=0
6062       \pgfkeysalso{#2}%
6063     \else
6064       \pgfkeysalso{delay={delay n'/.expand once=\expandafter{\number\numexpr#1-1\relax}{#2}}}%
6065     \fi
6066   },
6067   if have delayed/.style 2 args={if have delayed'={processing order}{#1}{#2}},
6068   if have delayed'/.code n args=3{%
6069     \forest@havedelayedoptionsfalse
6070     \forest@forthis{%
6071       \forest@nodewalk{#1}{%
6072         TeX={%
6073           \forest@toget{delay}\forest@temp@delayed
6074           \ifdefempty\forest@temp@delayed{}{\forest@havedelayedoptionstrue}%
6075         }%
6076       }%
6077     }%
6078     \ifforest@havedelayedoptions\pgfkeysalso{#2}\else\pgfkeysalso{#3}\fi
6079   },
6080   typeset nodes/.code={%
6081     \forest@drawtree@preservenodeboxes@false
6082     \forest@nodewalk
6083       {typeset nodes processing order/.try,processing order/.lastretry}%
6084       {TeX={\forest@node@typeset}}%
6085   },
6086   typeset nodes'/.code={%
6087     \forest@drawtree@preservenodeboxes@true
6088     \forest@nodewalk
6089       {typeset nodes processing order/.try,processing order/.lastretry}%
6090       {TeX={\forest@node@typeset}}%

```

```

6091 },
6092 typeset node/.code={%
6093   \forest@drawtree@preservenodeboxes@false
6094   \forest@node@typeset
6095 },
6096 pack/.code={\forest@pack},
6097 pack'/.code={\forest@pack@onlythisnode},
6098 compute xy/.code={\forest@node@computeabsolute positions},
6099 draw tree box/.store in=\forest@drawtreebox,
6100 draw tree box,
6101 draw tree/.code={%
6102   \forest@drawtree@preservenodeboxes@false
6103   \forest@node@drawtree
6104 },
6105 draw tree'/.code={%
6106   \forest@drawtree@preservenodeboxes@true
6107   \forest@node@drawtree
6108 },
6109 %%% begin listing region: draw_tree_method
6110 draw tree method/.style={
6111   for nodewalk={
6112     draw tree nodes processing order/.try,
6113     draw tree processing order/.retry,
6114     processing order/.lastretry
6115   }{draw tree node},
6116   for nodewalk={
6117     draw tree edges processing order/.try,
6118     draw tree processing order/.retry,
6119     processing order/.lastretry
6120   }{draw tree edge},
6121   for nodewalk={
6122     draw tree tikz processing order/.try,
6123     draw tree processing order/.retry,
6124     processing order/.lastretry
6125   }{draw tree tikz}
6126 },
6127 %%% end listing region: draw_tree_method
6128 draw tree node/.code={\forest@draw@node},
6129 draw tree node'/.code={\forest@draw@node@},
6130 if node drawn/.code n args={3}{%
6131   \forest@forthis{%
6132     \forest@configured@nodewalk{independent}{inherited}{fake}{#1}{}%
6133     \ifnum\forest@cn=0
6134       \forest@tempfalse
6135     \else
6136       \ifcsdef{forest@drawn@\forest@cn}{\forest@temptrue}{\forest@tempfalse}%
6137     \fi
6138   }%
6139   \ifforest@temp\pgfkeysalso{#2}\else\pgfkeysalso{#3}\fi
6140 },
6141 draw tree edge/.code={\forest@draw@edge},
6142 draw tree edge'/.code={\forest@draw@edge@},
6143 draw tree tikz/.code={\forest@draw@tikz@}, % always!
6144 draw tree tikz'/.code={\forest@draw@tikz@},
6145 processing order/.nodewalk style={tree},
6146 %given options processing order/.style={processing order},
6147 %before typesetting nodes processing order/.style={processing order},
6148 %before packing processing order/.style={processing order},
6149 %before computing xy processing order/.style={processing order},
6150 %before drawing tree processing order/.style={processing order},
6151 }

```

```

6152 \newtoks\forest@do@ dynamics
6153 \newif\ifforest@havedelayedoptions
6154 \def\forest@process@hook@keylist#1#2{%#1=keylist,#2=processing order nodewalk
6155   \safeloop
6156     \forest@for node{\forest@root}{\forest@process@hook@keylist@{#1}{#2}}%
6157     \expandafter\ifstreampy\expandafter{\the\forest@do@ dynamics}{}%
6158     \the\forest@do@ dynamics
6159     \forest@do@ dynamics={}%
6160     \forest@node@Compute@numeric@ts@info{\forest@root}%
6161   }%
6162   \forest@for node{\forest@root}{\forest@havedelayedoptions{#1}{#2}}%
6163   \ifforest@havedelayedoptions
6164   \saferepeat
6165 }
6166 \def\forest@process@hook@keylist@nodynamics#1#2{%#1=keylist,#2=processing order nodewalk
6167   % note: this macro works on (nodewalk starting at) the current node
6168   \safeloop
6169     \forest@for this{\forest@process@hook@keylist@{#1}{#2}}%
6170     \forest@havedelayedoptions{#1}{#2}%
6171     \ifforest@havedelayedoptions
6172     \saferepeat
6173 }
6174 \def\forest@process@hook@keylist@#1#2{%#1=keylist,#2=processing order nodewalk
6175   \forest@nodewalk{#2}{%
6176     TeX={%
6177       \forest@toget{#1}\forest@temp@keys
6178       \ifdefined\forest@temp@keys{%
6179         \forest@set{#1}{}%
6180         \expandafter\forest@set\expandafter{\forest@temp@keys}%
6181       }%
6182     }%
6183   }%
6184 }
6185 \def\forest@process@keylist@register#1{%
6186   \edef\forest@marshal{%
6187     \noexpand\forest@set{\forest@register{#1}}%
6188   }\forest@marshal
6189 }

```

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

```

6190 \def\forest@havedelayedoptions#1#2{%#1 = keylist, #2=nodewalk
6191   \forest@havedelayedoptionsfalse
6192   \forest@for this{%
6193     \forest@nodewalk{#2}{%
6194       TeX={%
6195         \forest@toget{delay}\forest@temp@delayed
6196         \ifdefined\forest@temp@delayed{\forest@havedelayedoptionstrue}%
6197         \forest@tolet{#1}\forest@temp@delayed
6198         \forest@set{delay}{}%
6199       }%
6200     }%
6201   }%
6202 }

```

8.1 Typesetting nodes

```

6203 \def\forest@node@typeset{%
6204   \let\forest@next\forest@node@typeset@
6205   \forest@ifdefined{@box}{%
6206     \forest@toget{@box}\forest@temp
6207     \ifdefined\forest@temp{%
6208       \locbox\forest@temp@box

```

```

6209     \forestolet{@box}\forest@temp@box
6210 }{%
6211     \ifforest@drawtree@preservenodeboxes@
6212     \let\forest@next\relax
6213     \fi
6214 }%
6215 }{%
6216     \locbox\forest@temp@box
6217     \forestolet{@box}\forest@temp@box
6218 }%
6219 \def\forest@node@typeset@restore{%
6220 \ifdefined\ifsa@tikz\forest@standalone@hack\fi
6221 \forest@next
6222 \forest@node@typeset@restore
6223 }
6224 \def\forest@standalone@hack{%
6225     \ifsa@tikz
6226     \let\forest@standalone@tikzpicture\tikzpicture
6227     \let\forest@standalone@endtikzpicture\endtikzpicture
6228     \let\tikzpicture\sa@orig@tikzpicture
6229     \let\endtikzpicture\sa@orig@endtikzpicture
6230     \def\forest@node@typeset@restore{%
6231         \let\tikzpicture\forest@standalone@tikzpicture
6232         \let\endtikzpicture\forest@standalone@endtikzpicture
6233     }%
6234     \fi
6235 }
6236 \newbox\forest@box
6237 \def\forest@pgf@notyetpositioned{not yet positionedPGFINTERNAL}
6238 \def\forest@node@typeset@{%
6239     \forestanchortotikzanchor{anchor}\forest@temp
6240     \edef\forest@marshal{%
6241         \noexpand\forestolet{anchor}\noexpand\forest@temp
6242         \noexpand\forest@node@typeset@@
6243         \noexpand\forestoset{anchor}{\forestov{anchor}}%
6244     }\forest@marshal
6245 }
6246 \def\forest@node@typeset@@{%
6247     \foresttoget{name}\forest@nodename
6248     \edef\forest@temp@nodeformat{\forestove{node format}}%
6249     \gdef\forest@smuggle{%
6250         \setbox0=\hbox{%
6251             \begin{tikzpicture}%
6252             /forest/copy command key={/tikz/anchor}/{/tikz/forest@orig@anchor},
6253             anchor/.style={%
6254                 /forest/TeX={\forestanchortotikzanchor{##1}\forest@temp@anchor},
6255                 forest@orig@anchor/.expand once=\forest@temp@anchor
6256             }%
6257             \pgfpositionnodelater{\forest@positionnodelater@save}%
6258             \forest@temp@nodeformat
6259             \pgfinterruptpath
6260             \pgfpointanchor{\forest@pgf@notyetpositioned\forest@nodename}{\forestcomputenodeboundary}%
6261             \endpgfinterruptpath
6262             \end{tikzpicture}%
6263         }%
6264         \setbox\forestove{@box}=\box\forest@box % smuggle the box
6265         \forestolet{@boundary}\forest@global@boundary
6266         \forest@smuggle % ... and the rest
6267     }
6268
6269

```



```

6270 \forestset{
6271   declare readonly dimen={min x}{0pt},
6272   declare readonly dimen={min y}{0pt},
6273   declare readonly dimen={max x}{0pt},
6274   declare readonly dimen={max y}{0pt},
6275 }
6276 \def\forest@patch@enormouscoordinateboxbounds@plus#1{%
6277   \expandafter\ifstrequal\expandafter{#1}{16000.0pt}{\edef#1{0.0\pgfmath@pt}}{}}%
6278 }
6279 \def\forest@patch@enormouscoordinateboxbounds@minus#1{%
6280   \expandafter\ifstrequal\expandafter{#1}{-16000.0pt}{\edef#1{0.0\pgfmath@pt}}{}}%
6281 }
6282 \def\forest@positionnodelater@save{%
6283   \global\setbox\forest@box=\box\pgfpositionnodelaterbox
6284   \xappto\forest@smuggle{\noexpand\forestoset{later@name}{\pgfpositionnodelatername}}%
6285   % a bug in pgf? ---well, here's a patch
6286   \forest@patch@enormouscoordinateboxbounds@plus\pgfpositionnodelaterminx
6287   \forest@patch@enormouscoordinateboxbounds@plus\pgfpositionnodelaterminy
6288   \forest@patch@enormouscoordinateboxbounds@minus\pgfpositionnodelatermaxx
6289   \forest@patch@enormouscoordinateboxbounds@minus\pgfpositionnodelatermaxy
6290   % end of patch
6291   \xappto\forest@smuggle{\noexpand\forestoset{min x}{\pgfpositionnodelaterminx}}%
6292   \xappto\forest@smuggle{\noexpand\forestoset{min y}{\pgfpositionnodelaterminy}}%
6293   \xappto\forest@smuggle{\noexpand\forestoset{max x}{\pgfpositionnodelatermaxx}}%
6294   \xappto\forest@smuggle{\noexpand\forestoset{max y}{\pgfpositionnodelatermaxy}}%
6295 }
6296 \def\forest@node@forest@positionnodelater@restore{%
6297   \ifforest@drawtree@preservenodeboxes@
6298     \let\forest@boxorcopy\copy
6299   \else
6300     \let\forest@boxorcopy\box
6301   \fi
6302   \forestoget{@box}\forest@temp
6303   \setbox\pgfpositionnodelaterbox=\forest@boxorcopy\forest@temp
6304   \edef\pgfpositionnodelatername{\forestove{later@name}}%
6305   \edef\pgfpositionnodelaterminx{\forestove{min x}}%
6306   \edef\pgfpositionnodelaterminy{\forestove{min y}}%
6307   \edef\pgfpositionnodelatermaxx{\forestove{max x}}%
6308   \edef\pgfpositionnodelatermaxy{\forestove{max y}}%
6309   \ifforest@drawtree@preservenodeboxes@
6310     \else
6311       \forestoset{@box}{}%
6312   \fi
6313 }

```

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.

```

6314 \def\forest@pack{%
6315   \pgfsyssoftpath@getcurrentpath\forest@pack@original@path
6316   \forest@pack@computetiers
6317   \forest@pack@computegrowthuniformity
6318   \forest@@@pack
6319   \pgfsyssoftpath@setcurrentpath\forest@pack@original@path
6320 }
6321 \def\forest@@@pack{%
6322   \ifnum\forestove{uniform growth}>0
6323     \ifnum\forestove{n children}>0
6324       \forest@pack@level@uniform
6325       \forest@pack@aligntiers@ofsubtree

```

```

6326     \forest@pack@sibling@uniform@recursive
6327     \fi
6328 \else
6329     \forest@node@foreachchild{\forest@pack}%
6330     \forest@process@hook@keylist@nodynamics{before packing node}{current}%
6331     \ifnum\forestove{n children}>0
6332         \forest@pack@level@nonuniform
6333         \forest@pack@aligntiers
6334         \forest@pack@sibling@uniform@applyreversed
6335     \fi
6336     \forestoget{after packing node}\forest@temp@keys
6337     \forest@process@hook@keylist@nodynamics{after packing node}{current}%
6338 \fi
6339 }
6340 % \forestset{recalculate tree boundary/.code={\forest@node@recalculate@edges}}
6341 % \def\forest@node@recalculate@edges{%
6342 %     \edef\forest@marshal{%
6343 %         \noexpand\forest@forthis{\noexpand\forest@node@getedges{\forestove{grow}}}%
6344 %     }\forest@marshal
6345 % }
6346 \def\forest@pack@onlythisnode{%
6347     \ifnum\forestove{n children}>0
6348         \forest@pack@computetiers
6349         \forest@pack@level@nonuniform
6350         \forest@pack@aligntiers
6351         \forest@node@foreachchild{\forestoset{s}{0\pgfmath@pt}}%
6352         \forest@pack@sibling@uniform@applyreversed
6353     \fi
6354 }

```

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

```

6355 \def\forest@pack@computegrowthuniformity{%
6356     \forest@node@foreachchild{\forest@pack@computegrowthuniformity}%
6357     \edef\forest@pack@cgu@uniformity{%
6358         \ifnum\forestove{n children}=0
6359             2\else 1\fi
6360     }%
6361     \forestoget{grow}\forest@pack@cgu@parentgrow
6362     \forest@node@foreachchild{%
6363         \ifnum\forestove{uniform growth}=0
6364             \def\forest@pack@cgu@uniformity{0}%
6365         \else
6366             \ifnum\forestove{uniform growth}=1
6367                 \ifnum\forestove{grow}=\forest@pack@cgu@parentgrow\relax\else
6368                     \def\forest@pack@cgu@uniformity{0}%
6369                 \fi
6370             \fi
6371         \fi
6372     }%
6373     \forestoget{before packing node}\forest@temp@a
6374     \forestoget{after packing node}\forest@temp@b
6375     \expandafter\expandafter\expandafter\ifstrempy\expandafter\expandafter\expandafter{\expandafter\forest@temp@a\forest@temp@b}
6376     \forestolet{uniform growth}\forest@pack@cgu@uniformity
6377 }{%
6378     \forestoset{uniform growth}{0}%
6379 }%
6380 }

```

Pack children in the level dimension in a uniform tree.

```

6381 \def\forest@pack@level@uniform{%

```

```

6382 \let\forest@plu@minchildl\relax
6383 \forestoget{grow}\forest@plu@grow
6384 \forest@node@foreachchild{%
6385   \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
6386   \advance\pgf@xa\forestove{1}\relax
6387   \ifx\forest@plu@minchildl\relax
6388     \edef\forest@plu@minchildl{\the\pgf@xa}%
6389   \else
6390     \ifdim\pgf@xa<\forest@plu@minchildl\relax
6391       \edef\forest@plu@minchildl{\the\pgf@xa}%
6392     \fi
6393   \fi
6394 }%
6395 \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
6396 \pgfutil@tempdima=\pgf@xb\relax
6397 \advance\pgfutil@tempdima -\forest@plu@minchildl\relax
6398 \advance\pgfutil@tempdima \forestove{1 sep}\relax
6399 \ifdim\pgfutil@tempdima>0pt
6400   \forest@node@foreachchild{%
6401     \forestoeset{1}{\the\dimexpr\forestove{1}+\the\pgfutil@tempdima}%
6402   }%
6403 \fi
6404 \forest@node@foreachchild{%
6405   \ifnum\forestove{n children}>0
6406     \forest@pack@level@uniform
6407   \fi
6408 }%
6409 }

```

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

```

6410 \def\forest@pack@level@nonuniform{%
6411   \let\forest@plu@minchildl\relax
6412   \forestoget{grow}\forest@plu@grow
6413   \forest@node@foreachchild{%
6414     \forest@node@getedge{negative}{\forest@plu@grow}{\forest@plnu@negativechilddedge}%
6415     \forest@node@getedge{positive}{\forest@plu@grow}{\forest@plnu@positivechilddedge}%
6416     \def\forest@plnu@childdedge{\forest@plnu@negativechilddedge\forest@plnu@positivechilddedge}%
6417     \forest@path@getboundingrectangle@ls\forest@plnu@childdedge{\forest@plu@grow}%
6418     \advance\pgf@xa\forestove{1}\relax
6419     \ifx\forest@plu@minchildl\relax
6420       \edef\forest@plu@minchildl{\the\pgf@xa}%
6421     \else
6422       \ifdim\pgf@xa<\forest@plu@minchildl\relax
6423         \edef\forest@plu@minchildl{\the\pgf@xa}%
6424       \fi
6425     \fi
6426   }%
6427   \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
6428   \pgfutil@tempdima=\pgf@xb\relax
6429   \advance\pgfutil@tempdima -\forest@plu@minchildl\relax
6430   \advance\pgfutil@tempdima \forestove{1 sep}\relax
6431   \ifdim\pgfutil@tempdima>0pt
6432     \forest@node@foreachchild{%
6433       \forestoeset{1}{\the\dimexpr\the\pgfutil@tempdima+\forestove{1}}%
6434     }%
6435   \fi
6436 }

```

Align tiers.

```

6437 \def\forest@pack@aligtiers{%
6438   \forestoget{grow}\forest@temp@parentgrow
6439   \forestoget{@tiers}\forest@temp@tiers

```

```

6440 \forlistloop\forest@pack@alignntier@\forest@temp@tiers
6441 }
6442 \def\forest@pack@alignntiers@ofsubtree{%
6443 \forest@node@foreach{\forest@pack@alignntiers}%
6444 }
6445 \def\forest@pack@alignntiers@computeabs1{%
6446 \forestoleto{abs@1}{1}%
6447 \forest@node@foreachdescendant{\forest@pack@alignntiers@computeabs1}%
6448 }
6449 \def\forest@pack@alignntiers@computeabs1o{%
6450 \forestoeset{abs@1}{\the\dimexpr\forestove{1}+\forestOve{\forestove{@parent}}{abs@1}}%
6451 }
6452 \def\forest@pack@alignntier@#1{%
6453 \forest@pack@alignntiers@computeabs1
6454 \pgfutil@tempdima=-\maxdimen\relax
6455 \def\forest@temp@currenttier{#1}%
6456 \forest@node@foreach{%
6457 \forestoget{tier}\forest@temp@tier
6458 \ifx\forest@temp@currenttier\forest@temp@tier
6459 \ifdim\pgfutil@tempdima<\forestove{abs@1}\relax
6460 \pgfutil@tempdima=\forestove{abs@1}\relax
6461 \fi
6462 \fi
6463 }%
6464 \ifdim\pgfutil@tempdima=-\maxdimen\relax\else
6465 \forest@node@foreach{%
6466 \forestoget{tier}\forest@temp@tier
6467 \ifx\forest@temp@currenttier\forest@temp@tier
6468 \forestoeset{1}{\the\dimexpr\pgfutil@tempdima-\forestove{abs@1}+\forestove{1}}%
6469 \fi
6470 }%
6471 \fi
6472 }

```

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

```

6473 \def\forest@pack@sibling@uniform@recursive{%
6474 \forest@node@foreachchild{\forest@pack@sibling@uniform@recursive}%
6475 \forest@pack@sibling@uniform@applyreversed
6476 }

```

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

```

6477 \def\forest@pack@sibling@uniform@applyreversed{%
6478 \ifnum\forestove{n children}>1
6479 \ifnum\forestove{reversed}=0
6480 \forest@pack@sibling@uniform@main{first}{last}{next}{previous}%
6481 \else
6482 \forest@pack@sibling@uniform@main{last}{first}{previous}{next}%
6483 \fi
6484 \else
6485 \ifnum\forestove{n children}=1

```

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

```

6486 \csname forest@calign@\forestove{calign}\endcsname
6487 \fi
6488 \fi
6489 }

```

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

```

6490 \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.

```

6491 \forestoget{@#1}\forest@child
6492 \edef\forest@marshal{%
6493   \noexpand\forest@fornode{\forestove{@#1}}{%
6494     \noexpand\forest@node@getedge
6495       {positive}%
6496       {\forestove{grow}}}%
6497   \noexpand\forest@temp@edge
6498 }%
6499 }\forest@marshal
6500 \forest@pack@pgfpoint@childposition\forest@child
6501 \let\forest@previous@positive@edge\pgfutil@empty
6502 \forest@extendpath\forest@previous@positive@edge\forest@temp@edge{}%
6503 \forest@get{\forest@child}{@#3}\forest@child

```

Loop until the current child is the null node.

```

6504 \edef\forest@previous@child@s{0\pgfmath@pt}%
6505 \safeloop
6506 \unless\ifnum\forest@child=0

```

Get the negative edge of the child.

```

6507 \edef\forest@temp{%
6508   \noexpand\forest@fornode{\forest@child}{%
6509     \noexpand\forest@node@getedge
6510       {negative}%
6511       {\forestove{grow}}}%
6512   \noexpand\forest@temp@edge
6513 }%
6514 }\forest@temp

```

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

```

6515 \forest@pack@pgfpoint@childposition\forest@child

```

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

```

6516 \let\forest@child@negative@edge\pgfutil@empty
6517 \forest@extendpath\forest@child@negative@edge\forest@temp@edge{}%

```

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

```

6518 \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!)

