

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

v2.1.1

Sašo Živanović*

December 18, 2016

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

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

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

Contents

1 Identification

```
1 \ProvidesPackage{forest}[2016/12/18 v2.1.1 Drawing (linguistic) trees]
2
3 \RequirePackage{tikz}[2013/12/13]
4 \usetikzlibrary{shapes}
5 \usetikzlibrary{fit}
6 \usetikzlibrary{calc}
7 \usepgflibrary{intersections}
8
9 \RequirePackage{pgfopts}
10 \RequirePackage{etoolbox}[2010/08/21]
11 \RequirePackage{elocalloc}% for \locbox
12 \RequirePackage{environ}
13 \RequirePackage{xparse}
14
15 \RequirePackage{inlinedef}
16 \newtoks\ID@usercommands{}
17 \newcommand\NewInlineCommand[3][0]{%
18   \newcommand#2[#1]{#3}%
19   \ID@usercommands\xa{%
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
```

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

```

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

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

```

2 Package options

```

85 \newif\ifforest@external@
86 \newif\ifforest@tikz@
87 \newif\ifforest@install@

```

```

88 \newif\ifforestdbugnodewalks
89 \newif\ifforestdbugdynamics
90 \newif\ifforestdbugprocess
91 \newif\ifforestdbugtemp
92 \newif\ifforestdbug
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\pgfqkeys{/forest/package@options/debug}{#1}},
107   debug/.default={nodewalks,dynamics,process},
108   debug/nodewalks/.is if=forestdbugnodewalks,
109   debug/dynamics/.is if=forestdbugdynamics,
110   debug/process/.is if=forestdbugprocess,
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{\ifcsedef{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}{#4}%
140   \csdef{forest@patched@#1}{#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 `\...loop...` 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}{####1relax\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     \edef\safeloop@marshal{%
196       \noexpand\csdef{safeiterate@#1}{%
197         \unexpanded{##1}\relax
198         \noexpand\expandafter
199         \expandonce{\csname safeiterate@#1\endcsname}%
200         \noexpand\fi
201       }%
202     }\safeloop@marshal
203     \csuse{safeiterate@#1}%
204     \advance\nsafeloop@depth-1\relax
205     \cslet{safeiterate@#1}\relax
206   }%
207 }%
208 \newcount\safeloop@depth
209 \def\safeloop{%
210   \advance\safeloop@depth1
211   \ifcsdef{safeloop@the\safeloop@depth}{\expandafter\nnewsafeloop\expandafter{\the\safeloop@depth}}%
212   \csdef{safeloopn@the\safeloop@depth}{0}%
213   \csuse{safeloop@the\safeloop@depth}%
214   \csdef{safeloopn@the\safeloop@depth}{\number\numexpr\csuse{safeloopn@the\safeloop@depth}+1}%
215 }
216 \let\saferepeat\fi
217 \def\safeloopn{\csuse{safeloopn@the\safeloop@depth}}%

```

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

```

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

```

Additional loops (for embedding).

245 \newloop\forest@loop
 New counters, dimens, ifs.

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

Appending and prepending to token lists.

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

Expanding number arguments.

262 \def\expandnumberarg#1#2{\expandafter#1\expandafter{\number#2}}
 263 \def\expandtwonumberargs#1#2#3{%
 264 \expandafter\expandtwonumberargs@\expandafter#1\expandafter{\number#3}{#2}}
 265 \def\expandtwonumberargs@#1#2#3{%
 266 \expandafter#1\expandafter{\number#3}{#2}}
 267 \def\expandthreenumberargs#1#2#3#4{%
 268 \expandafter\expandthreenumberargs@\expandafter#1\expandafter{\number#4}{#2}{#3}}
 269 \def\expandthreenumberargs@#1#2#3#4{%
 270 \expandafter\expandthreenumberargs@@\expandafter#1\expandafter{\number#4}{#2}{#3}}
 271 \def\expandthreenumberargs@@#1#2#3#4{%
 272 \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.

