

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

v2.1.4

Sašo Živanović\*

February 2, 2017

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

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

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

## Contents

1	Identification	1
2	Package options	2
3	Patches	3
4	Utilities	4
4.1	Arrays	10
4.2	Testing for numbers and dimensions	14
4.3	forestmath	17
4.4	Sorting	23
5	The bracket representation parser	27
5.1	The user interface macros	27
5.2	Parsing	28
5.3	The tree-structure interface	32
6	Nodes	33
6.1	Option setting and retrieval	33
6.2	Tree structure	36
6.3	Node options	45
6.3.1	Option-declaration mechanism	45
7	Handlers	52
7.1	.process	53
7.1.1	Registers	62
7.1.2	Declaring options	68
7.1.3	Option propagation	75
7.2	Aggregate functions	77
7.2.1	pgfmath extensions	80
7.3	Nodewalk	82
7.4	Dynamic tree	102

---

\*e-mail: [saso.zivanovic@guest.arnes.si](mailto:saso.zivanovic@guest.arnes.si); web: <http://spj.ff.uni-lj.si/zivanovic/>

8	Stages	107
8.1	Typesetting nodes	111
8.2	Packing	113
8.2.1	Tiers	125
8.2.2	Node boundary	128
8.3	Compute absolute positions	133
8.4	Drawing the tree	134
9	Geometry	136
9.1	Projections	136
9.2	Break path	139
9.3	Get tight edge of path	141
9.4	Get rectangle/band edge	147
9.5	Distance between paths	148
9.6	Utilities	151
10	The outer UI	153
10.1	Externalization	153
10.2	The forest environment	154
10.3	Standard node	157
10.4	ls coordinate system	159
10.5	Relative node names in <i>TikZ</i>	160
10.6	Anchors	161
11	Compatibility with previous versions	168

# 1 Identification

```

1 \ProvidesPackage{forest}[2017/02/02 v2.1.4 Drawing (linguistic) trees]
2
3 \RequirePackage{tikz}[2013/12/13]
4 \usetikzlibrary{shapes}
5 \usetikzlibrary{fit}
6 \usetikzlibrary{calc}
7 \usepgflibrary{intersections}
8
9 \RequirePackage{pgfplots}
10 \RequirePackage{etoolbox}[2010/08/21]
11 \RequirePackage{elocalloc}% for \locbox
12 \RequirePackage{environ}
13 \RequirePackage{xparse}
14
15 \RequirePackage{inlinedef}
16 \newtoks\ID@usercommands{}
17 \newcommand\NewInlineCommand[3][0]{%
18   \newcommand#2[#1]{#3}%
19   \ID@usercommands\x{%
20     \the\ID@usercommands
21     \ifx\@foo#2%
22       \def\next{\ID@expandunsafe#2}%
23     \fi
24   }%
25 }
26 \def\@ExpandIfTF#1{%
27   \csname
28     % I'm not 100% sure if this plays well in every situation
29     \csname if#1\endcsname
30     @firstoftwo%
31   \else
32     @secondoftwo%
```

```

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

/forest is the root of the key hierarchy.
83 \pgfkeys{/forest/.is family}
84 \def\forestset#1{\pgfkeys{/forest}{#1}}

```

## 2 Package options

```

85 \newif\ifforest@external@
86 \newif\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\csname #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\csname #1repeat#2\endcsname{%
184         \noexpand\csdef{#1iterate#2}{####1\relax\noexpand\expandafter\expandafter\noexpand\csname#1iterate#
185         \expandafter\noexpand\csname#1iterate#2\endcsname
186         \let\expandafter\noexpand\csname#1iterate#2\endcsname\relax
187       }%
188     }%
189     \newloop@marshal
190   }%
191 }%
192 }%

```

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

```

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

```

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

```

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

```

Additional loops (for embedding).

```

244 \newloop\forest@loop

```

New counters, *dimens*, *ifs*.

```

245 \newdimen\forest@temp@dimen
246 \newcount\forest@temp@count
247 \newcount\forest@n
248 \newif\ifforest@temp
249 \newcount\forest@temp@global@count
250 \newtoks\forest@temp@toks

```

Appending and prepending to token lists.

```

251 \def\etotoks#1#2{\edef\pot@temp{#2}\expandafter#1\expandafter{\pot@temp}}
252 \def\apptotoks#1#2{\expandafter#1\expandafter{\the#1#2}}
253 \long\def\lapptotoks#1#2{\expandafter#1\expandafter{\the#1#2}}
254 \def\eapptotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter#1\expandafter\expandafter\expandafter
255 \def\pretotoks#1#2{\toks@={#2}\expandafter\expandafter\expandafter#1\expandafter\expandafter\expandafter\expandafter{\exp
256 \def\epretotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter#1\expandafter\expandafter\expandafter\expandafter
257 \def\gapptotoks#1#2{\expandafter\global\expandafter#1\expandafter{\the#1#2}}
258 \def\xapptotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter\global\expandafter\expandafter\expandafter\exp
259 \def\gpretotoks#1#2{\toks@={#2}\expandafter\expandafter\expandafter\global\expandafter\expandafter\expandafter\expandafter
260 \def\xpretotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter\global\expandafter\expandafter\expandafter\exp

```

Expanding number arguments.

```

261 \def\expandnumberarg#1#2{\expandafter#1\expandafter{\number#2}}
262 \def\expandtwonumberargs#1#2#3{%
263   \expandafter\expandtwonumberargs@\expandafter#1\expandafter{\number#3}{#2}}
264 \def\expandtwonumberargs@#1#2#3{%
265   \expandafter#1\expandafter{\number#3}{#2}}
266 \def\expandthreenumberargs#1#2#3#4{%
267   \expandafter\expandthreenumberargs@\expandafter#1\expandafter{\number#4}{#2}{#3}}
268 \def\expandthreenumberargs@#1#2#3#4{%
269   \expandafter\expandthreenumberargs@@\expandafter#1\expandafter{\number#4}{#2}{#3}}
270 \def\expandthreenumberargs@@#1#2#3#4{%
271   \expandafter#1\expandafter{\number#4}{#2}{#3}}

```

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

```

272 \def\forest@convert@others@to@underscores#1#2{%
273   \def\forest@cotu@result{}%
274   \forest@cotu@first#1\forest@end
275   \let#2\forest@cotu@result
276 }
277 \def\forest@cotu{%
278   \let\forest@cotu@have@num\forest@cotu@have@alpha
279   \futurelet\forest@cotu@nextchar\forest@cotu@checkforspace
280 }
281 \def\forest@cotu@first{%
282   \let\forest@cotu@have@num\forest@cotu@have@other
283   \futurelet\forest@cotu@nextchar\forest@cotu@checkforspace
284 }
285 \def\forest@cotu@checkforspace{%
286   \expandafter\ifx\space\forest@cotu@nextchar
287     \let\forest@cotu@next\forest@cotu@havespace
288   \else
289     \let\forest@cotu@next\forest@cotu@nospace
290   \fi
291   \forest@cotu@next
292 }
293 \def\forest@cotu@havespace#1{%
294   \appto\forest@cotu@result{_}%
295   \forest@cotu#1%
296 }
297 \def\forest@cotu@nospace{%
298   \ifx\forest@cotu@nextchar\forest@end
299     \@escapeif@gobble

```

```

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

```

Additional list macros.

```

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

```

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

```

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

```

Utilities dealing with pgfkeys.

```

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

```



```

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

```

```

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

```

```

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

```

## 4.1 Arrays

```

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

```

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

```

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

```

```

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

```

```

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

```

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

```

\def\arrayvalues{% <-- \csedef{#1values}
\expandafter\expandafter\expandafter\arrayvaluesfromrange %\arrayvaluesfromrange <-- \expandonce{\csname#1
\expandafter\expandafter\expandafter}%

```

```

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

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

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

```

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

```

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

```

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

```

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

```

```

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

```

## 4.2 Testing for numbers and dimensions

Test if the argument is an integer (only base 10) that can be assigned to a 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`.

```

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

```

Expand in advance, like `pgfmath` does.

```

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

```

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

```

707 \expandafter\forest@isnum@a\forest@isnum@temp\forest@eon\forest@eon\forest@END
708 \ifforest@isnum
709 \expandafter\@firstoftwo
710 \else
711 \expandafter\@secondoftwo
712 \fi
713 }
714 \def\forest@isnum@a{\futurelet\forest@isnum@token\forest@isnum@b}

```

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

```

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

```

```

732 \forest@next
733 }

```

Eat + and -.

```

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

```

Eat the space!

```

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

```

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

```

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

```

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

```

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

```

Is it a dimension? We support all T<sub>E</sub>X'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<sub>E</sub>X-style \if-macro \forest@isdim and counter \forest@isdim@dimen.

```

761 \newcount\forest@isdim@nonintpart
762 \newif\ifforest@isdim
763 \def\forest@isdim#1{%
764   \forest@isdim>false
765   \forest@isnum@minus=false
766   \def\forest@isdim@leadingzeros{}%
767   \forest@isdim@nonintpart=0
768   \def\forest@isdim@unit{pt}%
769   \let\forest@isnum@next\forest@isdim@checkfordot
770   \edef\forest@isnum@temp{#1}%

```

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

```

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

```



```

777 }
778 \def\forest@isdim@checkfordot{%
779   \ifx.\forest@isnum@token
780     \expandafter\forest@isdim@dot
781   \else
782     \ifx,\forest@isnum@token
783       \expandafter\expandafter\expandafter\forest@isdim@dot
784     \else
785       \expandafter\expandafter\expandafter\forest@isdim@nodot
786     \fi
787   \fi
788 }
789 \def\forest@isdim@nodot{%
790   \ifforest@isnum

```

No number, no dot, so not a dimension.

```

791     \expandafter\forest@isdim@checkforunit
792   \else
793     \expandafter\forest@isdim@finish@nodim
794   \fi
795 }
796 \def\forest@isdim@dot#1{% #1=. or ,
797   \futurelet\forest@isnum@token\forest@isdim@collectzero
798 }
799 \def\forest@isdim@collectzero{%
800   \ifx0\forest@isnum@token
801     \expandafter\forest@isdim@collectzero@
802   \else
803     \expandafter\forest@isdim@getnonintpart
804   \fi
805 }
806 \def\forest@isdim@collectzero@#1{% #1 = 0
807   \appto\forest@isdim@leadingzeros{0}%
808   \futurelet\forest@isnum@token\forest@isdim@collectzero
809 }
810 \def\forest@isdim@getnonintpart{%
811   \afterassignment\forest@isdim@checkforunit\forest@isdim@nonintpart0%
812 }

```

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

```

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

```

```

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

```

### 4.3 forestmath

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

Evaluates #1 to a boolean: if true execute #2, otherwise #3. #2 and #3 are T<sub>E</sub>X code. Includes a shortcut for some common values.

```

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

```

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

```

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

```

These defer conversion (see aggregates).

```

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

```

Sets \pgfmathresulttype to the type of #1.

```

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

```

```

1051 }%
1052 }

```

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

```

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

```

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

```

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

```

Booleans.

```

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

```

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

Low level <.

```

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

```

Low level =.

```

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

```

High level <, > and =.

```

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

```

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

```

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

```

## 4.4 Sorting

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

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

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

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

```

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

```

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



```

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

```

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

```

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

```

Various counters and macros needed by the sorting algorithms.

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

Exchange the pivot and the first element.

```
1250 \forest@sort@xch{#1}{\the\forest@sort@p}%
```

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

```
1251 \forest@sort@m=#1\relax
```

Loop through the list.

```

1252 \forest@sort@k=#1\relax
1253 \forest@sort@loop
1254 \ifnum\forest@sort@k<#2\relax
1255 \advance\forest@sort@k 1

```

Compare the pivot and the current element.

```
1256 \forest@sort@cmp{#1}{\the\forest@sort@k}%
```

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

```

1257 \ifx\forest@sort@direction\forest@sort@cmp@result
1258 \advance\forest@sort@m 1
1259 \forest@sort@xch{\the\forest@sort@m}{\the\forest@sort@k}
1260 \fi
1261 \forest@sort@repeat

```

Move the pivot element into its final position.

```
1262 \forest@sort@xch{#1}{\the\forest@sort@m}%
```

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

```

1263 \forest@sort@k=\forest@sort@m
1264 \advance\forest@sort@k -1
1265 \advance\forest@sort@m 1
1266 \edef\forest@sort@marshal{%
1267 \noexpand\forest@sort@{#1}{\the\forest@sort@k}%
1268 \noexpand\forest@sort@{\the\forest@sort@m}{#2}%
1269 }%
1270 \forest@sort@marshal
1271 }
1272 % We defines the item-exchange macro in terms of the (user-provided)
1273 % array let macro.
1274 % \begin{macrocode}
1275 \def\forest@sort@aux{aux}
1276 \def\forest@sort@xch#1#2{%
1277 \forest@sort@let{\forest@sort@aux}{#1}%
1278 \forest@sort@let{#1}{#2}%
1279 \forest@sort@let{#2}{\forest@sort@aux}%
1280 }

```

Insertion sort.

```

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

```

```

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

```

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

Compare numbers.

```

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

```

Compare dimensions.

```

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

```

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

```

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

```

```

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

```

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

```

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

```

## 5 The bracket representation parser

### 5.1 The user interface macros

Settings.

```

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

```

`\bracketParse` is the macro that should be called to parse a balanced bracket representation. It takes two parameters: #1 is the code that will be run after parsing the bracket; #2 is a control sequence

that will receive the id of the root of the created tree structure. (The `bracket` representation should follow (after optional spaces), but is not a formal parameter of the macro.)

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

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

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

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

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

By default, don’t expand anything.

```
1400   \bracket@expandtokensfalse
```

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

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

Finally, we start the parser.

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

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

```
1407 \def\bracketResume{\bracket@Parse}%
```

## 5.2 Parsing

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

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

We either fully expand the next token (using a popular T<sub>E</sub>Xnical trick ...) or don’t expand it at all, depending on the state of `\ifbracket@expandtokens`.

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

```

1427 } \fi
1428 }

```

We then look ahead to see what's coming.

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

Declare states.

```

1496 \def\bracket@state@inContent{0}
1497 \def\bracket@state@inAfterthought{1}
1498 \def\bracket@state@starting{2}

```

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

```

1499 \def\bracket@Parse@openingBracketFound{%
1500 \bracket@haveSpacefalse
1501 \ifcase\bracket@state\relax% in content [ ... [

```

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

```

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

```

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

```

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

```

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

```

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

```

```

1521 \fi
1522 }
1523 \def\bracket@Parse@closingBracketFound{%
1524 \bracket@haveSpacefalse
1525 \ifcase\bracket@state\relax % in content [ ... ]

```

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

```

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

```

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

```

1540 \escapeif{%
1541 \bracket@addAfterthought
1542 \let\bracket@afterthoughtNode\bracket@parentNode
1543 \edef\bracket@parentNode{\forest@ve{\bracket@parentNode}{@parent}}%
1544 \ifnum\bracket@parentNode=0
1545 \expandafter\bracket@endParsingHook
1546 \else
1547 \expandafter\bracket@Parse
1548 \fi
1549 }%
1550 \else % starting
  {start}...]: something's obviously wrong with the input here...
1551 \PackageError{forest}{You're attempting to start a bracket representation
1552 with a closing bracket}{}%
1553 \fi
1554 }

```

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

```

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

```



```

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

```

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

```

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

```

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

```

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

```

Inhibit expansion of the next token.

```

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

```

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

```

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

```

Anything else will be expanded once.

```

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

```

### 5.3 The tree-structure interface

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

```

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

```

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

```

1608 \def\bracket@addAfterthought{%

```

```

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

```

## 6 Nodes

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

### 6.1 Option setting and retrieval

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

```

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

```

User macros for retrieving node options of the current node.

```

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

```

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

```

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

```

```

1656 }
1657 \def\forestOappto#1#2#3{%
1658   \forestOeset{#1}{#2}{\forestOv{#1}{#2}\unexpanded{#3}}%
1659 }
1660 \def\forestOeappto#1#2#3{%
1661   \forestOeset{#1}{#2}{\forestOv{#1}{#2}#3}%
1662 }
1663 \def\forestOpreto#1#2#3{%
1664   \forestOeset{#1}{#2}{\unexpanded{#3}\forestOv{#1}{#2}}%
1665 }
1666 \def\forestOepreto#1#2#3{%
1667   \forestOeset{#1}{#2}{#3\forestOv{#1}{#2}}%
1668 }
1669 \def\forestOifdefined#1#2#3#4
1670 {%
1671   \ifcsdef{fRsT#1/#2}%{#3}{#4}%
1672 }
1673 \def\forestOlet0#1#2#3#4{% option #2 of node #1 <-- option #4 of node #3
1674   \forestOget{#3}{#4}\forestoption@temp
1675   \forestOlet{#1}{#2}\forestoption@temp}
1676 \def\forestOleto#1#2#3{%
1677   \forestoget{#3}\forestoption@temp
1678   \forestOlet{#1}{#2}\forestoption@temp}
1679 \def\forestolet0#1#2#3{%
1680   \forestOget{#2}{#3}\forestoption@temp
1681   \forestolet{#1}\forestoption@temp}
1682 \def\forestoleto#1#2{%
1683   \forestoget{#2}\forestoption@temp
1684   \forestolet{#1}\forestoption@temp}

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

```

1715 }

Macros for retrieving/setting registers.

```

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

```

User macros for retrieving node options of the current node.

```

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

```

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

```

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

```

```

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

```

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

```

1796 \def\forest@cn{0}
1797 \forest@node@init

```

## 6.2 Tree structure

Node insertion/removal.

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

```

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

```

```

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

```

Do some stuff and return to the current node.

```

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

```

Looping methods: children.

```

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

```

```

1940 \fi
1941 }

```

Looping methods: (sub)tree and descendants.

```

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

```

Looping methods: breadth-first.

```

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

```



```

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

```

```

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

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

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

```

```

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

```

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

2269 \NewDocumentCommand\forestdebugtypeouttrees{o}{%
2270   \forestdebug@typeouttrees\forest@temp
2271   \typeout{%
2272     \forestdebugtypeouttreesprefix
2273     \IfValueTF{#1}{#1: }{}%
2274     \detokenize\expandafter{\forest@temp}%
2275     \forestdebugtypeouttreessuffix
2276   }%
2277 }
2278 \def\forestdebug@typeouttrees#1{% #1 = cs to store the result
2279   \begingroup
2280   \edef\forest@temp@message{}%
2281   \def\forestdebug@typeouttrees@n{0}%

```

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

```

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

```

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

```

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

```

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

```

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

```

## 6.3 Node options

### 6.3.1 Option-declaration mechanism

Common code for declaring options.

```

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

```

```

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

```

Option-declaration handlers.

```

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

```

```

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

```



```

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

```

```

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

```

```

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

```

Nothing else should be defined in this namespace.

```

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

```

```

2641 #2/not #3/.code={\forest0set{\forest@setter@node}{#3}{0}},
2642 #2/if #3/.code 2 args={%
2643   \forest0get{#3}\forest@temp@option@value
2644   \ifnum\forest@temp@option@value=0
2645     \pgfkeysalso{##2}%
2646   \else
2647     \pgfkeysalso{##1}%
2648   \fi
2649 },
2650 #2/where #3/.style 2 args={for tree={#2/if #3={##1}{##2}}}
2651 }%
2652 \pgfkeyssetvalue{#1/option@name}{#3}%
2653 \pgfkeyslet{#1/@type}\forestmathtype@count % for .process & co
2654 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@count{##1}{#3}}%
2655 }
2656 \forestset{
2657   declare toks/.code 2 args={%
2658     \forest@declarehandler\forest@declaretoks@handler{#1}{#2}%
2659   },
2660   declare autowrapped toks/.code 2 args={%
2661     \forest@declarehandler\forest@declareautowrappedtoks@handler{#1}{#2}%
2662   },
2663   declare keylist/.code 2 args={%
2664     \forest@declarehandler\forest@declarekeylist@handler{#1}{#2}%
2665   },
2666   declare readonly dimen/.code 2 args={%
2667     \forestmathsetlengthmacro\forest@temp{#2}%
2668     \edef\forest@marshal{%
2669       \unexpanded{\forest@declarehandler\forest@declarereadonlydimen@handler{#1}}{\forest@temp}%
2670     }\forest@marshal
2671   },
2672   declare dimen/.code 2 args={%
2673     \forestmathsetlengthmacro\forest@temp{#2}%
2674     \edef\forest@marshal{%
2675       \unexpanded{\forest@declarehandler\forest@declaredimen@handler{#1}}{\forest@temp}%
2676     }\forest@marshal
2677   },
2678   declare readonly count/.code 2 args={%
2679     \forestmathtruncatemacro\forest@temp{#2}%
2680     \edef\forest@marshal{%
2681       \unexpanded{\forest@declarehandler\forest@declarereadonlycount@handler{#1}}{\forest@temp}%
2682     }\forest@marshal
2683   },
2684   declare count/.code 2 args={%
2685     \forestmathtruncatemacro\forest@temp{#2}%
2686     \edef\forest@marshal{%
2687       \unexpanded{\forest@declarehandler\forest@declarecount@handler{#1}}{\forest@temp}%
2688     }\forest@marshal
2689   },
2690   declare boolean/.code 2 args={%
2691     \forestmath@if{#2}{%
2692       \def\forest@temp{1}%
2693     }{%
2694       \def\forest@temp{0}%
2695     }%
2696     \edef\forest@marshal{%
2697       \unexpanded{\forest@declarehandler\forest@declareboolean@handler{#1}}{\forest@temp}%
2698     }\forest@marshal
2699   },

```

## 7 Handlers

```

2700 /handlers/.restore default value/.code={%
2701   \edef\forest@handlers@currentpath{\pgfkeyscurrentpath}%
2702   \pgfkeysgetvalue{\pgfkeyscurrentpath/option@name}\forest@currentoptionname
2703   \pgfkeysgetvalue{/forest/\forest@currentoptionname}\forest@temp
2704   \expandafter\pgfkeysalso\expandafter{\forest@handlers@currentpath/.expand once=\forest@temp}%
2705 },
2706 /handlers/.pgfmath/.code={%
2707   \pgfmathparse{#1}%
2708   \pgfkeysalso{\pgfkeyscurrentpath/.expand once=\pgfmathresult}%
2709 },
2710 /handlers/.wrap value/.code={%
2711   \edef\forest@handlers@wrap@currentpath{\pgfkeyscurrentpath}%
2712   \pgfkeysgetvalue{\forest@handlers@wrap@currentpath/option@name}\forest@currentoptionname
2713   \forestOget{\pgfkeysvalueof{/forest/\forest@currentoptionname/node@or@reg}}{\forest@currentoptionname}\fo
2714   \forest@def@with@pgfeov\forest@wrap@code{#1}%
2715   \expandafter\edef\expandafter\forest@wrapped@value\expandafter{\expandafter\expandonce\expandafter{\expan
2716   \pgfkeysalso{\forest@handlers@wrap@currentpath/.expand once=\forest@wrapped@value}%
2717 },
2718 /handlers/.option/.code={%
2719   \edef\forest@temp{\pgfkeyscurrentpath}%
2720   \expandafter\forest@handlers@option\expandafter{\forest@temp}{#1}%
2721 },
2722 }
2723 \def\forest@handlers@option#1#2{%#1=pgfkeyscurrentpath,#2=relative node name
2724   \forestRNOget{#2}\forest@temp
2725   \pgfkeysalso{#1/.expand once={\forest@temp}}%
2726 }%
2727 \forestset{
2728   /handlers/.register/.code={%
2729     \edef\forest@marshal{%
2730       \noexpand\pgfkeysalso{\pgfkeyscurrentpath={\forestregister{#1}}}%
2731     }\forest@marshal
2732   },
2733   /handlers/.wrap pgfmath arg/.code 2 args={%
2734     \forestmathparse{#2}\let\forest@wrap@arg@i\forestmathresult
2735     \edef\forest@wrap@args{{\expandonce\forest@wrap@arg@i}}%
2736     \def\forest@wrap@code##1{#1}%
2737     % here we don't call \forest@wrap@pgfmath@args@@@wrapandpasson, as compat-2.0.2-wrappgfmathargs changes t
2738     \expandafter\expandafter\expandafter\forest@temp@toks\expandafter\expandafter\expandafter{\expandafter\fo
2739     \expandafter\pgfkeysalso\expandafter{\expandafter\pgfkeyscurrentpath\expandafter=\expandafter{\the\forest
2740   },
2741   /handlers/.wrap 2 pgfmath args/.code n args={3}{%
2742     \forestmathparse{#2}\let\forest@wrap@arg@i\forestmathresult
2743     \forestmathparse{#3}\let\forest@wrap@arg@ii\forestmathresult
2744     \edef\forest@wrap@args{{\expandonce\forest@wrap@arg@i}{\expandonce\forest@wrap@arg@ii}}%
2745     \def\forest@wrap@code##1##2{#1}%
2746     \forest@wrap@pgfmath@args@@@wrapandpasson
2747   },
2748   /handlers/.wrap 3 pgfmath args/.code n args={4}{%
2749     \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{-}{-}{-}{-}{3}%
2750     \forest@wrap@n@pgfmath@do{#1}{3}},
2751   /handlers/.wrap 4 pgfmath args/.code n args={5}{%
2752     \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{-}{-}{-}{4}%
2753     \forest@wrap@n@pgfmath@do{#1}{4}},
2754   /handlers/.wrap 5 pgfmath args/.code n args={6}{%
2755     \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{-}{-}{5}%
2756     \forest@wrap@n@pgfmath@do{#1}{5}},
2757   /handlers/.wrap 6 pgfmath args/.code n args={7}{%
2758     \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{#7}{-}{6}%

```

```

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

```

The following macro is redefined by compat key 2.0.2-wrappgfm@args.