```

6519 \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`.

```

6520 \ifx\forest@csdistance\relax
6521   %\forest@reset{\forest@child}{s}{\forest@previous@child@s}%
6522 \else
6523   \advance\pgfutil@tempdimb-\forest@csdistance\relax
6524   \advance\pgfutil@tempdimb\forestove{s_sep}\relax
6525   \forest@reset{\forest@child}{s}{\the\dimexpr\forest@ove{\forest@child}{s}-\forest@csdistance+\forestove{s}
6526 \fi

```

Retain monotonicity (is this ok?). (This problem arises when the adjacent children's 1 are too far apart.)

```

6527 \ifdim\forest@ove{\forest@child}{s}<\forest@previous@child@s\relax
6528   \forest@reset{\forest@child}{s}{\forest@previous@child@s}%
6529 \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.

```

6530 \forest@get{\forest@child}{s}\forest@child@s
6531 \edef\forest@previous@child@s{\forest@child@s}%

```

```

6532 \edef\forest@temp{%
6533   \noexpand\forest@fornode{\forest@child}{%
6534     \noexpand\forest@node@getedge
6535       {positive}}%
6536     {\forestove{grow}}}%
6537   \noexpand\forest@temp@edge
6538 }%
6539 }\forest@temp
6540 \forest@pack@pgfpoint@childposition\forest@child
6541 \forest@extendpath\forest@previous@positive@edge\forest@temp@edge{%
6542   \forest@getpositivetightedgeofpath\forest@previous@positive@edge\forest@previous@positive@edge
6543   \forest@get{\forest@child}{@#3}\forest@child
6544 \saferepeat

```

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

```

6545 \csname forest@calign@\forestove{calign}\endcsname
6546 }

```

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

```

6547 \def\forest@pack@pgfpoint@childposition#1{%
6548   {%
6549     \pgftransformreset
6550     \forest@pgfqtransformrotate{\forestove{grow}}%
6551     \forest@fornode{#1}{%
6552       \pgfpointransformed{\pgfqpoint{\forestove{1}}{\forestove{s}}}%
6553     }%
6554   }%
6555 }

```

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

```

6556 \def\forest@pack@pgfpoint@positioningrow#1{%
6557   {%
6558     \pgftransformreset
6559     \forest@pgfqtransformrotate{#1}%
6560     \pgfpointransformed{\pgfqpoint{\forestove{1}}{\forestove{s}}}%
6561   }%
6562 }

```

Child alignment.

```

6563 \def\forest@calign@s@shift#1{%
6564   \pgfutil@tempdima=#1\relax
6565   \forest@node@foreachchild{%
6566     \forestoeset{s}{\the\dimexpr\forestove{s}+\pgfutil@tempdima}%
6567   }%
6568 }
6569 \def\forest@calign@child{%
6570   \forest@calign@s@shift{-\forestove{\forest@node@nornbarthchildid{\forestove{calign primary child}}}{s}}%
6571 }
6572 \csdef{forest@calign@child edge}{%
6573   {%
6574     \edef\forest@temp@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
6575     \pgftransformreset
6576     \forest@pgfqtransformrotate{\forestove{grow}}%
6577     \pgfpointransformed{\pgfqpoint{\forestove{\forest@temp@child}{1}}{\forestove{\forest@temp@child}{s}}}%
6578     \pgf@xa=\pgf@x\relax\pgf@ya=\pgf@y\relax
6579     \forest@Pointanchor{\forest@temp@child}{child anchor}%
6580     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
6581     \forest@pointanchor{parent anchor}%
6582     \advance\pgf@xa-\pgf@x\relax\advance\pgf@ya-\pgf@y\relax
6583     \edef\forest@marshal{%
6584       \noexpand\pgftransformreset
6585       \noexpand\forest@pgfqtransformrotate{-\forestove{grow}}%

```

```

6586 \noexpand\pgfpointransformed{\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}%
6587 }\forest@marshal
6588 }%
6589 \forest@calign@s@shift{\the\dimexpr-\the\pgf@y}%
6590 }
6591 \csdef{forest@calign@midpoint}{%
6592 \forest@calign@s@shift{\the\dimexpr Opt -%
6593 (\forestOve{\forest@node@nornbarthchildid{\forestove{calign primary child}}}{s}%
6594 +\forestOve{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}{s}%
6595 )/2\relax
6596 }%
6597 }
6598 \csdef{forest@calign@edge midpoint}{%
6599 {%
6600 \edef\forest@temp@firstchild{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
6601 \edef\forest@temp@secondchild{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
6602 \pgftransformreset
6603 \forest@pgfqtransformrotate{\forestove{grow}}%
6604 \pgfpointransformed{\pgfqpoint{\forestOve{\forest@temp@firstchild}{l}}{\forestOve{\forest@temp@firstchild}{r}}}%
6605 \pgf@xa=\pgf@x\relax\pgf@ya=\pgf@y\relax
6606 \forest@Pointanchor{\forest@temp@firstchild}{child anchor}%
6607 \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
6608 \edef\forest@marshal{%
6609 \noexpand\pgfpointransformed{\noexpand\pgfqpoint{\forestOve{\forest@temp@secondchild}{l}}{\forestOve{\forest@temp@secondchild}{r}}}%
6610 }\forest@marshal
6611 \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
6612 \forest@Pointanchor{\forest@temp@secondchild}{child anchor}%
6613 \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
6614 \divide\pgf@xa2 \divide\pgf@ya2
6615 \forest@pointanchor{parent anchor}%
6616 \advance\pgf@xa-\pgf@x\relax\advance\pgf@ya-\pgf@y\relax
6617 \edef\forest@marshal{%
6618 \noexpand\pgftransformreset
6619 \noexpand\forest@pgfqtransformrotate{-\forestove{grow}}%
6620 \noexpand\pgfpointransformed{\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}%
6621 }\forest@marshal
6622 }%
6623 \forest@calign@s@shift{\the\dimexpr-\the\pgf@y}%
6624 }

```

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.

```

6625 \def\forest@edef@stript#1#2{%
6626 \edef#1{#2}%
6627 \edef#1{\expandafter\Pgf@geT#1}%
6628 }
6629 \csdef{forest@calign@fixed angles}{%
6630 \ifnum\forestove{n children}>1
6631 \edef\forest@ca@first@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
6632 \edef\forest@ca@second@child{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
6633 \ifnum\forestove{reversed}=1
6634 \let\forest@temp\forest@ca@first@child
6635 \let\forest@ca@first@child\forest@ca@second@child
6636 \let\forest@ca@second@child\forest@temp
6637 \fi
6638 \forest@get{\forest@ca@first@child}{l}\forest@ca@first@l
6639 \edef\forest@ca@first@l{\expandafter\Pgf@geT\forest@ca@first@l}%
6640 \forest@get{\forest@ca@second@child}{l}\forest@ca@second@l
6641 \edef\forest@ca@second@l{\expandafter\Pgf@geT\forest@ca@second@l}%

```

```

6642 \pgfmathtan@{\forestove{calign secondary angle}}%
6643 \pgfmathmultiply@{\pgfmathresult}{\forest@ca@second@l}%
6644 \let\forest@calign@temp\pgfmathresult
6645 \pgfmathtan@{\forestove{calign primary angle}}%
6646 \pgfmathmultiply@{\pgfmathresult}{\forest@ca@first@l}%
6647 \edef\forest@ca@desired@s@distance{\the\dimexpr
6648 \forest@calign@temp pt-\pgfmathresult pt}%
6649 % \pgfmathsetlengthmacro\forest@ca@desired@s@distance{%
6650 % \tan(\forestove{calign secondary angle})*\forest@ca@second@l
6651 % -\tan(\forestove{calign primary angle})*\forest@ca@first@l
6652 % }%
6653 \forest@get{\forest@ca@first@child}{s}\forest@ca@first@s
6654 \forest@get{\forest@ca@second@child}{s}\forest@ca@second@s
6655 \edef\forest@ca@actual@s@distance{\the\dimexpr
6656 \forest@ca@second@s-\forest@ca@first@s}%
6657 %\pgfmathsetlengthmacro\forest@ca@actual@s@distance{%
6658 % \forest@ca@second@s-\forest@ca@first@s}%
6659 \ifdim\forest@ca@desired@s@distance>\forest@ca@actual@s@distance\relax
6660 \ifdim\forest@ca@actual@s@distance=0pt
6661 \pgfmathtan@{\forestove{calign primary angle}}%
6662 \pgfmathmultiply@{\pgfmathresult}{\forest@ca@second@l}%
6663 \pgfutil@tempdima=\pgfmathresult pt
6664 % \pgfmathsetlength\pgfutil@tempdima{\tan(\forestove{calign primary angle})*\forest@ca@second@l}%
6665 \pgfutil@tempdimb=\dimexpr
6666 \forest@ca@desired@s@distance/(\forestove{n children}-1)\relax%
6667 %\pgfmathsetlength\pgfutil@tempdimb{\forest@ca@desired@s@distance/(\forestove{n children}-1)}%
6668 \forest@node@foreachchild{%
6669 \forestoeset{s}{\the\pgfutil@tempdima}%
6670 \advance\pgfutil@tempdima\pgfutil@tempdimb
6671 }%
6672 \def\forest@calign@anchor{0pt}%
6673 \else
6674 \edef\forest@marshal{\noexpand\pgfmathdivide@
6675 {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
6676 {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
6677 }\forest@marshal
6678 \let\forest@ca@ratio\pgfmathresult
6679 %\pgfmathsetmacro\forest@ca@ratio{%
6680 % \forest@ca@desired@s@distance/\forest@ca@actual@s@distance}%
6681 \forest@node@foreachchild{%
6682 \forest@edef@stript\forest@temp{\forestove{s}}%
6683 \pgfmathmultiply@{\forest@ca@ratio}{\forest@temp}%
6684 \forestoeset{s}{\the\dimexpr\pgfmathresult pt}%
6685 %\pgfmathsetlengthmacro\forest@temp{\forest@ca@ratio*\forestove{s}}%
6686 %\forestolet{s}\forest@temp
6687 }%
6688 \pgfmathtan@{\forestove{calign primary angle}}%
6689 \pgfmathmultiply@{\pgfmathresult}{\forest@ca@first@l}%
6690 \edef\forest@calign@anchor{\the\dimexpr-\pgfmathresult pt}%
6691 %\pgfmathsetlengthmacro\forest@calign@anchor{%
6692 % -\tan(\forestove{calign primary angle})*\forest@ca@first@l}%
6693 \fi
6694 \else
6695 \ifdim\forest@ca@desired@s@distance<\forest@ca@actual@s@distance\relax
6696 \edef\forest@marshal{\noexpand\pgfmathdivide@
6697 {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
6698 {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
6699 }\forest@marshal
6700 \let\forest@ca@ratio\pgfmathresult
6701 %\pgfmathsetlengthmacro\forest@ca@ratio{%
6702 % \forest@ca@actual@s@distance/\forest@ca@desired@s@distance}%

```



```

6703 \forest@node@foreachchild{%
6704 \forest@edef\stript\forest@temp{\forestove{1}}%
6705 \pgfmathmultiply@{\forest@ca@ratio}{\forest@temp}%
6706 \forestoeset{1}{\the\dimexpr\pgfmathresult pt}%
6707 %\pgfmathsetlengthmacro\forest@temp{\forest@ca@ratio*\forestove{1}}%
6708 %\forestolet{1}\forest@temp
6709 }%
6710 \forestOget{\forest@ca@first@child}{1}\forest@ca@first@1
6711 \edef\forest@ca@first@1{\expandafter\PgfeT\forest@ca@first@1}%
6712 \pgfmathatan@{\forestove{calign primary angle}}%
6713 \pgfmathmultiply@{\pgfmathresult}{\forest@ca@first@1}%
6714 \edef\forest@calign@anchor{\the\dimexpr-\pgfmathresult pt}%
6715 %\pgfmathsetlengthmacro\forest@calign@anchor{%
6716 % -tan(\forestove{calign primary angle})*\forest@ca@first@1}%
6717 \fi
6718 \fi
6719 \forest@calign@s@shift{-\forest@calign@anchor}%
6720 \fi
6721 }
6722 \csdef{forest@calign@fixed edge angles}{%
6723 \ifnum\forestove{n children}>1
6724 \edef\forest@ca@first@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
6725 \edef\forest@ca@second@child{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
6726 \ifnum\forestove{reversed}=1
6727 \let\forest@temp\forest@ca@first@child
6728 \let\forest@ca@first@child\forest@ca@second@child
6729 \let\forest@ca@second@child\forest@temp
6730 \fi
6731 \forestOget{\forest@ca@first@child}{1}\forest@ca@first@1
6732 \forestOget{\forest@ca@second@child}{1}\forest@ca@second@1
6733 \forest@pointanchor{parent anchor}%
6734 \edef\forest@ca@parent@anchor@s{\the\pgf@x}%
6735 \edef\forest@ca@parent@anchor@l{\the\pgf@y}%
6736 \forest@Pointanchor{\forest@ca@first@child}{child anchor}%
6737 \edef\forest@ca@first@child@anchor@s{\the\pgf@x}%
6738 \edef\forest@ca@first@child@anchor@l{\the\pgf@y}%
6739 \forest@Pointanchor{\forest@ca@second@child}{child anchor}%
6740 \edef\forest@ca@second@child@anchor@s{\the\pgf@x}%
6741 \edef\forest@ca@second@child@anchor@l{\the\pgf@y}%
6742 \pgfmathatan@{\forestove{calign secondary angle}}%
6743 \edef\forest@temp{\the\dimexpr
6744 \forest@ca@second@1-\forest@ca@second@child@anchor@l+\forest@ca@parent@anchor@l}%
6745 \pgfmathmultiply@{\pgfmathresult}{\expandafter\PgfeT\forest@temp}%
6746 \edef\forest@ca@desired@second@edge@s{\the\dimexpr\pgfmathresult pt}%
6747 %\pgfmathsetlengthmacro\forest@ca@desired@second@edge@s{%
6748 % tan(\forestove{calign secondary angle})*%
6749 % (\forest@ca@second@1-\forest@ca@second@child@anchor@l+\forest@ca@parent@anchor@l)}%
6750 \pgfmathatan@{\forestove{calign primary angle}}%
6751 \edef\forest@temp{\the\dimexpr
6752 \forest@ca@first@1-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@l}%
6753 \pgfmathmultiply@{\pgfmathresult}{\expandafter\PgfeT\forest@temp}%
6754 \edef\forest@ca@desired@first@edge@s{\the\dimexpr\pgfmathresult pt}%
6755 %\pgfmathsetlengthmacro\forest@ca@desired@first@edge@s{%
6756 % tan(\forestove{calign primary angle})*%
6757 % (\forest@ca@first@1-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@l)}%
6758 \edef\forest@ca@desired@s@distance{\the\dimexpr
6759 \forest@ca@desired@second@edge@s-\forest@ca@desired@first@edge@s}%
6760 %\pgfmathsetlengthmacro\forest@ca@desired@s@distance{\forest@ca@desired@second@edge@s-\forest@ca@desired@
6761 \forestOget{\forest@ca@first@child}{s}\forest@ca@first@s
6762 \forestOget{\forest@ca@second@child}{s}\forest@ca@second@s
6763 \edef\forest@ca@actual@s@distance{\the\dimexpr

```

```

6764 \forest@ca@second@s+\forest@ca@second@child@anchor@s
6765 -\forest@ca@first@s-\forest@ca@first@child@anchor@s}%
6766 %\pgfmthsetlengthmacro\forest@ca@actual@s@distance{%
6767 % \forest@ca@second@s+\forest@ca@second@child@anchor@s
6768 % -\forest@ca@first@s-\forest@ca@first@child@anchor@s}%
6769 \ifdim\forest@ca@desired@s@distance>\forest@ca@actual@s@distance\relax
6770 \ifdim\forest@ca@actual@s@distance=0pt
6771 \forest@toget{n children}\forest@temp@n@children
6772 \forest@node@foreachchild{%
6773 \forest@pointanchor{child anchor}%
6774 \edef\forest@temp@child@anchor@s{\the\pgf@x}%
6775 \forest@toeset{s}{\the\dimexpr
6776 \forest@ca@desired@first@edge@s+\forest@ca@desired@s@distance*(\forest@ve{n}-1)/(\forest@temp@n@children-1)}%
6777 %\pgfmthsetlengthmacro\forest@temp@{
6778 % \forest@ca@desired@first@edge@s+(\forest@ve{n}-1)*\forest@ca@desired@s@distance/(\forest@temp@n@children-1)}%
6779 %\forest@tolet{s}\forest@temp
6780 }%
6781 \def\forest@calign@anchor{0pt}%
6782 \else
6783 \edef\forest@marshal{\noexpand\pgfmthdivide@
6784 {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
6785 {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
6786 }\forest@marshal
6787 \let\forest@ca@ratio\pgfmthresult
6788 %\pgfmthsetmacro\forest@ca@ratio{%
6789 % \forest@ca@desired@s@distance/\forest@ca@actual@s@distance}%
6790 \forest@node@foreachchild{%
6791 \forest@pointanchor{child anchor}%
6792 \edef\forest@temp@child@anchor@s{\the\pgf@x}%
6793 \edef\forest@marshal{\noexpand\pgfmthmultiply@
6794 {\forest@ca@ratio}%
6795 {\expandafter\Pgf@geT\the\dimexpr
6796 \forest@ve{s}-\forest@ca@first@s+
6797 \forest@temp@child@anchor@s-\forest@ca@first@child@anchor@s}%
6798 }\forest@marshal
6799 \forest@toeset{s}{\the\dimexpr\pgfmthresult pt+\forest@ca@first@s
6800 +\forest@ca@first@child@anchor@s-\forest@temp@child@anchor@s}%
6801 % \pgfmthsetlengthmacro\forest@temp@{
6802 % \forest@ca@ratio*(
6803 % \forest@ve{s}-\forest@ca@first@s
6804 % +\forest@temp@child@anchor@s-\forest@ca@first@child@anchor@s)%
6805 % +\forest@ca@first@s
6806 % +\forest@ca@first@child@anchor@s-\forest@temp@child@anchor@s}%
6807 % \forest@tolet{s}\forest@temp
6808 }%
6809 \pgfmthtan@{\forest@ve{calign primary angle}}%
6810 \edef\forest@marshal{\noexpand\pgfmthmultiply@
6811 {\pgfmthresult}%
6812 {\expandafter\Pgf@geT\the\dimexpr
6813 \forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@l}%
6814 }\forest@marshal
6815 \edef\forest@calign@anchor{\the\dimexpr
6816 -\pgfmthresult pt+\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s}%
6817 % \pgfmthsetlengthmacro\forest@calign@anchor{
6818 % -tan(\forest@ve{calign primary angle})*(\forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@s)}%
6819 % +\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s
6820 % }%
6821 \fi
6822 \else
6823 \ifdim\forest@ca@desired@s@distance<\forest@ca@actual@s@distance\relax
6824 \edef\forest@marshal{\noexpand\pgfmthdivide@

```

```

6825     {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
6826     {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
6827 } \forest@marshal
6828 \let\forest@ca@ratio\pgfmathresult
6829 % \pgfmathsetlengthmacro\forest@ca@ratio{%
6830 %   \forest@ca@actual@s@distance/\forest@ca@desired@s@distance}%
6831 \forest@node@foreachchild{%
6832   \forest@pointanchor{child anchor}%
6833   \edef\forest@temp@child@anchor@1{\the\pgf@y}%
6834   \edef\forest@marshal{\noexpand\pgfmathmultiply@
6835     {\forest@ca@ratio}%
6836     {\expandafter\Pgf@geT\the\dimexpr\forestove{1}+\forest@ca@parent@anchor@1-\forest@temp@child@anch
6837   } \forest@marshal
6838   \forestoeset{1}{\the\dimexpr
6839     \pgfmathresult pt-\forest@ca@parent@anchor@1+\forest@temp@child@anchor@1}%
6840   % \pgfmathsetlengthmacro\forest@temp{%
6841   %   \forest@ca@ratio*(%
6842   %     \forestove{1}+\forest@ca@parent@anchor@1-\forest@temp@child@anchor@1)
6843   %   -\forest@ca@parent@anchor@1+\forest@temp@child@anchor@1}%
6844   % \forestolet{1}\forest@temp
6845 }%
6846 \forest@get{\forest@ca@first@child}{1}\forest@ca@first@1
6847 \pgfmathatan@{\forestove{calign primary angle}}%
6848 \edef\forest@marshal{\noexpand\pgfmathmultiply@
6849   {\pgfmathresult}%
6850   {\expandafter\Pgf@geT\the\dimexpr
6851     \forest@ca@first@1+\forest@ca@parent@anchor@1-\forest@temp@child@anchor@1}%
6852 } \forest@marshal
6853 \edef\forest@calign@anchor{\the\dimexpr
6854   -\pgfmathresult pt+\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s}%
6855 % \pgfmathsetlengthmacro\forest@calign@anchor{%
6856 %   -tan(\forestove{calign primary angle})*(\forest@ca@first@1+\forest@ca@parent@anchor@1-\forest@tem
6857 %   +\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s
6858 % }%
6859 \fi
6860 \fi
6861 \forest@calign@s@shift{-\forest@calign@anchor}%
6862 \fi
6863 }

```

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.