```

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

```

## 7.1 .process

```

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

```

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

```

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

```

```

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

```

Debugging and error messages.

```

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

```

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

```

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

```

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

```

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

```



```

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

```

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

No optional n:

```

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

```

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

```

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

```

```

2989 \fi
2990 }

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

Definitions of processing instructions. Processors should be defined using \forest@def@processor.
If they take arguments: yes, they follow, but they were scanned in \forest@process@catregime. Pro-
cessors should manipulate arrays \forest@process@left@ and \forest@process@right. They should
set \def\forestmathresulttype to _ not defined, n number, d dimension, P pgfmath or t text.
3003 \forest@def@processor{_}{1}{*}{*}{% no processing, no type
3004 \forest@process@right@bottompop\forest@temp
3005 \forest@process@left@letappend\forest@temp
3006 }
3007 \forest@def@processor{n}{1}{*}{*}{% numexpr
3008 \forest@process@right@bottompop\forest@temp
3009 \forest@process@left@esetappend{\number\numexpr\forest@temp}%
3010 \let\forestmathresulttype\forestmathtype@count
3011 }
3012 \forest@def@processor{d}{1}{*}{*}{% dimexpr
3013 \forest@process@right@bottompop\forest@temp
3014 \forest@process@left@esetappend{\the\dimexpr\forest@temp}%
3015 \let\forestmathresulttype\forestmathtype@dimen
3016 }
3017 \forest@def@processor{P}{1}{*}{*}{% pgfmath expression
3018 \forest@process@right@bottompop\forest@temp
3019 \pgfmathparse{\forest@temp}%
3020 \forest@process@left@letappend\pgfmathresult
3021 \let\forestmathresulttype\forestmathtype@unitless
3022 }
3023 \forest@def@processor{p}{1}{*}{*}{% process expression
3024 \forest@process@right@bottompop\forest@temp@a
3025 \def\forest@temp{\forest@process{true}}%
3026 \expandafter\forest@temp\forest@temp@a\forest@eov
3027 \let\forest@topextend@next\relax
3028 \edef\forest@temp{\forest@process@result@values}%
3029 \expandafter\forest@process@left@topextend\forest@temp\forest@eov
3030 \let\forestmathresulttype\forest@process@result@type
3031 }
3032 \forest@def@processor{t}{1}{*}{*}{% text
3033 \forest@process@right@bottompop\forest@temp
3034 \forest@process@left@letappend\forest@temp
3035 \let\forestmathresulttype\forestmathtype@textasc
3036 }
3037 \forest@def@processor{-}{1}{*}{*}{% toggle ascending/descending
3038 \forest@process@left@toppop\forestmathresult
3039 \csname forest@processor@-\forestmathresulttype\endcsname
3040 \forest@process@left@letappend\forestmathresult
3041 }
3042 \cslet{forest@processor@-\forestmathtype@generic}\relax

```

```

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

```

Expansions:

```

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

```

Access to FOREST data.

```

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

```

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

```

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

```

```

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

```

```

3161 \csdef{forest@def@n@4}#1{\def#1##1##2##3##4}
3162 \csdef{forest@def@n@5}#1{\def#1##1##2##3##4##5}
3163 \csdef{forest@def@n@6}#1{\def#1##1##2##3##4##5##6}
3164 \csdef{forest@def@n@7}#1{\def#1##1##2##3##4##5##6##7}
3165 \csdef{forest@def@n@8}#1{\def#1##1##2##3##4##5##6##7##8}
3166 \csdef{forest@def@n@9}#1{\def#1##1##2##3##4##5##6##7##8##9}

```

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

```

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

```

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

```

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

```

```

3217 \fi
3218 }

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

```

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

```

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

```

```

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

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

```

### 7.1.1 Registers

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

```

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

```

```

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

```



```

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

```

```

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

```

```

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

```

```

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

```

Declare some temporary registers.

```

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

```

## 7.1.2 Declaring options

```

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

```

```

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

```

```

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

```

```

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

```



```

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

```

```

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

```

```

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

```

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

### 7.1.3 Option propagation

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

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

```

```

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

```

```

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

```

## 7.2 Aggregate functions

```

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

```

```

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

```

```

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

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

```



```

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

```

Actual computation.

```

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

```

### 7.2.1 pgfmath extensions

```

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

```

```

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

```

```

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

```

### 7.3 Nodewalk

Setup machinery.

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

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

```

4742 \def\forest@forstepwrapper#1#2{\forest@forthis{\forest@nodewalk{#1}{#2}}}
4743 \def\forest@util@nargs#1#2#3{% #1 = prefix (#, ##, ...), #2 = n args, #3=start; returns {#start+1}...{#start+
4744     \ifnum#2>0 {#1\number\numexpr#3+1}\fi
4745     \ifnum#2>1 {#1\number\numexpr#3+2}\fi
4746     \ifnum#2>2 {#1\number\numexpr#3+3}\fi
4747     \ifnum#2>3 {#1\number\numexpr#3+4}\fi
4748     \ifnum#2>4 {#1\number\numexpr#3+5}\fi
4749     \ifnum#2>5 {#1\number\numexpr#3+6}\fi
4750     \ifnum#2>6 {#1\number\numexpr#3+7}\fi
4751     \ifnum#2>7 {#1\number\numexpr#3+8}\fi
4752     \ifnum#2>8 {#1\number\numexpr#3+9}\fi
4753 }
4754 \def\forest@nodewalk@start@oninvalid@fake#1{}
4755 \def\forest@nodewalk@start@oninvalid@compatfake#1{%
4756     \forest@deprecated{last step in stack "\forest@nodewalk@currentstepname", which started from an invalid nod
4757 }%
4758 \let\forest@nodewalk@start@oninvalid@errorifreal\forest@nodewalk@start@oninvalid@fake % the step will be to a
4759 \let\forest@nodewalk@start@oninvalid@lastvalid\forest@nodewalk@start@oninvalid@fake
4760 \def\forest@nodewalk@start@oninvalid@error#1{\PackageError{forest}{nodewalk step "#1" cannot start at the inv

```

Define long-form single-step walks.

```

4761 \forestset{
4762     define long step={current}{autostep}{},
4763     define long step={next}{autostep}{\edef\forest@cn{\forestove{@next}}},
4764     define long step={previous}{autostep}{\edef\forest@cn{\forestove{@previous}}},

```

```

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

```



```

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

```

```

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

```

```

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

```

```

5009     \pgfkeysalso{#2}%
5010 }%
5011 },
5012 define long step={strip fake steps}{n args=1,@bare}{%
5013     \forest@nodewalk@stripfakesteps{\pgfkeysalso{#1}}},
5014 define long step={unique}{n args=1}{%
5015     \begingroup
5016     \def\forest@nodewalk@unique@temp{%
5017         \forest@nodewalk{#1}{%
5018             TeX={%
5019                 \forest@get{unique@visited}\forest@temp
5020                 \ifx\forest@temp\relax
5021                     \forest@set{unique@visited}{1}%
5022                     \eappto\forest@nodewalk@unique@temp{,id=\forest@cn}%
5023                 \fi
5024             }%
5025         }%
5026         \global\let\forest@global@temp\forest@nodewalk@unique@temp
5027     \endgroup
5028     \pgfkeysalsofrom{\forest@global@temp}%
5029 },
5030 define long step={walk back}{n args=1,@bare}{%
5031     \forest@mathtruncatemacro\forest@temp@n{#1}%
5032     \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\ifnum\forest@cn=0\relax\forest@temp@n-1\fi}%
5033     \forest@nodewalk@back@updatehistory
5034 },
5035 nodewalk/walk back/.default=1,
5036 define long step={jump back}{n args=1,@bare}{%
5037     \forest@mathtruncatemacro\forest@temp@n{(#1)+\ifnum\forest@cn=0\relax0\else1\fi}%
5038     \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\forest@temp@n-1}%
5039     \forest@nodewalk@back@updatehistory
5040 },
5041 nodewalk/jump back/.default=1,
5042 define long step={back}{n args=1,@bare}{%
5043     \forest@mathtruncatemacro\forest@temp@n{#1}%
5044     \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\ifnum\forest@cn=0\relax\forest@temp@n-1\fi}%
5045     \forest@nodewalk@back@updatehistory
5046 },
5047 nodewalk/back/.default=1,
5048 define long step={walk forward}{n args=1,@bare}{%
5049     \forest@mathtruncatemacro\forest@temp@n{#1}%
5050     \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{0}{\forest@temp@n}%
5051     \forest@nodewalk@forward@updatehistory
5052 },
5053 nodewalk/walk forward/.default=1,
5054 define long step={jump forward}{n args=1,@bare}{%
5055     \forest@mathtruncatemacro\forest@temp@n{#1}%
5056     \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{\forest@temp@n}%
5057     \forest@nodewalk@forward@updatehistory
5058 },
5059 nodewalk/jump forward/.default=1,
5060 define long step={forward}{n args=1,@bare}{%
5061     \forest@mathtruncatemacro\forest@temp@n{#1}%
5062     \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{0}{\forest@temp@n}%
5063     \forest@nodewalk@forward@updatehistory
5064 },
5065 nodewalk/forward/.default=1,
5066 define long step={last valid'}{@bare}{%
5067     \ifnum\forest@cn=0
5068         \forest@nodewalk@tolastvalid
5069     \forest@nodewalk@makestep

```

```

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

```

```

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

```

```

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

```

```

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

```

Find minimal/maximal node in a walk.

```

5296 \forestset{
5297   define long step={min}{n args=1,@bare,make for}{% the first min in the argument nodewalk
5298     \forest@nodewalk{#1,TeX={%
5299       \global\let\forest@global@temp\forest@nodewalk@historyback
5300     }}}%
5301     \forest@nodewalk@minmax{\forest@global@temp}{-1}{<}{\forest@nodewalk@minmax@node,}%
5302   },
5303   define long step={mins}{n args=1,@bare,make for}{% all mins in the argument nodewalk
5304     \forest@nodewalk{#1,TeX={%
5305       \global\let\forest@global@temp\forest@nodewalk@historyback
5306     }}}%
5307     \forest@nodewalk@minmax{\forest@global@temp}{-1}{<}{\forest@nodewalk@minmax@nodes}%
5308   },
5309   define long step={walk and min}{n args=1,@bare}{%
5310     \edef\forest@marshal{%
5311       \noexpand\pgfkeysalso{\unexpanded{#1}}%

```



```

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

```

```

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

```

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.

```

5420 \newtoks\forest@nodewalk@shortsteps@resolution
5421 \newif\ifforest@nodewalk@areshortsteps
5422 \pgfqkeys{/forest/nodewalk}{
5423 .unknown/.code={%
5424 \forest@nodewalk@areshortstepsfalse
5425 \ifx\pgfkeyscurrentvalue\pgfkeysnovalue@text % no value, so possibly short steps
5426 \forest@nodewalk@shortsteps@resolution}%
5427 \forest@nodewalk@areshortstepstrue
5428 \expandafter\forest@nodewalk@shortsteps\pgfkeyscurrentname=====,% "=" and "," cannot be short step
5429 \fi

```

```

5430 \ifforest@nodewalk@aeshortsteps
5431 \@escapeif{\expandafter\pgfkeysalso\expandafter{\the\forest@nodewalk@shortsteps@resolution}}%
5432 \else
5433 \@escapeif{\pgfkeysalso{/forest/\pgfkeyscurrentname={#1}}}%
5434 \fi
5435 },
5436 }
5437 \def\forest@nodewalk@shortsteps{%
5438 \futurelet\forest@nodewalk@nexttoken\forest@nodewalk@shortsteps@
5439 }
5440 \def\forest@nodewalk@shortsteps@{%
5441 \ifx\forest@nodewalk@nexttoken=%
5442 \let\forest@nodewalk@nextop\forest@nodewalk@shortsteps@end
5443 \else
5444 \ifx\forest@nodewalk@nexttoken\bgroup
5445 \letcs\forest@nodewalk@nextop{forest@shortstep@group}%
5446 \else
5447 \let\forest@nodewalk@nextop\forest@nodewalk@shortsteps@@
5448 \fi
5449 \fi
5450 \forest@nodewalk@nextop
5451 }
5452 \def\forest@nodewalk@shortsteps@@#1{%
5453 \ifcsdef{forest@shortstep@#1}{%
5454 \csname forest@shortstep@#1\endcsname
5455 }{%
5456 \forest@nodewalk@aeshortstepsfalse
5457 \forest@nodewalk@shortsteps@end
5458 }%
5459 }
5460 % in the following definitions:
5461 % #1 = short step
5462 % #2 = (long) step, or a style in /forest/nodewalk (taking n args)
5463 \csdef{forest@nodewalk@defshortstep@0@args}#1#2{%
5464 \csdef{forest@shortstep@#1}{%
5465 \apptotoks\forest@nodewalk@shortsteps@resolution{,#2}%
5466 \forest@nodewalk@shortsteps}}
5467 \csdef{forest@nodewalk@defshortstep@1@args}#1#2{%
5468 \csdef{forest@shortstep@#1}##1{%
5469 \edef\forest@marshal####1{#2}%
5470 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}}%
5471 \forest@nodewalk@shortsteps}}
5472 \csdef{forest@nodewalk@defshortstep@2@args}#1#2{%
5473 \csdef{forest@shortstep@#1}##1##2{%
5474 \edef\forest@marshal####1####2{#2}%
5475 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}}%
5476 \forest@nodewalk@shortsteps}}
5477 \csdef{forest@nodewalk@defshortstep@3@args}#1#2{%
5478 \csdef{forest@shortstep@#1}##1##2##3{%
5479 \edef\forest@marshal####1####2####3{#2}%
5480 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}}%
5481 \forest@nodewalk@shortsteps}}
5482 \csdef{forest@nodewalk@defshortstep@4@args}#1#2{%
5483 \csdef{forest@shortstep@#1}##1##2##3##4{%
5484 \edef\forest@marshal####1####2####3####4{#2}%
5485 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}}%
5486 \forest@nodewalk@shortsteps}}
5487 \csdef{forest@nodewalk@defshortstep@5@args}#1#2{%
5488 \csdef{forest@shortstep@#1}##1##2##3##4##5{%
5489 \edef\forest@marshal####1####2####3####4####5{#2}%
5490 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}}%

```

```

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

```

```

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

```

Define multiple-step walks.

```

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

```

```

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

```

```

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

```

## 7.4 Dynamic tree

```

5719 \def\forest@last@node{0}
5720 \csdef{forest@nodewalk@saved@dynamic nodes}{-}
5721 \def\forest@nodehandleby@name@nodewalk@or@bracket#1{%
5722 \ifx\pgfkeysnovalue#1%
5723 \edef\forest@last@node{\forest@node@Nametoid{forest@last@node}}%
5724 \else
5725 \forest@nodehandleby@nnb@checkfirst#1\forest@END
5726 \fi

```

```

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

```



```

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

```

```

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

```

```

5910 }%
5911 },
5912 sort'/.code={%
5913 \eapptotoks\forest@do@ynamics{%
5914 \def\noexpand\forest@nodesort@sortkey{\forestrv{sort by}}%
5915 \noexpand\forest@nodesort\noexpand\forest@sort@descending{\forest@cn}
5916 }%
5917 },
5918 }
5919 \def\forest@replaceby@code#1#2{%#1=node spec,#2=insert after['']['']
5920 \ifnum\forestove{@parent}=0
5921 \cseappto{\forest@nodewalk@saved@dynamic nodes}{\forest@cn,}%
5922 \pgfkeysalso{alias=forest@last@node,set root={#1}}%
5923 \else
5924 \cseappto{\forest@nodewalk@saved@dynamic nodes}{\forest@cn,}%
5925 \pgfkeysalso{alias=forest@last@node,#2={#1}}%
5926 \eapptotoks\forest@do@ynamics{%
5927 \noexpand\ifnum\noexpand\forestOve{\forest@cn}{@parent}=\forestove{@parent}
5928 \noexpand\forest@remove@node{\forest@cn}%
5929 \noexpand\fi
5930 }%
5931 \fi
5932 }
5933 \def\forest@nodesort#1#2{% #1 = direction, #2 = parent node
5934 \ifforestdebugdynamics\forestdebug@dynamics{before sorting children of #2}\fi
5935 \forest@for@node{#2}{\forest@nodesort@#1}%
5936 \ifforestdebugdynamics\forestdebug@dynamics{after sorting children of #2}\fi
5937 }
5938 \def\forest@nodesort@#1{%
5939 % prepare the array of child ids
5940 \c@pgf@counta=0
5941 \forest@toget{@first}\forest@nodesort@id
5942 \forest@loop
5943 \ifnum\forest@nodesort@id>0
5944 \advance\c@pgf@counta 1
5945 \csedef{\forest@nodesort@the\c@pgf@counta}{\forest@nodesort@id}%
5946 \forest@get{\forest@nodesort@id}{@next}\forest@nodesort@id
5947 \forest@repeat
5948 % sort
5949 \forest@toget{n children}\forest@nodesort@n
5950 \forest@sort\forest@nodesort@cmpnodes\forest@nodesort@let#1{1}{\forest@nodesort@n}%
5951 % remove all children
5952 \forest@toget{@first}\forest@nodesort@id
5953 \forest@loop
5954 \ifnum\forest@nodesort@id>0
5955 \forest@node@Remove{\forest@nodesort@id}%
5956 \forest@toget{@first}\forest@nodesort@id
5957 \forest@repeat
5958 % insert the children in new order
5959 \c@pgf@counta=0
5960 \forest@loop
5961 \ifnum\c@pgf@counta<\forest@nodesort@n\relax
5962 \advance\c@pgf@counta 1
5963 \forest@node@append{\csname forest@nodesort@the\c@pgf@counta\endcsname}%
5964 \forest@repeat
5965 }
5966 \def\forest@nodesort@cmpnodes#1#2{%
5967 \expandafter\forest@nodesort@cmpnodes@forest@nodesort@sortkey,\forest@END{#1}{#2}%
5968 }
5969 \def\forest@nodesort@cmpnodes@#1,#2\forest@END#3#4{%
5970 % #1=process ins+arg for this dimension, #2=for next dimensions

```

```

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

```

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

```

6000 \def\forest@nodesort@cmp{\csname fRsT@nsc@\forestmathresulttype\endcsname}
6001 \def\fRsT@nsc@#1{\csname fRsT@nsc@#1\endcsname}
6002 \def\fRsT@nsc@n#1#2{\ifnum#1<#2 <\else\ifnum#1=#2 =\else>\fi\fi}
6003 \def\fRsT@nsc@d#1#2{\ifdim#1<#2 <\else\ifdim#1=#2 =\else>\fi\fi}
6004 \def\fRsT@nsc@P#1#2{\ifdim#1pt<#2pt <\else\ifdim#1pt=#2pt =\else>\fi\fi}
6005 \def\fRsT@nsc@t#1#2{\csname fRsT@nsc@\pdfstrcmp{#1}{#2}\endcsname}
6006 \def\fRsT@nsc@T#1#2{\csname fRsT@nsc@\pdfstrcmp{#2}{#1}\endcsname}
6007 \csdef{fRsT@nsc@-1}{<}
6008 \csdef{fRsT@nsc@0}{=}
6009 \csdef{fRsT@nsc@1}{>}
6010 \def\forest@nodesort@let#1#2{%
6011   \csletcs{forest@nodesort@#1}{forest@nodesort@#2}%
6012 }
6013 \forestset{
6014   define long step={last dynamic node}{style,must start at valid node=false}{%
6015     name=forest@last@node
6016   }
6017 }

```

## 8 Stages

```

6018 \def\forest@root{0}
6019 %%% begin listing region: stages
6020 \forestset{
6021   stages/.style={
6022     for root'={
6023       process keylist register=default preamble,
6024       process keylist register=preamble
6025     },

```

```

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

```

```

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

```

```

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

```

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

```

6201 \def\forest@havedelayedoptions#1#2{%#1 = keylist, #2=nodewalk
6202   \forest@havedelayedoptionsfalse
6203   \forest@for@this{%
6204     \forest@nodewalk{#2}{%
6205       TeX={%
6206         \forest@get{delay}\forest@temp@delayed