```

6864 \def\forest@node@getedge#1#2#3{%
6865   \forestoget{#1@edge@#2}#3%
6866   \ifx#3\relax
6867     \forest@node@foreachchild{%
6868       \forest@node@getedge{#1}{#2}{\forest@temp@edge}%
6869     }%
6870     \forest@forthis{\forest@node@getedges{#2}}%
6871     \forestoget{#1@edge@#2}#3%
6872   \fi
6873 }

```

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.

```

6874 \def\forest@node@getedges#1{%

```

Run the computation in a \TeX group.

```

6875   {%

```

Setup the grow line.

```
6876 \forest@setupgrowline{#1}%
```

Get the edge of the node itself.

```
6877 \ifnum\forestove{ignore}=0
6878 \forestoget{@boundary}\forest@node@boundary
6879 \else
6880 \def\forest@node@boundary{}%
6881 \fi
6882 \csname forest@getboth\forestove{fit}edgesofpath\endcsname
6883 \forest@node@boundary\forest@negative@node@edge\forest@positive@node@edge
6884 \forestolet{negative@edge@#1}\forest@negative@node@edge
6885 \forestolet{positive@edge@#1}\forest@positive@node@edge
```

Add the edges of the children.

```
6886 \get@edges@merge{negative}{#1}%
6887 \get@edges@merge{positive}{#1}%
6888 }%
6889 }
```

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

```
6890 \def\get@edges@merge#1#2{%
6891 \ifnum\forestove{n children}>0
6892 \forestoget{#1@edge@#2}\forest@node@edge
```

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

```
6893 \forest@pointanchor{parent anchor}%
6894 \edef\forest@getedge@pa@l{\the\pgf@x}%
6895 \edef\forest@getedge@pa@s{\the\pgf@y}%
6896 \eappto\forest@node@edge{\noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@pa@l}{\forest@getedge@pa@s}}
```

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

```
6897 \pgfgettransform\forest@temp@transform
6898 \pgftransformreset
6899 \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.

```
6900 \def\forest@all@edges{%
6901 \forest@node@foreachchild{%
6902 \forestoget{#1@edge@#2}\forest@temp@edge
6903 \pgfpointtransformed{\pgfqpoint{\forestove{l}}{\forestove{s}}}%
6904 \forest@extendpath\forest@node@edge\forest@temp@edge}%
6905 \ifnum\forestove{ignore edge}=0
6906 \pgfpointadd
6907 {\pgfpointtransformed{\pgfqpoint{\forestove{l}}{\forestove{s}}}%
6908 {\forest@pointanchor{child anchor}}}%
6909 \pgfgetlastxy{\forest@getedge@ca@l}{\forest@getedge@ca@s}%
6910 \eappto\forest@all@edges{%
6911 \noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@pa@l}{\forest@getedge@pa@s}%
6912 \noexpand\pgfsyssoftpath@linetotoken{\forest@getedge@ca@l}{\forest@getedge@ca@s}%
6913 }%
6914 % this deals with potential overlap of the edges:
6915 \eappto\forest@node@edge{\noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@ca@l}{\forest@getedge@ca@s}}
6916 \fi
6917 }%
6918 \ifdefempty{\forest@all@edges}{-}%
6919 \pgfintersectionofpaths{\pgfsetpath\forest@all@edges}{\pgfsetpath\forest@node@edge}%
6920 \def\forest@edgenode@intersections{%
6921 \forest@merge@intersectionloop
6922 \eappto\forest@node@edge{\expandonce{\forest@all@edges}\expandonce{\forest@edgenode@intersections}}%
```

```

6923 }%
6924 \pgfsettransform\forest@temp@transform
    Process the path into an edge and store the edge.
6925 \csname forest@get#1\forestove{fit}edgeofpath\endcsname\forest@node@edge\forest@node@edge
6926 \forestolet{#1@edge@#2}\forest@node@edge
6927 \fi
6928 }
6929 %\newloop\forest@merge@loop
6930 \def\forest@merge@intersectionloop{%
6931 \c@pgf@counta=0
6932 \forest@loop
6933 \ifnum\c@pgf@counta<\pgfintersectionsolutions\relax
6934 \advance\c@pgf@counta1
6935 \pgfpointintersectionsolution{\the\c@pgf@counta}%
6936 \eappto\forest@edgenode@intersections{\noexpand\pgfsyssoftpath@movetotoken
6937 {\the\pgf@x}{\the\pgf@y}}%
6938 \forest@repeat
6939 }

    Get the bounding rectangle of the node (without descendants). #1 = grow.
6940 \def\forest@node@getboundingrectangle@ls#1{%
6941 \forestoget{@boundary}\forest@node@boundary
6942 \forest@path@getboundingrectangle@ls\forest@node@boundary{#1}%
6943 }

    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).
6944 \def\forest@pgfpathtransformed#1{%
6945 \forest@save@pgfsyssoftpath@tokendef
6946 \let\pgfsyssoftpath@movetotoken\forest@pgfpathtransformed@moveto
6947 \let\pgfsyssoftpath@linetotoken\forest@pgfpathtransformed@lineto
6948 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
6949 #1%
6950 \forest@restore@pgfsyssoftpath@tokendef
6951 }
6952 \def\forest@pgfpathtransformed@moveto#1#2{%
6953 \forest@pgfpathtransformed@op\pgfsyssoftpath@moveto{#1}{#2}%
6954 }
6955 \def\forest@pgfpathtransformed@lineto#1#2{%
6956 \forest@pgfpathtransformed@op\pgfsyssoftpath@lineto{#1}{#2}%
6957 }
6958 \def\forest@pgfpathtransformed@op#1#2#3{%
6959 \pgfpointtransformed{\pgfqpoint{#2}{#3}}%
6960 \edef\forest@temp{%
6961 \noexpand#1{\the\pgf@x}{\the\pgf@y}%
6962 }%
6963 \forest@temp
6964 }

```

8.2.1 Tiers

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

```

6965 \def\forest@pack@computetiers{%
6966 {%
6967 \forest@pack@tiers@getalltiersinsubtree
6968 \forest@pack@tiers@computetierhierarchy
6969 \forest@pack@tiers@findcontainers
6970 \forest@pack@tiers@raisecontainers
6971 \forest@pack@tiers@computeprocessingorder
6972 \gdef\forest@smuggle{}%
6973 \forest@pack@tiers@write

```

```

6974 }%
6975 \forest@node@foreach{\forestoset{@tiers}}{}%
6976 \forest@smuggle
6977 }

```

Puts all tiers contained in the subtree into attribute tiers.

```

6978 \def\forest@pack@tiers@getalltiersinsubtree{%
6979   \ifnum\forestove{n children}>0
6980     \forest@node@foreachchild{\forest@pack@tiers@getalltiersinsubtree}%
6981     \fi
6982     \forestoget{tier}\forest@temp@mytier
6983     \def\forest@temp@mytiers{}%
6984     \ifdefempty\forest@temp@mytier{}{%
6985       \listead\forest@temp@mytiers\forest@temp@mytier
6986     }%
6987     \ifnum\forestove{n children}>0
6988       \forest@node@foreachchild{%
6989         \forestoget{tiers}\forest@temp@tiers
6990         \forlistloop\forest@pack@tiers@forhandlerA\forest@temp@tiers
6991       }%
6992     \fi
6993     \forestolet{tiers}\forest@temp@mytiers
6994 }
6995 \def\forest@pack@tiers@forhandlerA#1{%
6996   \ifinlist{#1}\forest@temp@mytiers{}{%
6997     \listead\forest@temp@mytiers{#1}%
6998   }%
6999 }

```

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.

```

7000 \def\forest@pack@tiers@computetierhierarchy{%
7001   \def\forest@tiers@ancestors{}%
7002   \forestoget{tiers}\forest@temp@mytiers
7003   \forlistloop\forest@pack@tiers@cth@init\forest@temp@mytiers
7004   \forest@pack@tiers@computetierhierarchy@
7005 }
7006 \def\forest@pack@tiers@cth@init#1{%
7007   \csdef{forest@tiers@higher@#1}{}%
7008   \csdef{forest@tiers@lower@#1}{}%
7009 }
7010 \def\forest@pack@tiers@computetierhierarchy@{%
7011   \forestoget{tier}\forest@temp@mytier
7012   \ifdefempty\forest@temp@mytier{}{%
7013     \forlistloop\forest@pack@tiers@forhandlerB\forest@tiers@ancestors
7014     \listead\forest@tiers@ancestors\forest@temp@mytier
7015   }%
7016   \forest@node@foreachchild{%
7017     \forest@pack@tiers@computetierhierarchy@
7018   }%
7019   \forestoget{tier}\forest@temp@mytier
7020   \ifdefempty\forest@temp@mytier{}{%
7021     \forest@listedel\forest@tiers@ancestors\forest@temp@mytier
7022   }%
7023 }
7024 \def\forest@pack@tiers@forhandlerB#1{%
7025   \def\forest@temp@tier{#1}%
7026   \ifx\forest@temp@tier\forest@temp@mytier
7027     \PackageError{forest}{Circular tier hierarchy (tier \forest@temp@mytier)}{}%
7028   \fi
7029   \ifinlistcs{#1}{forest@tiers@higher@\forest@temp@mytier}{}{%

```

```

7030 \listcsadd{forest@tiers@higher@\forest@temp@mytier}{#1}}%
7031 \xifinlistcs{forest@temp@mytier{forest@tiers@lower@#1}}{ }{%
7032 \listcseadd{forest@tiers@lower@#1}{\forest@temp@mytier}}%
7033 }
7034 \def\forest@pack@tiers@findcontainers{%
7035 \foresttoget{tiers}\forest@temp@tiers
7036 \forlistloop\forest@pack@tiers@findcontainer\forest@temp@tiers
7037 }
7038 \def\forest@pack@tiers@findcontainer#1{%
7039 \def\forest@temp@tier{#1}%
7040 \foresttoget{tier}\forest@temp@mytier
7041 \ifx\forest@temp@tier\forest@temp@mytier
7042 \csedef{forest@tiers@container@#1}{\forest@cn}%
7043 \else\@escapeif{%
7044 \forest@pack@tiers@findcontainerA{#1}%
7045 }\fi%
7046 }
7047 \def\forest@pack@tiers@findcontainerA#1{%
7048 \c@pgf@counta=0
7049 \forest@node@foreachchild{%
7050 \foresttoget{tiers}\forest@temp@tiers
7051 \ifinlist{#1}\forest@temp@tiers{%
7052 \advance\c@pgf@counta 1
7053 \let\forest@temp@child\forest@cn
7054 } }%
7055 }%
7056 \ifnum\c@pgf@counta>1
7057 \csedef{forest@tiers@container@#1}{\forest@cn}%
7058 \else\@escapeif{% surely =1
7059 \forest@fornode{\forest@temp@child}{%
7060 \forest@pack@tiers@findcontainer{#1}%
7061 } }%
7062 }\fi
7063 }
7064 \def\forest@pack@tiers@raisecontainers{%
7065 \foresttoget{tiers}\forest@temp@mytiers
7066 \forlistloop\forest@pack@tiers@rc@forhandlerA\forest@temp@mytiers
7067 }
7068 \def\forest@pack@tiers@rc@forhandlerA#1{%
7069 \edef\forest@tiers@temptier{#1}%
7070 \letcs\forest@tiers@containernodeoftier{forest@tiers@container@#1}%
7071 \letcs\forest@temp@lowertiers{forest@tiers@lower@#1}%
7072 \forlistloop\forest@pack@tiers@rc@forhandlerB\forest@temp@lowertiers
7073 }
7074 \def\forest@pack@tiers@rc@forhandlerB#1{%
7075 \letcs\forest@tiers@containernodeoflowertier{forest@tiers@container@#1}%
7076 \forest@get{\forest@tiers@containernodeoflowertier}{content}\lowercontent
7077 \forest@get{\forest@tiers@containernodeoftier}{content}\uppercontent
7078 \forest@fornode{\forest@tiers@containernodeoflowertier}{%
7079 \forest@ifancestorof
7080 {\forest@tiers@containernodeoftier}
7081 {\csletcs{forest@tiers@container@\forest@tiers@temptier}{forest@tiers@container@#1}}%
7082 } }%
7083 }%
7084 }
7085 \def\forest@pack@tiers@computeprocessingorder{%
7086 \def\forest@tiers@processingorder{ }%
7087 \foresttoget{tiers}\forest@tiers@cpo@tierstodo
7088 \safeloop
7089 \ifdefempty\forest@tiers@cpo@tierstodo{\forest@tempfalse}{\forest@temptrue}%
7090 \ifforest@temp

```

```

7091 \def\forest@tiers@cpo@tiersremaining{%
7092 \def\forest@tiers@cpo@tiersindependent{%
7093 \forlistloop\forest@pack@tiers@cpo@forhandlerA\forest@tiers@cpo@tierstodo
7094 \ifdefempty\forest@tiers@cpo@tiersindependent{%
7095 \PackageError{forest}{Circular tiers!}{}}}%
7096 \forlistloop\forest@pack@tiers@cpo@forhandlerB\forest@tiers@cpo@tiersremaining
7097 \let\forest@tiers@cpo@tierstodo\forest@tiers@cpo@tiersremaining
7098 \saferepeat
7099 }
7100 \def\forest@pack@tiers@cpo@forhandlerA#1{%
7101 \ifcempty\forest@tiers@higher@#1{%
7102 \listadd\forest@tiers@cpo@tiersindependent{#1}%
7103 \listadd\forest@tiers@processingorder{#1}%
7104 }{%
7105 \listadd\forest@tiers@cpo@tiersremaining{#1}%
7106 }%
7107 }
7108 \def\forest@pack@tiers@cpo@forhandlerB#1{%
7109 \def\forest@pack@tiers@cpo@aremainingtier{#1}%
7110 \forlistloop\forest@pack@tiers@cpo@forhandlerC\forest@tiers@cpo@tiersindependent
7111 }
7112 \def\forest@pack@tiers@cpo@forhandlerC#1{%
7113 \ifinlistcs{#1}{\forest@tiers@higher@\forest@pack@tiers@cpo@aremainingtier}{%
7114 \forest@listcsdel{\forest@tiers@higher@\forest@pack@tiers@cpo@aremainingtier}{#1}%
7115 }{}%
7116 }
7117 \def\forest@pack@tiers@write{%
7118 \forlistloop\forest@pack@tiers@write@forhandler\forest@tiers@processingorder
7119 }
7120 \def\forest@pack@tiers@write@forhandler#1{%
7121 \forest@fornode{\csname forest@tiers@container@#1\endcsname}{%
7122 \forest@pack@tiers@check{#1}%
7123 }%
7124 \xappto\forest@smuggle{%
7125 \noexpand\listadd
7126 \forest@om{\csname forest@tiers@container@#1\endcsname}{@tiers}%
7127 {#1}%
7128 }%
7129 }
7130 % checks if the tier is compatible with growth changes and calign=node/edge angle
7131 \def\forest@pack@tiers@check#1{%
7132 \def\forest@temp@currenttier{#1}%
7133 \forest@node@foreachdescendant{%
7134 \ifnum\forestove{grow}=\forestove{\forestove{@parent}}{grow}
7135 \else
7136 \forest@pack@tiers@check@grow
7137 \fi
7138 \ifnum\forestove{n children}>1
7139 \forest@toget{calign}\forest@temp
7140 \ifx\forest@temp\forest@pack@tiers@check@nodeangle
7141 \forest@pack@tiers@check@calign
7142 \fi
7143 \ifx\forest@temp\forest@pack@tiers@check@edgeangle
7144 \forest@pack@tiers@check@calign
7145 \fi
7146 \fi
7147 }%
7148 }
7149 \def\forest@pack@tiers@check@nodeangle{node angle}%
7150 \def\forest@pack@tiers@check@edgeangle{edge angle}%
7151 \def\forest@pack@tiers@check@grow{%

```



```

7152 \foresttoget{content}\forest@temp@content
7153 \let\forest@temp@currentnode\forest@cn
7154 \forest@node@foreachdescendant{%
7155   \foresttoget{tier}\forest@temp
7156   \ifx\forest@temp@currentttier\forest@temp
7157     \forest@pack@tiers@check@grow@error
7158   \fi
7159 }%
7160 }
7161 \def\forest@pack@tiers@check@grow@error{%
7162   \PackageError{forest}{Tree growth direction changes in node \forest@temp@currentnode\space
7163     (content: \forest@temp@content), while tier '\forest@temp' is specified for nodes both
7164     out- and inside the subtree rooted in node \forest@temp@currentnode. This will not work.}{}%
7165 }
7166 \def\forest@pack@tiers@check@calign{%
7167   \forest@node@foreachchild{%
7168     \foresttoget{tier}\forest@temp
7169     \ifx\forest@temp@currentttier\forest@temp
7170       \forest@pack@tiers@check@calign@warning
7171     \fi
7172   }%
7173 }
7174 \def\forest@pack@tiers@check@calign@warning{%
7175   \PackageWarning{forest}{Potential option conflict: node \forestove{@parent} (content:
7176     '\forestOve{\forestove{@parent}}{content}') was given 'calign=\forestove{calign}', while its
7177     child \forest@cn\space (content: '\forestove{content}') was given 'tier=\forestove{tier}'.
7178     The parent's 'calign' will only work if the child was the lowest node on its tier before the
7179     alignment.}%
7180 }

```

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.

```

7181 \pgfdeclaregenericanchor{forestcomputenodeboundary}{%
7182   \letcs\forest@temp@boundary@macro{forest@compute@node@boundary@#1}%
7183   \ifcsname forest@compute@node@boundary@#1\endcsname
7184     \csname forest@compute@node@boundary@#1\endcsname
7185   \else
7186     \forest@compute@node@boundary@rectangle
7187   \fi
7188   \pgfsyssoftpath@getcurrentpath\forest@temp
7189   \global\let\forest@global@boundary\forest@temp
7190 }
7191 \def\forest@mt#1{%
7192   \expandafter\pgfpointanchor\expandafter{\pgfreferencednodename}{#1}%
7193   \pgfsyssoftpath@moveto{\the\pgf@x}{\the\pgf@y}%
7194 }%
7195 \def\forest@lt#1{%
7196   \expandafter\pgfpointanchor\expandafter{\pgfreferencednodename}{#1}%
7197   \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
7198 }%
7199 \def\forest@compute@node@boundary@coordinate{%
7200   \forest@mt{center}%
7201 }
7202 \def\forest@compute@node@boundary@circle{%
7203   \forest@mt{east}%
7204   \forest@lt{north east}%
7205   \forest@lt{north}%
7206   \forest@lt{north west}%