```

```

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

```

## 8.1 Typesetting nodes

```

6214 \def\forest@node@typeset{%
6215     \let\forest@next\forest@node@typeset@
6216     \forestoifdefined{@box}{%
6217         \forestoget{@box}\forest@temp
6218         \ifdefempty\forest@temp{%
6219             \locbox\forest@temp@box
6220             \forestolet{@box}\forest@temp@box
6221         }{%
6222             \ifforest@drawtree@preservenodeboxes@
6223                 \let\forest@next\relax
6224             \fi
6225         }%
6226     }{%
6227         \locbox\forest@temp@box
6228         \forestolet{@box}\forest@temp@box
6229     }%
6230     \def\forest@node@typeset@restore{%
6231         \ifdefined\ifsa@tikz\forest@standalone@hack\fi
6232         \forest@next
6233     \forest@node@typeset@restore
6234 }
6235 \def\forest@standalone@hack{%
6236     \ifsa@tikz
6237         \let\forest@standalone@tikzpicture\tikzpicture
6238         \let\forest@standalone@endtikzpicture\endtikzpicture
6239         \let\tikzpicture\sa@orig@tikzpicture
6240         \let\endtikzpicture\sa@orig@endtikzpicture
6241         \def\forest@node@typeset@restore{%
6242             \let\tikzpicture\forest@standalone@tikzpicture
6243             \let\endtikzpicture\forest@standalone@endtikzpicture
6244         }%
6245     \fi
6246 }
6247 \newbox\forest@box
6248 \def\forest@pgf@notyetpositioned{not yet positionedPGFINTERNAL}
6249 \def\forest@node@typeset@{%
6250     \forestanchortotikzanchor{anchor}\forest@temp
6251     \edef\forest@marshal{%
6252         \noexpand\forestolet{anchor}\noexpand\forest@temp
6253         \noexpand\forest@node@typeset@@
6254         \noexpand\forestoset{anchor}{\forestov{anchor}}%
6255     }\forest@marshal
6256 }
6257 \def\forest@node@typeset@@{%
6258     \forestoget{name}\forest@nodename
6259     \edef\forest@temp@nodeformat{\forestove{node format}}%
6260     \gdef\forest@smuggle{%
6261         \setbox0=\hbox{%
6262             \begin{tikzpicture}[%
6263                 /forest/copy command key={/tikz/anchor}{/tikz/forest@orig@anchor},
6264                 anchor/.style={%
6265                     /forest/TeX={\forestanchortotikzanchor{##1}\forest@temp@anchor},

```



```

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

```

```

6327 \ifforest@drawtree@preservenodeboxes@
6328 \else
6329 \forestset{@box}{}%
6330 \fi
6331 }

```

## 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.

```

6332 \def\forest@pack{%
6333 \pgfsyssoftpath@getcurrentpath\forest@pack@original@path
6334 \forest@pack@computetiers
6335 \forest@pack@computegrowthuniformity
6336 \forest@@pack
6337 \pgfsyssoftpath@setcurrentpath\forest@pack@original@path
6338 }
6339 \def\forest@@pack{%
6340 \ifnum\forestove{uniform growth}>0
6341 \ifnum\forestove{n children}>0
6342 \forest@pack@level@uniform
6343 \forest@pack@aligntiers@ofsubtree
6344 \forest@pack@sibling@uniform@recursive
6345 \fi
6346 \else
6347 \forest@node@foreachchild{\forest@@pack}%
6348 \forest@process@hook@keylist@nodynamics{before packing node}{current}%
6349 \ifnum\forestove{n children}>0
6350 \forest@pack@level@nonuniform
6351 \forest@pack@aligntiers
6352 \forest@pack@sibling@uniform@applyreversed
6353 \fi
6354 \forestoget{after packing node}\forest@temp@keys
6355 \forest@process@hook@keylist@nodynamics{after packing node}{current}%
6356 \fi
6357 }
6358 % \forestset{recalculate tree boundary/.code={\forest@node@recalculate@edges}}
6359 % \def\forest@node@recalculate@edges{%
6360 % \edef\forest@marshal{%
6361 % \noexpand\forest@forthis{\noexpand\forest@node@getedges{\forestove{grow}}}%
6362 % }\forest@marshal
6363 % }
6364 \def\forest@pack@onlythisnode{%
6365 \ifnum\forestove{n children}>0
6366 \forest@pack@computetiers
6367 \forest@pack@level@nonuniform
6368 \forest@pack@aligntiers
6369 \forest@node@foreachchild{\forestoset{s}{0\pgfm@h@pt}}%
6370 \forest@pack@sibling@uniform@applyreversed
6371 \fi
6372 }

```

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

```

6373 \def\forest@pack@computegrowthuniformity{%
6374 \forest@node@foreachchild{\forest@pack@computegrowthuniformity}%
6375 \edef\forest@pack@cgu@uniformity{%
6376 \ifnum\forestove{n children}=0
6377 2\else 1\fi
6378 }%
6379 \forestoget{grow}\forest@pack@cgu@parentgrow

```

```

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

```

Pack children in the level dimension in a uniform tree.

```

6399 \def\forest@pack@level@uniform{%
6400   \let\forest@plu@minchildl\relax
6401   \forestoget{grow}\forest@plu@grow
6402   \forest@node@foreachchild{%
6403     \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
6404     \advance\pgf@xa\forestove{1}\relax
6405     \ifx\forest@plu@minchildl\relax
6406       \edef\forest@plu@minchildl{\the\pgf@xa}%
6407     \else
6408       \ifdim\pgf@xa<\forest@plu@minchildl\relax
6409         \edef\forest@plu@minchildl{\the\pgf@xa}%
6410       \fi
6411     \fi
6412   }%
6413   \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
6414   \pgfutil@tempdima=\pgf@xb\relax
6415   \advance\pgfutil@tempdima -\forest@plu@minchildl\relax
6416   \advance\pgfutil@tempdima \forestove{1 sep}\relax
6417   \ifdim\pgfutil@tempdima>0pt
6418     \forest@node@foreachchild{%
6419       \forestoeset{1}{\the\dimexpr\forestove{1}+\the\pgfutil@tempdima}%
6420     }%
6421   \fi
6422   \forest@node@foreachchild{%
6423     \ifnum\forestove{n children}>0
6424       \forest@pack@level@uniform
6425     \fi
6426   }%
6427 }

```

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

```

6428 \def\forest@pack@level@nonuniform{%
6429   \let\forest@plu@minchildl\relax
6430   \forestoget{grow}\forest@plu@grow
6431   \forest@node@foreachchild{%
6432     \forest@node@getedge[negative]{\forest@plu@grow}{\forest@plnu@negativechilddedge}%
6433     \forest@node@getedge[positive]{\forest@plu@grow}{\forest@plnu@positivechilddedge}%
6434     \def\forest@plnu@childdedge{\forest@plnu@negativechilddedge\forest@plnu@positivechilddedge}%
6435     \forest@path@getboundingrectangle@ls\forest@plnu@childdedge{\forest@plu@grow}%
6436     \advance\pgf@xa\forestove{1}\relax
6437     \ifx\forest@plu@minchildl\relax

```

```

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

Align tiers.
6455 \def\forest@pack@aligntiers{%
6456     \forestoget{grow}\forest@temp@parentgrow
6457     \forestoget{@tiers}\forest@temp@tiers
6458     \forlistloop\forest@pack@aligntier@\forest@temp@tiers
6459 }
6460 \def\forest@pack@aligntiers@ofsubtree{%
6461     \forest@node@foreach{\forest@pack@aligntiers}%
6462 }
6463 \def\forest@pack@aligntiers@computeabs1{%
6464     \forestoleto{abs@1}{1}%
6465     \forest@node@foreachdescendant{\forest@pack@aligntiers@computeabs1}%
6466 }
6467 \def\forest@pack@aligntiers@computeabs1@{%
6468     \forestoeset{abs@1}{\the\dimexpr\forestove{1}+\forestove{\forestove{@parent}}{abs@1}}%
6469 }
6470 \def\forest@pack@aligntier@#1{%
6471     \forest@pack@aligntiers@computeabs1
6472     \pgfutil@tempdima=-\maxdimen\relax
6473     \def\forest@temp@currenttier{#1}%
6474     \forest@node@foreach{%
6475         \forestoget{tier}\forest@temp@tier
6476         \ifx\forest@temp@currenttier\forest@temp@tier
6477             \ifdim\pgfutil@tempdima<\forestove{abs@1}\relax
6478                 \pgfutil@tempdima=\forestove{abs@1}\relax
6479             \fi
6480         \fi
6481     }%
6482     \ifdim\pgfutil@tempdima=-\maxdimen\relax\else
6483         \forest@node@foreach{%
6484             \forestoget{tier}\forest@temp@tier
6485             \ifx\forest@temp@currenttier\forest@temp@tier
6486                 \forestoeset{1}{\the\dimexpr\pgfutil@tempdima-\forestove{abs@1}+\forestove{1}}%
6487             \fi
6488         }%
6489     \fi
6490 }

```

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

```

6491 \def\forest@pack@sibling@uniform@recursive{%
6492     \forest@node@foreachchild{\forest@pack@sibling@uniform@recursive}%
6493     \forest@pack@sibling@uniform@applyreversed
6494 }

```

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

```

6495 \def\forest@pack@sibling@uniform@applyreversed{%
6496   \ifnum\forestove{n children}>1
6497     \ifnum\forestove{reversed}=0
6498       \forest@pack@sibling@uniform@main{first}{last}{next}{previous}%
6499     \else
6500       \forest@pack@sibling@uniform@main{last}{first}{previous}{next}%
6501     \fi
6502   \else
6503     \ifnum\forestove{n children}=1

```

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

```

6504     \csname forest@calign@\forestove{calign}\endcsname
6505   \fi
6506 \fi
6507 }

```

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

```

6508 \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.

```

6509   \forestoget{#1}\forest@child
6510   \edef\forest@marshal{%
6511     \noexpand\forest@fornode{\forestove{#1}}{%
6512       \noexpand\forest@node@getedge
6513         {positive}%
6514         {\forestove{grow}}}%
6515     \noexpand\forest@temp@edge
6516   }%
6517   }\forest@marshal
6518   \forest@pack@pgfpoint@childposition\forest@child
6519   \let\forest@previous@positive@edge\pgfutil@empty
6520   \forest@extendpath\forest@previous@positive@edge\forest@temp@edge{}%
6521   \forestoget{\forest@child}{#3}\forest@child

```

Loop until the current child is the null node.

```

6522   \edef\forest@previous@child@s{0\pgfmath@pt}%
6523   \safeloop
6524   \unless\ifnum\forest@child=0

```

Get the negative edge of the child.

```

6525     \edef\forest@temp{%
6526       \noexpand\forest@fornode{\forest@child}{%
6527         \noexpand\forest@node@getedge
6528           {negative}%
6529           {\forestove{grow}}}%
6530       \noexpand\forest@temp@edge
6531     }%
6532     }\forest@temp

```

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

```

6533     \forest@pack@pgfpoint@childposition\forest@child

```

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

```

6534     \let\forest@child@negative@edge\pgfutil@empty
6535     \forest@extendpath\forest@child@negative@edge\forest@temp@edge{}%

```

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

```

6536     \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!)

```
6537 \forest@distance@between@edge@paths\forest@previous@positive@edge\forest@child@negative@edge\forest@c
```

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`.

```
6538 \ifx\forest@csdistance\relax
6539 %\forest@reset{\forest@child}{s}{\forest@previous@child@s}%
6540 \else
6541 \advance\pgfutil@tempdimb-\forest@csdistance\relax
6542 \advance\pgfutil@tempdimb\forestove{s_sep}\relax
6543 \forest@reset{\forest@child}{s}{\the\dimexpr\forestove{\forest@child}{s}-\forest@csdistance+\forestove{s}
6544 \fi
```

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

```
6545 \ifdim\forestove{\forest@child}{s}<\forest@previous@child@s\relax
6546 \forest@reset{\forest@child}{s}{\forest@previous@child@s}%
6547 \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.

```
6548 \forest@get{\forest@child}{s}\forest@child@s
6549 \edef\forest@previous@child@s{\forest@child@s}%
6550 \edef\forest@temp{%
6551 \noexpand\forest@for@node{\forest@child}{%
6552 \noexpand\forest@node@get@edge
6553 {positive}%
6554 {\forestove{grow}}}%
6555 \noexpand\forest@temp@edge
6556 }%
6557 }\forest@temp
6558 \forest@pack@pgfpoint@child@position\forest@child
6559 \forest@extend@path\forest@previous@positive@edge\forest@temp@edge{}%
6560 \forest@get@positive@tight@edge@of@path\forest@previous@positive@edge\forest@previous@positive@edge
6561 \forest@get{\forest@child}{#3}\forest@child
6562 \saferepeat
```

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

```
6563 \csname forest@calign\forestove{calign}\endcsname
6564 }
```

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

```
6565 \def\forest@pack@pgfpoint@child@position#1{%
6566 {%
6567 \pgftransformreset
6568 \forest@pgfqtransformrotate{\forestove{grow}}%
6569 \forest@for@node{#1}{%
6570 \pgfpointtransformed{\pgfqpoint{\forestove{1}}{\forestove{s}}}%
6571 }%
6572 }%
6573 }
```

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

```
6574 \def\forest@pack@pgfpoint@positioningrow#1{%
6575 {%
6576 \pgftransformreset
6577 \forest@pgfqtransformrotate{#1}%
6578 \pgfpointtransformed{\pgfqpoint{\forestove{1}}{\forestove{s}}}%
6579 }%
6580 }
```

Child alignment.

```
6581 \def\forest@calign@s@shift#1{%
```

```

6582 \pgfutil@tempdima=#1\relax
6583 \forest@node@foreachchild{%
6584   \forestoeset{s}{\the\dimexpr\forestove{s}+\pgfutil@tempdima}%
6585 }%
6586 }
6587 \def\forest@calign@child{%
6588   \forest@calign@s@shift{-\forestOve{\forest@node@nornbarthchildid{\forestove{calign primary child}}}{s}}%
6589 }
6590 \csdef{forest@calign@child edge}{%
6591   {%
6592     \edef\forest@temp@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
6593     \pgftransformreset
6594     \forest@pgfqtransformrotate{\forestove{grow}}%
6595     \pgfpointrtransformed{\pgfqpoint{\forestOve{\forest@temp@child}{1}}{\forestOve{\forest@temp@child}{s}}}%
6596     \pgf@xa=\pgf@x\relax\pgf@ya=\pgf@y\relax
6597     \forest@Pointanchor{\forest@temp@child}{child anchor}%
6598     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
6599     \forest@pointanchor{parent anchor}%
6600     \advance\pgf@xa-\pgf@x\relax\advance\pgf@ya-\pgf@y\relax
6601     \edef\forest@marshal{%
6602       \noexpand\pgftransformreset
6603       \noexpand\forest@pgfqtransformrotate{-\forestove{grow}}%
6604       \noexpand\pgfpointrtransformed{\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}%
6605     }\forest@marshal
6606   }%
6607   \forest@calign@s@shift{\the\dimexpr-\the\pgf@y}%
6608 }
6609 \csdef{forest@calign@midpoint}{%
6610   \forest@calign@s@shift{\the\dimexpr Opt -%
6611     (\forestOve{\forest@node@nornbarthchildid{\forestove{calign primary child}}}{s}%
6612     +\forestOve{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}{s}%
6613     )/2\relax
6614   }%
6615 }
6616 \csdef{forest@calign@edge midpoint}{%
6617   {%
6618     \edef\forest@temp@firstchild{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
6619     \edef\forest@temp@secondchild{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
6620     \pgftransformreset
6621     \forest@pgfqtransformrotate{\forestove{grow}}%
6622     \pgfpointrtransformed{\pgfqpoint{\forestOve{\forest@temp@firstchild}{1}}{\forestOve{\forest@temp@firstchild}{s}}}%
6623     \pgf@xa=\pgf@x\relax\pgf@ya=\pgf@y\relax
6624     \forest@Pointanchor{\forest@temp@firstchild}{child anchor}%
6625     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
6626     \edef\forest@marshal{%
6627       \noexpand\pgfpointrtransformed{\noexpand\pgfqpoint{\forestOve{\forest@temp@secondchild}{1}}{\forestOve{\forest@temp@secondchild}{s}}}%
6628     }\forest@marshal
6629     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
6630     \forest@Pointanchor{\forest@temp@secondchild}{child anchor}%
6631     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
6632     \divide\pgf@xa2 \divide\pgf@ya2
6633     \forest@pointanchor{parent anchor}%
6634     \advance\pgf@xa-\pgf@x\relax\advance\pgf@ya-\pgf@y\relax
6635     \edef\forest@marshal{%
6636       \noexpand\pgftransformreset
6637       \noexpand\forest@pgfqtransformrotate{-\forestove{grow}}%
6638       \noexpand\pgfpointrtransformed{\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}%
6639     }\forest@marshal
6640   }%
6641   \forest@calign@s@shift{\the\dimexpr-\the\pgf@y}%
6642 }

```

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.

```

6643 \def\forest@edef@strippt#1#2{%
6644   \edef#1{#2}%
6645   \edef#1{\expandafter\Pgf@geT#1}%
6646 }
6647 \csdef{forest@calign@fixed angles}{%
6648   \ifnum\forest@ve{n children}>1
6649     \edef\forest@ca@first@child{\forest@node@nornbarthchildid{\forest@ve{calign primary child}}}%
6650     \edef\forest@ca@second@child{\forest@node@nornbarthchildid{\forest@ve{calign secondary child}}}%
6651     \ifnum\forest@ve{reversed}=1
6652       \let\forest@temp\forest@ca@first@child
6653       \let\forest@ca@first@child\forest@ca@second@child
6654       \let\forest@ca@second@child\forest@temp
6655     \fi
6656     \forest@get{\forest@ca@first@child}{l}\forest@ca@first@l
6657     \edef\forest@ca@first@l{\expandafter\Pgf@geT\forest@ca@first@l}%
6658     \forest@get{\forest@ca@second@child}{l}\forest@ca@second@l
6659     \edef\forest@ca@second@l{\expandafter\Pgf@geT\forest@ca@second@l}%
6660     \pgfmathatan@{\forest@ve{calign secondary angle}}%
6661     \pgfmathmultiply@{\pgfmathresult}{\forest@ca@second@l}%
6662     \let\forest@calign@temp\pgfmathresult
6663     \pgfmathatan@{\forest@ve{calign primary angle}}%
6664     \pgfmathmultiply@{\pgfmathresult}{\forest@ca@first@l}%
6665     \edef\forest@ca@desired@s@distance{\the\dimexpr
6666       \forest@calign@temp pt-\pgfmathresult pt}%
6667     % \pgfmathsetlengthmacro\forest@ca@desired@s@distance{%
6668     %   tan(\forest@ve{calign secondary angle})*\forest@ca@second@l
6669     %   -tan(\forest@ve{calign primary angle})*\forest@ca@first@l
6670     % }%
6671     \forest@get{\forest@ca@first@child}{s}\forest@ca@first@s
6672     \forest@get{\forest@ca@second@child}{s}\forest@ca@second@s
6673     \edef\forest@ca@actual@s@distance{\the\dimexpr
6674       \forest@ca@second@s-\forest@ca@first@s}%
6675     % \pgfmathsetlengthmacro\forest@ca@actual@s@distance{%
6676     %   \forest@ca@second@s-\forest@ca@first@s}%
6677     \ifdim\forest@ca@desired@s@distance>\forest@ca@actual@s@distance\relax
6678       \ifdim\forest@ca@actual@s@distance=0pt
6679         \pgfmathatan@{\forest@ve{calign primary angle}}%
6680         \pgfmathmultiply@{\pgfmathresult}{\forest@ca@second@l}%
6681         \pgfutil@tempdima=\pgfmathresult pt
6682         % \pgfmathsetlength\pgfutil@tempdima{tan(\forest@ve{calign primary angle})*\forest@ca@second@l}%
6683         \pgfutil@tempdimb=\dimexpr
6684         \forest@ca@desired@s@distance/(\forest@ve{n children}-1)\relax%
6685         %\pgfmathsetlength\pgfutil@tempdimb{\forest@ca@desired@s@distance/(\forest@ve{n children}-1)}%
6686         \forest@node@foreachchild{%
6687           \forest@set{s}{\the\pgfutil@tempdima}%
6688           \advance\pgfutil@tempdima\pgfutil@tempdimb
6689         }%
6690         \def\forest@calign@anchor{0pt}%
6691       \else
6692         \edef\forest@marshal{\noexpand\pgfmathdivide@
6693           {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
6694           {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
6695         }\forest@marshal
6696         \let\forest@ca@ratio\pgfmathresult
6697         %\pgfmathsetmacro\forest@ca@ratio{%
6698         %   \forest@ca@desired@s@distance/\forest@ca@actual@s@distance}%

```



```

6699 \forest@node@foreachchild{%
6700 \forest@edef@strippt\forest@temp{\forestove{s}}%
6701 \pgfmathmultiply@{\forest@ca@ratio}{\forest@temp}%
6702 \forestoaset{s}{\the\dimexpr\pgfmathresult pt}%
6703 %\pgfmathsetlengthmacro\forest@temp{\forest@ca@ratio*\forestove{s}}%
6704 %\forestolet{s}\forest@temp
6705 }%
6706 \pgfmathatan@{\forestove{calign primary angle}}%
6707 \pgfmathmultiply@{\pgfmathresult}{\forest@ca@first@l}%
6708 \edef\forest@calign@anchor{\the\dimexpr-\pgfmathresult pt}%
6709 %\pgfmathsetlengthmacro\forest@calign@anchor{%
6710 % -tan(\forestove{calign primary angle})*\forest@ca@first@l}%
6711 \fi
6712 \else
6713 \ifdim\forest@ca@desired@s@distance<\forest@ca@actual@s@distance\relax
6714 \edef\forest@marshal{\noexpand\pgfmathdivide@
6715 {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
6716 {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
6717 }\forest@marshal
6718 \let\forest@ca@ratio\pgfmathresult
6719 %\pgfmathsetlengthmacro\forest@ca@ratio{%
6720 % \forest@ca@actual@s@distance/\forest@ca@desired@s@distance}%
6721 \forest@node@foreachchild{%
6722 \forest@edef@strippt\forest@temp{\forestove{l}}%
6723 \pgfmathmultiply@{\forest@ca@ratio}{\forest@temp}%
6724 \forestoaset{l}{\the\dimexpr\pgfmathresult pt}%
6725 %\pgfmathsetlengthmacro\forest@temp{\forest@ca@ratio*\forestove{l}}%
6726 %\forestolet{l}\forest@temp
6727 }%
6728 \forest@get{\forest@ca@first@child}{l}\forest@ca@first@l
6729 \edef\forest@ca@first@l{\expandafter\Pgf@geT\forest@ca@first@l}%
6730 \pgfmathatan@{\forestove{calign primary angle}}%
6731 \pgfmathmultiply@{\pgfmathresult}{\forest@ca@first@l}%
6732 \edef\forest@calign@anchor{\the\dimexpr-\pgfmathresult pt}%
6733 %\pgfmathsetlengthmacro\forest@calign@anchor{%
6734 % -tan(\forestove{calign primary angle})*\forest@ca@first@l}%
6735 \fi
6736 \fi
6737 \forest@calign@s@shift{-\forest@calign@anchor}%
6738 \fi
6739 }
6740 \csdef{forest@calign@fixed edge angles}{%
6741 \ifnum\forestove{n children}>1
6742 \edef\forest@ca@first@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
6743 \edef\forest@ca@second@child{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
6744 \ifnum\forestove{reversed}=1
6745 \let\forest@temp\forest@ca@first@child
6746 \let\forest@ca@first@child\forest@ca@second@child
6747 \let\forest@ca@second@child\forest@temp
6748 \fi
6749 \forest@get{\forest@ca@first@child}{l}\forest@ca@first@l
6750 \forest@get{\forest@ca@second@child}{l}\forest@ca@second@l
6751 \forest@pointanchor{parent anchor}%
6752 \edef\forest@ca@parent@anchor@s{\the\pgf@x}%
6753 \edef\forest@ca@parent@anchor@l{\the\pgf@y}%
6754 \forest@Pointanchor{\forest@ca@first@child}{child anchor}%
6755 \edef\forest@ca@first@child@anchor@s{\the\pgf@x}%
6756 \edef\forest@ca@first@child@anchor@l{\the\pgf@y}%
6757 \forest@Pointanchor{\forest@ca@second@child}{child anchor}%
6758 \edef\forest@ca@second@child@anchor@s{\the\pgf@x}%
6759 \edef\forest@ca@second@child@anchor@l{\the\pgf@y}%

```