```

```

7207 \forest@lt{west}%
7208 \forest@lt{south west}%
7209 \forest@lt{south}%
7210 \forest@lt{south east}%
7211 \forest@lt{east}%
7212 }
7213 \def\forest@compute@node@boundary@rectangle{%
7214 \forest@mt{south west}%
7215 \forest@lt{south east}%
7216 \forest@lt{north east}%
7217 \forest@lt{north west}%
7218 \forest@lt{south west}%
7219 }
7220 \def\forest@compute@node@boundary@diamond{%
7221 \forest@mt{east}%
7222 \forest@lt{north}%
7223 \forest@lt{west}%
7224 \forest@lt{south}%
7225 \forest@lt{east}%
7226 }
7227 \let\forest@compute@node@boundary@ellipse\forest@compute@node@boundary@circle
7228 \def\forest@compute@node@boundary@trapezium{%
7229 \forest@mt{top right corner}%
7230 \forest@lt{top left corner}%
7231 \forest@lt{bottom left corner}%
7232 \forest@lt{bottom right corner}%
7233 \forest@lt{top right corner}%
7234 }
7235 \def\forest@compute@node@boundary@semicircle{%
7236 \forest@mt{arc start}%
7237 \forest@lt{north}%
7238 \forest@lt{east}%
7239 \forest@lt{north east}%
7240 \forest@lt{apex}%
7241 \forest@lt{north west}%
7242 \forest@lt{west}%
7243 \forest@lt{arc end}%
7244 \forest@lt{arc start}%
7245 }
7246 %\newloop\forest@computenodeboundary@loop
7247 \csdef{forest@compute@node@boundary@regular polygon}{%
7248 \forest@mt{corner 1}%
7249 \c@pgf@counta=\sides\relax
7250 \forest@loop
7251 \ifnum\c@pgf@counta>0
7252 \forest@lt{corner \the\c@pgf@counta}%
7253 \advance\c@pgf@counta-1
7254 \forest@repeat
7255 }%
7256 \def\forest@compute@node@boundary@star{%
7257 \forest@mt{outer point 1}%
7258 \c@pgf@counta=\totalstarpoints\relax
7259 \divide\c@pgf@counta2
7260 \forest@loop
7261 \ifnum\c@pgf@counta>0
7262 \forest@lt{inner point \the\c@pgf@counta}%
7263 \forest@lt{outer point \the\c@pgf@counta}%
7264 \advance\c@pgf@counta-1
7265 \forest@repeat
7266 }%
7267 \csdef{forest@compute@node@boundary@isosceles triangle}{%

```

```

7268 \forest@mt{apex}%
7269 \forest@lt{left corner}%
7270 \forest@lt{right corner}%
7271 \forest@lt{apex}%
7272 }
7273 \def\forest@compute@node@boundary@kite{%
7274 \forest@mt{upper vertex}%
7275 \forest@lt{left vertex}%
7276 \forest@lt{lower vertex}%
7277 \forest@lt{right vertex}%
7278 \forest@lt{upper vertex}%
7279 }
7280 \def\forest@compute@node@boundary@dart{%
7281 \forest@mt{tip}%
7282 \forest@lt{left tail}%
7283 \forest@lt{tail center}%
7284 \forest@lt{right tail}%
7285 \forest@lt{tip}%
7286 }
7287 \csdef{forest@compute@node@boundary@circular sector}{%
7288 \forest@mt{sector center}%
7289 \forest@lt{arc start}%
7290 \forest@lt{arc center}%
7291 \forest@lt{arc end}%
7292 \forest@lt{sector center}%
7293 }
7294 \def\forest@compute@node@boundary@cylinder{%
7295 \forest@mt{top}%
7296 \forest@lt{after top}%
7297 \forest@lt{before bottom}%
7298 \forest@lt{bottom}%
7299 \forest@lt{after bottom}%
7300 \forest@lt{before top}%
7301 \forest@lt{top}%
7302 }
7303 \cslet{forest@compute@node@boundary@forbidden sign}\forest@compute@node@boundary@circle
7304 \cslet{forest@compute@node@boundary@magnifying glass}\forest@compute@node@boundary@circle
7305 \def\forest@compute@node@boundary@cloud{%
7306 \getradii
7307 \forest@mt{puff 1}%
7308 \c@pgf@counta=\puffs\relax
7309 \forest@loop
7310 \ifnum\c@pgf@counta>0
7311 \forest@lt{puff \the\c@pgf@counta}%
7312 \advance\c@pgf@counta-1
7313 \forest@repeat
7314 }
7315 \def\forest@compute@node@boundary@starburst{
7316 \calculatestarburstpoints
7317 \forest@mt{outer point 1}%
7318 \c@pgf@counta=\totalpoints\relax
7319 \divide\c@pgf@counta2
7320 \forest@loop
7321 \ifnum\c@pgf@counta>0
7322 \forest@lt{inner point \the\c@pgf@counta}%
7323 \forest@lt{outer point \the\c@pgf@counta}%
7324 \advance\c@pgf@counta-1
7325 \forest@repeat
7326 }%
7327 \def\forest@compute@node@boundary@signal{%
7328 \forest@mt{east}%

```

```

7329 \forest@lt{south east}%
7330 \forest@lt{south west}%
7331 \forest@lt{west}%
7332 \forest@lt{north west}%
7333 \forest@lt{north east}%
7334 \forest@lt{east}%
7335 }
7336 \def\forest@compute@node@boundary@tape{%
7337 \forest@mt{north east}%
7338 \forest@lt{60}%
7339 \forest@lt{north}%
7340 \forest@lt{120}%
7341 \forest@lt{north west}%
7342 \forest@lt{south west}%
7343 \forest@lt{240}%
7344 \forest@lt{south}%
7345 \forest@lt{310}%
7346 \forest@lt{south east}%
7347 \forest@lt{north east}%
7348 }
7349 \csdef{forest@compute@node@boundary@single arrow}{%
7350 \forest@mt{tip}%
7351 \forest@lt{after tip}%
7352 \forest@lt{after head}%
7353 \forest@lt{before tail}%
7354 \forest@lt{after tail}%
7355 \forest@lt{before head}%
7356 \forest@lt{before tip}%
7357 \forest@lt{tip}%
7358 }
7359 \csdef{forest@compute@node@boundary@double arrow}{%
7360 \forest@mt{tip 1}%
7361 \forest@lt{after tip 1}%
7362 \forest@lt{after head 1}%
7363 \forest@lt{before head 2}%
7364 \forest@lt{before tip 2}%
7365 \forest@mt{tip 2}%
7366 \forest@lt{after tip 2}%
7367 \forest@lt{after head 2}%
7368 \forest@lt{before head 1}%
7369 \forest@lt{before tip 1}%
7370 \forest@lt{tip 1}%
7371 }
7372 \csdef{forest@compute@node@boundary@arrow box}{%
7373 \forest@mt{before north arrow}%
7374 \forest@lt{before north arrow head}%
7375 \forest@lt{before north arrow tip}%
7376 \forest@lt{north arrow tip}%
7377 \forest@lt{after north arrow tip}%
7378 \forest@lt{after north arrow head}%
7379 \forest@lt{after north arrow}%
7380 \forest@lt{north east}%
7381 \forest@lt{before east arrow}%
7382 \forest@lt{before east arrow head}%
7383 \forest@lt{before east arrow tip}%
7384 \forest@lt{east arrow tip}%
7385 \forest@lt{after east arrow tip}%
7386 \forest@lt{after east arrow head}%
7387 \forest@lt{after east arrow}%
7388 \forest@lt{south east}%
7389 \forest@lt{before south arrow}%

```

```

7390 \forest@lt{before south arrow head}%
7391 \forest@lt{before south arrow tip}%
7392 \forest@lt{south arrow tip}%
7393 \forest@lt{after south arrow tip}%
7394 \forest@lt{after south arrow head}%
7395 \forest@lt{after south arrow}%
7396 \forest@lt{south west}%
7397 \forest@lt{before west arrow}%
7398 \forest@lt{before west arrow head}%
7399 \forest@lt{before west arrow tip}%
7400 \forest@lt{west arrow tip}%
7401 \forest@lt{after west arrow tip}%
7402 \forest@lt{after west arrow head}%
7403 \forest@lt{after west arrow}%
7404 \forest@lt{north west}%
7405 \forest@lt{before north arrow}%
7406 }
7407 \cslet{forest@compute@node@boundary@circle split}\forest@compute@node@boundary@circle
7408 \cslet{forest@compute@node@boundary@circle solidus}\forest@compute@node@boundary@circle
7409 \cslet{forest@compute@node@boundary@ellipse split}\forest@compute@node@boundary@ellipse
7410 \cslet{forest@compute@node@boundary@rectangle split}\forest@compute@node@boundary@rectangle
7411 \def\forest@compute@node@boundary@@callout{%
7412 \beforecalloutpointer
7413 \pgfsyssoftpath@moveto{\the\pgf@x}{\the\pgf@y}%
7414 \calloutpointeranchor
7415 \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
7416 \aftercalloutpointer
7417 \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
7418 }
7419 \csdef{forest@compute@node@boundary@rectangle callout}{%
7420 \forest@compute@node@boundary@rectangle
7421 \rectanglecalloutpoints
7422 \forest@compute@node@boundary@@callout
7423 }
7424 \csdef{forest@compute@node@boundary@ellipse callout}{%
7425 \forest@compute@node@boundary@ellipse
7426 \ellipsecalloutpoints
7427 \forest@compute@node@boundary@@callout
7428 }
7429 \csdef{forest@compute@node@boundary@cloud callout}{%
7430 \forest@compute@node@boundary@cloud
7431 % at least a first approx...
7432 \forest@mt{center}%
7433 \forest@lt{pointer}%
7434 }%
7435 \csdef{forest@compute@node@boundary@cross out}{%
7436 \forest@mt{south east}%
7437 \forest@lt{north west}%
7438 \forest@mt{south west}%
7439 \forest@lt{north east}%
7440 }%
7441 \csdef{forest@compute@node@boundary@strike out}{%
7442 \forest@mt{north east}%
7443 \forest@lt{south west}%
7444 }%
7445 \csdef{forest@compute@node@boundary@rounded rectangle}{%
7446 \forest@mt{east}%
7447 \forest@lt{north east}%
7448 \forest@lt{north}%
7449 \forest@lt{north west}%
7450 \forest@lt{west}%

```

```

7451 \forest@lt{south west}%
7452 \forest@lt{south}%
7453 \forest@lt{south east}%
7454 \forest@lt{east}%
7455 }%
7456 \csdef{forest@compute@node@boundary@chamfered rectangle}{%
7457 \forest@mt{before south west}%
7458 \forest@mt{after south west}%
7459 \forest@lt{before south east}%
7460 \forest@lt{after south east}%
7461 \forest@lt{before north east}%
7462 \forest@lt{after north east}%
7463 \forest@lt{before north west}%
7464 \forest@lt{after north west}%
7465 \forest@lt{before south west}%
7466 }%

```

8.3 Compute absolute positions

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

```

7467 \def\forest@node@computeabsolutepositions{%
7468 \edef\forest@marshal{%
7469 \noexpand\forest@node@foreachchild{%
7470 \noexpand\forest@node@computeabsolutepositions@{\forestove{x}}{\forestove{y}}{\forestove{grow}}}%
7471 }%
7472 }\forest@marshal
7473 }
7474 \def\forest@node@computeabsolutepositions@#1#2#3{%
7475 \pgfpointadd
7476 {\pgfqpoint{#1}{#2}}%
7477 {\pgfpointadd
7478 {\pgfqpointpolar{#3}{\forestove{1}}}%
7479 {\pgfqpointpolar{\numexpr 90+#3\relax}{\forestove{s}}}%
7480 }%
7481 \pgfgetlastxy\forest@temp@x\forest@temp@y
7482 \forestolet{x}\forest@temp@x
7483 \forestolet{y}\forest@temp@y
7484 \edef\forest@marshal{%
7485 \noexpand\forest@node@foreachchild{%
7486 \noexpand\forest@node@computeabsolutepositions@{\forest@temp@x}{\forest@temp@y}{\forestove{grow}}}%
7487 }%
7488 }\forest@marshal
7489 }

```

8.4 Drawing the tree

```

7490 \newif\ifforest@drawtree@preservenodeboxes@
7491 \def\forest@node@drawtree{%
7492 \expandafter\ifstrequal\expandafter{\forest@drawtreebox}{\pgfkeysnovalue}{%
7493 \let\forest@drawtree@beginbox\relax
7494 \let\forest@drawtree@endbox\relax
7495 }%
7496 \edef\forest@drawtree@beginbox{\global\setbox\forest@drawtreebox=\hbox\bgroup}%
7497 \let\forest@drawtree@endbox\egroup
7498 }%
7499 \ifforest@external@
7500 \ifforest@externalize@tree@
7501 \forest@tempttrue
7502 \else

```

```

7503 \tikzifexternalizing{%
7504 \ifforest@was@tikzexternalwasenable
7505 \forest@temptrue
7506 \pgfkeys{/tikz/external/optimize=false}%
7507 \let\forest@drawtree@beginbox\relax
7508 \let\forest@drawtree@endbox\relax
7509 \else
7510 \forest@tempfalse
7511 \fi
7512 }{%
7513 \forest@tempfalse
7514 }%
7515 \fi
7516 \ifforest@temp
7517 \advance\forest@externalize@inner@n 1
7518 \edef\forest@externalize@filename{%
7519 \tikzexternalrealjob-forest-\forest@externalize@outer@n
7520 \ifnum\forest@externalize@inner@n=0 \else.\the\forest@externalize@inner@n\fi}%
7521 \expandafter\tikzsetnextfilename\expandafter{\forest@externalize@filename}%
7522 \tikzexternalenable
7523 \pgfkeysalso{/tikz/external/remake next,/tikz/external/export next}%
7524 \fi
7525 \ifforest@externalize@tree@
7526 \typeout{forest: Invoking a recursive call to generate the external picture
7527 '\forest@externalize@filename' for the following context+code:
7528 '\expandafter\detokenize\expandafter{\forest@externalize@id}'}%
7529 \fi
7530 \fi
7531 %
7532 \ifforest\tikzcsstack
7533 \let\forest@original@tikz@parse@node\tikz@parse@node
7534 \let\tikz@parse@node\forest@tikz@parse@node
7535 \fi
7536 \pgfkeysgetvalue{/forest/begin draw/.@cmd}\forest@temp@begindraw
7537 \pgfkeysgetvalue{/forest/end draw/.@cmd}\forest@temp@enddraw
7538 \edef\forest@marshal{%
7539 \noexpand\forest@drawtree@beginbox
7540 \expandonce{\forest@temp@begindraw\pgfkeysnovalue\pgfeov}%
7541 \noexpand\forest@node@drawtree@
7542 \expandonce{\forest@temp@enddraw\pgfkeysnovalue\pgfeov}%
7543 \noexpand\forest@drawtree@endbox
7544 }\forest@marshal
7545 \ifforest\tikzcsstack
7546 \let\tikz@parse@node\forest@original@tikz@parse@node
7547 \fi
7548 %
7549 \ifforest@external@
7550 \ifforest@externalize@tree@
7551 \tikzexternaldisable
7552 \eappto\forest@externalize@checkimages{%
7553 \noexpand\forest@includeexternal@check{\forest@externalize@filename}%
7554 }%
7555 \expandafter\ifstrequal\expandafter{\forest@drawtreebox}{\pgfkeysnovalue}{%
7556 \eappto\forest@externalize@loadimages{%
7557 \noexpand\forest@includeexternal{\forest@externalize@filename}%
7558 }%
7559 }{%
7560 \eappto\forest@externalize@loadimages{%
7561 \noexpand\forest@includeexternal@box\forest@drawtreebox{\forest@externalize@filename}%
7562 }%
7563 }%

```

```

7564 \fi
7565 \fi
7566 }
7567 \def\forest@drawtree@root{0}
7568 \def\forest@node@drawtree@{%
7569 \def\forest@clear@drawn{%
7570 \forest@forthis{%
7571 \forest@saveandrestoremacro\forest@drawtree@root{%
7572 \edef\forest@drawtree@root{\forest@cn}%
7573 \forestset{draw tree method}%
7574 }%
7575 }%
7576 \forest@node@ifnamedefined{forest@baseline@node}{%
7577 \edef\forest@baseline@id{\forest@node@Nametoid{forest@baseline@node}}%
7578 \ifcsdef{forest@drawn@\forest@baseline@id}{%
7579 \edef\forest@marshal{%
7580 \noexpand\pgfsetbaselinepointlater{%
7581 \noexpand\pgfpointanchor
7582 {\forestOve{\forest@baseline@id}{name}}%
7583 {\forestOve{\forest@baseline@id}{anchor}}%
7584 }%
7585 }\forest@marshal
7586 }{}%
7587 }{}%
7588 \forest@clear@drawn
7589 }
7590 \def\forest@draw@node{%
7591 \ifnum\forestove{phantom}=0
7592 \forest@draw@node@
7593 \fi
7594 }
7595 \def\forest@draw@node@{%
7596 \forest@node@forest@positionnodelater@restore
7597 \ifforest@drawtree@preservenodeboxes@
7598 \pgfnodealias{forest@temp}{\forestove{later@name}}%
7599 \fi
7600 \pgfpositionnodenow{\pgfqpoint{\forestove{x}}{\forestove{y}}}%
7601 \ifforest@drawtree@preservenodeboxes@
7602 \pgfnodealias{\forestove{later@name}}{forest@temp}%
7603 \fi
7604 \csdef{forest@drawn@\forest@cn}{}%
7605 \eappto\forest@clear@drawn{\noexpand\csundef{forest@drawn@\forest@cn}}%
7606 }
7607 \def\forest@draw@edge{%
7608 \ifcsdef{forest@drawn@\forest@cn}{% was the current node drawn?
7609 \ifnum\forestove{@parent}=0 % do we have a parent?
7610 \else
7611 \ifcsdef{forest@drawn@\forestove{@parent}}{% was the parent drawn?
7612 \forest@draw@edge@
7613 }{}%
7614 \fi
7615 }{}%
7616 }
7617 \def\forest@draw@edge@{%
7618 \edef\forest@temp{\forestove{edge path}}\forest@temp
7619 }
7620 \def\forest@draw@tikz{%
7621 \ifnum\forestove{phantom}=0
7622 \forest@draw@tikz@
7623 \fi
7624 }

```



```

7625 \def\forest@draw@tikz{%
7626   \forestove{tikz}%
7627 }

```

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.

```

7628 \newdimen\forest@xg
7629 \newdimen\forest@yg
7630 \newdimen\forest@xs
7631 \newdimen\forest@ys
7632 \def\forest@setupgrowline#1{%
7633   \edef\forest@grow{#1}%
7634   \pgfqpointpolar{\forest@grow}{1pt}%
7635   \forest@xg=\pgf@x
7636   \forest@yg=\pgf@y
7637   \forest@xs=-\pgf@y
7638   \forest@ys=\pgf@x
7639 }

```

9.1 Projections

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

```

7640 \def\forest@pgfpointprojectiontogrowline#1{%
7641   \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.

```

7642   \pgfutil@tempdima=\pgf@sys@tonumber{\pgf@x}\forest@xg%
7643   \advance\pgfutil@tempdima by\pgf@sys@tonumber{\pgf@y}\forest@yg%

```

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

```

7644   \global\pgf@x=\pgf@sys@tonumber{\pgfutil@tempdima}\forest@xg%
7645   \global\pgf@y=\pgf@sys@tonumber{\pgfutil@tempdima}\forest@yg%
7646 }

```

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.