```

6760 \pgfmathatan@{\forestove{calign secondary angle}}%
6761 \edef\forest@temp{\the\dimexpr
6762 \forest@ca@second@l-\forest@ca@second@child@anchor@l+\forest@ca@parent@anchor@l}%
6763 \pgfmathmultiply@{\pgfmathresult}{\expandafter\Pgf@geT\forest@temp}%
6764 \edef\forest@ca@desired@second@edge@s{\the\dimexpr\pgfmathresult pt}%
6765 %\pgfmathsetlengthmacro\forest@ca@desired@second@edge@s{%
6766 % tan(\forestove{calign secondary angle})*%
6767 % (\forest@ca@second@l-\forest@ca@second@child@anchor@l+\forest@ca@parent@anchor@l)}%
6768 \pgfmathatan@{\forestove{calign primary angle}}%
6769 \edef\forest@temp{\the\dimexpr
6770 \forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@l}%
6771 \pgfmathmultiply@{\pgfmathresult}{\expandafter\Pgf@geT\forest@temp}%
6772 \edef\forest@ca@desired@first@edge@s{\the\dimexpr\pgfmathresult pt}%
6773 %\pgfmathsetlengthmacro\forest@ca@desired@first@edge@s{%
6774 % tan(\forestove{calign primary angle})*%
6775 % (\forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@l)}%
6776 \edef\forest@ca@desired@s@distance{\the\dimexpr
6777 \forest@ca@desired@second@edge@s-\forest@ca@desired@first@edge@s}%
6778 %\pgfmathsetlengthmacro\forest@ca@desired@s@distance{\forest@ca@desired@second@edge@s-\forest@ca@desired@
6779 \forest@lget{\forest@ca@first@child}{s}\forest@ca@first@s
6780 \forest@lget{\forest@ca@second@child}{s}\forest@ca@second@s
6781 \edef\forest@ca@actual@s@distance{\the\dimexpr
6782 \forest@ca@second@s+\forest@ca@second@child@anchor@s
6783 -\forest@ca@first@s-\forest@ca@first@child@anchor@s}%
6784 %\pgfmathsetlengthmacro\forest@ca@actual@s@distance{%
6785 % \forest@ca@second@s+\forest@ca@second@child@anchor@s
6786 % -\forest@ca@first@s-\forest@ca@first@child@anchor@s}%
6787 \ifdim\forest@ca@desired@s@distance>\forest@ca@actual@s@distance\relax
6788 \ifdim\forest@ca@actual@s@distance=0pt
6789 \forest@toget{n children}\forest@temp@n@children
6790 \forest@node@foreachchild{%
6791 \forest@pointanchor{child anchor}%
6792 \edef\forest@temp@child@anchor@s{\the\pgf@x}%
6793 \forest@oaset{s}{\the\dimexpr
6794 \forest@ca@desired@first@edge@s+\forest@ca@desired@s@distance*(\forestove{n}-1)/(\forest@temp@n@
6795 %\pgfmathsetlengthmacro\forest@temp{%
6796 % \forest@ca@desired@first@edge@s+(\forestove{n}-1)*\forest@ca@desired@s@distance/(\forest@temp@n@
6797 %\forest@oaset{s}\forest@temp
6798 }%
6799 \def\forest@calign@anchor{0pt}%
6800 \else
6801 \edef\forest@marshal{\noexpand\pgfmathdivide@
6802 {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
6803 {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
6804 }\forest@marshal
6805 \let\forest@ca@ratio\pgfmathresult
6806 %\pgfmathsetmacro\forest@ca@ratio{%
6807 % \forest@ca@desired@s@distance/\forest@ca@actual@s@distance}%
6808 \forest@node@foreachchild{%
6809 \forest@pointanchor{child anchor}%
6810 \edef\forest@temp@child@anchor@s{\the\pgf@x}%
6811 \edef\forest@marshal{\noexpand\pgfmathmultiply@
6812 {\forest@ca@ratio}%
6813 {\expandafter\Pgf@geT\the\dimexpr
6814 \forestove{s}-\forest@ca@first@s+
6815 \forest@temp@child@anchor@s-\forest@ca@first@child@anchor@s}%
6816 }\forest@marshal
6817 \forest@oaset{s}{\the\dimexpr\pgfmathresult pt+\forest@ca@first@s
6818 +\forest@ca@first@child@anchor@s-\forest@temp@child@anchor@s}%
6819 %\pgfmathsetlengthmacro\forest@temp{%
6820 % \forest@ca@ratio*(%

```

```

6821 % \forestove{s}-\forest@ca@first@s
6822 % +\forest@temp@child@anchor@s-\forest@ca@first@child@anchor@s}%
6823 % +\forest@ca@first@s
6824 % +\forest@ca@first@child@anchor@s-\forest@temp@child@anchor@s}%
6825 % \forestolet{s}\forest@temp
6826 }%
6827 \pgfmathatan@{\forestove{calign primary angle}}%
6828 \edef\forest@marshal{\noexpand\pgfmathmultiply@
6829 {\pgfmathresult}%
6830 {\expandafter\Pgf@geT\the\dimexpr
6831 \forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@l}%
6832 }\forest@marshal
6833 \edef\forest@calign@anchor{\the\dimexpr
6834 -\pgfmathresult pt+\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s}%
6835 % \pgfmathsetlengthmacro\forest@calign@anchor{%
6836 % -tan(\forestove{calign primary angle})*(\forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@s)
6837 % +\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s
6838 % }%
6839 \fi
6840 \else
6841 \ifdim\forest@ca@desired@s@distance<\forest@ca@actual@s@distance\relax
6842 \edef\forest@marshal{\noexpand\pgfmathdivide@
6843 {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
6844 {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
6845 }\forest@marshal
6846 \let\forest@ca@ratio\pgfmathresult
6847 %\pgfmathsetlengthmacro\forest@ca@ratio{%
6848 % \forest@ca@actual@s@distance/\forest@ca@desired@s@distance}%
6849 \forest@node@foreachchild{%
6850 \forest@pointanchor{child anchor}%
6851 \edef\forest@temp@child@anchor@l{\the\pgf@y}%
6852 \edef\forest@marshal{\noexpand\pgfmathmultiply@
6853 {\forest@ca@ratio}%
6854 {\expandafter\Pgf@geT\the\dimexpr\forestove{l}+\forest@ca@parent@anchor@l-\forest@temp@child@anchor@l}%
6855 }\forest@marshal
6856 \forestoeset{l}{\the\dimexpr
6857 \pgfmathresult pt-\forest@ca@parent@anchor@l+\forest@temp@child@anchor@l}%
6858 % \pgfmathsetlengthmacro\forest@temp@l{%
6859 % \forest@ca@ratio*(%
6860 % \forestove{l}+\forest@ca@parent@anchor@l-\forest@temp@child@anchor@l)
6861 % -\forest@ca@parent@anchor@l+\forest@temp@child@anchor@l}%
6862 % \forestolet{l}\forest@temp
6863 }%
6864 \forest@get{\forest@ca@first@child}{l}\forest@ca@first@l
6865 \pgfmathatan@{\forestove{calign primary angle}}%
6866 \edef\forest@marshal{\noexpand\pgfmathmultiply@
6867 {\pgfmathresult}%
6868 {\expandafter\Pgf@geT\the\dimexpr
6869 \forest@ca@first@l+\forest@ca@parent@anchor@l-\forest@temp@child@anchor@l}%
6870 }\forest@marshal
6871 \edef\forest@calign@anchor{\the\dimexpr
6872 -\pgfmathresult pt+\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s}%
6873 % \pgfmathsetlengthmacro\forest@calign@anchor{%
6874 % -tan(\forestove{calign primary angle})*(\forest@ca@first@l+\forest@ca@parent@anchor@l-\forest@temp@child@anchor@l)
6875 % +\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s
6876 % }%
6877 \fi
6878 \fi
6879 \forest@calign@s@shift{-\forest@calign@anchor}%
6880 \fi
6881 }

```

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.

```

6882 \def\forest@node@getedge#1#2#3{%
6883   \forestoget{#1@edge@#2}#3%
6884   \ifx#3\relax
6885     \forest@node@foreachchild{%
6886       \forest@node@getedge{#1}{#2}{\forest@temp@edge}%
6887     }%
6888     \forest@forthis{\forest@node@getedges{#2}}%
6889     \forestoget{#1@edge@#2}#3%
6890   \fi
6891 }

```

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.

```

6892 \def\forest@node@getedges#1{%

```

Run the computation in a  $\TeX$  group.

```

6893   %{}%

```

Setup the grow line.

```

6894     \forest@setupgrowline{#1}%

```

Get the edge of the node itself.

```

6895     \ifnum\forestove{ignore}=0
6896       \forestoget{@boundary}\forest@node@boundary
6897     \else
6898       \def\forest@node@boundary{}%
6899     \fi
6900     \csname forest@getboth\forestove{fit}edgesofpath\endcsname
6901     \forest@node@boundary\forest@negative@node@edge\forest@positive@node@edge
6902     \forestolet{negative@edge@#1}\forest@negative@node@edge
6903     \forestolet{positive@edge@#1}\forest@positive@node@edge

```

Add the edges of the children.

```

6904     \forest@get@edges@merge{negative}{#1}%
6905     \forest@get@edges@merge{positive}{#1}%
6906   %}%
6907 }

```

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

```

6908 \def\forest@get@edges@merge#1#2{%
6909   \ifnum\forestove{n children}>0
6910     \forestoget{#1@edge@#2}\forest@node@edge

```

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

```

6911     \forest@pointanchor{parent anchor}%
6912     \edef\forest@getedge@pa@l{\the\pgf@x}%
6913     \edef\forest@getedge@pa@s{\the\pgf@y}%
6914     \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).

```

6915     \pgfgettransform\forest@temp@transform
6916     \pgftransformreset
6917     \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.

```

6918     \def\forest@all@edges{}%

```

```

6919 \forest@node@foreachchild{%
6920   \forest@toget{#1@edge@#2}\forest@temp@edge
6921   \pgfpointtransformed{\pgfpoint{\forest@ve{1}}{\forest@ve{s}}}%
6922   \forest@extendpath\forest@node@edge\forest@temp@edge}%
6923   \ifnum\forest@ve{ignore edge}=0
6924     \pgfpointadd
6925     {\pgfpointtransformed{\pgfpoint{\forest@ve{1}}{\forest@ve{s}}}}%
6926     {\forest@pointanchor{child anchor}}%
6927     \pgfgetlastxy{\forest@getedge@ca@1}\forest@getedge@ca@s}%
6928     \eappto\forest@all@edges{%
6929       \noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@pa@1}\forest@getedge@pa@s}%
6930       \noexpand\pgfsyssoftpath@linetotoken{\forest@getedge@ca@1}\forest@getedge@ca@s}%
6931     }%
6932     % this deals with potential overlap of the edges:
6933     \eappto\forest@node@edge{\noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@ca@1}\forest@getedge@ca@s}%
6934   \fi
6935 }%
6936 \ifdefempty{\forest@all@edges}{}%
6937   \pgfintersectionofpaths{\pgfsetpath\forest@all@edges}{\pgfsetpath\forest@node@edge}%
6938   \def\forest@edgenode@intersections{%
6939     \forest@merge@intersectionloop
6940     \eappto\forest@node@edge{\expandonce{\forest@all@edges}\expandonce{\forest@edgenode@intersections}}%
6941   }%
6942   \pgfsettransform\forest@temp@transform

```

Process the path into an edge and store the edge.

```

6943   \csname forest@get#1\forest@ve{fit}edgeofpath\endcsname\forest@node@edge\forest@node@edge
6944   \forest@tolet{#1@edge@#2}\forest@node@edge
6945 \fi
6946 }
6947 %\newloop\forest@merge@loop
6948 \def\forest@merge@intersectionloop{%
6949   \c@pgf@counta=0
6950   \forest@loop
6951   \ifnum\c@pgf@counta<\pgfintersectionofsolutions\relax
6952     \advance\c@pgf@counta1
6953     \pgfpointintersectionsolution{\the\c@pgf@counta}%
6954     \eappto\forest@edgenode@intersections{\noexpand\pgfsyssoftpath@movetotoken
6955       {\the\pgf@x}{\the\pgf@y}}%
6956     \forest@repeat
6957 }

```

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

```

6958 \def\forest@node@getboundingrectangle@ls#1{%
6959   \forest@toget{@boundary}\forest@node@boundary
6960   \forest@path@getboundingrectangle@ls\forest@node@boundary{#1}%
6961 }

```

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).

```

6962 \def\forest@pgfpathtransformed#1{%
6963   \forest@save@pgfsyssoftpath@tokendefs
6964   \let\pgfsyssoftpath@movetotoken\forest@pgfpathtransformed@moveto
6965   \let\pgfsyssoftpath@linetotoken\forest@pgfpathtransformed@lineto
6966   \pgfsyssoftpath@setcurrentpath\pgfutil@empty
6967   #1%
6968   \forest@restore@pgfsyssoftpath@tokendefs
6969 }
6970 \def\forest@pgfpathtransformed@moveto#1#2{%
6971   \forest@pgfpathtransformed@op\pgfsyssoftpath@moveto{#1}{#2}%
6972 }
6973 \def\forest@pgfpathtransformed@lineto#1#2{%

```

```

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

```

### 8.2.1 Tiers

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

```

6983 \def\forest@pack@computetiers{%
6984 {%
6985 \forest@pack@tiers@getalltiersinsubtree
6986 \forest@pack@tiers@computetierhierarchy
6987 \forest@pack@tiers@findcontainers
6988 \forest@pack@tiers@raisecontainers
6989 \forest@pack@tiers@computeprocessingorder
6990 \gdef\forest@smuggle{%
6991 \forest@pack@tiers@write
6992 }%
6993 \forest@node@foreach{\forestoset{@tiers}{}}%
6994 \forest@smuggle
6995 }

```

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

```

6996 \def\forest@pack@tiers@getalltiersinsubtree{%
6997 \ifnum\forestove{n children}>0
6998 \forest@node@foreachchild{\forest@pack@tiers@getalltiersinsubtree}%
6999 \fi
7000 \forestoget{tier}\forest@temp@mytier
7001 \def\forest@temp@mytiers{%
7002 \ifdefempty\forest@temp@mytier{}{%
7003 \listead\forest@temp@mytiers\forest@temp@mytier
7004 }%
7005 \ifnum\forestove{n children}>0
7006 \forest@node@foreachchild{%
7007 \forestoget{tiers}\forest@temp@tiers
7008 \forlistloop\forest@pack@tiers@forhandlerA\forest@temp@tiers
7009 }%
7010 \fi
7011 \forestolet{tiers}\forest@temp@mytiers
7012 }
7013 \def\forest@pack@tiers@forhandlerA#1{%
7014 \ifinlist{#1}\forest@temp@mytiers{}{%
7015 \listead\forest@temp@mytiers{#1}%
7016 }%
7017 }

```

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.

```

7018 \def\forest@pack@tiers@computetierhierarchy{%
7019 \def\forest@tiers@ancestors{%
7020 \forestoget{tiers}\forest@temp@mytiers
7021 \forlistloop\forest@pack@tiers@cth@init\forest@temp@mytiers
7022 \forest@pack@tiers@computetierhierarchy@
7023 }
7024 \def\forest@pack@tiers@cth@init#1{%
7025 \csdef{forest@tiers@higher@#1}{}%

```

```

7026 \csdef{forest@tiers@lower@#1}{}%
7027 }
7028 \def\forest@pack@tiers@computetierhierarchy{%
7029 \forestoget{tier}\forest@temp@mytier
7030 \ifdefempty\forest@temp@mytier}{}%
7031 \forlistloop\forest@pack@tiers@forhandlerB\forest@tiers@ancestors
7032 \listead\forest@tiers@ancestors\forest@temp@mytier
7033 }%
7034 \forest@node@foreachchild{%
7035 \forest@pack@tiers@computetierhierarchy@
7036 }%
7037 \forestoget{tier}\forest@temp@mytier
7038 \ifdefempty\forest@temp@mytier}{}%
7039 \forest@listedel\forest@tiers@ancestors\forest@temp@mytier
7040 }%
7041 }
7042 \def\forest@pack@tiers@forhandlerB#1{%
7043 \def\forest@temp@tier{#1}%
7044 \ifx\forest@temp@tier\forest@temp@mytier
7045 \PackageError{forest}{Circular tier hierarchy (tier \forest@temp@mytier)}{ }%
7046 \fi
7047 \ifinlistcs{#1}\forest@tiers@higher@\forest@temp@mytier}{ }{%
7048 \listcsadd\forest@tiers@higher@\forest@temp@mytier}{#1}%
7049 \xifinlistcs\forest@temp@mytier\forest@tiers@lower@#1}{ }{%
7050 \listcseadd\forest@tiers@lower@#1}\forest@temp@mytier}%
7051 }
7052 \def\forest@pack@tiers@findcontainers{%
7053 \forestoget{tiers}\forest@temp@tiers
7054 \forlistloop\forest@pack@tiers@findcontainer\forest@temp@tiers
7055 }
7056 \def\forest@pack@tiers@findcontainer#1{%
7057 \def\forest@temp@tier{#1}%
7058 \forestoget{tier}\forest@temp@mytier
7059 \ifx\forest@temp@tier\forest@temp@mytier
7060 \csdef{forest@tiers@container@#1}\forest@cn}%
7061 \else\@escapeif{%
7062 \forest@pack@tiers@findcontainerA{#1}%
7063 }\fi%
7064 }
7065 \def\forest@pack@tiers@findcontainerA#1{%
7066 \c@pgf@counta=0
7067 \forest@node@foreachchild{%
7068 \forestoget{tiers}\forest@temp@tiers
7069 \ifinlist{#1}\forest@temp@tiers{%
7070 \advance\c@pgf@counta 1
7071 \let\forest@temp@child\forest@cn
7072 } }%
7073 }%
7074 \ifnum\c@pgf@counta>1
7075 \csdef{forest@tiers@container@#1}\forest@cn}%
7076 \else\@escapeif{% surely =1
7077 \forest@fornode{\forest@temp@child}{%
7078 \forest@pack@tiers@findcontainer{#1}%
7079 }%
7080 }\fi
7081 }
7082 \def\forest@pack@tiers@raisecontainers{%
7083 \forestoget{tiers}\forest@temp@mytiers
7084 \forlistloop\forest@pack@tiers@rc@forhandlerA\forest@temp@mytiers
7085 }
7086 \def\forest@pack@tiers@rc@forhandlerA#1{%

```

```

7087 \edef\forest@tiers@temptier{#1}%
7088 \letcs\forest@tiers@containernodeoftier{forest@tiers@container@#1}%
7089 \letcs\forest@temp@lowertiers{forest@tiers@lower@#1}%
7090 \forlistloop\forest@pack@tiers@rc@forhandlerB\forest@temp@lowertiers
7091 }
7092 \def\forest@pack@tiers@rc@forhandlerB#1{%
7093 \letcs\forest@tiers@containernodeoflowertier{forest@tiers@container@#1}%
7094 \forest@get{\forest@tiers@containernodeoflowertier}{content}\lowercontent
7095 \forest@get{\forest@tiers@containernodeoftier}{content}\uppercontent
7096 \forest@fornode{\forest@tiers@containernodeoflowertier}{%
7097 \forest@ifancestorof
7098 {\forest@tiers@containernodeoftier}
7099 {\csletcs{forest@tiers@container@\forest@tiers@temptier}{forest@tiers@container@#1}}%
7100 {}}%
7101 }%
7102 }
7103 \def\forest@pack@tiers@computeprocessingorder{%
7104 \def\forest@tiers@processingorder{%
7105 \forest@toget{tiers}\forest@tiers@cpo@tierstodo
7106 \safeloop
7107 \ifdefempty\forest@tiers@cpo@tierstodo{\forest@tempfalse}{\forest@temptrue}%
7108 \ifforest@temp
7109 \def\forest@tiers@cpo@tiersremaining{%
7110 \def\forest@tiers@cpo@tiersindependent{%
7111 \forlistloop\forest@pack@tiers@cpo@forhandlerA\forest@tiers@cpo@tierstodo
7112 \ifdefempty\forest@tiers@cpo@tiersindependent{%
7113 \PackageError{forest}{Circular tiers!}{}}}%
7114 \forlistloop\forest@pack@tiers@cpo@forhandlerB\forest@tiers@cpo@tiersremaining
7115 \let\forest@tiers@cpo@tierstodo\forest@tiers@cpo@tiersremaining
7116 \saferepeat
7117 }
7118 \def\forest@pack@tiers@cpo@forhandlerA#1{%
7119 \ifcseempty{forest@tiers@higher@#1}%
7120 \listadd\forest@tiers@cpo@tiersindependent{#1}%
7121 \listadd\forest@tiers@processingorder{#1}%
7122 }{%
7123 \listadd\forest@tiers@cpo@tiersremaining{#1}%
7124 }%
7125 }
7126 \def\forest@pack@tiers@cpo@forhandlerB#1{%
7127 \def\forest@pack@tiers@cpo@aremainingtier{#1}%
7128 \forlistloop\forest@pack@tiers@cpo@forhandlerC\forest@tiers@cpo@tiersindependent
7129 }
7130 \def\forest@pack@tiers@cpo@forhandlerC#1{%
7131 \ifinlistcs{#1}{forest@tiers@higher@\forest@pack@tiers@cpo@aremainingtier}{%
7132 \forest@listcsdel{forest@tiers@higher@\forest@pack@tiers@cpo@aremainingtier}{#1}%
7133 }{}%
7134 }
7135 \def\forest@pack@tiers@write{%
7136 \forlistloop\forest@pack@tiers@write@forhandler\forest@tiers@processingorder
7137 }
7138 \def\forest@pack@tiers@write@forhandler#1{%
7139 \forest@fornode{\csname forest@tiers@container@#1\endcsname}{%
7140 \forest@pack@tiers@check{#1}%
7141 }%
7142 \xappto\forest@smuggle{%
7143 \noexpand\listadd
7144 \forest@om{\csname forest@tiers@container@#1\endcsname}{@tiers}%
7145 {#1}}%
7146 }%
7147 }

```



```

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

```

## 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.

```

7199 \pgfdeclaregenericanchor{forestcomputenodeboundary}{%
7200   \letcs\forest@temp@boundary@macro\forest@compute@node@boundary@#1}%
7201   \ifcsname forest@compute@node@boundary@#1\endcsname
7202     \csname forest@compute@node@boundary@#1\endcsname

```

```

7203 \else
7204 \forest@compute@node@boundary@rectangle
7205 \fi
7206 \pgfsyssoftpath@getcurrentpath\forest@temp
7207 \global\let\forest@global@boundary\forest@temp
7208 }
7209 \def\forest@mt#1{%
7210 \expandafter\pgfpointanchor\expandafter{\pgfreferencednodename}{#1}%
7211 \pgfsyssoftpath@moveto{\the\pgf@x}{\the\pgf@y}%
7212 }%
7213 \def\forest@lt#1{%
7214 \expandafter\pgfpointanchor\expandafter{\pgfreferencednodename}{#1}%
7215 \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
7216 }%
7217 \def\forest@compute@node@boundary@coordinate{%
7218 \forest@mt{center}}%
7219 }
7220 \def\forest@compute@node@boundary@circle{%
7221 \forest@mt{east}}%
7222 \forest@lt{north east}%
7223 \forest@lt{north}%
7224 \forest@lt{north west}%
7225 \forest@lt{west}%
7226 \forest@lt{south west}%
7227 \forest@lt{south}%
7228 \forest@lt{south east}%
7229 \forest@lt{east}%
7230 }
7231 \def\forest@compute@node@boundary@rectangle{%
7232 \forest@mt{south west}}%
7233 \forest@lt{south east}%
7234 \forest@lt{north east}%
7235 \forest@lt{north west}%
7236 \forest@lt{south west}%
7237 }
7238 \def\forest@compute@node@boundary@diamond{%
7239 \forest@mt{east}}%
7240 \forest@lt{north}%
7241 \forest@lt{west}%
7242 \forest@lt{south}%
7243 \forest@lt{east}%
7244 }
7245 \let\forest@compute@node@boundary@ellipse\forest@compute@node@boundary@circle
7246 \def\forest@compute@node@boundary@trapezium{%
7247 \forest@mt{top right corner}}%
7248 \forest@lt{top left corner}%
7249 \forest@lt{bottom left corner}%
7250 \forest@lt{bottom right corner}%
7251 \forest@lt{top right corner}%
7252 }
7253 \def\forest@compute@node@boundary@semicircle{%
7254 \forest@mt{arc start}}%
7255 \forest@lt{north}%
7256 \forest@lt{east}%
7257 \forest@lt{north east}%
7258 \forest@lt{apex}%
7259 \forest@lt{north west}%
7260 \forest@lt{west}%
7261 \forest@lt{arc end}}%
7262 \forest@lt{arc start}}%
7263 }

```

```

7264 %\newloop\forest@computenodeboundary@loop
7265 \csdef{forest@compute@node@boundary@regular polygon}{%
7266   \forest@mt{corner 1}%
7267   \c@pgf@counta=\sides\relax
7268   \forest@loop
7269   \ifnum\c@pgf@counta>0
7270     \forest@lt{corner \the\c@pgf@counta}%
7271     \advance\c@pgf@counta-1
7272   \forest@repeat
7273 }%
7274 \def\forest@compute@node@boundary@star{%
7275   \forest@mt{outer point 1}%
7276   \c@pgf@counta=\totalstarpoints\relax
7277   \divide\c@pgf@counta2
7278   \forest@loop
7279   \ifnum\c@pgf@counta>0
7280     \forest@lt{inner point \the\c@pgf@counta}%
7281     \forest@lt{outer point \the\c@pgf@counta}%
7282     \advance\c@pgf@counta-1
7283   \forest@repeat
7284 }%
7285 \csdef{forest@compute@node@boundary@isosceles triangle}{%
7286   \forest@mt{apex}%
7287   \forest@lt{left corner}%
7288   \forest@lt{right corner}%
7289   \forest@lt{apex}%
7290 }
7291 \def\forest@compute@node@boundary@kite{%
7292   \forest@mt{upper vertex}%
7293   \forest@lt{left vertex}%
7294   \forest@lt{lower vertex}%
7295   \forest@lt{right vertex}%
7296   \forest@lt{upper vertex}%
7297 }
7298 \def\forest@compute@node@boundary@dart{%
7299   \forest@mt{tip}%
7300   \forest@lt{left tail}%
7301   \forest@lt{tail center}%
7302   \forest@lt{right tail}%
7303   \forest@lt{tip}%
7304 }
7305 \csdef{forest@compute@node@boundary@circular sector}{%
7306   \forest@mt{sector center}%
7307   \forest@lt{arc start}%
7308   \forest@lt{arc center}%
7309   \forest@lt{arc end}%
7310   \forest@lt{sector center}%
7311 }
7312 \def\forest@compute@node@boundary@cylinder{%
7313   \forest@mt{top}%
7314   \forest@lt{after top}%
7315   \forest@lt{before bottom}%
7316   \forest@lt{bottom}%
7317   \forest@lt{after bottom}%
7318   \forest@lt{before top}%
7319   \forest@lt{top}%
7320 }
7321 \cslet{forest@compute@node@boundary@forbidden sign}\forest@compute@node@boundary@circle
7322 \cslet{forest@compute@node@boundary@magnifying glass}\forest@compute@node@boundary@circle
7323 \def\forest@compute@node@boundary@cloud{%
7324   \getradii

```

```

7325 \forest@mt{puff 1}%
7326 \c@pgf@counta=\puffs\relax
7327 \forest@loop
7328 \ifnum\c@pgf@counta>0
7329 \forest@lt{puff \the\c@pgf@counta}%
7330 \advance\c@pgf@counta-1
7331 \forest@repeat
7332 }
7333 \def\forest@compute@node@boundary@starburst{
7334 \calculatestarburstpoints
7335 \forest@mt{outer point 1}%
7336 \c@pgf@counta=\totalpoints\relax
7337 \divide\c@pgf@counta2
7338 \forest@loop
7339 \ifnum\c@pgf@counta>0
7340 \forest@lt{inner point \the\c@pgf@counta}%
7341 \forest@lt{outer point \the\c@pgf@counta}%
7342 \advance\c@pgf@counta-1
7343 \forest@repeat
7344 }%
7345 \def\forest@compute@node@boundary@signal{%
7346 \forest@mt{east}%
7347 \forest@lt{south east}%
7348 \forest@lt{south west}%
7349 \forest@lt{west}%
7350 \forest@lt{north west}%
7351 \forest@lt{north east}%
7352 \forest@lt{east}%
7353 }
7354 \def\forest@compute@node@boundary@tape{%
7355 \forest@mt{north east}%
7356 \forest@lt{60}%
7357 \forest@lt{north}%
7358 \forest@lt{120}%
7359 \forest@lt{north west}%
7360 \forest@lt{south west}%
7361 \forest@lt{240}%
7362 \forest@lt{south}%
7363 \forest@lt{310}%
7364 \forest@lt{south east}%
7365 \forest@lt{north east}%
7366 }
7367 \csdef{forest@compute@node@boundary@single arrow}{%
7368 \forest@mt{tip}%
7369 \forest@lt{after tip}%
7370 \forest@lt{after head}%
7371 \forest@lt{before tail}%
7372 \forest@lt{after tail}%
7373 \forest@lt{before head}%
7374 \forest@lt{before tip}%
7375 \forest@lt{tip}%
7376 }
7377 \csdef{forest@compute@node@boundary@double arrow}{%
7378 \forest@mt{tip 1}%
7379 \forest@lt{after tip 1}%
7380 \forest@lt{after head 1}%
7381 \forest@lt{before head 2}%
7382 \forest@lt{before tip 2}%
7383 \forest@mt{tip 2}%
7384 \forest@lt{after tip 2}%
7385 \forest@lt{after head 2}%

```

```

7386 \forest@lt{before head 1}%
7387 \forest@lt{before tip 1}%
7388 \forest@lt{tip 1}%
7389 }
7390 \csdef{forest@compute@node@boundary@arrow box}{%
7391 \forest@mt{before north arrow}%
7392 \forest@lt{before north arrow head}%
7393 \forest@lt{before north arrow tip}%
7394 \forest@lt{north arrow tip}%
7395 \forest@lt{after north arrow tip}%
7396 \forest@lt{after north arrow head}%
7397 \forest@lt{after north arrow}%
7398 \forest@lt{north east}%
7399 \forest@lt{before east arrow}%
7400 \forest@lt{before east arrow head}%
7401 \forest@lt{before east arrow tip}%
7402 \forest@lt{east arrow tip}%
7403 \forest@lt{after east arrow tip}%
7404 \forest@lt{after east arrow head}%
7405 \forest@lt{after east arrow}%
7406 \forest@lt{south east}%
7407 \forest@lt{before south arrow}%
7408 \forest@lt{before south arrow head}%
7409 \forest@lt{before south arrow tip}%
7410 \forest@lt{south arrow tip}%
7411 \forest@lt{after south arrow tip}%
7412 \forest@lt{after south arrow head}%
7413 \forest@lt{after south arrow}%
7414 \forest@lt{south west}%
7415 \forest@lt{before west arrow}%
7416 \forest@lt{before west arrow head}%
7417 \forest@lt{before west arrow tip}%
7418 \forest@lt{west arrow tip}%
7419 \forest@lt{after west arrow tip}%
7420 \forest@lt{after west arrow head}%
7421 \forest@lt{after west arrow}%
7422 \forest@lt{north west}%
7423 \forest@lt{before north arrow}%
7424 }
7425 \cslet{forest@compute@node@boundary@circle split}\forest@compute@node@boundary@circle
7426 \cslet{forest@compute@node@boundary@circle solidus}\forest@compute@node@boundary@circle
7427 \cslet{forest@compute@node@boundary@ellipse split}\forest@compute@node@boundary@ellipse
7428 \cslet{forest@compute@node@boundary@rectangle split}\forest@compute@node@boundary@rectangle
7429 \def\forest@compute@node@boundary@@callout{%
7430 \beforecalloutpointer
7431 \pgfsyssoftpath@moveto{\the\pgf@x}{\the\pgf@y}%
7432 \calloutpointeranchor
7433 \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
7434 \aftercalloutpointer
7435 \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
7436 }
7437 \csdef{forest@compute@node@boundary@rectangle callout}{%
7438 \forest@compute@node@boundary@rectangle
7439 \rectanglecalloutpoints
7440 \forest@compute@node@boundary@@callout
7441 }
7442 \csdef{forest@compute@node@boundary@ellipse callout}{%
7443 \forest@compute@node@boundary@ellipse
7444 \ellipsecalloutpoints
7445 \forest@compute@node@boundary@@callout
7446 }

```

```

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

```

### 8.3 Compute absolute positions

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

```

7485 \def\forest@node@computeabsolutepositions{%
7486   \edef\forest@marshal{%
7487     \noexpand\forest@node@foreachchild{%
7488       \noexpand\forest@node@computeabsolutepositions@{\forestove{x}}{\forestove{y}}{\forestove{grow}}}%
7489     }%
7490   }\forest@marshal
7491 }
7492 \def\forest@node@computeabsolutepositions@#1#2#3{%
7493   \pgfpointadd
7494     {\pgfqpoint{#1}{#2}}%
7495     {\pgfpointadd
7496       {\pgfqpointpolar{#3}{\forestove{1}}}%
7497       {\pgfqpointpolar{\numexpr 90+#3\relax}{\forestove{s}}}%
7498     }%
7499   \pgfgetlastxy\forest@temp@x\forest@temp@y
7500   \forestolet{x}\forest@temp@x
7501   \forestolet{y}\forest@temp@y

```

```

7502 \edef\forest@marshal{%
7503   \noexpand\forest@node@foreachchild{%
7504     \noexpand\forest@node@computeabsolute positions@{\forest@temp@x}{\forest@temp@y}{\forest@temp@z}{\forest@temp@grow}}%
7505   }%
7506 }\forest@marshal
7507 }

```

## 8.4 Drawing the tree

```

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

```

```

7559 \noexpand\forest@node@drawtree@
7560 \expandonce{\forest@temp@enddraw\pgfkeysnovalue\pgfeov}%
7561 \noexpand\forest@drawtree@endbox
7562 }\forest@marshal
7563 \ifforesttikzcshack
7564 \let\tikz@parse@node\forest@original@tikz@parse@node
7565 \fi
7566 %
7567 \ifforest@external@
7568 \ifforest@externalize@tree@
7569 \tikzexternaldisable
7570 \eappto\forest@externalize@checkimages{%
7571 \noexpand\forest@includeexternal@check{\forest@externalize@filename}%
7572 }%
7573 \expandafter\ifstrequal\expandafter{\forest@drawtreebox}{\pgfkeysnovalue}{%
7574 \eappto\forest@externalize@loadimages{%
7575 \noexpand\forest@includeexternal{\forest@externalize@filename}%
7576 }%
7577 }{%
7578 \eappto\forest@externalize@loadimages{%
7579 \noexpand\forest@includeexternal@box\forest@drawtreebox{\forest@externalize@filename}%
7580 }%
7581 }%
7582 \fi
7583 \fi
7584 }
7585 \def\forest@drawtree@root{0}
7586 \def\forest@node@drawtree{%
7587 \def\forest@clear@drawn{%
7588 \forest@forthis{%
7589 \forest@saveandrestoremacro\forest@drawtree@root{%
7590 \edef\forest@drawtree@root{\forest@cn}%
7591 \forestset{draw tree method}%
7592 }%
7593 }%
7594 \forest@node@ifnamedefined{forest@baseline@node}{%
7595 \edef\forest@baseline@id{\forest@node@Nametoid{forest@baseline@node}}%
7596 \ifcsdef{forest@drawn@}\forest@baseline@id{%
7597 \edef\forest@marshal{%
7598 \noexpand\pgfsetbaselinepointlater{%
7599 \noexpand\pgfpointanchor
7600 {\forest@ve{\forest@baseline@id}{name}}%
7601 {\forest@ve{\forest@baseline@id}{anchor}}%
7602 }%
7603 }\forest@marshal
7604 }{}%
7605 }{}%
7606 \forest@clear@drawn
7607 }
7608 \def\forest@draw@node{%
7609 \ifnum\forest@ve{phantom}=0
7610 \forest@draw@node@
7611 \fi
7612 }
7613 \def\forest@draw@node@{%
7614 \forest@node@forest@position@node@later@restore
7615 \ifforest@drawtree@preservenodeboxes@
7616 \pgfnodealias{forest@temp}{\forest@ve{later@name}}%
7617 \fi
7618 \pgfpositionnodenow{\pgfpqpoint{\forest@ve{x}}{\forest@ve{y}}}%
7619 \ifforest@drawtree@preservenodeboxes@

```



```

7620 \pgfnodealias{\forestove{later@name}}{\forest@temp}%
7621 \fi
7622 \csdef{forest@drawn@{\forest@cn}}{}%
7623 \eappto\forest@clear@drawn{\noexpand\csundef{forest@drawn@{\forest@cn}}}%
7624 }
7625 \def\forest@draw@edge{%
7626 \ifcsdef{forest@drawn@{\forest@cn}}{% was the current node drawn?
7627 \ifnum\forestove{@parent}=0 % do we have a parent?
7628 \else
7629 \ifcsdef{forest@drawn@{\forestove{@parent}}}{% was the parent drawn?
7630 \forest@draw@edge@
7631 }{}%
7632 \fi
7633 }{}%
7634 }
7635 \def\forest@draw@edge@{%
7636 \edef\forest@temp{\forestove{edge path}}\forest@temp
7637 }
7638 \def\forest@draw@tikz{%
7639 \ifnum\forestove{phantom}=0
7640 \forest@draw@tikz@
7641 \fi
7642 }
7643 \def\forest@draw@tikz@{%
7644 \forestove{tikz}%
7645 }

```

## 9 Geometry

A  $\alpha$  *grow line* is a line through the origin at angle  $\alpha$ . The following macro sets up the grow line, which can then be used by other code (the change is local to the  $\text{\TeX}$  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  $(x_g, y_g)$   $90^\circ$  counter-clockwise.

```

7646 \newdimen\forest@xg
7647 \newdimen\forest@yg
7648 \newdimen\forest@xs
7649 \newdimen\forest@ys
7650 \def\forest@setupgrowline#1{%
7651 \edef\forest@grow{#1}%
7652 \pgfqpointpolar{\forest@grow}{1pt}%
7653 \forest@xg=\pgf@x
7654 \forest@yg=\pgf@y
7655 \forest@xs=-\pgf@y
7656 \forest@ys=\pgf@x
7657 }

```

### 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`.

```

7658 \def\forest@pgfpointprojectiontogrowline#1{%
7659 \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.

```

7660 \pgfutil@tempdima=\pgf@sys@tonumber{\pgf@x}\forest@xg%
7661 \advance\pgfutil@tempdima by\pgf@sys@tonumber{\pgf@y}\forest@yg%

```

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

```

7662 \global\pgf@x=\pgf@sys@tonumber{\pgfutil@tempdima}\forest@xg%

```

```

7663 \global\pgf@y=\pgf@sys@tonumber{\pgfutil@tempdima}\forest@yg%
7664 }}

```

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

```

7665 \def\forest@distancetogrowline#1#2{%
7666   \pgf@process{#2}%
7667   #1=\pgf@sys@tonumber{\pgf@x}\forest@xs\relax
7668   \advance#1 by\pgf@sys@tonumber{\pgf@y}\forest@ys\relax
7669 }

```

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

The following macro projects all segment edges (“points”) of a simple<sup>2</sup> path #1 onto the grow line. The result is an array of tuples  $(x_o, y_o, x_p, y_p)$ , where  $x_o$  and  $y_o$  stand for the *original* point, and  $x_p$  and  $y_p$  stand for its *projection*. The prefix of the array is given by #2. If the array already exists, the new items are appended to it. The array is not sorted: the order of original points in the array is their order in the path. The computation does not destroy the current path. All result-macros have local scope.

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

```

7670 \let\forest@pp@n\relax
7671 \def\forest@projectpath@togrowline#1#2{%
7672   \edef\forest@pp@prefix{#2}%
7673   \forest@save@pgfsyssoftpath@tokendefs
7674   \let\pgfsyssoftpath@movetotoken\forest@projectpath@processpoint
7675   \let\pgfsyssoftpath@linetotoken\forest@projectpath@processpoint
7676   \c@pgf@counta=0
7677   #1%
7678   \csedef{#2n}{\the\c@pgf@counta}%
7679   \forest@restore@pgfsyssoftpath@tokendefs
7680 }

```

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

```

7681 \def\forest@projectpath@processpoint#1#2{%
7682   \pgfqpoint{#1}{#2}%
7683   \expandafter\edef\c@pgf@counta\forest@pp@prefix\the\c@pgf@counta x\endcsname{\the\pgf@x}%
7684   \expandafter\edef\c@pgf@counta\forest@pp@prefix\the\c@pgf@counta y\endcsname{\the\pgf@y}%
7685   \forest@pgfpointprojectiontogrowline{}%
7686   \expandafter\edef\c@pgf@counta\forest@pp@prefix\the\c@pgf@counta xp\endcsname{\the\pgf@x}%
7687   \expandafter\edef\c@pgf@counta\forest@pp@prefix\the\c@pgf@counta yp\endcsname{\the\pgf@y}%
7688   \advance\c@pgf@counta 1\relax
7689 }

```

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

```

7690 \def\forest@sortprojections#1{%
7691   % todo: optimize in cases when we know that the array is actually a
7692   % merger of sorted arrays; when does this happen? in
7693   % distance_between_paths, and when merging the edges of the parent
7694   % and its children in a uniform growth tree
7695   \edef\forest@ppi@inputprefix{#1}%
7696   \c@pgf@counta=\c@pgf@counta#1n\endcsname\relax
7697   \advance\c@pgf@counta -1
7698   \forest@sort\forest@ppiraw@cmp\forest@ppiraw@let\forest@sort@ascending{0}{\the\c@pgf@counta}%
7699 }

```

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)

---

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

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

```
7700 \def\forest@processprojectioninfo#1#2#3{%
7701   \edef\forest@ppi

```

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

```
7702   \c@pgf@counta=0
7703   \c@pgf@countb=-1
7704   \safeloop
7705   \ifnum\c@pgf@counta<\csname#1n\endcsname\relax
```

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

```
7706   \letcs\forest@xo{#1\the\c@pgf@counta xo}%
7707   \letcs\forest@yo{#1\the\c@pgf@counta yo}%
7708   \letcs\forest@xp{#1\the\c@pgf@counta xp}%
7709   \letcs\forest@yp{#1\the\c@pgf@counta yp}%
7710   \ifnum\c@pgf@countb<0
7711     \forest@equaltotolerancefalse
7712   \else
7713     \forest@equaltotolerance
7714     {\pgfqpoint\forest@xp\forest@yp}%
7715     {\pgfqpoint
7716       {\csname#2\the\c@pgf@countb x\endcsname}%
7717       {\csname#2\the\c@pgf@countb y\endcsname}%
7718     }%
7719   \fi
7720   \ifforest@equaltotolerance\else
```

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

```
7721   \advance\c@pgf@countb 1
7722   \cslet{#2\the\c@pgf@countb x}\forest@xp
7723   \cslet{#2\the\c@pgf@countb y}\forest@yp
7724   \csdef{#2\the\c@pgf@countb @n}{0}%
7725   \fi
```

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

```
7726   % todo: this is ugly!
7727   \ifdefined\forest@xo\ifx\forest@xo\relax\else
7728   \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.

```
7729   \forest@append@point@to@inner@array
7730   \forest@xo\forest@yo
7731   {#2\the\c@pgf@countb @}%
```

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

```
7732   \eapptotoks#3{(\forest@xo,\forest@yo){\the\c@pgf@countb}}%
7733   \fi\fi
7734   \fi\fi
```

---

<sup>3</sup>At first sight, this information could be cached “at the source”: by  $\backslash forest@pgfqpointprojectiontogrowline$ . However, due to imprecise intersecting (in  $\backslash breakpath$ ), 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.

Clean-up the raw array item.

```

7735 % todo: is this really necessary? (yes: see the "ugly" thing above)
7736 \cslet{#1\the\c@pgf@counta xo}\relax
7737 \cslet{#1\the\c@pgf@counta yo}\relax
7738 \cslet{#1\the\c@pgf@counta xp}\relax
7739 \cslet{#1\the\c@pgf@counta yp}\relax
7740 \advance\c@pgf@counta 1
7741 \saferepeat

```

Clean up the raw array length.

```

7742 % todo: is this really necessary?
7743 \cslet{#1n}\relax

```

Store the length of the outer result array.

```

7744 \advance\c@pgf@countb 1
7745 \csedef{#2n}{\the\c@pgf@countb}%
7746 }

```

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

```

7747 \def\forest@ppiraw@let#1#2{%
7748 \csletcs{\forest@ppi@inputprefix#1xo}{\forest@ppi@inputprefix#2xo}%
7749 \csletcs{\forest@ppi@inputprefix#1yo}{\forest@ppi@inputprefix#2yo}%
7750 \csletcs{\forest@ppi@inputprefix#1xp}{\forest@ppi@inputprefix#2xp}%
7751 \csletcs{\forest@ppi@inputprefix#1yp}{\forest@ppi@inputprefix#2yp}%
7752 }

```

Item comparison macro for sorting the raw projection data.