```

7647 \def\forest@distancetogrowline#1#2{%
7648   \pgf@process{#2}%
7649   #1=\pgf@sys@tonumber{\pgf@x}\forest@xs\relax
7650   \advance#1 by\pgf@sys@tonumber{\pgf@y}\forest@ys\relax
7651 }

```

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 `(xo, yo, xp, yp)`, where `xo` and `yo` stand for the *original* point, and `xp` and `yp` 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`.

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

```

7652 \let\forest@pp@n\relax
7653 \def\forest@projectpath@togrowline#1#2{%
7654   \edef\forest@pp@prefix{#2}%
7655   \forest@save@pgfsyssoftpath@tokendefs
7656   \let\pgfsyssoftpath@movetotoken\forest@projectpath@processpoint
7657   \let\pgfsyssoftpath@linetotoken\forest@projectpath@processpoint
7658   \c@pgf@counta=0
7659   #1%
7660   \csedef{#2n}{\the\c@pgf@counta}%
7661   \forest@restore@pgfsyssoftpath@tokendefs
7662 }

```

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

```

7663 \def\forest@projectpath@processpoint#1#2{%
7664   \pgfqpoint{#1}{#2}%
7665   \expandafter\edef\c@name\forest@pp@prefix\the\c@pgf@counta xo\endcsname{\the\pgf@x}%
7666   \expandafter\edef\c@name\forest@pp@prefix\the\c@pgf@counta yo\endcsname{\the\pgf@y}%
7667   \forest@pgfpointprojection@togrowline{}%
7668   \expandafter\edef\c@name\forest@pp@prefix\the\c@pgf@counta xp\endcsname{\the\pgf@x}%
7669   \expandafter\edef\c@name\forest@pp@prefix\the\c@pgf@counta yp\endcsname{\the\pgf@y}%
7670   \advance\c@pgf@counta 1\relax
7671 }

```

Sort the array (prefix #1) produced by `\forest@projectpath@togrowline` by (x,y), in the ascending order.

```

7672 \def\forest@sort@projections#1{%
7673   % todo: optimize in cases when we know that the array is actually a
7674   % merger of sorted arrays; when does this happen? in
7675   % distance_between_paths, and when merging the edges of the parent
7676   % and its children in a uniform growth tree
7677   \edef\forest@ppi@inputprefix{#1}%
7678   \c@pgf@counta=\c@name#1n\endcsname\relax
7679   \advance\c@pgf@counta -1
7680   \forest@sort\forest@ppiraw@cmp\forest@ppiraw\let\forest@sort@ascending{0}{\the\c@pgf@counta}%
7681 }

```

The following macro processes the data gathered by (possibly more than one invocation of) `\forest@projectpath@togrowline` 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 #2: 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

```

7682 \def\forest@processprojectioninfo#1#2{%
7683   \edef\forest@ppi@inputprefix{#1}%

```

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

```

7684   \c@pgf@counta=0
7685   \c@pgf@countb=-1
7686   \safeloop
7687   \ifnum\c@pgf@counta<\c@name#1n\endcsname\relax

```

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

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

```

7688 \letcs\forest@xo{#1\the\c@pgf@counta xo}%
7689 \letcs\forest@yo{#1\the\c@pgf@counta yo}%
7690 \letcs\forest@xp{#1\the\c@pgf@counta xp}%
7691 \letcs\forest@yp{#1\the\c@pgf@counta yp}%
7692 \ifnum\c@pgf@countb<0
7693 \forest@equaltotolerancefalse
7694 \else
7695 \forest@equaltotolerance
7696 {\pgfqpoint\forest@xp\forest@yp}%
7697 {\pgfqpoint
7698 {\csname#2\the\c@pgf@countb x\endcsname}%
7699 {\csname#2\the\c@pgf@countb y\endcsname}%
7700 }%
7701 \fi
7702 \ifforest@equaltotolerance\else

```

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

```

7703 \advance\c@pgf@countb 1
7704 \cslet{#2\the\c@pgf@countb x}\forest@xp
7705 \cslet{#2\the\c@pgf@countb y}\forest@yp
7706 \csdef{#2\the\c@pgf@countb @n}{0}%
7707 \fi

```

If the projection is actually a projection of one a point in our path:

```

7708 % todo: this is ugly!
7709 \ifdefined\forest@xo\ifx\forest@xo\relax\else
7710 \ifdefined\forest@yo\ifx\forest@yo\relax\else

```

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

```

7711 \forest@append@point@to@inner@array
7712 \forest@xo\forest@yo
7713 {#2\the\c@pgf@countb @}%

```

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

```

7714 \csedef{#2(\forest@xo,\forest@yo)}{\the\c@pgf@countb}%
7715 \fi\fi
7716 \fi\fi

```

Clean-up the raw array item.

```

7717 \cslet{#1\the\c@pgf@counta xo}\relax
7718 \cslet{#1\the\c@pgf@counta yo}\relax
7719 \cslet{#1\the\c@pgf@counta xp}\relax
7720 \cslet{#1\the\c@pgf@counta yp}\relax
7721 \advance\c@pgf@counta 1
7722 \saferepeat

```

Clean up the raw array length.

```

7723 \cslet{#1n}\relax

```

Store the length of the outer result array.

```

7724 \advance\c@pgf@countb 1
7725 \csedef{#2n}{\the\c@pgf@countb}%
7726 }

```

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

```

7727 \def\forest@ppiraw@let#1#2{%
7728 \csletcs{\forest@ppi@inputprefix#1xo}{\forest@ppi@inputprefix#2xo}%
7729 \csletcs{\forest@ppi@inputprefix#1yo}{\forest@ppi@inputprefix#2yo}%
7730 \csletcs{\forest@ppi@inputprefix#1xp}{\forest@ppi@inputprefix#2xp}%
7731 \csletcs{\forest@ppi@inputprefix#1yp}{\forest@ppi@inputprefix#2yp}%
7732 }

```

Item comparison macro for quicksorting the raw projection data.

```

7733 \def\forest@ppiraw@cmp#1#2{%
7734   \forest@sort@cmptwodimcs
7735   {\forest@ppi@inputprefix#1xp}{\forest@ppi@inputprefix#1yp}%
7736   {\forest@ppi@inputprefix#2xp}{\forest@ppi@inputprefix#2yp}%
7737 }

```

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

```

7738 \def\forest@append@point@to@inner@array#1#2#3{%
7739   \c@pgf@countc=\csname#3n\endcsname\relax
7740   \csedef{#3\the\c@pgf@countc x}{#1}%
7741   \csedef{#3\the\c@pgf@countc y}{#2}%
7742   \forest@distancetogrowline\pgfutil@tempdima{\pgfqpoint#1#2}%
7743   \csedef{#3\the\c@pgf@countc d}{\the\pgfutil@tempdima}%
7744   \advance\c@pgf@countc 1
7745   \csedef{#3n}{\the\c@pgf@countc}%
7746 }

```

9.2 Break path

The following macro computes from the given path (#1) a “broken” path (#3) 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 with prefix #2. The macro updates the information compiled by `\forest@processprojectioninfo` with information about points added by path-breaking.

```

7747 \def\forest@breakpath#1#2#3{%
7748   \edef\forest@bp@prefix{#2}%
7749   \forest@save@pgfsyssoftpath@tokendefs
7750   \let\pgfsyssoftpath@movetotoken\forest@breakpath@processfirstpoint
7751   \let\pgfsyssoftpath@linetotoken\forest@breakpath@processfirstpoint
7752   %\pgfusepath{}% empty the current path. ok?
7753   #1%
7754   \forest@restore@pgfsyssoftpath@tokendefs
7755   \pgfsyssoftpath@getcurrentpath#3%
7756 }

```

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.)

```

7757 \def\forest@breakpath@processfirstpoint#1#2{%
7758   \forest@breakpath@processmoveto{#1}{#2}%
7759   \let\pgfsyssoftpath@movetotoken\forest@breakpath@processmoveto
7760   \let\pgfsyssoftpath@linetotoken\forest@breakpath@processlineto
7761 }

```

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.

```

7762 \def\forest@breakpath@processmoveto#1#2{%
7763   \pgfsyssoftpath@moveto{#1}{#2}%
7764   \def\forest@previous@x{#1}%
7765   \def\forest@previous@y{#2}%
7766   \expandafter\let\expandafter\forest@previous@i
7767     \csname\forest@bp@prefix(#1,#2)\endcsname
7768   \expandafter\let\expandafter\forest@previous@px
7769     \csname\forest@bp@prefix\forest@previous@i x\endcsname

```

```

7770 \expandafter\let\expandafter\forest@previous@py
7771 \csname\forest@bp@prefix\forest@previous@i y\endcsname
7772 }

```

This is the heart of the path-breaking procedure.

```

7773 \def\forest@breakpath@processlineto#1#2{%

```

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

```

7774 \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.)

```

7775 \expandafter\let\expandafter\forest@i
7776 \csname\forest@bp@prefix(#1,#2)\endcsname
7777 \expandafter\let\expandafter\forest@px
7778 \csname\forest@bp@prefix\forest@i x\endcsname
7779 \expandafter\let\expandafter\forest@py
7780 \csname\forest@bp@prefix\forest@i y\endcsname

```

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

```

7781 \forest@equaltotolerance
7782 {\pgfqpoint{\forest@previous@px}{\forest@previous@py}}%
7783 {\pgfqpoint{\forest@px}{\forest@py}}%
7784 \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.

```

7785 \let\forest@breakpath@op\pgfsyssoftpath@moveto
7786 \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.

```

7787 \forest@temp@count=\forest@previous@i\relax
7788 \ifnum\forest@previous@i<\forest@i\relax
7789 \def\forest@breakpath@step{1}%
7790 \def\forest@breakpath@test{\forest@temp@count<\forest@i\relax}%
7791 \else
7792 \def\forest@breakpath@step{-1}%
7793 \def\forest@breakpath@test{\forest@temp@count>\forest@i\relax}%
7794 \fi

```

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

```

7795 \safeloop
7796 \advance\forest@temp@count\forest@breakpath@step\relax
7797 \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.)

```

7798 \pgfpointintersectionoflines
7799 {\pgfqpoint
7800 {\csname\forest@bp@prefix\the\forest@temp@count x\endcsname}%
7801 {\csname\forest@bp@prefix\the\forest@temp@count y\endcsname}%
7802 }%
7803 {\pgfpointadd
7804 {\pgfqpoint
7805 {\csname\forest@bp@prefix\the\forest@temp@count x\endcsname}%
7806 {\csname\forest@bp@prefix\the\forest@temp@count y\endcsname}%
7807 }%
7808 {\pgfqpoint{\forest@xs}{\forest@ys}}%
7809 }%
7810 {\pgfqpoint{\forest@previous@x}{\forest@previous@y}}%
7811 {\pgfqpoint{#1}{#2}}%

```

Break the segment at the intersection.

```

7812 \pgfgetlastxy\forest@last@x\forest@last@y
7813 \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
Append the breaking point to the inner array for the projection.
7814 \forest@append@point@to@inner@array
7815 \forest@last@x\forest@last@y
7816 {\forest@bp@prefix\the\forest@temp@count @}%
Cache the projection of the new segment edge.
7817 \csedef{\forest@bp@prefix(\the\pgf@x,\the\pgf@y)}{\the\forest@temp@count}%
7818 \saferepeat
7819 \fi

```

Add the current point.

```

7820 \forest@breakpath@op{#1}{#2}%
Setup new “previous” info: the segment edge, its projection’s index, and the projection.
7821 \def\forest@previous@x{#1}%
7822 \def\forest@previous@y{#2}%
7823 \let\forest@previous@i\forest@i
7824 \let\forest@previous@px\forest@px
7825 \let\forest@previous@py\forest@py
7826 }

```

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

```

7827 \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.

```

7828 \def\forest@getnegativetightedgeofpath#1#2{%
7829 \forest@get@onetightedgeofpath#1\forest@sort@ascending#2}
7830 \def\forest@getpositivetightedgeofpath#1#2{%
7831 \forest@get@onetightedgeofpath#1\forest@sort@descending#2}
7832 \def\forest@get@onetightedgeofpath#1#2#3{%
7833 {%
7834 \forest@get@one@tightedgeofpath#1#2\forest@gep@edge
7835 \global\let\forest@gep@global@edge\forest@gep@edge
7836 }%
7837 \let#3\forest@gep@global@edge
7838 }
7839 \def\forest@get@one@tightedgeofpath#1#2#3{%

```

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

```

7840 \forest@projectpathtogrowline#1{\forest@pp@}%
7841 \forest@sortprojections{\forest@pp@}%
7842 \forest@processprojectioninfo{\forest@pp@}{\forest@pi@}%

```

Break the path.

```

7843 \forest@breakpath#1{\forest@pi@}\forest@brokenpath

```

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

Compile some more useful information.

```
7844 \forest@sort@inner@arrays{forest@pi@}#2%
7845 \forest@pathtodict\forest@brokenpath{forest@pi@}%
```

The auxiliary data is set up: do the work!

```
7846 \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).

```
7847 \forest@simplifypath\forest@edge#3%
7848 }
```

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

```
7849 \def\forest@getbothtightededgesofpath#1#2#3{%
7850 {%
7851 \forest@get@one@tightedgeofpath#1\forest@sort@ascending\forest@gep@firstedge
```

Reverse the order of items in the inner arrays.

```
7852 \c@pgf@counta=0
7853 \forest@loop
7854 \ifnum\c@pgf@counta<\forest@pi@n\relax
7855 \forest@ppi@deflet{forest@pi@\the\c@pgf@counta @}%
7856 \forest@reversearray\forest@ppi@let
7857 {0}%
7858 {\csname forest@pi@\the\c@pgf@counta @n\endcsname}%
7859 \advance\c@pgf@counta 1
7860 \forest@repeat
```

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

```
7861 \forest@gettightedgeofpath@getedge\forest@edge
7862 \forest@simplifypath\forest@edge\forest@gep@secondedge
```

Smuggle the results out of the enclosing `TEX` group.

```
7863 \global\let\forest@gep@global@firstedge\forest@gep@firstedge
7864 \global\let\forest@gep@global@secondedge\forest@gep@secondedge
7865 }%
7866 \let#2\forest@gep@global@firstedge
7867 \let#3\forest@gep@global@secondedge
7868 }
```

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

```
7869 \def\forest@sort@inner@arrays#1#2{%
7870 \c@pgf@counta=0
7871 \safeloop
7872 \ifnum\c@pgf@counta<\csname#1n\endcsname
7873 \c@pgf@countb=\csname#1\the\c@pgf@counta @n\endcsname\relax
7874 \ifnum\c@pgf@countb>1
7875 \advance\c@pgf@countb -1
7876 \forest@ppi@deflet{#1\the\c@pgf@counta @}%
7877 \forest@ppi@defcmp{#1\the\c@pgf@counta @}%
7878 \forest@sort\forest@ppi@cmp\forest@ppi@let#2{0}{\the\c@pgf@countb}%
7879 \fi
7880 \advance\c@pgf@counta 1
7881 \saferepeat
7882 }
```

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.

```
7883 \def\forest@ppi@deflet#1{%
7884 \edef\forest@ppi@let##1##2{%
7885 \noexpand\csletcs{#1##1x}{#1##2x}%
7886 \noexpand\csletcs{#1##1y}{#1##2y}%
```

```

7887 \noexpand\csletcs{#1##1d}{#1##2d}%
7888 }%
7889 }

```

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.

```

7890 \def\forest@ppi@defcmp#1{%
7891 \edef\forest@ppi@cmp##1##2{%
7892 \noexpand\forest@sort@cmpdimcs{#1##1d}{#1##2d}%
7893 }%
7894 }

```

Put path segments into a “segment dictionary”: for each segment of the path from (x_1, y_1) to (x_2, y_2) let $\text{\forest@}(x_1, y_1) \text{--}(x_2, y_2)$ be \forest@inpath (which can be anything but \relax).

```

7895 \let\forest@inpath\advance

```

This macro is just a wrapper to process the path.

```

7896 \def\forest@pathtodict#1#2{%
7897 \edef\forest@pathtodict@prefix{#2}%
7898 \forest@save@pgfsyssoftpath@tokendefs
7899 \let\pgfsyssoftpath@movetotoken\forest@pathtodict@movetoop
7900 \let\pgfsyssoftpath@linetotoken\forest@pathtodict@linetoop
7901 \def\forest@pathtodict@subpathstart{}%
7902 #1%
7903 \forest@restore@pgfsyssoftpath@tokendefs
7904 }

```

When a move-to operation is encountered:

```

7905 \def\forest@pathtodict@movetoop#1#2{%

```

If a subpath had just started, it was a degenerate one (a point). No need to store that (i.e. no code would use this information). So, just remember that a new subpath has started.

```

7906 \def\forest@pathtodict@subpathstart{(#1,#2)-}%
7907 }

```

When a line-to operation is encountered:

```

7908 \def\forest@pathtodict@linetoop#1#2{%

```

If the subpath has just started, its start is also the start of the current segment.

```

7909 \if\relax\forest@pathtodict@subpathstart\relax\else
7910 \let\forest@pathtodict@from\forest@pathtodict@subpathstart
7911 \fi

```

Mark the segment as existing.

```

7912 \expandafter\let\csname\forest@pathtodict@prefix\forest@pathtodict@from-(#1,#2)\endcsname\forest@inpath

```

Set the start of the next segment to the current point, and mark that we are in the middle of a subpath.

```

7913 \def\forest@pathtodict@from{(#1,#2)-}%
7914 \def\forest@pathtodict@subpathstart{}%
7915 }

```

In this macro, the edge is actually computed.

```

7916 \def\forest@gettightedgeofpath@getedge#1{% cs to store the edge into

```

Clear the path and the last projection.

```

7917 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
7918 \let\forest@last@x\relax
7919 \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.)

```

7920 \c@pgf@counta=0
7921 \forest@temp@count=\forest@pi@n\relax
7922 \advance\forest@temp@count -1

```



```

7923 \edef\forest@nminusone{\the\forest@temp@count}%
7924 \safeloop
7925 \ifnum\c@pgf@counta<\forest@nminusone\relax
7926   \forest@gettightedgeofpath@getedge@loopa
7927 \saferepeat

```

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

```

7928 \ifnum\forest@nminusone<\forest@n\relax\else
7929   \ifnum\csname forest@pi@\forest@nminusone @n\endcsname>0
7930     \forest@gettightedgeofpath@maybemoveto{\forest@nminusone}{0}%
7931   \fi
7932 \fi
7933 \pgfsyssoftpath@getcurrentpath#1%
7934 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
7935 }

```

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.

```

7936 \def\forest@gettightedgeofpath@getedge@loopa{%
7937   \ifnum\csname forest@pi@\the\c@pgf@counta @n\endcsname>0

```

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

```

7938   \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.