```

7753 \def\forest@ppiraw@cmp#1#2{%
7754 \forest@sort@cmptwodimcs
7755 {\forest@ppi@inputprefix#1xp}{\forest@ppi@inputprefix#1yp}%
7756 {\forest@ppi@inputprefix#2xp}{\forest@ppi@inputprefix#2yp}%
7757 }

```

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

```

7758 \def\forest@append@point@to@inner@array#1#2#3{%
7759 \c@pgf@countc=\csname#3n\endcsname\relax
7760 \csedef{#3\the\c@pgf@countc x}{#1}%
7761 \csedef{#3\the\c@pgf@countc y}{#2}%
7762 \forest@distancetogrowline\pgfutil@tempdima{\pgfqpoint#1#2}%
7763 \csedef{#3\the\c@pgf@countc d}{\the\pgfutil@tempdima}%
7764 \advance\c@pgf@countc 1
7765 \csedef{#3n}{\the\c@pgf@countc}%
7766 }

```

## 9.2 Break path

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

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

```

7767 \def\forest@breakpath#1#2#3#4{%

```

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

```

7768 \edef\forest@bp@prefix{#2}%
7769 \let\forest@breakpath@toks#3%

```

```

7770 \forest@save@pgfsyssoftpath@tokendefs
7771 \let\pgfsyssoftpath@movetotoken\forest@breakpath@processfirstpoint
7772 \let\pgfsyssoftpath@linetotoken\forest@breakpath@processfirstpoint
7773 %\pgfusepath{}% empty the current path. ok?
7774 #1%
7775 \forest@restore@pgfsyssoftpath@tokendefs
7776 \pgfsyssoftpath@getcurrentpath#4%
7777 }

```

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.)

```

7778 \def\forest@breakpath@processfirstpoint#1#2{%
7779 \forest@breakpath@processmoveto{#1}{#2}%
7780 \let\pgfsyssoftpath@movetotoken\forest@breakpath@processmoveto
7781 \let\pgfsyssoftpath@linetotoken\forest@breakpath@processlineto
7782 }

```

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.

```

7783 \def\forest@breakpath@processmoveto#1#2{%
7784 \pgfsyssoftpath@moveto{#1}{#2}%
7785 \def\forest@previous@x{#1}%
7786 \def\forest@previous@y{#2}%
7787 \forest@breakpath@getfromtoks\forest@breakpath@toks\forest@previous@i{#1}{#2}%
7788 \expandafter\let\expandafter\forest@previous@px
7789 \csname\forest@bp@prefix\forest@previous@i x\endcsname
7790 \expandafter\let\expandafter\forest@previous@py
7791 \csname\forest@bp@prefix\forest@previous@i y\endcsname
7792 }
7793 \def\forest@breakpath@getfromtoks#1#2#3#4{%
7794 % #1=cache toks register, #2=receiving cs, (#3,#4)=point;
7795 % we rely on the fact that the point we’re looking up should always be present
7796 \def\forest@breakpath@getfromtoks@##1(#3,#4)##2##3\forest@END{##2}%
7797 \edef#2{\expandafter\forest@breakpath@getfromtoks@the#1\forest@END}%
7798 }

```

This is the heart of the path-breaking procedure.

```

7799 \def\forest@breakpath@processlineto#1#2{%

```

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

```

7800 \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.)

```

7801 \forest@breakpath@getfromtoks\forest@breakpath@toks\forest@i{#1}{#2}%
7802 \expandafter\let\expandafter\forest@px
7803 \csname\forest@bp@prefix\forest@i x\endcsname
7804 \expandafter\let\expandafter\forest@py
7805 \csname\forest@bp@prefix\forest@i y\endcsname

```

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

```

7806 \forest@equaltotolerance
7807 {\pgfqpoint{\forest@previous@px}{\forest@previous@py}}%
7808 {\pgfqpoint{\forest@px}{\forest@py}}%
7809 \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.

```

7810 \let\forest@breakpath@op\pgfsyssoftpath@moveto
7811 \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.

```

7812 \forest@temp@count=\forest@previous@i\relax
7813 \ifnum\forest@previous@i<\forest@i\relax
7814 \def\forest@breakpath@step{1}%
7815 \def\forest@breakpath@test{\forest@temp@count<\forest@i\relax}%
7816 \else
7817 \def\forest@breakpath@step{-1}%
7818 \def\forest@breakpath@test{\forest@temp@count>\forest@i\relax}%
7819 \fi

```

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

```

7820 \safeloop
7821 \advance\forest@temp@count\forest@breakpath@step\relax
7822 \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.)

```

7823 \pgfpointintersectionoflines
7824 {\pgfqpoint
7825 {\csname\forest@bp@prefix\the\forest@temp@count x\endcsname}%
7826 {\csname\forest@bp@prefix\the\forest@temp@count y\endcsname}%
7827 }%
7828 {\pgfpointadd
7829 {\pgfqpoint
7830 {\csname\forest@bp@prefix\the\forest@temp@count x\endcsname}%
7831 {\csname\forest@bp@prefix\the\forest@temp@count y\endcsname}%
7832 }%
7833 {\pgfqpoint{\forest@xs}{\forest@ys}}%
7834 }%
7835 {\pgfqpoint{\forest@previous@x}{\forest@previous@y}}%
7836 {\pgfqpoint{#1}{#2}}%

```

Break the segment at the intersection.

```

7837 \pgfgetlastxy\forest@last@x\forest@last@y
7838 \pgfsyssoftpath@lineto\forest@last@x\forest@last@y

```

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

```

7839 \forest@append@point@to@inner@array
7840 \forest@last@x\forest@last@y
7841 {\forest@bp@prefix\the\forest@temp@count @}%
7842 \saferepeat
7843 \fi

```

Add the current point.

```

7844 \forest@breakpath@op{#1}{#2}%

```

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

```

7845 \def\forest@previous@x{#1}%
7846 \def\forest@previous@y{#2}%
7847 \let\forest@previous@i\forest@i
7848 \let\forest@previous@px\forest@px
7849 \let\forest@previous@py\forest@py
7850 }

```

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

```

7851 \patchcmd{\pgfpointintersectionoflines}{\pgfqpoint}{\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.<sup>4</sup> 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.

```

7852 \def\forest@getnegativetightedgeofpath#1#2{%
7853   \forest@get@onetightedgeofpath#1\forest@sort@ascending#2}
7854 \def\forest@getpositivetightedgeofpath#1#2{%
7855   \forest@get@onetightedgeofpath#1\forest@sort@descending#2}
7856 \def\forest@get@onetightedgeofpath#1#2#3{%
7857   {%
7858     \forest@get@one@tightedgeofpath#1#2\forest@gep@edge
7859     \global\let\forest@gep@global@edge\forest@gep@edge
7860   }%
7861   \let#3\forest@gep@global@edge
7862 }
7863 \newtoks\forest@pi@toks
7864 \newtoks\forest@segment@toks
7865 \def\forest@get@one@tightedgeofpath#1#2#3{%

```

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

```

7866 \forest@projectpathtogrowline#1\forest@pp@}%
7867 \forest@sortprojections\forest@pp@}%
7868 \forest@processprojectioninfo\forest@pp@\forest@pi@\forest@pi@toks

```

Break the path.

```

7869 \forest@breakpath#1\forest@pi@\forest@pi@toks\forest@brokenpath

```

Compile some more useful information.

```

7870 \forest@sort@inner@arrays\forest@pi@}%2%
7871 \forest@pathtodict\forest@brokenpath\forest@segment@toks

```

The auxiliary data is set up: do the work!

```

7872 \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).

```

7873 \forest@simplifypath\forest@edge#3%
7874 }

```

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

```

7875 \def\forest@getbothtightedgesofpath#1#2#3{%
7876   {%
7877     \forest@get@one@tightedgeofpath#1\forest@sort@ascending\forest@gep@firstedge

```

Reverse the order of items in the inner arrays.

```

7878   \c@pgf@counta=0
7879   \forest@loop
7880   \ifnum\c@pgf@counta<\forest@pi@n\relax
7881     \forest@ppi@deflet\forest@pi@\the\c@pgf@counta @}%
7882     \forest@reversearray\forest@ppi@let
7883     {}%
7884     {\csname forest@pi@\the\c@pgf@counta @\endcsname}%
7885     \advance\c@pgf@counta 1
7886   \forest@repeat

```

---

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

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

```
7887 \forest@gettightedgeofpath@getedge\forest@edge
7888 \forest@simplifypath\forest@edge\forest@gep@secondedge
```

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

```
7889 \global\let\forest@gep@global@firstedge\forest@gep@firstedge
7890 \global\let\forest@gep@global@secondedge\forest@gep@secondedge
7891 }%
7892 \let#2\forest@gep@global@firstedge
7893 \let#3\forest@gep@global@secondedge
7894 }
```

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

```
7895 \def\forest@sort@inner@arrays#1#2{%
7896 \c@pgf@counta=0
7897 \safeloop
7898 \ifnum\c@pgf@counta<\csname#1n\endcsname
7899 \c@pgf@countb=\csname#1\the\c@pgf@counta @n\endcsname\relax
7900 \ifnum\c@pgf@countb>1
7901 \advance\c@pgf@countb -1
7902 \forest@ppi@deflet{#1\the\c@pgf@counta @}%
7903 \forest@ppi@defcmp{#1\the\c@pgf@counta @}%
7904 \forest@sort\forest@ppi@cmp\forest@ppi@let#2{0}{\the\c@pgf@countb}%
7905 \fi
7906 \advance\c@pgf@counta 1
7907 \saferepeat
7908 }
```

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.

```
7909 \def\forest@ppi@deflet#1{%
7910 \edef\forest@ppi@let##1##2{%
7911 \noexpand\csletcs{#1##1x}{#1##2x}%
7912 \noexpand\csletcs{#1##1y}{#1##2y}%
7913 \noexpand\csletcs{#1##1d}{#1##2d}%
7914 }%
7915 }
```

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.

```
7916 \def\forest@ppi@defcmp#1{%
7917 \edef\forest@ppi@cmp##1##2{%
7918 \noexpand\forest@sort@cmpdimcs{#1##1d}{#1##2d}%
7919 }%
7920 }
```

Put path segments into a “segment dictionary”: for each segment of the pgf path (given in `#1`) from  $(x_1, y_1)$  to  $(x_2, y_2)$  we put  $(x_1, y_1) \text{--} (x_2, y_2)$  into toks `#2`. (The “dictionary” was reimplemented in v2.1.4. It’s based on a toks register now, we search using `\pgfutil@in@`.)

```
7921 \def\forest@pathtodict#1#2{%
7922 \let\forest@pathtodict@toks#2%
7923 \forest@save@pgfsyssoftpath@tokendef
7924 \let\pgfsyssoftpath@movetotoken\forest@pathtodict@movetoo
7925 \let\pgfsyssoftpath@linetotoken\forest@pathtodict@linetoo
7926 \def\forest@pathtodict@subpathstart{}%
7927 #1%
7928 \forest@restore@pgfsyssoftpath@tokendef
7929 }
```

When a move-to operation is encountered:

```
7930 \def\forest@pathtodict@movetoo#1#2{%
7931 \apptotoks\forest@pathtodict@toks{(#1,#2)}%
7932 }
```



When a line-to operation is encountered:

```
7933 \def\forest@pathtodict@linetoo#1#2{%
7934   \apptotoks\forest@pathtodict@toks{--(#1,#2)}%
7935 }
```

In this macro, the edge is actually computed.

```
7936 \def\forest@gettightedgeofpath@getedge#1{% cs to store the edge into
```

Clear the path and the last projection.

```
7937   \pgfsyssoftpath@setcurrentpath\pgfutil@empty
7938   \let\forest@last@x\relax
7939   \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.)

```
7940   \c@pgf@counta=0
7941   \forest@temp@count=\forest@pi@n\relax
7942   \advance\forest@temp@count -1
7943   \edef\forest@nminusone{\the\forest@temp@count}%
7944   \safeloop
7945   \ifnum\c@pgf@counta<\forest@nminusone\relax
7946     \forest@gettightedgeofpath@getedge@loopa
7947   \saferepeat
```

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

```
7948   \ifnum\forest@nminusone<\forest@n\relax\else
7949     \ifnum\csname forest@pi@\forest@nminusone @n\endcsname>0
7950       \forest@gettightedgeofpath@maybemoveto{\forest@nminusone}{0}%
7951     \fi
7952   \fi
7953   \pgfsyssoftpath@getcurrentpath#1%
7954   \pgfsyssoftpath@setcurrentpath\pgfutil@empty
7955 }
```

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.

```
7956 \def\forest@gettightedgeofpath@getedge@loopa{%
7957   \ifnum\csname forest@pi@\the\c@pgf@counta @n\endcsname>0
```

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

```
7958   \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.

```
7959   \c@pgf@countb=0
7960   \safeloop
7961   \ifnum\c@pgf@countb<\csname forest@pi@\the\c@pgf@counta @n\endcsname\relax
7962     \forest@gettightedgeofpath@getedge@loopb
7963   \saferepeat
7964   \fi
7965   \advance\c@pgf@counta 1
7966 }
```

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

```
7967 \def\forest@gettightedgeofpath@getedge@loopb{%
7968   \c@pgf@countc=0
7969   \advance\c@pgf@counta 1
7970   \edef\forest@aplusone{\the\c@pgf@counta}%
7971   \advance\c@pgf@counta -1
7972   \safeloop
7973   \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!

```

7974 \edef\forest@temp{%
7975   (\csname forest@pi@the\c@pgf@counta @the\c@pgf@countb x\endcsname,%
7976   \csname forest@pi@the\c@pgf@counta @the\c@pgf@countb y\endcsname)--(%
7977   \csname forest@pi@forest@aplusone @the\c@pgf@countc x\endcsname,%
7978   \csname forest@pi@forest@aplusone @the\c@pgf@countc y\endcsname)%
7979 }%
7980 \expandafter\expandafter\expandafter\pgfutil@in@\expandafter\expandafter\expandafter
7981   {\expandafter\forest@temp\expandafter}\expandafter
7982   {\the\forest@segment@toks}%
7983 \ifpgfutil@in@
7984 \else
7985   \edef\forest@temp{%
7986     (\csname forest@pi@\forest@aplusone @the\c@pgf@countc x\endcsname,%
7987     \csname forest@pi@\forest@aplusone @the\c@pgf@countc y\endcsname)--(%
7988     \csname forest@pi@the\c@pgf@counta @the\c@pgf@countb x\endcsname,%
7989     \csname forest@pi@the\c@pgf@counta @the\c@pgf@countb y\endcsname)%
7990   }%
7991   \expandafter\expandafter\expandafter\pgfutil@in@\expandafter\expandafter\expandafter
7992     {\expandafter\forest@temp\expandafter}\expandafter
7993     {\the\forest@segment@toks}%
7994 \fi
7995 \ifpgfutil@in@

```

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

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

```

7996 \forest@gettightedgeofpath@maybemoveto{\the\c@pgf@counta}{\the\c@pgf@countb}%
Second, create a line to the end point.
7997 \edef\forest@last@x{%
7998   \csname forest@pi@\forest@aplusone @the\c@pgf@countc x\endcsname}%
7999 \edef\forest@last@y{%
8000   \csname forest@pi@\forest@aplusone @the\c@pgf@countc y\endcsname}%
8001 \pgfsyssoftpath@lineto\forest@last@x\forest@last@y

```

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

```

8002 \c@pgf@countc=\csname forest@pi@\forest@aplusone @n\endcsname
8003 \c@pgf@countb=\csname forest@pi@the\c@pgf@counta @n\endcsname
8004 \fi
8005 \advance\c@pgf@countc 1
8006 \saferepeat
8007 \advance\c@pgf@countb 1
8008 }

```

`\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.

```

8009 \def\forest@gettightedgeofpath@maybemoveto#1#2{%
8010   \forest@temptrue
8011   \ifx\forest@last@x\relax\else

```

```

8012 \ifdim\forest@last@x=\csname forest@pi@#1@#2x\endcsname\relax
8013 \ifdim\forest@last@y=\csname forest@pi@#1@#2y\endcsname\relax
8014 \forest@tempfalse
8015 \fi
8016 \fi
8017 \fi
8018 \ifforest@temp
8019 \edef\forest@last@x{\csname forest@pi@#1@#2x\endcsname}%
8020 \edef\forest@last@y{\csname forest@pi@#1@#2y\endcsname}%
8021 \pgfsyssoftpath@moveto\forest@last@x\forest@last@y
8022 \fi
8023 }

```

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

```

8024 \def\forest@simplifypath#1#2{%
8025 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
8026 \forest@save\pgfsyssoftpath@tokendefs
8027 \let\pgfsyssoftpath@movetotoken\forest@simplifypath@moveto
8028 \let\pgfsyssoftpath@linetotoken\forest@simplifypath@lineto
8029 \let\forest@last@x\relax
8030 \let\forest@last@y\relax
8031 \let\forest@last@atan\relax
8032 #1%
8033 \ifx\forest@last@x\relax\else
8034 \ifx\forest@last@atan\relax\else
8035 \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
8036 \fi
8037 \fi
8038 \forest@restore\pgfsyssoftpath@tokendefs
8039 \pgfsyssoftpath@getcurrentpath#2%
8040 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
8041 }

```

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.

```

8042 \def\forest@simplifypath@moveto#1#2{%
8043 \ifx\forest@last@x\relax\else
8044 \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
8045 \fi
8046 \pgfsyssoftpath@moveto{#1}{#2}%
8047 \def\forest@last@x{#1}%
8048 \def\forest@last@y{#2}%
8049 \let\forest@last@atan\relax
8050 }

```

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

```

8051 \def\forest@getedgeofpath@precision{1pt}

```

When a line-to is encountered...

```

8052 \def\forest@simplifypath@lineto#1#2{%
8053 \ifx\forest@last@x\relax

```

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

```

8054 \def\forest@last@x{#1}%
8055 \def\forest@last@y{#2}%
8056 \let\forest@last@atan\relax
8057 \else

```

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

```

8058 \pgfpointdiff{\pgfpoint{#1}{#2}}{\pgfpoint{\forest@last@x}{\forest@last@y}}%

```

```

8059 \ifdim\pgf@x<\pgfintersectiontolerance
8060 \ifdim-\pgf@x<\pgfintersectiontolerance
8061 \pgf@x=0pt
8062 \fi
8063 \fi
8064 \edef\forest@marshal{%
8065 \noexpand\pgfmthatantwo@
8066 {\expandafter\pgf@geT\the\pgf@x}%
8067 {\expandafter\pgf@geT\the\pgf@y}%
8068 }\forest@marshal
8069 \let\forest@current@atan\pgfmthatresult
8070 \ifx\forest@last@atan\relax

```

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

```

8071 \def\forest@last@x{#1}%
8072 \def\forest@last@y{#2}%
8073 \let\forest@last@atan\forest@current@atan
8074 \else

```

Otherwise, compare the first and the current slope.

```

8075 \pgfutil@tempdima=\forest@current@atan pt
8076 \advance\pgfutil@tempdima -\forest@last@atan pt
8077 \ifdim\pgfutil@tempdima<0pt\relax
8078 \multiply\pgfutil@tempdima -1
8079 \fi
8080 \ifdim\pgfutil@tempdima<\forest@getedgeofpath@precision\relax
8081 \else

```

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

```

8082 \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
8083 \let\forest@last@atan\forest@current@atan
8084 \fi

```

In any event, update the last point.

```

8085 \def\forest@last@x{#1}%
8086 \def\forest@last@y{#2}%
8087 \fi
8088 \fi
8089 }

```

## 9.4 Get rectangle/band edge

```

8090 \def\forest@getnegativerectangleedgeofpath#1#2{%
8091 \forest@getnegativerectangleorbandededgeofpath{#1}{#2}{\the\pgf@xb}}
8092 \def\forest@getpositiverectangleedgeofpath#1#2{%
8093 \forest@getpositiverectangleorbandededgeofpath{#1}{#2}{\the\pgf@xb}}
8094 \def\forest@getbothrectangleedgesofpath#1#2#3{%
8095 \forest@getbothrectangleorbandededgesofpath{#1}{#2}{#3}{\the\pgf@xb}}
8096 \def\forest@bandlength{5000pt} % something large (ca. 180cm), but still manageable for TeX without producing
8097 \def\forest@getnegativebandededgeofpath#1#2{%
8098 \forest@getnegativerectangleorbandededgeofpath{#1}{#2}{\forest@bandlength}}
8099 \def\forest@getpositivebandededgeofpath#1#2{%
8100 \forest@getpositiverectangleorbandededgeofpath{#1}{#2}{\forest@bandlength}}
8101 \def\forest@getbothbandededgesofpath#1#2#3{%
8102 \forest@getbothrectangleorbandededgesofpath{#1}{#2}{#3}{\forest@bandlength}}
8103 \def\forest@getnegativerectangleorbandededgeofpath#1#2#3{%
8104 \forest@path@getboundingrectangle@ls#1{\forest@grow}%
8105 \edef\forest@gre@path{%
8106 \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@ya}%
8107 \noexpand\pgfsyssoftpath@linetotoken{#3}{\the\pgf@ya}%
8108 }%
8109 {%

```

```

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

```

## 9.5 Distance between paths

Another crucial part of the package.

```

8152 \newtoks\forest@PIi@toks
8153 \newtoks\forest@PIii@toks
8154 \def\forest@distance@between@edge@paths#1#2#3{%
8155 \begingroup
8156 % #1, #2 = (edge) paths
8157 %
8158 % project paths
8159 \forest@projectpathtogrowline#1{\forest@p1@}%
8160 \forest@projectpathtogrowline#2{\forest@p2@}%
8161 % merge projections (the lists are sorted already, because edge
8162 % paths are |sorted|)
8163 \forest@dbep@mergeprojections
8164 {\forest@p1@}{\forest@p2@}%
8165 {\forest@P1@}{\forest@P2@}%
8166 % process projections

```

```

8167 \forest@processprojectioninfo{forest@PI1@}{forest@PI1@}\forest@PIi@toks
8168 \forest@processprojectioninfo{forest@PI2@}{forest@PI2@}\forest@PIii@toks
8169 % break paths
8170 \forest@breakpath#1{forest@PI1@}\forest@PIi@toks\forest@broken@one
8171 \forest@breakpath#2{forest@PI2@}\forest@PIii@toks\forest@broken@two
8172 % sort inner arrays ---optimize: it's enough to find max and min
8173 \forest@sort@inner@arrays{forest@PI1@}\forest@sort@descending
8174 \forest@sort@inner@arrays{forest@PI2@}\forest@sort@ascending
8175 % compute the distance
8176 \let\forest@distance\relax
8177 \c@pgf@countc=0
8178 \forest@loop
8179 \ifnum\c@pgf@countc<\csname forest@PI1@n\endcsname\relax
8180 \ifnum\csname forest@PI1@\the\c@pgf@countc @n\endcsname=0 \else
8181 \ifnum\csname forest@PI2@\the\c@pgf@countc @n\endcsname=0 \else
8182 \pgfutil@tempdima=\csname forest@PI2@\the\c@pgf@countc @0d\endcsname\relax
8183 \advance\pgfutil@tempdima -\csname forest@PI1@\the\c@pgf@countc @0d\endcsname\relax
8184 \ifx\forest@distance\relax
8185 \edef\forest@distance{\the\pgfutil@tempdima}%
8186 \else
8187 \ifdim\pgfutil@tempdima<\forest@distance\relax
8188 \edef\forest@distance{\the\pgfutil@tempdima}%
8189 \fi
8190 \fi
8191 \fi
8192 \fi
8193 \advance\c@pgf@countc 1
8194 \forest@repeat
8195 \global\let\forest@global@temp\forest@distance
8196 \endgroup
8197 \let#3\forest@global@temp
8198 }
8199 % merge projections: we need two projection arrays, both containing
8200 % projection points from both paths, but each with the original
8201 % points from only one path
8202 \def\forest@dbep@mergeprojections#1#2#3#4{%
8203 % TODO: optimize: v bistvu ni treba sortirat, ker je edge path e sortiran
8204 \forest@sortprojections{#1}%
8205 \forest@sortprojections{#2}%
8206 \c@pgf@counta=0
8207 \c@pgf@countb=0
8208 \c@pgf@countc=0
8209 \edef\forest@input@prefix@one{#1}%
8210 \edef\forest@input@prefix@two{#2}%
8211 \edef\forest@output@prefix@one{#3}%
8212 \edef\forest@output@prefix@two{#4}%
8213 \forest@dbep@mp@iterate
8214 \csedef{#3n}{\the\c@pgf@countc}%
8215 \csedef{#4n}{\the\c@pgf@countc}%
8216 }
8217 \def\forest@dbep@mp@iterate{%
8218 \let\forest@dbep@mp@next\forest@dbep@mp@iterate
8219 \ifnum\c@pgf@counta<\csname\forest@input@prefix@one n\endcsname\relax
8220 \ifnum\c@pgf@countb<\csname\forest@input@prefix@two n\endcsname\relax
8221 \let\forest@dbep@mp@next\forest@dbep@mp@do
8222 \else
8223 \let\forest@dbep@mp@next\forest@dbep@mp@iteratefirst
8224 \fi
8225 \else
8226 \ifnum\c@pgf@countb<\csname\forest@input@prefix@two n\endcsname\relax
8227 \let\forest@dbep@mp@next\forest@dbep@mp@iteratesecond

```

```

8228 \else
8229 \let\forest@dbep@mp@next\relax
8230 \fi
8231 \fi
8232 \forest@dbep@mp@next
8233 }
8234 \def\forest@dbep@mp@do{%
8235 \forest@sort@cmtwodimcs%
8236 {\forest@input@prefix@one\the\c@pgf@counta xp}%
8237 {\forest@input@prefix@one\the\c@pgf@counta yp}%
8238 {\forest@input@prefix@two\the\c@pgf@countb xp}%
8239 {\forest@input@prefix@two\the\c@pgf@countb yp}%
8240 \if\forest@sort@cmp@result=%
8241 \forest@dbep@mp@@store@p\forest@input@prefix@one\c@pgf@counta
8242 \forest@dbep@mp@@store@o\forest@input@prefix@one
8243 \c@pgf@counta\forest@output@prefix@one
8244 \forest@dbep@mp@@store@o\forest@input@prefix@two
8245 \c@pgf@countb\forest@output@prefix@two
8246 \advance\c@pgf@counta 1
8247 \advance\c@pgf@countb 1
8248 \else
8249 \if\forest@sort@cmp@result>%
8250 \forest@dbep@mp@@store@p\forest@input@prefix@two\c@pgf@countb
8251 \forest@dbep@mp@@store@o\forest@input@prefix@two
8252 \c@pgf@countb\forest@output@prefix@two
8253 \advance\c@pgf@countb 1
8254 \else%<
8255 \forest@dbep@mp@@store@p\forest@input@prefix@one\c@pgf@counta
8256 \forest@dbep@mp@@store@o\forest@input@prefix@one
8257 \c@pgf@counta\forest@output@prefix@one
8258 \advance\c@pgf@counta 1
8259 \fi
8260 \fi
8261 \advance\c@pgf@countc 1
8262 \forest@dbep@mp@iterate
8263 }
8264 \def\forest@dbep@mp@@store@p#1#2{%
8265 \csletcs
8266 {\forest@output@prefix@one\the\c@pgf@countc xp}%
8267 {#1\the#2xp}%
8268 \csletcs
8269 {\forest@output@prefix@one\the\c@pgf@countc yp}%
8270 {#1\the#2yp}%
8271 \csletcs
8272 {\forest@output@prefix@two\the\c@pgf@countc xp}%
8273 {#1\the#2xp}%
8274 \csletcs
8275 {\forest@output@prefix@two\the\c@pgf@countc yp}%
8276 {#1\the#2yp}%
8277 }
8278 \def\forest@dbep@mp@@store@o#1#2#3{%
8279 \csletcs{#3\the\c@pgf@countc xo}{#1\the#2xo}%
8280 \csletcs{#3\the\c@pgf@countc yo}{#1\the#2yo}%
8281 }
8282 \def\forest@dbep@mp@iteratefirst{%
8283 \forest@dbep@mp@iterateone\forest@input@prefix@one\c@pgf@counta\forest@output@prefix@one
8284 }
8285 \def\forest@dbep@mp@iteratesecond{%
8286 \forest@dbep@mp@iterateone\forest@input@prefix@two\c@pgf@countb\forest@output@prefix@two
8287 }
8288 \def\forest@dbep@mp@iterateone#1#2#3{%

```

```

8289 \forest@loop
8290 \ifnum#2<\csname#1n\endcsname\relax
8291 \forest@dbep@mp@@store@p#1#2%
8292 \forest@dbep@mp@@store@o#1#2#3%
8293 \advance\c@pgf@countc 1
8294 \advance#21
8295 \forest@repeat
8296 }

```

## 9.6 Utilities

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

```

8297 \newif\ifforest@equaltotolerance
8298 \def\forest@equaltotolerance#1#2{%
8299 \pgfpointdiff{#1}{#2}%
8300 \ifdim\pgf@x<0pt \multiply\pgf@x -1 \fi
8301 \ifdim\pgf@y<0pt \multiply\pgf@y -1 \fi
8302 \global\forest@equaltotolerancefalse
8303 \ifdim\pgf@x<\pgfintersectiontolerance\relax
8304 \ifdim\pgf@y<\pgfintersectiontolerance\relax
8305 \global\forest@equaltolerancetrue
8306 \fi
8307 \fi
8308 }}

```

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

```

8309 \def\forest@save@pgfsyssoftpath@tokendefs{%
8310 \let\forest@origmovetotoken\pgfsyssoftpath@movetotoken
8311 \let\forest@origlinetotoken\pgfsyssoftpath@linetotoken
8312 \let\forest@origcurvetosupportatoken\pgfsyssoftpath@curvetosupportatoken
8313 \let\forest@origcurvetosupportbtoken\pgfsyssoftpath@curvetosupportbtoken
8314 \let\forest@origcurvetotoken\pgfsyssoftpath@curvetototoken
8315 \let\forest@origrectcornertoken\pgfsyssoftpath@rectcornertoken
8316 \let\forest@origrectsizetoken\pgfsyssoftpath@rectsizetoken
8317 \let\forest@origclosepathtoken\pgfsyssoftpath@closepathtoken
8318 \let\pgfsyssoftpath@movetotoken\forest@badtoken
8319 \let\pgfsyssoftpath@linetotoken\forest@badtoken
8320 \let\pgfsyssoftpath@curvetosupportatoken\forest@badtoken
8321 \let\pgfsyssoftpath@curvetosupportbtoken\forest@badtoken
8322 \let\pgfsyssoftpath@curvetototoken\forest@badtoken
8323 \let\pgfsyssoftpath@rectcornertoken\forest@badtoken
8324 \let\pgfsyssoftpath@rectsizetoken\forest@badtoken
8325 \let\pgfsyssoftpath@closepathtoken\forest@badtoken
8326 }
8327 \def\forest@badtoken{%
8328 \PackageError{forest}{This token should not be in this path}{}%
8329 }
8330 \def\forest@restore@pgfsyssoftpath@tokendefs{%
8331 \let\pgfsyssoftpath@movetotoken\forest@origmovetotoken
8332 \let\pgfsyssoftpath@linetotoken\forest@origlinetotoken
8333 \let\pgfsyssoftpath@curvetosupportatoken\forest@origcurvetosupportatoken
8334 \let\pgfsyssoftpath@curvetosupportbtoken\forest@origcurvetosupportbtoken
8335 \let\pgfsyssoftpath@curvetototoken\forest@origcurvetotoken
8336 \let\pgfsyssoftpath@rectcornertoken\forest@origrectcornertoken
8337 \let\pgfsyssoftpath@rectsizetoken\forest@origrectsizetoken
8338 \let\pgfsyssoftpath@closepathtoken\forest@origclosepathtoken
8339 }

```

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

```

8340 \def\forest@extendpath#1#2#3{%

```



```

8341 \pgf@process{#3}%
8342 \pgfsyssoftpath@setcurrentpath#1%
8343 \forest@save\pgfsyssoftpath@tokendefs
8344 \let\pgfsyssoftpath@movetotoken\forest@extendpath@moveto
8345 \let\pgfsyssoftpath@linetotoken\forest@extendpath@lineto
8346 #2%
8347 \forest@restore\pgfsyssoftpath@tokendefs
8348 \pgfsyssoftpath@getcurrentpath#1%
8349 }
8350 \def\forest@extendpath@moveto#1#2{%
8351 \forest@extendpath@do{#1}{#2}\pgfsyssoftpath@moveto
8352 }
8353 \def\forest@extendpath@lineto#1#2{%
8354 \forest@extendpath@do{#1}{#2}\pgfsyssoftpath@lineto
8355 }
8356 \def\forest@extendpath@do#1#2#3{%
8357 {%
8358 \advance\pgf@x #1
8359 \advance\pgf@y #2
8360 #3{\the\pgf@x}{\the\pgf@y}%
8361 }%
8362 }

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.)
8363 \def\forest@path@getboundingrectangle@ls#1#2{%
8364 {%
8365 \pgftransformreset
8366 \forest@pgfqtransformrotate{-#2}%
8367 \forest@pgfpathtransformed#1%
8368 }%
8369 \pgfsyssoftpath@getcurrentpath\forest@gbr@rotatedpath
8370 \forest@path@getboundingrectangle@xy\forest@gbr@rotatedpath
8371 }
8372 \def\forest@path@getboundingrectangle@xy#1{%
8373 \forest@save\pgfsyssoftpath@tokendefs
8374 \let\pgfsyssoftpath@movetotoken\forest@gbr@firstpoint
8375 \let\pgfsyssoftpath@linetotoken\forest@gbr@firstpoint
8376 #1%
8377 \forest@restore\pgfsyssoftpath@tokendefs
8378 }
8379 \def\forest@gbr@firstpoint#1#2{%
8380 \pgf@xa=#1 \pgf@xb=#1 \pgf@ya=#2 \pgf@yb=#2
8381 \let\pgfsyssoftpath@movetotoken\forest@gbr@point
8382 \let\pgfsyssoftpath@linetotoken\forest@gbr@point
8383 }
8384 \def\forest@gbr@point#1#2{%
8385 \ifdim#1<\pgf@xa\relax\pgf@xa=#1 \fi
8386 \ifdim#1>\pgf@xb\relax\pgf@xb=#1 \fi
8387 \ifdim#2<\pgf@ya\relax\pgf@ya=#2 \fi
8388 \ifdim#2>\pgf@yb\relax\pgf@yb=#2 \fi
8389 }

```

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.

```

8390 \let\forest@pgfqtransformrotate\pgftransformrotate
8391 \let\forest@pgftransformcm\pgftransformcm
8392 \let\forest@pgf@transformcm\pgf@transformcm
8393 \patchcmd{\forest@pgfqtransformrotate}{\pgfmathparse{#1}}{\edef\pgfmathresult{\number\numexpr#1}}{}{}
8394 \patchcmd{\forest@pgfqtransformrotate}{\pgftransformcm}{\forest@pgftransformcm}{}{}
8395 \patchcmd{\forest@pgftransformcm}{\pgf@transformcm}{\forest@pgf@transformcm}{}{}

```

```

8396 \patchcmd{\forest@pgf@transformcm}{\pgfmathsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x
8397 \patchcmd{\forest@pgf@transformcm}{\pgfmathsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x
8398 \patchcmd{\forest@pgf@transformcm}{\pgfmathsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x
8399 \patchcmd{\forest@pgf@transformcm}{\pgfmathsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x
8400 \def\forest@pgf@transformcm@setlength#1#2{#1=#2pt}

```

## 10 The outer UI

### 10.1 Externalization

```

8401 \pgfkeys{/forest/external/.cd,
8402   %copy command/.initial={cp "\source" "\target"},
8403   copy command/.initial={},
8404   optimize/.is if=forest@external@optimize@,
8405   context/.initial={%
8406     \forestOve{\csname forest@id@of@standard node\endcsname}{environment@formula}},
8407   depends on macro/.style={context/.append/.expanded={%
8408     \expandafter\detokenize\expandafter{#1}}},
8409 }
8410 \def\forest@file@copy#1#2{%
8411   \IfFileExists{#1}{%
8412     \pgfkeysgetvalue{/forest/external/copy command}\forest@copy@command
8413     \ifdefempty\forest@copy@command{%
8414       \forest@file@copy@{#1}{#2}%
8415     }{ % copy by external command
8416       \def\source{#1}%
8417       \def\target{#2}%
8418       \immediate\write18{\forest@copy@command}%
8419     }%
8420   }{}%
8421 }
8422 \newread\forest@copy@in
8423 \newwrite\forest@copy@out
8424 \def\forest@file@copy@#1#2{%
8425   \begingroup
8426   \openin\forest@copy@in=#1
8427   \immediate\openout\forest@copy@out#2
8428   \endlinechar-1
8429   \loop
8430   \unless\ifeof\forest@copy@in
8431     \readline\forest@copy@in to\forest@temp
8432     \immediate\write\forest@copy@out{\forest@temp}%
8433   \repeat
8434   \immediate\closeout\forest@copy@out
8435   \closein\forest@copy@in
8436   \endgroup
8437 }
8438 \newif\ifforest@external@optimize@
8439 \forest@external@optimize@true
8440 \ifforest@install@keys@to@tikz@path@
8441 \tikzset{
8442   fit to/.style={
8443     /forest/for nodewalk=%
8444     {TeX={\def\forest@fitto{}}},#1}%
8445     {TeX={\eappto\forest@fitto{(\forestove{name})}}},
8446     fit/.expanded={\forest@fitto}
8447   },
8448 }
8449 \fi
8450 \ifforest@external@

```

```

8451 \ifdefined\tikzexternal\tikz@replacement\else
8452 \usetikzlibrary{external}%
8453 \fi
8454 \pgfkeys{%
8455 /tikz/external/failed ref warnings for={},
8456 /pgf/images/aux in dpth=false,
8457 }%
8458 \tikzifexternalizing{}{%
8459 \forest@file@copy{\jobname.aux}{\jobname.aux.copy}%
8460 }%
8461 \AtBeginDocument{%
8462 \tikzifexternalizing{%
8463 \IfFileExists{\tikzexternalrealjob.aux.copy}{%
8464 \makeatletter
8465 \input\tikzexternalrealjob.aux.copy\relax
8466 \makeatother
8467 }{}%
8468 }{%
8469 \newwrite\forest@auxout
8470 \immediate\openout\forest@auxout=\tikzexternalrealjob.for.tmp
8471 }%
8472 \IfFileExists{\tikzexternalrealjob.for}{%
8473 {%
8474 \makehashother\makeatletter
8475 \input\tikzexternalrealjob.for\relax
8476 }%
8477 }{}%
8478 }%
8479 \AtEndDocument{%
8480 \tikzifexternalizing{}{%
8481 \immediate\closeout\forest@auxout
8482 \forest@file@copy{\jobname.for.tmp}{\jobname.for}%
8483 }%
8484 }%
8485 \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.

```

8486 \NewDocumentEnvironment{forest}{D(){} }{%
8487 \forest@config{#1}%
8488 \Collect@Body
8489 \forest@env
8490 }{}
8491 \NewDocumentCommand{\Forest}{s D(){} m}{%
8492 \forest@config{#2}%
8493 \IfBooleanTF{#1}{\let\forest@next\forest@env}{\let\forest@next\forest@group@env}%
8494 \forest@next{#3}%
8495 }
8496 \def\forest@config#1{%
8497 \forest@defstages{stages}%
8498 \forestset{@config/.cd,#1}%
8499 }
8500 \def\forest@defstages#1{%
8501 \def\forest@stages{#1}%
8502 }
8503 \forestset{@config/.cd,
8504 %stages/.store in=\forest@stages,
8505 stages/.code={\forest@defstages{#1}},

```

```

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

```

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

```

8540 \long\def\forest@begin@externalizing#1{%
8541   \forest@external@setup{#1}%
8542   \let\forest@external@next\forest@begin
8543   \forest@externalize@inner@n=-1
8544   \ifforest@external@optimize@\forest@externalizing@maybeoptimize\fi
8545   \forest@external@next{#1}%
8546   \tikzexternalenable
8547 }
8548 \def\forest@externalizing@maybeoptimize{%
8549   \edef\forest@temp{\tikzexternalrealjob-forest-\forest@externalize@outer@n}%
8550   \edef\forest@marshal{%
8551     \noexpand\pgfutil@in@
8552     {\expandafter\detokenize\expandafter{\forest@temp}.}
8553     {\expandafter\detokenize\expandafter{\pgfactualjobname}.}%
8554   }\forest@marshal
8555   \ifpgfutil@in@
8556   \else
8557     \let\forest@external@next\@gobble
8558   \fi
8559 }

```

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

```

8560 \long\def\forest@begin@externalize#1{%
8561   \forest@external@setup{#1}%
8562   \iftikzexternal@file@isuptodate

```

```

8563 \setbox0=\hbox{%
8564 \csname forest@externalcheck@\forest@externalize@outer@n\endcsname
8565 }%
8566 \fi
8567 \iftikzexternal@file@isuptodate
8568 \csname forest@externalload@\forest@externalize@outer@n\endcsname
8569 \else
8570 \forest@externalize@tree@true
8571 \forest@externalize@inner@n=-1
8572 \forest@begin{#1}%
8573 \ifcsdef{forest@externalize@@\forest@externalize@id}{}%
8574 \immediate\write\forest@auxout{%
8575 \noexpand\forest@external
8576 {\forest@externalize@outer@n}%
8577 {\expandafter\detokenize\expandafter{\forest@externalize@id}}%
8578 {\expandonce\forest@externalize@checkimages}%
8579 {\expandonce\forest@externalize@loadimages}%
8580 }%
8581 }%
8582 \fi
8583 \tikzexternalenable
8584 }
8585 \def\forest@includeexternal@check#1{%
8586 \tikzsetnextfilename{#1}%
8587 \IfFileExists{\tikzexternal@filenameprefix/#1}{\tikzexternal@file@isuptodatetrue}{\tikzexternal@file@isuptodatefalse}
8588 }
8589 \def\makehashother{\catcode'\# =12}%
8590 \long\def\forest@external@setup#1{%
8591 % set up \forest@externalize@id and \forest@externalize@outer@n
8592 % we need to deal with #s correctly (\write doubles them)
8593 \setbox0=\hbox{\makehashother\makeatletter
8594 \scantokens{\forest@temp@toks{#1}}\expandafter
8595 }%
8596 \expandafter\forest@temp@toks\expandafter{\the\forest@temp@toks}%
8597 \edef\forest@temp{\pgfkeysvalueof{/forest/external/context}}%
8598 \edef\forest@externalize@id{%
8599 \expandafter\detokenize\expandafter{\forest@temp}%
8600 @@%
8601 \expandafter\detokenize\expandafter{\the\forest@temp@toks}%
8602 }%
8603 \letcs\forest@externalize@outer@n{forest@externalize@@\forest@externalize@id}%
8604 \ifdefined\forest@externalize@outer@n
8605 \global\tikzexternal@file@isuptodatetrue
8606 \else
8607 \global\advance\forest@externalize@max@outer@n 1
8608 \edef\forest@externalize@outer@n{\the\forest@externalize@max@outer@n}%
8609 \global\tikzexternal@file@isuptodatefalse
8610 \fi
8611 \def\forest@externalize@loadimages{}%
8612 \def\forest@externalize@checkimages{}%
8613 }
8614 \newcount\forest@externalize@max@outer@n
8615 \global\forest@externalize@max@outer@n=0
8616 \newcount\forest@externalize@inner@n

```

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

```

8617 \long\def\forest@external#1#2#3#4{% #1=n,#2=context+source code,#3=update check code, #4=load code
8618 \ifnum\forest@externalize@max@outer@n<#1
8619 \global\forest@externalize@max@outer@n=#1
8620 \fi
8621 \global\csdef{forest@externalize@@\detokenize{#2}}{#1}%

```

```

8622 \global\csdef{forest@externalcheck@#1}{#3}%
8623 \global\csdef{forest@externalload@#1}{#4}%
8624 \tikzifexternalizing{}{%
8625   \immediate\write\forest@auxout{%
8626     \noexpand\forest@external{#1}%
8627     \expandafter\detokenize\expandafter{#2}}}%
8628   \unexpanded{#3}}%
8629   \unexpanded{#4}}%
8630 }%
8631 }%
8632 }

```

```

8633 \def\forest@includeexternal#1{%
8634   \edef\forest@temp{\pgfkeysvalueof{/forest/external/context}}%
8635   %\typeout{forest: Including external picture '#1' for forest context+code: '\expandafter\detokenize\expandafter\includeexternal'#1}%
8636   {%
8637     %\def\pgf@declaredraftimage##1##2{\def\pgf@image{\hbox{}}}%
8638     \tikzsetnextfilename{#1}%
8639     \tikzexternalenable
8640     \tikz{}%
8641   }%
8642 }
8643 \def\forest@includeexternal@box#1#2{%
8644   \global\setbox#1=\hbox{\forest@includeexternal{#2}}%
8645 }

```

```

8646 \long\def\forest@begin#1{%
8647   \iffalse{\fi\forest@parsebracket#1}%
8648 }
8649 \def\forest@parsebracket{%
8650   \bracketParse{\forest@get@root@afterthought}\forest@root=%
8651 }
8652 \def\forest@get@root@afterthought{%
8653   \expandafter\forest@get@root@afterthought\expandafter{\iffalse}\fi
8654 }
8655 \long\def\forest@get@root@afterthought#1{%
8656   \ifblank{#1}{-}{%
8657     \forest@oappto{\forest@root}{given options}{,afterthought={\unexpanded{#1}}}%
8658   }%
8659   \forest@do
8660 }
8661 \def\forest@do{%
8662   \forest@node@Compute@numeric@ts@info{\forest@root}%
8663   \expandafter\forestset\expandafter{\forest@stages}%
8664   \ifforest@was@tikzexternalwasenable
8665     \tikzexternalenable
8666   \fi
8667 }

```

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

```

8674 }%
8675 }
8676 \def\forest@standardnode@calibrate{%
8677   \forest@fornode{\forest@node@Nametoid{standard node}}{%
8678     \edef\forest@environment{\forest@ve{environment@formula}}%
8679     \forest@toget{previous@environment}\forest@previous@environment
8680     \ifx\forest@environment\forest@previous@environment\else
8681       \forest@tolet{previous@environment}\forest@environment
8682       \forest@node@typeset
8683       \forest@toget{calibration@procedure}\forest@temp
8684       \expandafter\forestset\expandafter{\forest@temp}%
8685     \fi
8686 }%
8687 }

```

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  $\text{\TeX}$  group.)