```

7939   \c@pgf@countb=0
7940   \safeloop
7941   \ifnum\c@pgf@countb<\csname forest@pi@\the\c@pgf@counta @n\endcsname\relax
7942     \forest@gettightedgeofpath@getedge@loopb
7943   \saferepeat
7944 \fi
7945 \advance\c@pgf@counta 1
7946 }

```

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

```

7947 \def\forest@gettightedgeofpath@getedge@loopb{%
7948   \c@pgf@countc=0
7949   \advance\c@pgf@counta 1
7950   \edef\forest@aplusone{\the\c@pgf@counta}%
7951   \advance\c@pgf@counta -1
7952   \safeloop
7953   \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!

```

7954   \forest@tempfalse
7955   \expandafter\ifx\csname forest@pi@(%
7956     \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb x\endcsname,%
7957     \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb y\endcsname)--(%
7958     \csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname,%
7959     \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname)%
7960   \endcsname\forest@inpath

```

```

7961         \forest@temptrue
7962     \else
7963         \expandafter\ifx\csname forest@pi@(%
7964             \csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname,%
7965             \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname)--(%
7966             \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb x\endcsname,%
7967             \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb y\endcsname)%
7968             \endcsname\forest@inpath
7969         \forest@temptrue
7970     \fi
7971 \fi
7972 \ifforest@temp

```

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.

```

7973     \forest@gettightedgeofpath@maybemoveto{\the\c@pgf@counta}{\the\c@pgf@countb}%

```

Second, create a line to the end point.

```

7974     \edef\forest@last@x{%
7975         \csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname}%
7976     \edef\forest@last@y{%
7977         \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname}%
7978     \pgfsyssoftpath@lineto\forest@last@x\forest@last@y

```

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

```

7979         \c@pgf@countc=\csname forest@pi@\forest@aplusone @n\endcsname
7980         \c@pgf@countb=\csname forest@pi@\the\c@pgf@counta @n\endcsname
7981     \fi
7982     \advance\c@pgf@countc 1
7983     \saferepeat
7984     \advance\c@pgf@countb 1
7985 }

```

\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.

```

7986 \def\forest@gettightedgeofpath@maybemoveto#1#2{%
7987     \forest@temptrue
7988     \ifx\forest@last@x\relax\else
7989         \ifdim\forest@last@x=\csname forest@pi@#1@#2x\endcsname\relax
7990             \ifdim\forest@last@y=\csname forest@pi@#1@#2y\endcsname\relax
7991                 \forest@tempfalse
7992             \fi
7993         \fi
7994     \fi
7995     \ifforest@temp
7996         \edef\forest@last@x{\csname forest@pi@#1@#2x\endcsname}%
7997         \edef\forest@last@y{\csname forest@pi@#1@#2y\endcsname}%
7998         \pgfsyssoftpath@moveto\forest@last@x\forest@last@y
7999     \fi
8000 }

```

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

```

8001 \def\forest@simplifypath#1#2{%
8002     \pgfsyssoftpath@setcurrentpath\pgfutil@empty
8003     \forest@save\pgfsyssoftpath@tokendef
8004     \let\pgfsyssoftpath@movetotoken\forest@simplifypath@moveto
8005     \let\pgfsyssoftpath@linetotoken\forest@simplifypath@lineto

```

```

8006 \let\forest@last@x\relax
8007 \let\forest@last@y\relax
8008 \let\forest@last@atan\relax
8009 #1%
8010 \ifx\forest@last@x\relax\else
8011 \ifx\forest@last@atan\relax\else
8012 \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
8013 \fi
8014 \fi
8015 \forest@restore@pgfsyssoftpath@tokendef
8016 \pgfsyssoftpath@getcurrentpath#2%
8017 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
8018 }

```

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.

```

8019 \def\forest@simplifypath@moveto#1#2{%
8020 \ifx\forest@last@x\relax\else
8021 \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
8022 \fi
8023 \pgfsyssoftpath@moveto{#1}{#2}%
8024 \def\forest@last@x{#1}%
8025 \def\forest@last@y{#2}%
8026 \let\forest@last@atan\relax
8027 }

```

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

```

8028 \def\forest@getedgeofpath@precision{1pt}

```

When a line-to is encountered...

```

8029 \def\forest@simplifypath@lineto#1#2{%
8030 \ifx\forest@last@x\relax

```

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

```

8031 \def\forest@last@x{#1}%
8032 \def\forest@last@y{#2}%
8033 \let\forest@last@atan\relax
8034 \else

```

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

```

8035 \pgfpointdiff{\pgfpoint{#1}{#2}}{\pgfpoint{\forest@last@x}{\forest@last@y}}%
8036 \ifdim\pgf@x<\pgfintersectiontolerance
8037 \ifdim-\pgf@x<\pgfintersectiontolerance
8038 \pgf@x=0pt
8039 \fi
8040 \fi
8041 \edef\forest@marshal{%
8042 \noexpand\pgfmthatantwo@
8043 {\expandafter\pgf@geT\the\pgf@x}%
8044 {\expandafter\pgf@geT\the\pgf@y}%
8045 }\forest@marshal
8046 \let\forest@current@atan\pgfmathresult
8047 \ifx\forest@last@atan\relax

```

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

```

8048 \def\forest@last@x{#1}%
8049 \def\forest@last@y{#2}%
8050 \let\forest@last@atan\forest@current@atan
8051 \else

```

Otherwise, compare the first and the current slope.

```

8052 \pgfutil@tempdima=\forest@current@atan pt
8053 \advance\pgfutil@tempdima -\forest@last@atan pt
8054 \ifdim\pgfutil@tempdima<0pt\relax
8055 \multiply\pgfutil@tempdima -1
8056 \fi
8057 \ifdim\pgfutil@tempdima<\forest@getedgeofpath@precision\relax
8058 \else

```

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

```

8059 \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
8060 \let\forest@last@atan\forest@current@atan
8061 \fi

```

In any event, update the last point.

```

8062 \def\forest@last@x{#1}%
8063 \def\forest@last@y{#2}%
8064 \fi
8065 \fi
8066 }

```

9.4 Get rectangle/band edge

```

8067 \def\forest@getnegativerectangleedgeofpath#1#2{%
8068 \forest@getnegativerectangleorbandedgeofpath{#1}{#2}{\the\pgf@xb}}
8069 \def\forest@getpositiverectangleedgeofpath#1#2{%
8070 \forest@getpositiverectangleorbandedgeofpath{#1}{#2}{\the\pgf@xb}}
8071 \def\forest@getbothrectangleedgesofpath#1#2#3{%
8072 \forest@getbothrectangleorbandedgesofpath{#1}{#2}{#3}{\the\pgf@xb}}
8073 \def\forest@bandlength{5000pt} % something large (ca. 180cm), but still manageable for TeX without producing
8074 \def\forest@getnegativebandedgeofpath#1#2{%
8075 \forest@getnegativerectangleorbandedgeofpath{#1}{#2}{\forest@bandlength}}
8076 \def\forest@getpositivebandedgeofpath#1#2{%
8077 \forest@getpositiverectangleorbandedgeofpath{#1}{#2}{\forest@bandlength}}
8078 \def\forest@getbothbandedgesofpath#1#2#3{%
8079 \forest@getbothrectangleorbandedgesofpath{#1}{#2}{#3}{\forest@bandlength}}
8080 \def\forest@getnegativerectangleorbandedgeofpath#1#2#3{%
8081 \forest@path@getboundingrectangle@ls#1{\forest@grow}%
8082 \edef\forest@gre@path{%
8083 \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@ya}%
8084 \noexpand\pgfsyssoftpath@linetotoken{#3}{\the\pgf@ya}%
8085 }%
8086 {%
8087 \pgftransformreset
8088 \forest@pgfqtransformrotate{\forest@grow}%
8089 \forest@pgfpathtransformed\forest@gre@path
8090 }%
8091 \pgfsyssoftpath@getcurrentpath#2%
8092 }
8093 \def\forest@getpositiverectangleorbandedgeofpath#1#2#3{%
8094 \forest@path@getboundingrectangle@ls#1{\forest@grow}%
8095 \edef\forest@gre@path{%
8096 \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@yb}%
8097 \noexpand\pgfsyssoftpath@linetotoken{#3}{\the\pgf@yb}%
8098 }%
8099 {%
8100 \pgftransformreset
8101 \forest@pgfqtransformrotate{\forest@grow}%
8102 \forest@pgfpathtransformed\forest@gre@path
8103 }%
8104 \pgfsyssoftpath@getcurrentpath#2%
8105 }

```

```

8106 \def\forest@getbothrectangleorbandedgesofpath#1#2#3#4{%
8107   \forest@path@getboundingrectangle@ls#1{\forest@grow}%
8108   \edef\forest@gre@negpath{%
8109     \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@ya}%
8110     \noexpand\pgfsyssoftpath@linetotoken{#4}{\the\pgf@ya}%
8111   }%
8112   \edef\forest@gre@pospath{%
8113     \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@yb}%
8114     \noexpand\pgfsyssoftpath@linetotoken{#4}{\the\pgf@yb}%
8115   }%
8116   {%
8117     \pgftransformreset
8118     \forest@pgfqtransformrotate{\forest@grow}%
8119     \forest@pgfpathtransformed\forest@gre@negpath
8120   }%
8121   \pgfsyssoftpath@getcurrentpath#2%
8122   {%
8123     \pgftransformreset
8124     \forest@pgfqtransformrotate{\forest@grow}%
8125     \forest@pgfpathtransformed\forest@gre@pospath
8126   }%
8127   \pgfsyssoftpath@getcurrentpath#3%
8128 }

```

9.5 Distance between paths

Another crucial part of the package.

```

8129 \def\forest@distance@between@edge@paths#1#2#3{%
8130   % #1, #2 = (edge) paths
8131   %
8132   % project paths
8133   \forest@projectpathtogrowline#1{\forest@p1@}%
8134   \forest@projectpathtogrowline#2{\forest@p2@}%
8135   % merge projections (the lists are sorted already, because edge
8136   % paths are |sorted|)
8137   \forest@dbep@mergeprojections
8138   {\forest@p1@}{\forest@p2@}%
8139   {\forest@P1@}{\forest@P2@}%
8140   % process projections
8141   \forest@processprojectioninfo{\forest@P1@}{\forest@PI1@}%
8142   \forest@processprojectioninfo{\forest@P2@}{\forest@PI2@}%
8143   % break paths
8144   \forest@breakpath#1{\forest@PI1@}\forest@broken@one
8145   \forest@breakpath#2{\forest@PI2@}\forest@broken@two
8146   % sort inner arrays ---optimize: it's enough to find max and min
8147   \forest@sort@inner@arrays{\forest@PI1@}\forest@sort@descending
8148   \forest@sort@inner@arrays{\forest@PI2@}\forest@sort@ascending
8149   % compute the distance
8150   \let\forest@distance\relax
8151   \c@pgf@countc=0
8152   \forest@loop
8153   \ifnum\c@pgf@countc<\csname forest@PI1@n\endcsname\relax
8154     \ifnum\csname forest@PI1@the\c@pgf@countc @n\endcsname=0 \else
8155       \ifnum\csname forest@PI2@the\c@pgf@countc @n\endcsname=0 \else
8156         \pgfutil@tempdima=\csname forest@PI2@the\c@pgf@countc @0d\endcsname\relax
8157         \advance\pgfutil@tempdima -\csname forest@PI1@the\c@pgf@countc @0d\endcsname\relax
8158         \ifx\forest@distance\relax
8159           \edef\forest@distance{\the\pgfutil@tempdima}%
8160         \else
8161           \ifdim\pgfutil@tempdima<\forest@distance\relax
8162             \edef\forest@distance{\the\pgfutil@tempdima}%

```

```

8163         \fi
8164     \fi
8165     \fi
8166     \fi
8167     \advance\c@pgf@countc 1
8168     \forest@repeat
8169     \let#3\forest@distance
8170 }
8171 % merge projections: we need two projection arrays, both containing
8172 % projection points from both paths, but each with the original
8173 % points from only one path
8174 \def\forest@dbep@mergeprojections#1#2#3#4{%
8175 % TODO: optimize: v bistvu ni treba sortirat, ker je edge path e sortiran
8176 \forest@sortprojections{#1}%
8177 \forest@sortprojections{#2}%
8178 \c@pgf@counta=0
8179 \c@pgf@countb=0
8180 \c@pgf@countc=0
8181 \edef\forest@input@prefix@one{#1}%
8182 \edef\forest@input@prefix@two{#2}%
8183 \edef\forest@output@prefix@one{#3}%
8184 \edef\forest@output@prefix@two{#4}%
8185 \forest@dbep@mp@iterate
8186 \csedef{#3n}{\the\c@pgf@countc}%
8187 \csedef{#4n}{\the\c@pgf@countc}%
8188 }
8189 \def\forest@dbep@mp@iterate{%
8190 \let\forest@dbep@mp@next\forest@dbep@mp@iterate
8191 \ifnum\c@pgf@counta<\csname\forest@input@prefix@one n\endcsname\relax
8192 \ifnum\c@pgf@countb<\csname\forest@input@prefix@two n\endcsname\relax
8193 \let\forest@dbep@mp@next\forest@dbep@mp@do
8194 \else
8195 \let\forest@dbep@mp@next\forest@dbep@mp@iteratefirst
8196 \fi
8197 \else
8198 \ifnum\c@pgf@countb<\csname\forest@input@prefix@two n\endcsname\relax
8199 \let\forest@dbep@mp@next\forest@dbep@mp@iteratesecond
8200 \else
8201 \let\forest@dbep@mp@next\relax
8202 \fi
8203 \fi
8204 \forest@dbep@mp@next
8205 }
8206 \def\forest@dbep@mp@do{%
8207 \forest@sort@cmptwodimcs%
8208 {\forest@input@prefix@one\the\c@pgf@counta xp}%
8209 {\forest@input@prefix@one\the\c@pgf@counta yp}%
8210 {\forest@input@prefix@two\the\c@pgf@countb xp}%
8211 {\forest@input@prefix@two\the\c@pgf@countb yp}%
8212 \if\forest@sort@cmp@result=%
8213 \forest@dbep@mp@@store@p\forest@input@prefix@one\c@pgf@counta
8214 \forest@dbep@mp@@store@o\forest@input@prefix@one
8215 \c@pgf@counta\forest@output@prefix@one
8216 \forest@dbep@mp@@store@o\forest@input@prefix@two
8217 \c@pgf@countb\forest@output@prefix@two
8218 \advance\c@pgf@counta 1
8219 \advance\c@pgf@countb 1
8220 \else
8221 \if\forest@sort@cmp@result>%
8222 \forest@dbep@mp@@store@p\forest@input@prefix@two\c@pgf@countb
8223 \forest@dbep@mp@@store@o\forest@input@prefix@two

```

```

8224      \c@pgf@countb\forest@output@prefix@two
8225      \advance\c@pgf@countb 1
8226      \else%<
8227      \forest@dbep@mp@@store@p\forest@input@prefix@one\c@pgf@counta
8228      \forest@dbep@mp@@store@o\forest@input@prefix@one
8229      \c@pgf@counta\forest@output@prefix@one
8230      \advance\c@pgf@counta 1
8231      \fi
8232      \fi
8233      \advance\c@pgf@countc 1
8234      \forest@dbep@mp@iterate
8235  }
8236  \def\forest@dbep@mp@@store@p#1#2{%
8237      \csletcs
8238      {\forest@output@prefix@one\the\c@pgf@countc xp}%
8239      {#1\the#2xp}%
8240      \csletcs
8241      {\forest@output@prefix@one\the\c@pgf@countc yp}%
8242      {#1\the#2yp}%
8243      \csletcs
8244      {\forest@output@prefix@two\the\c@pgf@countc xp}%
8245      {#1\the#2xp}%
8246      \csletcs
8247      {\forest@output@prefix@two\the\c@pgf@countc yp}%
8248      {#1\the#2yp}%
8249  }
8250  \def\forest@dbep@mp@@store@o#1#2#3{%
8251      \csletcs{#3\the\c@pgf@countc xo}{#1\the#2xo}%
8252      \csletcs{#3\the\c@pgf@countc yo}{#1\the#2yo}%
8253  }
8254  \def\forest@dbep@mp@iteratefirst{%
8255      \forest@dbep@mp@iterateone\forest@input@prefix@one\c@pgf@counta\forest@output@prefix@one
8256  }
8257  \def\forest@dbep@mp@iteratesecond{%
8258      \forest@dbep@mp@iterateone\forest@input@prefix@two\c@pgf@countb\forest@output@prefix@two
8259  }
8260  \def\forest@dbep@mp@iterateone#1#2#3{%
8261      \forest@loop
8262      \ifnum#2<\csname#1n\endcsname\relax
8263      \forest@dbep@mp@@store@p#1#2%
8264      \forest@dbep@mp@@store@o#1#2#3%
8265      \advance\c@pgf@countc 1
8266      \advance#21
8267      \forest@repeat
8268  }

```

9.6 Utilities

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

```

8269 \newif\ifforest@equaltotolerance
8270 \def\forest@equaltotolerance#1#2{%
8271     \pgfpointdiff{#1}{#2}%
8272     \ifdim\pgf@x<0pt \multiply\pgf@x -1 \fi
8273     \ifdim\pgf@y<0pt \multiply\pgf@y -1 \fi
8274     \global\forest@equaltotolerancefalse
8275     \ifdim\pgf@x<\pgfintersectiontolerance\relax
8276     \ifdim\pgf@y<\pgfintersectiontolerance\relax
8277         \global\forest@equaltolerancetrue
8278     \fi

```

```

8279 \fi
8280 }}

      Save/restore pgfs \pgfsyssoftpath@...token definitions.
8281 \def\forest@save@pgfsyssoftpath@tokendefs{%
8282   \let\forest@origmovetotoken\pgfsyssoftpath@movetotoken
8283   \let\forest@origlinetotoken\pgfsyssoftpath@linetotoken
8284   \let\forest@origcurvetosupportatoken\pgfsyssoftpath@curvetosupportatoken
8285   \let\forest@origcurvetosupportbtoken\pgfsyssoftpath@curvetosupportbtoken
8286   \let\forest@origcurvetotoken\pgfsyssoftpath@curvetotoken
8287   \let\forest@origrectcornertoken\pgfsyssoftpath@rectcornertoken
8288   \let\forest@origrectsizetoken\pgfsyssoftpath@rectsizetoken
8289   \let\forest@origclosepathtoken\pgfsyssoftpath@closepathtoken
8290   \let\pgfsyssoftpath@movetotoken\forest@badtoken
8291   \let\pgfsyssoftpath@linetotoken\forest@badtoken
8292   \let\pgfsyssoftpath@curvetosupportatoken\forest@badtoken
8293   \let\pgfsyssoftpath@curvetosupportbtoken\forest@badtoken
8294   \let\pgfsyssoftpath@curvetotoken\forest@badtoken
8295   \let\pgfsyssoftpath@rectcornertoken\forest@badtoken
8296   \let\pgfsyssoftpath@rectsizetoken\forest@badtoken
8297   \let\pgfsyssoftpath@closepathtoken\forest@badtoken
8298 }
8299 \def\forest@badtoken{%
8300   \PackageError{forest}{This token should not be in this path}{}%
8301 }
8302 \def\forest@restore@pgfsyssoftpath@tokendefs{%
8303   \let\pgfsyssoftpath@movetotoken\forest@origmovetotoken
8304   \let\pgfsyssoftpath@linetotoken\forest@origlinetotoken
8305   \let\pgfsyssoftpath@curvetosupportatoken\forest@origcurvetosupportatoken
8306   \let\pgfsyssoftpath@curvetosupportbtoken\forest@origcurvetosupportbtoken
8307   \let\pgfsyssoftpath@curvetotoken\forest@origcurvetotoken
8308   \let\pgfsyssoftpath@rectcornertoken\forest@origrectcornertoken
8309   \let\pgfsyssoftpath@rectsizetoken\forest@origrectsizetoken
8310   \let\pgfsyssoftpath@closepathtoken\forest@origclosepathtoken
8311 }

```

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

```

8312 \def\forest@extendpath#1#2#3{%
8313   \pgf@process{#3}%
8314   \pgfsyssoftpath@setcurrentpath#1%
8315   \forest@save@pgfsyssoftpath@tokendefs
8316   \let\pgfsyssoftpath@movetotoken\forest@extendpath@moveto
8317   \let\pgfsyssoftpath@linetotoken\forest@extendpath@lineto
8318   #2%
8319   \forest@restore@pgfsyssoftpath@tokendefs
8320   \pgfsyssoftpath@getcurrentpath#1%
8321 }
8322 \def\forest@extendpath@moveto#1#2{%
8323   \forest@extendpath@do{#1}{#2}\pgfsyssoftpath@moveto
8324 }
8325 \def\forest@extendpath@lineto#1#2{%
8326   \forest@extendpath@do{#1}{#2}\pgfsyssoftpath@lineto
8327 }
8328 \def\forest@extendpath@do#1#2#3{%
8329   {%
8330     \advance\pgf@x #1
8331     \advance\pgf@y #2
8332     #3{\the\pgf@x}{\the\pgf@y}%
8333   }%
8334 }

```

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.)

```

8335 \def\forest@path@getboundingrectangle@ls#1#2{%
8336   {%
8337     \pgftransformreset
8338     \forest@pgfqtransformrotate{-#2}%
8339     \forest@pgfpathtransformed#1%
8340   }%
8341   \pgfsyssoftpath@getcurrentpath\forest@gbr@rotatedpath
8342   \forest@path@getboundingrectangle@xy\forest@gbr@rotatedpath
8343 }
8344 \def\forest@path@getboundingrectangle@xy#1{%
8345   \forest@save@pgfsyssoftpath@tokendefs
8346   \let\pgfsyssoftpath@movetotoken\forest@gbr@firstpoint
8347   \let\pgfsyssoftpath@linetotoken\forest@gbr@firstpoint
8348   #1%
8349   \forest@restore@pgfsyssoftpath@tokendefs
8350 }
8351 \def\forest@gbr@firstpoint#1#2{%
8352   \pgf@xa=#1 \pgf@xb=#1 \pgf@ya=#2 \pgf@yb=#2
8353   \let\pgfsyssoftpath@movetotoken\forest@gbr@point
8354   \let\pgfsyssoftpath@linetotoken\forest@gbr@point
8355 }
8356 \def\forest@gbr@point#1#2{%
8357   \ifdim#1<\pgf@xa\relax\pgf@xa=#1 \fi
8358   \ifdim#1>\pgf@xb\relax\pgf@xb=#1 \fi
8359   \ifdim#2<\pgf@ya\relax\pgf@ya=#2 \fi
8360   \ifdim#2>\pgf@yb\relax\pgf@yb=#2 \fi
8361 }

```

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.

```

8362 \let\forest@pgfqtransformrotate\pgftransformrotate
8363 \let\forest@pgftransformcm\pgftransformcm
8364 \let\forest@pgf@transformcm\pgf@transformcm
8365 \patchcmd{\forest@pgfqtransformrotate}{\pgfmathparse{#1}}{\edef\pgfmathresult{\number\numexpr#1}}{}{}
8366 \patchcmd{\forest@pgfqtransformrotate}{\pgftransformcm}{\forest@pgftransformcm}{}{}
8367 \patchcmd{\forest@pgftransformcm}{\pgf@transformcm}{\forest@pgf@transformcm}{}{}
8368 \patchcmd{\forest@pgf@transformcm}{\pgfmathsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x
8369 \patchcmd{\forest@pgf@transformcm}{\pgfmathsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x
8370 \patchcmd{\forest@pgf@transformcm}{\pgfmathsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x
8371 \patchcmd{\forest@pgf@transformcm}{\pgfmathsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x
8372 \def\forest@pgf@transformcm@setlength#1#2{#1=#2pt}

```

10 The outer UI

10.1 Externalization

```

8373 \pgfkeys{/forest/external/.cd,
8374   %copy command/.initial={cp "\source" "\target"},
8375   copy command/.initial={},
8376   optimize/.is if=forest@external@optimize@,
8377   context/.initial={%
8378     \forestOve{csname forest@id@of@standard node\endcsname}{environment@formula}},
8379   depends on macro/.style={context/.append/.expanded={%
8380     \expandafter\detokenize\expandafter{#1}}},
8381 }
8382 \def\forest@file@copy#1#2{%
8383   \IfFileExists{#1}{%
8384     \pgfkeysgetvalue{/forest/external/copy command}\forest@copy@command
8385     \ifdefempty\forest@copy@command{%

```

```

8386     \forest@file@copy@{#1}{#2}%
8387   }{ % copy by external command
8388     \def\source{#1}%
8389     \def\target{#2}%
8390     \immediate\write18{\forest@copy@command}%
8391   }%
8392 }{}%
8393 }
8394 \newread\forest@copy@in
8395 \newwrite\forest@copy@out
8396 \def\forest@file@copy@#1#2{%
8397   \begingroup
8398   \openin\forest@copy@in=#1
8399   \immediate\openout\forest@copy@out=#2
8400   \endlinechar-1
8401   \loop
8402   \unless\ifeof\forest@copy@in
8403     \readline\forest@copy@in to\forest@temp
8404     \immediate\write\forest@copy@out{\forest@temp}%
8405   \repeat
8406   \immediate\closeout\forest@copy@out
8407   \closein\forest@copy@in
8408   \endgroup
8409 }
8410 \newif\ifforest@external@optimize@
8411 \forest@external@optimize@true
8412 \ifforest@install@keys@to@tikz@path@
8413 \tikzset{
8414   fit to/.style={
8415     /forest/for nodewalk=%
8416     {TeX={\def\forest@fitto{}}\,#1}%
8417     {TeX={\eappto\forest@fitto{(\forestove{name})}}},
8418     fit/.expanded={\forest@fitto}
8419   },
8420 }
8421 \fi
8422 \ifforest@external@
8423   \ifdefined\tikzexternal@tikz@replacement\else
8424     \usetikzlibrary{external}%
8425   \fi
8426   \pgfkeys{%
8427     /tikz/external/failed ref warnings for={},
8428     /pgf/images/aux in dpth=false,
8429   }%
8430   \tikzifexternalizing{}{%
8431     \forest@file@copy{\jobname.aux}{\jobname.aux.copy}%
8432   }%
8433   \AtBeginDocument{%
8434     \tikzifexternalizing{%
8435       \IfFileExists{\tikzexternalrealjob.aux.copy}{%
8436         \makeatletter
8437         \input\tikzexternalrealjob.aux.copy\relax
8438         \makeatother
8439       }{}%
8440     }{%
8441       \newwrite\forest@auxout
8442       \immediate\openout\forest@auxout=\tikzexternalrealjob.for.tmp
8443     }%
8444     \IfFileExists{\tikzexternalrealjob.for}{%
8445       {%
8446         \makehashother\makeatletter

```

```

8447     \input\tikzexternalrealjob.for\relax
8448   }%
8449 }{}%
8450 }%
8451 \AtEndDocument{%
8452   \tikzifexternalizing{}{%
8453     \immediate\closeout\forest@auxout
8454     \forest@file@copy{\jobname.for.tmp}{\jobname.for}%
8455   }%
8456 }%
8457 \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.

```

8458 \NewDocumentEnvironment{forest}{D(){} }{%
8459   \forest@config{#1}%
8460   \Collect@Body
8461   \forest@env
8462 }{}
8463 \NewDocumentCommand{\Forest}{s D(){} m}{%
8464   \forest@config{#2}%
8465   \IfBooleanTF{#1}{\let\forest@next\forest@env}{\let\forest@next\forest@group@env}%
8466   \forest@next{#3}%
8467 }
8468 \def\forest@config#1{%
8469   \forest@defstages{stages}%
8470   \forestset{@config/.cd,#1}%
8471 }
8472 \def\forest@defstages#1{%
8473   \def\forest@stages{#1}%
8474 }
8475 \forestset{@config/.cd,
8476   %stages/.store in=\forest@stages,
8477   stages/.code={\forest@defstages{#1}},
8478   .unknown/.code={\PackageError{forest}{Unknown config option for forest environment/command.}{In Forest v2.0}}
8479 }
8480 \def\forest@group@env#1{{\forest@env{#1}}}
8481 \newif\ifforest@externalize@tree@
8482 \newif\ifforest@was@tikzexternalwasenable
8483 \newcommand\forest@env[1]{%
8484   \let\forest@external@next\forest@begin
8485   \forest@was@tikzexternalwasenablefalse
8486   \ifdefined\tikzexternal@tikz@replacement
8487     \ifx\tikz\tikzexternal@tikz@replacement
8488       \forest@was@tikzexternalwasenabletrue
8489       \tikzexternaldisable
8490     \fi
8491   \fi
8492   \forest@externalize@tree@false
8493   \ifforest@external@
8494     \ifforest@was@tikzexternalwasenable
8495     \forest@env@
8496   \fi
8497   \fi
8498   \forest@standardnode@calibrate
8499   \forest@external@next{#1}%
8500 }
8501 \def\forest@env@{%

```

```

8502 \iftikzexternalexportnext
8503   \tikzifexternalizing{%
8504     \let\forest@external@next\forest@begin@externalizing
8505   }{%
8506     \let\forest@external@next\forest@begin@externalize
8507   }%
8508 \else
8509   \tikzexternalexportnexttrue
8510 \fi
8511 }

```

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

```

8512 \long\def\forest@begin@externalizing#1{%
8513   \forest@external@setup{#1}%
8514   \let\forest@external@next\forest@begin
8515   \forest@externalize@inner@n=-1
8516   \ifforest@external@optimize@\forest@externalizing@maybeoptimize\fi
8517   \forest@external@next{#1}%
8518   \tikzexternalenable
8519 }
8520 \def\forest@externalizing@maybeoptimize{%
8521   \edef\forest@temp{\tikzexternalrealjob-forest-\forest@externalize@outer@n}%
8522   \edef\forest@marshal{%
8523     \noexpand\pgfutil@in@
8524     {\expandafter\detokenize\expandafter{\forest@temp}.}
8525     {\expandafter\detokenize\expandafter{\pgfactualjobname}.}%
8526   }\forest@marshal
8527   \ifpgfutil@in@
8528   \else
8529     \let\forest@external@next\@gobble
8530   \fi
8531 }

```

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

```

8532 \long\def\forest@begin@externalize#1{%
8533   \forest@external@setup{#1}%
8534   \iftikzexternal@file@isuptodate
8535     \setbox0=\hbox{%
8536       \csname forest@externalcheck@\forest@externalize@outer@n\endcsname
8537     }%
8538   \fi
8539   \iftikzexternal@file@isuptodate
8540     \csname forest@externalload@\forest@externalize@outer@n\endcsname
8541   \else
8542     \forest@externalize@tree@true
8543     \forest@externalize@inner@n=-1
8544     \forest@begin{#1}%
8545     \ifcsdef{forest@externalize@@\forest@externalize@id}{-}{%
8546       \immediate\write\forest@auxout{%
8547         \noexpand\forest@external
8548         {\forest@externalize@outer@n}%
8549         {\expandafter\detokenize\expandafter{\forest@externalize@id}}%
8550         {\expandonce\forest@externalize@checkimages}%
8551         {\expandonce\forest@externalize@loadimages}%
8552       }%
8553     }%
8554   \fi
8555   \tikzexternalenable
8556 }
8557 \def\forest@includeexternal@check#1{%
8558   \tikzsetnextfilename{#1}%

```

```

8559 \IfFileExists{\tikzexternal@filenameprefix/#1}{\tikzexternal@file@isuptodatetrue}{\tikzexternal@file@isupto
8560 }
8561 \def\makehashother{\catcode'\# =12}%
8562 \long\def\forest@external@setup#1{%
8563   % set up \forest@externalize@id and \forest@externalize@outer@n
8564   % we need to deal with #s correctly (\write doubles them)
8565   \setbox0=\hbox{\makehashother\makeatletter
8566     \scantokens{\forest@temp@toks{#1}}\expandafter
8567   }%
8568   \expandafter\forest@temp@toks\expandafter{\the\forest@temp@toks}%
8569   \edef\forest@temp{\pgfkeysvalueof{/forest/external/context}}%
8570   \edef\forest@externalize@id{%
8571     \expandafter\detokenize\expandafter{\forest@temp}%
8572     @@%
8573     \expandafter\detokenize\expandafter{\the\forest@temp@toks}%
8574   }%
8575   \letcs\forest@externalize@outer@n\forest@externalize@@\forest@externalize@id}%
8576   \ifdefined\forest@externalize@outer@n
8577     \global\tikzexternal@file@isuptodatetrue
8578   \else
8579     \global\advance\forest@externalize@max@outer@n 1
8580     \edef\forest@externalize@outer@n{\the\forest@externalize@max@outer@n}%
8581     \global\tikzexternal@file@isuptodatefalse
8582   \fi
8583   \def\forest@externalize@loadimages{}%
8584   \def\forest@externalize@checkimages{}%
8585 }
8586 \newcount\forest@externalize@max@outer@n
8587 \global\forest@externalize@max@outer@n=0
8588 \newcount\forest@externalize@inner@n

```

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

```

8589 \long\def\forest@external#1#2#3#4{% #1=n,#2=context+source code,#3=update check code, #4=load code
8590   \ifnum\forest@externalize@max@outer@n<#1
8591     \global\forest@externalize@max@outer@n=#1
8592   \fi
8593   \global\csdef{forest@externalize@@\detokenize{#2}}{#1}%
8594   \global\csdef{forest@externalcheck@#1}{#3}%
8595   \global\csdef{forest@externalload@#1}{#4}%
8596   \tikzifexternalizing{}{%
8597     \immediate\write\forest@auxout{%
8598       \noexpand\forest@external{#1}%
8599       {\expandafter\detokenize\expandafter{#2}}%
8600       {\unexpanded{#3}}%
8601       {\unexpanded{#4}}%
8602     }%
8603   }%
8604 }

```

These two macros include the external picture.

```

8605 \def\forest@includeexternal#1{%
8606   \edef\forest@temp{\pgfkeysvalueof{/forest/external/context}}%
8607   %\typeout{forest: Including external picture '#1' for forest context+code: '\expandafter\detokenize\expanda
8608   {%
8609     %\def\pgf@declaredraftimage##1##2{\def\pgf@image{\hbox{}}}%
8610     \tikzsetnextfilename{#1}%
8611     \tikzexternalenable
8612     \tikz{}%
8613   }%
8614 }
8615 \def\forest@includeexternal@box#1#2{%
8616   \global\setbox#1=\hbox{\forest@includeexternal{#2}}%

```

8617 }

This code runs the bracket parser and stage processing.

```

8618 \long\def\forest@begin#1{%
8619   \iffalse{\fi\forest@parsebracket#1}%
8620 }
8621 \def\forest@parsebracket{%
8622   \bracketParse{\forest@get@root@afterthought}\forest@root=%
8623 }
8624 \def\forest@get@root@afterthought{%
8625   \expandafter\forest@get@root@afterthought\expandafter{\iffalse}\fi
8626 }
8627 \long\def\forest@get@root@afterthought@#1{%
8628   \ifblank{#1}{\}%
8629   \forest@eappto{\forest@root}{given options}{,afterthought={\unexpanded{#1}}}%
8630 }%
8631 \forest@do
8632 }
8633 \def\forest@do{%
8634   \forest@node@Compute@numeric@ts@info{\forest@root}%
8635   \expandafter\forestset\expandafter{\forest@stages}%
8636   \ifforest@was@tikzexternalwasenable
8637     \tikzexternalenable
8638   \fi
8639 }

```

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!

```

8640 \def\forest@standardnode@new{%
8641   \advance\forest@node@maxid1
8642   \forest@fornode{\the\forest@node@maxid}{%
8643     \forest@node@init
8644     \forest@set{id}{\forest@cn}%
8645     \forest@node@setname@silent{standard node}%
8646   }%
8647 }
8648 \def\forest@standardnode@calibrate{%
8649   \forest@fornode{\forest@node@Nametoid{standard node}}{%
8650     \edef\forest@environment{\forest@ve{environment@formula}}%
8651     \forest@toget{previous@environment}\forest@previous@environment
8652     \ifx\forest@environment\forest@previous@environment\else
8653       \forest@tolet{previous@environment}\forest@environment
8654       \forest@node@typeset
8655       \forest@toget{calibration@procedure}\forest@temp
8656       \expandafter\forestset\expandafter{\forest@temp}%
8657     \fi
8658   }%
8659 }

```

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 $\text{T}_{\text{E}}\text{X}$ group.)

```

8660 \def\forestStandardNode[#1]#2#3#4{%
8661   \let\forest@standardnode@restoretikzexternal\relax

```

```

8662 \ifdefined\tikzexternaldisable
8663 \ifx\tikz\tikzexternal\tikz@replacement
8664 \tikzexternaldisable
8665 \let\forest@standardnode@restoretikzexternal\tikzexternalenable
8666 \fi
8667 \fi
8668 \forest@standardnode@new
8669 \forest@for node{\forest@node@Nametoid{standard node}}{%
8670 \forestset{content=#1}%
8671 \forestset{environment@formula}{#2}%
8672 \edef\forest@temp{\unexpanded{#3}}%
8673 \forest@let{calibration@procedure}\forest@temp
8674 \def\forest@calibration@initializing@code{%
8675 \pgfqkeys{/forest/initializing@code}{#4}%
8676 \forest@let{initializing@code}\forest@calibration@initializing@code
8677 \forest@standardnode@restoretikzexternal
8678 }
8679 }
8680 \forestset{initializing@code/.unknown/.code={%
8681 \eappto\forest@calibration@initializing@code{%
8682 \noexpand\forest@get{\forest@node@Nametoid{standard node}}{\pgfkeyscurrentname}\noexpand\forest@temp
8683 \noexpand\forest@let{\pgfkeyscurrentname}\noexpand\forest@temp
8684 }%
8685 }
8686 }

```

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

```

8687 \def\forest@initializefromstandardnode{%
8688 \forest@ve{\forest@node@Nametoid{standard node}}{initializing@code}%
8689 }

```

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) 1 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 1 on the standard node should be the same as the effect of 1 sep, thus, we derive 1 from 1 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.)

```

8690 \forestStandardNode[dj]
8691 {%
8692 \forest@ve{\forest@node@Nametoid{standard node}}{content},%
8693 \the\ht\strutbox,\the\pgflinewidth,%
8694 \pgfkeysvalueof{/pgf/inner ysep},\pgfkeysvalueof{/pgf/outer ysep},%
8695 \pgfkeysvalueof{/pgf/inner xsep},\pgfkeysvalueof{/pgf/outer xsep}%
8696 }
8697 {
8698 1 sep'/.expanded={\the\dimexpr\the\ht\strutbox+\pgfkeysvalueof{/pgf/inner ysep}},
8699 l={1_sep()+abs(max_y()-min_y())+2*\pgfkeysvalueof{/pgf/outer ysep}},
8700 s sep'/.expanded={\the\dimexpr \pgfkeysvalueof{/pgf/inner xsep}*2}
8701 }
8702 {1 sep,l,s sep}

```

10.4 ls coordinate system

```

8703 \pgfqkeys{/forest/@cs}{%
8704 name/.code={%
8705 \edef\forest@cn{\forest@node@Nametoid{#1}}%
8706 \forest@forestcs@resetxy},
8707 id/.code={%

```

```

8708 \edef\forest@cn{#1}%
8709 \forest@forestcs@resetxy},
8710 go/.code={%
8711 \forest@go{#1}%
8712 \forest@forestcs@resetxy},
8713 anchor/.code={\forest@forestcs@anchor{#1}},
8714 l/.code={%
8715 \forestmathsetlengthmacro\forest@forestcs@l{#1}%
8716 \forest@forestcs@ls
8717 },
8718 s/.code={%
8719 \forestmathsetlengthmacro\forest@forestcs@s{#1}%
8720 \forest@forestcs@ls
8721 },
8722 .unknown/.code={%
8723 \expandafter\pgfutil@in@\expandafter.\expandafter{\pgfkeyscurrentname}%
8724 \ifpgfutil@in@
8725 \expandafter\forest@forestcs@namegoanchor\pgfkeyscurrentname\forest@end
8726 \else
8727 \expandafter\forest@nameandgo\expandafter{\pgfkeyscurrentname}%
8728 \forest@forestcs@resetxy
8729 \fi
8730 }
8731 }
8732 \def\forest@forestcs@resetxy{%
8733 \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
8734 \global\pgf@x\forestove{x}\relax
8735 \global\pgf@y\forestove{y}\relax
8736 }
8737 \def\forest@forestcs@ls{%
8738 \ifdefined\forest@forestcs@l
8739 \ifdefined\forest@forestcs@s
8740 {%
8741 \pgftransformreset
8742 \forest@pgfqtransformrotate{\forestove{grow}}}%
8743 \pgfpointransformed{\pgfqpoint{\forest@forestcs@l}{\forest@forestcs@s}}}%
8744 }%
8745 \global\advance\pgf@x\forestove{x}%
8746 \global\advance\pgf@y\forestove{y}%
8747 \fi
8748 \fi
8749 }
8750 \def\forest@forestcs@anchor#1{%
8751 \edef\forest@marshal{%
8752 \noexpand\forest@original@tikz@parse@node\relax
8753 (\forestove{name})\ifx\relax#1\relax\else.\fi#1}%
8754 }\forest@marshal
8755 }
8756 \def\forest@forestcs@namegoanchor#1.#2\forest@end{%
8757 \forest@nameandgo{#1}%
8758 \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
8759 \forest@forestcs@anchor{#2}%
8760 }
8761 \def\forest@cs@invalidnodeerror{%
8762 \PackageError{forest}{Attempt to refer to the invalid node by "forest cs"}{}}%
8763 }
8764 \tikzdeclarecoordinatesystem{forest}{%
8765 \forest@forthis{%
8766 \forest@forestcs@resetxy
8767 \ifdefined\forest@forestcs@l\undef\forest@forestcs@l\fi
8768 \ifdefined\forest@forestcs@s\undef\forest@forestcs@s\fi

```



```

8769 \pgfqkeys{/forest/@cs}{#1}%
8770 }%
8771 }

```

10.5 Relative node names in TikZ

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

```

8772 \def\forest@tikz@parse@node#1(#2){%
8773 \pgfutil@in@.#2}%
8774 \ifpgfutil@in@
8775 \expandafter\forest@tikz@parse@node@checkiftikzname@withdot
8776 \else%
8777 \expandafter\forest@tikz@parse@node@checkiftikzname@withoutdot
8778 \fi%
8779 #1(#2)\forest@end
8780 }
8781 \def\forest@tikz@parse@node@checkiftikzname@withdot#1(#2.#3)\forest@end{%
8782 \forest@tikz@parse@node@checkiftikzname#1{#2}{.#3}}
8783 \def\forest@tikz@parse@node@checkiftikzname@withoutdot#1(#2)\forest@end{%
8784 \forest@tikz@parse@node@checkiftikzname#1{#2}{}}
8785 \def\forest@tikz@parse@node@checkiftikzname#1#2#3{%
8786 \expandafter\ifx\csname pgf@sh@ns@#2\endcsname\relax
8787 \forest@forthis{%
8788 \forest@nameandgo{#2}%
8789 \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
8790 \edef\forest@temp@relativenodename{\forestove{name}}}%
8791 }%
8792 \else
8793 \def\forest@temp@relativenodename{#2}%
8794 \fi
8795 \expandafter\forest@original@tikz@parse@node\expandafter#1\expandafter(\forest@temp@relativenodename#3)%
8796 }
8797 \def\forest@nameandgo#1{%
8798 \pgfutil@in@!{#1}%
8799 \ifpgfutil@in@
8800 \forest@nameandgo@(#1)%
8801 \else
8802 \ifstrempy{#1}{-}{\edef\forest@cn{\forest@node@Nametoid{#1}}}%
8803 \fi
8804 }
8805 \def\forest@nameandgo@(#1!#2){%
8806 \ifstrempy{#1}{-}{\edef\forest@cn{\forest@node@Nametoid{#1}}}%
8807 \forest@go{#2}%
8808 }

```

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.