```

8688 \def\forestStandardNode[#1]#2#3#4{%
8689   \let\forest@standardnode@restoretikzexternal\relax
8690   \ifdefined\tikzexternaldisable
8691     \ifx\tikz\tikzexternal\tikz@replacement
8692       \tikzexternaldisable
8693       \let\forest@standardnode@restoretikzexternal\tikzexternalenable
8694     \fi
8695   \fi
8696   \forest@standardnode@new
8697   \forest@fornode{\forest@node@Nametoid{standard node}}{%
8698     \forestset{content=#1}%
8699     \forest@set{environment@formula}{#2}%
8700     \edef\forest@temp{\unexpanded{#3}}%
8701     \forest@tolet{calibration@procedure}\forest@temp
8702     \def\forest@calibration@initializing@code{%
8703       \pgfkeys{/forest/initializing@code}{#4}%
8704       \forest@tolet{initializing@code}\forest@calibration@initializing@code
8705     \forest@standardnode@restoretikzexternal
8706   }
8707 }
8708 \forestset{initializing@code/.unknown/.code={%
8709   \eappto\forest@calibration@initializing@code{%
8710     \noexpand\forest@toget{\forest@node@Nametoid{standard node}}{\pgfkeyscurrentname}\noexpand\forest@temp
8711     \noexpand\forest@tolet{\pgfkeyscurrentname}\noexpand\forest@temp
8712   }%
8713 }
8714 }

```

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

```

8715 \def\forest@initializefromstandardnode{%
8716   \forest@ve{\forest@node@Nametoid{standard node}}{initializing@code}%
8717 }

```

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

```

8718 \forestStandardNode[dj]
8719 {%
8720   \forestOve{\forest@node@Nametoid{standard node}}{content},%
8721   \the\ht\strutbox,\the\pgflinewidth,%
8722   \pgfkeysvalueof{/pgf/inner ysep},\pgfkeysvalueof{/pgf/outer ysep},%
8723   \pgfkeysvalueof{/pgf/inner xsep},\pgfkeysvalueof{/pgf/outer xsep}%
8724 }
8725 {
8726   l sep'/.expanded={\the\dimexpr\the\ht\strutbox+\pgfkeysvalueof{/pgf/inner ysep}},
8727   l={l_sep()+abs(max_y()-min_y())+2*\pgfkeysvalueof{/pgf/outer ysep}},
8728   s sep'/.expanded={\the\dimexpr \pgfkeysvalueof{/pgf/inner xsep}*2}
8729 }
8730 {l sep,l,s sep}

```

## 10.4 ls coordinate system

```

8731 \pgfkeys{/forest/@cs}{%
8732   name/.code={%
8733     \edef\forest@cn{\forest@node@Nametoid{#1}}%
8734     \forest@forestcs@resetxy},
8735   id/.code={%
8736     \edef\forest@cn{#1}%
8737     \forest@forestcs@resetxy},
8738   go/.code={%
8739     \forest@go{#1}%
8740     \forest@forestcs@resetxy},
8741   anchor/.code={\forest@forestcs@anchor{#1}},
8742   l/.code={%
8743     \forestmathsetlengthmacro\forest@forestcs@l{#1}%
8744     \forest@forestcs@ls
8745   },
8746   s/.code={%
8747     \forestmathsetlengthmacro\forest@forestcs@s{#1}%
8748     \forest@forestcs@ls
8749   },
8750   .unknown/.code={%
8751     \expandafter\pgfutil@in@\expandafter.\expandafter{\pgfkeyscurrentname}%
8752     \ifpgfutil@in@
8753       \expandafter\forest@forestcs@namegoanchor\pgfkeyscurrentname\forest@end
8754     \else
8755       \expandafter\forest@nameandgo\expandafter{\pgfkeyscurrentname}%
8756       \forest@forestcs@resetxy
8757     \fi
8758   }
8759 }
8760 \def\forest@forestcs@resetxy{%
8761   \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
8762   \global\pgf@x\forestove{x}\relax
8763   \global\pgf@y\forestove{y}\relax
8764 }
8765 \def\forest@forestcs@ls{%
8766   \ifdefined\forest@forestcs@l
8767     \ifdefined\forest@forestcs@s
8768       {%
8769         \pgftransformreset
8770         \forest@pgfqtransformrotate{\forestove{grow}}%
8771         \pgfpointransformed{\pgfpqpoint{\forest@forestcs@l}{\forest@forestcs@s}}%

```



```

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

```

## 10.5 Relative node names in TikZ

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

```

8800 \def\forest@tikz@parse@node#1(#2){%
8801     \pgfutil@in@.#2}%
8802     \ifpgfutil@in@
8803         \expandafter\forest@tikz@parse@node@checkiftikzname@withdot
8804     \else%
8805         \expandafter\forest@tikz@parse@node@checkiftikzname@withoutdot
8806     \fi%
8807     #1(#2)\forest@end
8808 }
8809 \def\forest@tikz@parse@node@checkiftikzname@withdot#1(#2.#3)\forest@end{%
8810     \forest@tikz@parse@node@checkiftikzname#1{#2}{.#3}}
8811 \def\forest@tikz@parse@node@checkiftikzname@withoutdot#1(#2)\forest@end{%
8812     \forest@tikz@parse@node@checkiftikzname#1{#2}{}}
8813 \def\forest@tikz@parse@node@checkiftikzname#1#2#3{%
8814     \expandafter\ifx\csname pgf@sh@ns@#2\endcsname\relax
8815         \forest@forthis{%
8816             \forest@nameandgo{#2}%
8817             \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
8818             \edef\forest@temp@relativenodename{\forestove{name}}}%
8819         }%
8820     \else
8821         \def\forest@temp@relativenodename{#2}%
8822     \fi
8823     \expandafter\forest@original@tikz@parse@node\expandafter#1\expandafter(\forest@temp@relativenodename#3)%
8824 }
8825 \def\forest@nameandgo#1{%
8826     \pgfutil@in@!{#1}%
8827     \ifpgfutil@in@
8828         \forest@nameandgo@(#1)%

```

```

8829 \else
8830 \ifstrempy{#1}{\edef\forest@cn{\forest@node@Nametoid{#1}}}%
8831 \fi
8832 }
8833 \def\forest@nameandgo@(#1!#2){%
8834 \ifstrempy{#1}{\edef\forest@cn{\forest@node@Nametoid{#1}}}%
8835 \forest@go{#2}%
8836 }

```

## 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.

```

8837 \forestset{
8838   declare count={rotate}{0},
8839   autoforward'={rotate}{node options},
8840 }

```

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

```

8841 \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.

```

8842 \newif\ifforest@anchor@forwardtotikz

```

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

```

8843 \newif\ifforest@anchor@isborder

```

The UI macro.

```

8844 \def\forestanchortotikzanchor#1#2{% #1 = forest anchor, #2 = macro to receive the tikz anchor
8845 \forest@anchor@forwardtotikzfalse
8846 \forest@anchor@do{#1}{\forest@cn}%
8847 \let#2\forest@temp@anchor
8848 }

```

Generic anchors.

```

8849 \pgfdeclaregenericanchor{child anchor}{%
8850 \forest@anchor@forwardtotikztrue
8851 \forest@anchor@do{#1}{child anchor}{\forest@referencednodeid}%
8852 }
8853 \pgfdeclaregenericanchor{parent anchor}{%
8854 \forest@anchor@forwardtotikztrue
8855 \forest@anchor@do{#1}{parent anchor}{\forest@referencednodeid}%
8856 }
8857 \pgfdeclaregenericanchor{anchor}{%
8858 \forest@anchor@forwardtotikztrue
8859 \forest@anchor@do{#1}{anchor}{\forest@referencednodeid}%
8860 }
8861 \pgfdeclaregenericanchor{children}{%
8862 \forest@anchor@forwardtotikztrue
8863 \forest@anchor@do{#1}{children}{\forest@referencednodeid}%
8864 }
8865 \pgfdeclaregenericanchor{-children}{%
8866 \forest@anchor@forwardtotikztrue
8867 \forest@anchor@do{#1}{-children}{\forest@referencednodeid}%
8868 }
8869 \pgfdeclaregenericanchor{children first}{%
8870 \forest@anchor@forwardtotikztrue
8871 \forest@anchor@do{#1}{children first}{\forest@referencednodeid}%

```

```

8872 }
8873 \pgfdeclaregenericanchor{-children first}{%
8874   \forest@anchor@forwardtotikztrue
8875   \forest@anchor@do{#1}{-children first}{\forest@referencednodeid}%
8876 }
8877 \pgfdeclaregenericanchor{first}{%
8878   \forest@anchor@forwardtotikztrue
8879   \forest@anchor@do{#1}{first}{\forest@referencednodeid}%
8880 }
8881 \pgfdeclaregenericanchor{parent first}{%
8882   \forest@anchor@forwardtotikztrue
8883   \forest@anchor@do{#1}{parent first}{\forest@referencednodeid}%
8884 }
8885 \pgfdeclaregenericanchor{-parent first}{%
8886   \forest@anchor@forwardtotikztrue
8887   \forest@anchor@do{#1}{-parent first}{\forest@referencednodeid}%
8888 }
8889 \pgfdeclaregenericanchor{parent}{%
8890   \forest@anchor@forwardtotikztrue
8891   \forest@anchor@do{#1}{parent}{\forest@referencednodeid}%
8892 }
8893 \pgfdeclaregenericanchor{-parent}{%
8894   \forest@anchor@forwardtotikztrue
8895   \forest@anchor@do{#1}{-parent}{\forest@referencednodeid}%
8896 }
8897 \pgfdeclaregenericanchor{parent last}{%
8898   \forest@anchor@forwardtotikztrue
8899   \forest@anchor@do{#1}{parent last}{\forest@referencednodeid}%
8900 }
8901 \pgfdeclaregenericanchor{-parent last}{%
8902   \forest@anchor@forwardtotikztrue
8903   \forest@anchor@do{#1}{-parent last}{\forest@referencednodeid}%
8904 }
8905 \pgfdeclaregenericanchor{last}{%
8906   \forest@anchor@forwardtotikztrue
8907   \forest@anchor@do{#1}{last}{\forest@referencednodeid}%
8908 }
8909 \pgfdeclaregenericanchor{children last}{%
8910   \forest@anchor@forwardtotikztrue
8911   \forest@anchor@do{#1}{children last}{\forest@referencednodeid}%
8912 }
8913 \pgfdeclaregenericanchor{-children last}{%
8914   \forest@anchor@forwardtotikztrue
8915   \forest@anchor@do{#1}{-children last}{\forest@referencednodeid}%
8916 }
8917 \pgfdeclaregenericanchor{children'}{%
8918   \forest@anchor@forwardtotikztrue
8919   \forest@anchor@do{#1}{children'}{\forest@referencednodeid}%
8920 }
8921 \pgfdeclaregenericanchor{-children'}{%
8922   \forest@anchor@forwardtotikztrue
8923   \forest@anchor@do{#1}{-children'}{\forest@referencednodeid}%
8924 }
8925 \pgfdeclaregenericanchor{children first'}{%
8926   \forest@anchor@forwardtotikztrue
8927   \forest@anchor@do{#1}{children first'}{\forest@referencednodeid}%
8928 }
8929 \pgfdeclaregenericanchor{-children first'}{%
8930   \forest@anchor@forwardtotikztrue
8931   \forest@anchor@do{#1}{-children first'}{\forest@referencednodeid}%
8932 }

```

```

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

```

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

```

8973 \def\forest@anchor@do#1#2#3{% #1 = shape name, #2 = (potentially) forest anchor, #3 = node id
8974   \forest@for node{#3}{%
8975     \def\forest@temp@anchor{#2}%
8976     \forest@anchor@snapbordertocompassfalse
8977     \forest@anchor@isborderfalse
8978     \forest@anchor@to@tikz@anchor
8979     \forest@anchor@border@to@compass
8980     \ifforest@anchor@forwardtotikz
8981       \forest@anchor@forward{#1}%
8982     \else
8983       \fi
8984   }%
8985 }

```

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

```

8986 \def\forest@anchor@to@tikz@anchor{%
8987   \ifcsdef{forest@anchor@@\forest@temp@anchor}{%
8988     \csuse{forest@anchor@@\forest@temp@anchor}%
8989     \forest@anchor@to@tikz@anchor
8990   }{}%

```

8991 }

Actual computation.

```
8992 \csdef{forest@anchor@@parent anchor}{%
8993   \foresttoget{parent anchor}\forest@temp@anchor}
8994 \csdef{forest@anchor@@child anchor}{%
8995   \foresttoget{child anchor}\forest@temp@anchor}
8996 \csdef{forest@anchor@@anchor}{%
8997   \foresttoget{anchor}\forest@temp@anchor}
8998 \csdef{forest@anchor@@children'}{%
8999   \forest@anchor@isbordertrue
9000   \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}}}%
9001 }
9002 \csdef{forest@anchor@@-children'}{%
9003   \forest@anchor@isbordertrue
9004   \edef\forest@temp@anchor{\number\numexpr 180+\forestove{grow}-\forestove{rotate}}}%
9005 }
9006 \csdef{forest@anchor@@parent'}{%
9007   \forest@anchor@isbordertrue
9008   \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestove{\forestove{@parent}}\gro
9009   \edef\forest@temp@anchor{\number\numexpr\forest@temp@grow-\forestove{rotate}+180}%
9010 }
9011 \csdef{forest@anchor@@-parent'}{%
9012   \forest@anchor@isbordertrue
9013   \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestove{\forestove{@parent}}\gro
9014   \edef\forest@temp@anchor{\number\numexpr\forest@temp@grow-\forestove{rotate}}}%
9015 }
9016 \csdef{forest@anchor@@first'}{%
9017   \forest@anchor@isbordertrue
9018   \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=0 -\ve
9019 }
9020 \csdef{forest@anchor@@last'}{%
9021   \forest@anchor@isbordertrue
9022   \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=0 +\ve
9023 }
9024 \csdef{forest@anchor@@parent first'}{%
9025   \forest@anchor@isbordertrue
9026   \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestove{\forestove{@parent}}\gro
9027   \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestove{\forestove{@pare
9028   \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}+180}%
9029   \edef\forest@temp@anchor@first{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@revers
9030   \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@first}\forest@temp@anchor
9031 }
9032 \csdef{forest@anchor@@-parent first'}{%
9033   \forest@anchor@isbordertrue
9034   \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestove{\forestove{@parent}}\gro
9035   \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestove{\forestove{@pare
9036   \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}}}%
9037   \edef\forest@temp@anchor@first{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@revers
9038   \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@first}\forest@temp@anchor
9039 }
9040 \csdef{forest@anchor@@parent last'}{%
9041   \forest@anchor@isbordertrue
9042   \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestove{\forestove{@parent}}\gro
9043   \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestove{\forestove{@pare
9044   \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}+180}%
9045   \edef\forest@temp@anchor@last{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@reverse
9046   \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@last}\forest@temp@anchor
9047 }
9048 \csdef{forest@anchor@@-parent last'}{%
9049   \forest@anchor@isbordertrue
```

```

9050 \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestove{\forestove{@parent}}\gro
9051 \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestove{\forestove{@pare
9052 \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}}}%
9053 \edef\forest@temp@anchor@last{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@reverse
9054 \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@last}\forest@temp@anchor
9055 }
9056 \csdef{forest@anchor@@children first'}{%
9057 \forest@anchor@isbordertrue
9058 \edef\forest@temp@anchor@first{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}
9059 \forest@getaverageangle{\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@first}\forest@temp@anchor
9060 }
9061 \csdef{forest@anchor@@-children first'}{%
9062 \forest@anchor@isbordertrue
9063 \edef\forest@temp@anchor@first{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}
9064 \forest@getaverageangle{180+\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@first}\forest@temp@anch
9065 }
9066 \csdef{forest@anchor@@children last'}{%
9067 \forest@anchor@isbordertrue
9068 \edef\forest@temp@anchor@last{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=
9069 \forest@getaverageangle{\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@last}\forest@temp@anchor
9070 }
9071 \csdef{forest@anchor@@-children last'}{%
9072 \forest@anchor@isbordertrue
9073 \edef\forest@temp@anchor@last{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=
9074 \forest@getaverageangle{180+\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@last}\forest@temp@anch
9075 }
9076 \csdef{forest@anchor@@children}{%
9077 \forest@anchor@snapbordertocompasstrue
9078 \csuse{forest@anchor@@children'}%
9079 }
9080 \csdef{forest@anchor@@-children}{%
9081 \forest@anchor@snapbordertocompasstrue
9082 \csuse{forest@anchor@@-children'}%
9083 }
9084 \csdef{forest@anchor@@parent}{%
9085 \forest@anchor@snapbordertocompasstrue
9086 \csuse{forest@anchor@@parent'}%
9087 }
9088 \csdef{forest@anchor@@-parent}{%
9089 \forest@anchor@snapbordertocompasstrue
9090 \csuse{forest@anchor@@-parent'}%
9091 }
9092 \csdef{forest@anchor@@first}{%
9093 \forest@anchor@snapbordertocompasstrue
9094 \csuse{forest@anchor@@first'}%
9095 }
9096 \csdef{forest@anchor@@last}{%
9097 \forest@anchor@snapbordertocompasstrue
9098 \csuse{forest@anchor@@last'}%
9099 }
9100 \csdef{forest@anchor@@parent first}{%
9101 \forest@anchor@snapbordertocompasstrue
9102 \csuse{forest@anchor@@parent first'}%
9103 }
9104 \csdef{forest@anchor@@-parent first}{%
9105 \forest@anchor@snapbordertocompasstrue
9106 \csuse{forest@anchor@@-parent first'}%
9107 }
9108 \csdef{forest@anchor@@parent last}{%
9109 \forest@anchor@snapbordertocompasstrue
9110 \csuse{forest@anchor@@parent last'}%

```

```

9111 }
9112 \csdef{forest@anchor@@-parent last}{%
9113   \forest@anchor@snapbordertocompasstrue
9114   \csuse{forest@anchor@@-parent last'}%
9115 }
9116 \csdef{forest@anchor@@children first}{%
9117   \forest@anchor@snapbordertocompasstrue
9118   \csuse{forest@anchor@@children first'}%
9119 }
9120 \csdef{forest@anchor@@-children first}{%
9121   \forest@anchor@snapbordertocompasstrue
9122   \csuse{forest@anchor@@-children first'}%
9123 }
9124 \csdef{forest@anchor@@children last}{%
9125   \forest@anchor@snapbordertocompasstrue
9126   \csuse{forest@anchor@@children last'}%
9127 }
9128 \csdef{forest@anchor@@-children last}{%
9129   \forest@anchor@snapbordertocompasstrue
9130   \csuse{forest@anchor@@-children last'}%
9131 }

```

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>.

```

9132 \def\forest@getaverageangle#1#2#3{%
9133   \edef\forest@temp{\number\numexpr #1-#2+540}%
9134   \expandafter\pgfmathMod@\expandafter{\forest@temp}{360}%
9135   \forest@truncatepgfmathresult
9136   \edef\forest@temp{\number\numexpr 360+#2+((\pgfmathresult-180)/2)}%
9137   \expandafter\pgfmathMod@\expandafter{\forest@temp}{360}%
9138   \forest@truncatepgfmathresult
9139   \let#3\pgfmathresult
9140 }
9141 \def\forest@truncatepgfmathresult{%
9142   \afterassignment\forest@gobbletoEND
9143   \forest@temp@count=\pgfmathresult\forest@END
9144   \def\pgfmathresult{\the\forest@temp@count}%
9145 }
9146 \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.

```

9147 \def\forest@anchor@border@to@compass{%
9148   \ifforest@anchor@isborder % snap to 45 deg, to range 0-360
9149     \ifforest@anchor@snapbordertocompass
9150       \forest@anchor@snap@border@to@compass
9151     \else % to range 0-360
9152       \pgfmathMod@{\forest@temp@anchor}{360}%
9153       \forest@truncatepgfmathresult
9154       \let\forest@temp@anchor\pgfmathresult
9155     \fi
9156     \ifforest@anchor@snapbordertocompass
9157       \ifforest@anchor@forwardtotikz
9158         \ifcsname pgf@anchor%
9159           @\csname pgf@sh@ns@\pgfreferencednodename\endcsname
9160           @\csname forest@compass@\forest@temp@anchor\endcsname
9161         \endcsname
9162         \letcs\forest@temp@anchor{forest@compass@\forest@temp@anchor}%
9163       \fi
9164     \else

```

```

9165      \letcs\forest@temp@anchor{forest@compass@\forest@temp@anchor}%
9166      \fi
9167      \fi
9168      \fi
9169 }
9170 \csdef{forest@compass@0}{east}
9171 \csdef{forest@compass@45}{north east}
9172 \csdef{forest@compass@90}{north}
9173 \csdef{forest@compass@135}{north west}
9174 \csdef{forest@compass@180}{west}
9175 \csdef{forest@compass@225}{south west}
9176 \csdef{forest@compass@270}{south}
9177 \csdef{forest@compass@315}{south east}
9178 \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).

```

9179 \def\forest@anchor@snap@border@to@compass{%
9180   \pgfmathMod@{\forest@temp@anchor}{360}%
9181   \pgfmathdivide@{\pgfmathresult}{45}%
9182   \pgfmathround@{\pgfmathresult}%
9183   \pgfmathmultiply@{\pgfmathresult}{45}%
9184   \forest@truncatepgfmathresult
9185   \let\forest@temp@anchor\pgfmathresult
9186 }

```

This macro forwards the resulting anchor to TikZ.

```

9187 \def\forest@anchor@forward#1{% #1 = shape name
9188   \ifdefempty\forest@temp@anchor{%
9189     \pgf@sh@reanchor{#1}{center}%
9190     \xdef\forest@hack@tikzshapeborder{%
9191       \noexpand\tikz@shapebordertrue
9192       \def\noexpand\tikz@shapeborder@name{\pgfreferencednodename}%
9193     }\aftergroup\forest@hack@tikzshapeborder
9194   }{%
9195     \pgf@sh@reanchor{#1}{\forest@temp@anchor}%
9196   }%
9197 }

```

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

```

9198 \def\forest@referencednodeid{\forest@node@Nametoid{\forest@referencednodename}}%
9199 \def\forest@referencednodename{%
9200   \expandafter\expandafter\expandafter\forest@referencednodename@\expandafter\pgfreferencednodename\forest@pgf@notyetpositioned
9201 }
9202 \expandafter\def\expandafter\forest@referencednodename@\expandafter#\expandafter1\forest@pgf@notyetpositioned
9203   \if\relax#1\relax\forest@referencednodename@stripafter#2\relax\fi
9204   \if\relax#2\relax#1\fi
9205 }
9206 \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.

```

9207 \def\forest@pointanchor#1{% #1 = anchor
9208   \forest@Pointanchor{\forest@cn}{#1}%
9209 }
9210 \def\forest@Pointanchor#1#2{% #1 = node id, #2 = anchor
9211   \def\forest@pa@temp@name{name}%
9212   \forestOifdefined{#1}{@box}{%
9213     \forestOget{#1}{@box}\forest@temp
9214     \ifdefempty\forest@temp{}{%
9215       \def\forest@pa@temp@name{later@name}%

```



```

9216     }%
9217   }{}%
9218   \setbox0\hbox{%
9219     \begin{pgfpicture}%
9220       \if\relax\forestOve{#1}{#2}\relax
9221         \pgfpointanchor{\forestOve{#1}{\forest@pa@temp@name}}{center}%
9222       \else
9223         \pgfpointanchor{\forestOve{#1}{\forest@pa@temp@name}}{\forestOve{#1}{#2}}%
9224       \fi
9225       \xdef\forest@global@marshal{%
9226         \noexpand\global\noexpand\pgf@x=\the\pgf@x\relax
9227         \noexpand\global\noexpand\pgf@y=\the\pgf@y\relax\relax
9228       }%
9229     \end{pgfpicture}%
9230   }%
9231   \forest@global@marshal
9232 }

```

## 11 Compatibility with previous versions

```

9233 \ifdefempty{\forest@compat}{}{%
9234   \RequirePackage{forest-compat}
9235 }

```