```

8809 \forestset{
8810 declare count={rotate}{0},
8811 autoforward'={rotate}{node options},
8812 }

```

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

```

8813 \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`.

```
8814 \newif\ifforest@anchor@forwardtotikz
```

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

```
8815 \newif\ifforest@anchor@isborder
```

The UI macro.

```
8816 \def\forestanchortotikzanchor#1#2{% #1 = forest anchor, #2 = macro to receive the tikz anchor
8817   \forest@anchor@forwardtotikzfalse
8818   \forest@anchor@do{#1}{\forest@cn}%
8819   \let#2\forest@temp@anchor
8820 }
```

Generic anchors.

```
8821 \pgfdeclaregenericanchor{child anchor}{%
8822   \forest@anchor@forwardtotikztrue
8823   \forest@anchor@do{#1}{child anchor}{\forest@referencednodeid}%
8824 }
8825 \pgfdeclaregenericanchor{parent anchor}{%
8826   \forest@anchor@forwardtotikztrue
8827   \forest@anchor@do{#1}{parent anchor}{\forest@referencednodeid}%
8828 }
8829 \pgfdeclaregenericanchor{anchor}{%
8830   \forest@anchor@forwardtotikztrue
8831   \forest@anchor@do{#1}{anchor}{\forest@referencednodeid}%
8832 }
8833 \pgfdeclaregenericanchor{children}{%
8834   \forest@anchor@forwardtotikztrue
8835   \forest@anchor@do{#1}{children}{\forest@referencednodeid}%
8836 }
8837 \pgfdeclaregenericanchor{-children}{%
8838   \forest@anchor@forwardtotikztrue
8839   \forest@anchor@do{#1}{-children}{\forest@referencednodeid}%
8840 }
8841 \pgfdeclaregenericanchor{children first}{%
8842   \forest@anchor@forwardtotikztrue
8843   \forest@anchor@do{#1}{children first}{\forest@referencednodeid}%
8844 }
8845 \pgfdeclaregenericanchor{-children first}{%
8846   \forest@anchor@forwardtotikztrue
8847   \forest@anchor@do{#1}{-children first}{\forest@referencednodeid}%
8848 }
8849 \pgfdeclaregenericanchor{first}{%
8850   \forest@anchor@forwardtotikztrue
8851   \forest@anchor@do{#1}{first}{\forest@referencednodeid}%
8852 }
8853 \pgfdeclaregenericanchor{parent first}{%
8854   \forest@anchor@forwardtotikztrue
8855   \forest@anchor@do{#1}{parent first}{\forest@referencednodeid}%
8856 }
8857 \pgfdeclaregenericanchor{-parent first}{%
8858   \forest@anchor@forwardtotikztrue
8859   \forest@anchor@do{#1}{-parent first}{\forest@referencednodeid}%
8860 }
8861 \pgfdeclaregenericanchor{parent}{%
8862   \forest@anchor@forwardtotikztrue
8863   \forest@anchor@do{#1}{parent}{\forest@referencednodeid}%
8864 }
8865 \pgfdeclaregenericanchor{-parent}{%
8866   \forest@anchor@forwardtotikztrue
8867   \forest@anchor@do{#1}{-parent}{\forest@referencednodeid}%
```

```

8868 }
8869 \pgfdeclaregenericanchor{parent last}{%
8870   \forest@anchor@forwardtotikztrue
8871   \forest@anchor@do{#1}{parent last}{\forest@referencednodeid}%
8872 }
8873 \pgfdeclaregenericanchor{-parent last}{%
8874   \forest@anchor@forwardtotikztrue
8875   \forest@anchor@do{#1}{-parent last}{\forest@referencednodeid}%
8876 }
8877 \pgfdeclaregenericanchor{last}{%
8878   \forest@anchor@forwardtotikztrue
8879   \forest@anchor@do{#1}{last}{\forest@referencednodeid}%
8880 }
8881 \pgfdeclaregenericanchor{children last}{%
8882   \forest@anchor@forwardtotikztrue
8883   \forest@anchor@do{#1}{children last}{\forest@referencednodeid}%
8884 }
8885 \pgfdeclaregenericanchor{-children last}{%
8886   \forest@anchor@forwardtotikztrue
8887   \forest@anchor@do{#1}{-children last}{\forest@referencednodeid}%
8888 }
8889 \pgfdeclaregenericanchor{children'}{%
8890   \forest@anchor@forwardtotikztrue
8891   \forest@anchor@do{#1}{children'}{\forest@referencednodeid}%
8892 }
8893 \pgfdeclaregenericanchor{-children'}{%
8894   \forest@anchor@forwardtotikztrue
8895   \forest@anchor@do{#1}{-children'}{\forest@referencednodeid}%
8896 }
8897 \pgfdeclaregenericanchor{children first'}{%
8898   \forest@anchor@forwardtotikztrue
8899   \forest@anchor@do{#1}{children first'}{\forest@referencednodeid}%
8900 }
8901 \pgfdeclaregenericanchor{-children first'}{%
8902   \forest@anchor@forwardtotikztrue
8903   \forest@anchor@do{#1}{-children first'}{\forest@referencednodeid}%
8904 }
8905 \pgfdeclaregenericanchor{first'}{%
8906   \forest@anchor@forwardtotikztrue
8907   \forest@anchor@do{#1}{first'}{\forest@referencednodeid}%
8908 }
8909 \pgfdeclaregenericanchor{parent first'}{%
8910   \forest@anchor@forwardtotikztrue
8911   \forest@anchor@do{#1}{parent first'}{\forest@referencednodeid}%
8912 }
8913 \pgfdeclaregenericanchor{-parent first'}{%
8914   \forest@anchor@forwardtotikztrue
8915   \forest@anchor@do{#1}{-parent first'}{\forest@referencednodeid}%
8916 }
8917 \pgfdeclaregenericanchor{parent'}{%
8918   \forest@anchor@forwardtotikztrue
8919   \forest@anchor@do{#1}{parent'}{\forest@referencednodeid}%
8920 }
8921 \pgfdeclaregenericanchor{-parent'}{%
8922   \forest@anchor@forwardtotikztrue
8923   \forest@anchor@do{#1}{-parent'}{\forest@referencednodeid}%
8924 }
8925 \pgfdeclaregenericanchor{parent last'}{%
8926   \forest@anchor@forwardtotikztrue
8927   \forest@anchor@do{#1}{parent last'}{\forest@referencednodeid}%
8928 }

```

```

8929 \pgfdeclaregenericanchor{-parent last'}{%
8930   \forest@anchor@forwardtotikztrue
8931   \forest@anchor@do{#1}{-parent last'}{\forest@referencednodeid}%
8932 }
8933 \pgfdeclaregenericanchor{last'}{%
8934   \forest@anchor@forwardtotikztrue
8935   \forest@anchor@do{#1}{last'}{\forest@referencednodeid}%
8936 }
8937 \pgfdeclaregenericanchor{children last'}{%
8938   \forest@anchor@forwardtotikztrue
8939   \forest@anchor@do{#1}{children last'}{\forest@referencednodeid}%
8940 }
8941 \pgfdeclaregenericanchor{-children last'}{%
8942   \forest@anchor@forwardtotikztrue
8943   \forest@anchor@do{#1}{-children last'}{\forest@referencednodeid}%
8944 }

```

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

```

8945 \def\forest@anchor@do#1#2#3{% #1 = shape name, #2 = (potentially) forest anchor, #3 = node id
8946   \forest@fornode{#3}{%
8947     \def\forest@temp@anchor{#2}%
8948     \forest@anchor@snapbordertocompassfalse
8949     \forest@anchor@isborderfalse
8950     \forest@anchor@to@tikz@anchor
8951     \forest@anchor@border@to@compass
8952     \ifforest@anchor@forwardtotikz
8953       \forest@anchor@forward{#1}%
8954     \else
8955       \fi
8956   }%
8957 }

```

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

```

8958 \def\forest@anchor@to@tikz@anchor{%
8959   \ifcsdef{forest@anchor@@\forest@temp@anchor}{%
8960     \csuse{forest@anchor@@\forest@temp@anchor}%
8961     \forest@anchor@to@tikz@anchor
8962   }{}%
8963 }

```

Actual computation.

```

8964 \csdef{forest@anchor@@parent anchor}{%
8965   \foresttoget{parent anchor}\forest@temp@anchor}
8966 \csdef{forest@anchor@@child anchor}{%
8967   \foresttoget{child anchor}\forest@temp@anchor}
8968 \csdef{forest@anchor@@anchor}{%
8969   \foresttoget{anchor}\forest@temp@anchor}
8970 \csdef{forest@anchor@@children'}{%
8971   \forest@anchor@isbordertrue
8972   \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}}}%
8973 }
8974 \csdef{forest@anchor@@-children'}{%
8975   \forest@anchor@isbordertrue
8976   \edef\forest@temp@anchor{\number\numexpr 180+\forestove{grow}-\forestove{rotate}}}%
8977 }
8978 \csdef{forest@anchor@@parent'}{%
8979   \forest@anchor@isbordertrue
8980   \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestove{\forestove{@parent}}\gro
8981   \edef\forest@temp@anchor{\number\numexpr\forest@temp@grow-\forestove{rotate}+180}%
8982 }
8983 \csdef{forest@anchor@@-parent'}{%
8984   \forest@anchor@isbordertrue

```

```

8985 \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestOve{\forestove{@parent}}\gro
8986 \edef\forest@temp@anchor{\number\numexpr\forest@temp@grow-\forestove{rotate}}}%
8987 }
8988 \csdef{forest@anchor@@first'}{%
8989 \forest@anchor@isbordertrue
8990 \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=0 -\e
8991 }
8992 \csdef{forest@anchor@@last'}{%
8993 \forest@anchor@isbordertrue
8994 \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=0 +\e
8995 }
8996 \csdef{forest@anchor@@parent first'}{%
8997 \forest@anchor@isbordertrue
8998 \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestOve{\forestove{@parent}}\gro
8999 \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestOve{\forestove{@pare
9000 \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}+180}%
9001 \edef\forest@temp@anchor@first{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@revers
9002 \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@first}\forest@temp@anchor
9003 }
9004 \csdef{forest@anchor@@-parent first'}{%
9005 \forest@anchor@isbordertrue
9006 \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestOve{\forestove{@parent}}\gro
9007 \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestOve{\forestove{@pare
9008 \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}}}%
9009 \edef\forest@temp@anchor@first{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@revers
9010 \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@first}\forest@temp@anchor
9011 }
9012 \csdef{forest@anchor@@parent last'}{%
9013 \forest@anchor@isbordertrue
9014 \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestOve{\forestove{@parent}}\gro
9015 \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestOve{\forestove{@pare
9016 \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}+180}%
9017 \edef\forest@temp@anchor@last{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@reverse
9018 \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@last}\forest@temp@anchor
9019 }
9020 \csdef{forest@anchor@@-parent last'}{%
9021 \forest@anchor@isbordertrue
9022 \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestOve{\forestove{@parent}}\gro
9023 \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestOve{\forestove{@pare
9024 \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}}}%
9025 \edef\forest@temp@anchor@last{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@reverse
9026 \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@last}\forest@temp@anchor
9027 }
9028 \csdef{forest@anchor@@children first'}{%
9029 \forest@anchor@isbordertrue
9030 \edef\forest@temp@anchor@first{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}
9031 \forest@getaverageangle{\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@first}\forest@temp@anchor
9032 }
9033 \csdef{forest@anchor@@-children first'}{%
9034 \forest@anchor@isbordertrue
9035 \edef\forest@temp@anchor@first{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}
9036 \forest@getaverageangle{180+\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@first}\forest@temp@anc
9037 }
9038 \csdef{forest@anchor@@children last'}{%
9039 \forest@anchor@isbordertrue
9040 \edef\forest@temp@anchor@last{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=
9041 \forest@getaverageangle{\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@last}\forest@temp@anchor
9042 }
9043 \csdef{forest@anchor@@-children last'}{%
9044 \forest@anchor@isbordertrue
9045 \edef\forest@temp@anchor@last{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=

```

```

9046 \forest@getaverageangle{180+\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@last}\forest@temp@anch
9047 }
9048 \csdef{forest@anchor@@children}{%
9049 \forest@anchor@snapbordertocompasstrue
9050 \csuse{forest@anchor@@children'}%
9051 }
9052 \csdef{forest@anchor@@-children}{%
9053 \forest@anchor@snapbordertocompasstrue
9054 \csuse{forest@anchor@@-children'}%
9055 }
9056 \csdef{forest@anchor@@parent}{%
9057 \forest@anchor@snapbordertocompasstrue
9058 \csuse{forest@anchor@@parent'}%
9059 }
9060 \csdef{forest@anchor@@-parent}{%
9061 \forest@anchor@snapbordertocompasstrue
9062 \csuse{forest@anchor@@-parent'}%
9063 }
9064 \csdef{forest@anchor@@first}{%
9065 \forest@anchor@snapbordertocompasstrue
9066 \csuse{forest@anchor@@first'}%
9067 }
9068 \csdef{forest@anchor@@last}{%
9069 \forest@anchor@snapbordertocompasstrue
9070 \csuse{forest@anchor@@last'}%
9071 }
9072 \csdef{forest@anchor@@parent first}{%
9073 \forest@anchor@snapbordertocompasstrue
9074 \csuse{forest@anchor@@parent first'}%
9075 }
9076 \csdef{forest@anchor@@-parent first}{%
9077 \forest@anchor@snapbordertocompasstrue
9078 \csuse{forest@anchor@@-parent first'}%
9079 }
9080 \csdef{forest@anchor@@parent last}{%
9081 \forest@anchor@snapbordertocompasstrue
9082 \csuse{forest@anchor@@parent last'}%
9083 }
9084 \csdef{forest@anchor@@-parent last}{%
9085 \forest@anchor@snapbordertocompasstrue
9086 \csuse{forest@anchor@@-parent last'}%
9087 }
9088 \csdef{forest@anchor@@children first}{%
9089 \forest@anchor@snapbordertocompasstrue
9090 \csuse{forest@anchor@@children first'}%
9091 }
9092 \csdef{forest@anchor@@-children first}{%
9093 \forest@anchor@snapbordertocompasstrue
9094 \csuse{forest@anchor@@-children first'}%
9095 }
9096 \csdef{forest@anchor@@children last}{%
9097 \forest@anchor@snapbordertocompasstrue
9098 \csuse{forest@anchor@@children last'}%
9099 }
9100 \csdef{forest@anchor@@-children last}{%
9101 \forest@anchor@snapbordertocompasstrue
9102 \csuse{forest@anchor@@-children last'}%
9103 }

```

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

624872.

```

9104 \def\forest@getaverageangle#1#2#3{%
9105   \edef\forest@temp{\number\numexpr #1-#2+540}%
9106   \expandafter\pgfmathMod@\expandafter{\forest@temp}{360}%
9107   \forest@truncatepgfmathresult
9108   \edef\forest@temp{\number\numexpr 360+#2+((\pgfmathresult-180)/2)}%
9109   \expandafter\pgfmathMod@\expandafter{\forest@temp}{360}%
9110   \forest@truncatepgfmathresult
9111   \let#3\pgfmathresult
9112 }
9113 \def\forest@truncatepgfmathresult{%
9114   \afterassignment\forest@gobbletoEND
9115   \forest@temp@count=\pgfmathresult\forest@END
9116   \def\pgfmathresult{\the\forest@temp@count}%
9117 }
9118 \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.

```

9119 \def\forest@anchor@border@to@compass{%
9120   \ifforest@anchor@isborder % snap to 45 deg, to range 0-360
9121     \ifforest@anchor@snapbordertocompass
9122       \forest@anchor@snap@border@to@compass
9123     \else % to range 0-360
9124       \pgfmathMod@{\forest@temp@anchor}{360}%
9125       \forest@truncatepgfmathresult
9126       \let\forest@temp@anchor\pgfmathresult
9127     \fi
9128     \ifforest@anchor@snapbordertocompass
9129     \ifforest@anchor@forwardtotikz
9130     \ifcsname pgf@anchor%
9131       @\csname pgf@sh@ns@\pgfreferencednodename\endcsname
9132       @\csname forest@compass@\forest@temp@anchor\endcsname
9133     \endcsname
9134     \letcs\forest@temp@anchor{forest@compass@\forest@temp@anchor}%
9135     \fi
9136   \else
9137     \letcs\forest@temp@anchor{forest@compass@\forest@temp@anchor}%
9138   \fi
9139 \fi
9140 \fi
9141 }
9142 \csdef{forest@compass@0}{east}
9143 \csdef{forest@compass@45}{north east}
9144 \csdef{forest@compass@90}{north}
9145 \csdef{forest@compass@135}{north west}
9146 \csdef{forest@compass@180}{west}
9147 \csdef{forest@compass@225}{south west}
9148 \csdef{forest@compass@270}{south}
9149 \csdef{forest@compass@315}{south east}
9150 \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).

```

9151 \def\forest@anchor@snap@border@to@compass{%
9152   \pgfmathMod@{\forest@temp@anchor}{360}%
9153   \pgfmathdivide@{\pgfmathresult}{45}%
9154   \pgfmathround@{\pgfmathresult}%
9155   \pgfmathmultiply@{\pgfmathresult}{45}%
9156   \forest@truncatepgfmathresult
9157   \let\forest@temp@anchor\pgfmathresult

```

9158 }

This macro forwards the resulting anchor to TikZ.

```

9159 \def\forest@anchor@forward#1{% #1 = shape name
9160   \ifdefempty\forest@temp@anchor{%
9161     \pgf@sh@reanchor{#1}{center}%
9162     \xdef\forest@hack@tikzshapeborder{%
9163       \noexpand\tikz@shapebordertrue
9164       \def\noexpand\tikz@shapeborder@name{\pgfreferencednodename}%
9165     }\aftergroup\forest@hack@tikzshapeborder
9166   }{%
9167     \pgf@sh@reanchor{#1}{\forest@temp@anchor}%
9168   }%
9169 }
```

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

```

9170 \def\forest@referencednodeid{\forest@node@Nametoid{\forest@referencednodename}}%
9171 \def\forest@referencednodename{%
9172   \expandafter\expandafter\expandafter\forest@referencednodename\expandafter\pgfreferencednodename\forest@pgf@notyetpositioned
9173 }
9174 \expandafter\def\expandafter\forest@referencednodename\expandafter#\expandafter1\forest@pgf@notyetpositioned
9175   \if\relax#1\relax\forest@referencednodename@stripafter#2\relax\fi
9176   \if\relax#2\relax#1\fi
9177 }
9178 \expandafter\def\expandafter\forest@referencednodename@stripafter\expandafter#\expandafter1\forest@pgf@notyetpositioned
```

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.

```

9179 \def\forest@pointanchor#1{% #1 = anchor
9180   \forest@Pointanchor{\forest@cn}{#1}%
9181 }
9182 \def\forest@Pointanchor#1#2{% #1 = node id, #2 = anchor
9183   \def\forest@pa@temp@name{name}%
9184   \forest@Oifdefined{#1}{@box}{%
9185     \forest@Oget{#1}{@box}\forest@temp
9186     \ifdefempty\forest@temp{}{%
9187       \def\forest@pa@temp@name{later@name}%
9188     }%
9189   }{}%
9190   \setbox0\hbox{%
9191     \begin{pgfpicture}% this is expensive: so that's why caching
9192       \if\relax\forest@Ove{#1}{#2}\relax
9193         \pgfpointanchor{\forest@Ove{#1}}{\forest@pa@temp@name}{center}%
9194       \else
9195         \pgfpointanchor{\forest@Ove{#1}}{\forest@pa@temp@name}{\forest@Ove{#1}{#2}}%
9196       \fi
9197       \xdef\forest@global@marshal{%
9198         \noexpand\global\noexpand\pgf@x=\the\pgf@x\relax
9199         \noexpand\global\noexpand\pgf@y=\the\pgf@y\relax\relax
9200       }%
9201     \end{pgfpicture}%
9202   }%
9203   \forest@global@marshal
9204 }
```

11 Compatibility with previous versions

```

9205 \ifdefempty{\forest@compat}{}{%
9206   \RequirePackage{forest-compat}
9207 }
```