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

```

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

```

Additional list macros.

```

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

```

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

```

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

```

Utilities dealing with pgfkeys.

```

353 \def\forest@copycommandkey#1#2{% copies command of #1 into #2

```

```

354 \pgfkeysifdefined{#1/.@cmd}{\%
355   \PackageError{forest}{Key #1 is not a command key}{\%
356 }%
357 \pgfkeysgetvalue{#1/.@cmd}\forest@temp
358 \pgfkeyslet{#2/.@cmd}\forest@temp
359 \pgfkeysifdefined{#1/.@args}{\%
360   \pgfkeysgetvalue{#1/.@args}\forest@temp
361   \pgfkeyslet{#2/.@args}\forest@temp
362 }{}%
363 \pgfkeysifdefined{#1/.@body}{\%
364   \pgfkeysgetvalue{#1/.@body}\forest@temp
365   \pgfkeyslet{#2/.@body}\forest@temp
366 }{}%
367 \pgfkeysifdefined{#1/.@body}{\%
368   \pgfkeysgetvalue{#1/.@body}\forest@temp
369   \pgfkeyslet{#2/.@body}\forest@temp
370 }{}%
371 \pgfkeysifdefined{#1/.@def}{\%
372   \pgfkeysgetvalue{#1/.@def}\forest@temp
373   \pgfkeyslet{#2/.@def}\forest@temp
374 }{}%
375 }
376 \forestset{
377   copy command key/.code 2 args={\forest@copycommandkey{#1}{#2}},
378   autoforward/.code 2 args={\forest@autoforward{#1}{#2={#1={##1}}}{true}},
379   autoforward'/.code 2 args={\forest@autoforward{#1}{#2=#1,#2={#1={##1}}}{true}},
380   Autoforward/.code 2 args={\forest@autoforward{#1}{#2}{true}},
381   autoforward register/.code 2 args={\forest@autoforward{#1}{#2={#1={##1}}}{false}},
382   autoforward register'/.code 2 args={\forest@autoforward{#1}{#2=#1,#2={#1={##1}}}{false}},
383   Autoforward register/.code 2 args={\forest@autoforward{#1}{#2}{false}},
384   copy command key@if it exists/.code 2 args={%
385     \pgfkeysifdefined{#1/.@cmd}{\%
386       \forest@copycommandkey{#1}{#2}%
387     }{}%
388   },
389   unautoforward/.style={
390     typeout={unautoforwarding #1},
391     copy command key@if it exists={/forest/autoforwarded #1}{/forest/#1},
392     copy command key@if it exists={/forest/autoforwarded #1+}{/forest/#1+},
393     copy command key@if it exists={/forest/autoforwarded #1-}{/forest/#1-},
394     copy command key@if it exists={/forest/autoforwarded #1*}{/forest/#1*},
395     copy command key@if it exists={/forest/autoforwarded #1:}{/forest/#1:},
396     copy command key@if it exists={/forest/autoforwarded #1'}{/forest/#1'},
397     copy command key@if it exists={/forest/autoforwarded #1+'}{/forest/#1+'},
398     copy command key@if it exists={/forest/autoforwarded #1-'}{/forest/#1-'},
399     copy command key@if it exists={/forest/autoforwarded #1*'}{/forest/#1*'},
400     copy command key@if it exists={/forest/autoforwarded #1:'}{/forest/#1:'},
401     copy command key@if it exists={/forest/autoforwarded +#1}{/forest/+ #1},
402   },
403   /handlers/.undef/.code={\csundef{pgfk@\pgfkeyscurrentpath}},
404   undef option/.style={
405     /forest/#1/.undef,
406     /forest/#1/.@cmd/.undef,
407     /forest/#1+/.@cmd/.undef,
408     /forest/#1-/.@cmd/.undef,
409     /forest/#1*/.@cmd/.undef,
410     /forest/#1:/.@cmd/.undef,
411     /forest/#1'/.@cmd/.undef,
412     /forest/#1+'/.@cmd/.undef,
413     /forest/#1-'/.@cmd/.undef,
414     /forest/#1*'/.@cmd/.undef,

```



```

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

```

```

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

```

4.1 Arrays

```

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

```

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

```

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

```

```

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

```

```

592 }%
593 }}%
594 % \csdef{#1bottompop}##1{% we need this as \Inline chokes on \let\macro=\relax
595 % \expandafter\Inline\expandafter\def\csname#1bottompop\endcsname##1{% ##1 = receiving cs
596 % \Expand{\csname#1ifempty\endcsname}{%
597 % \forest@array@empty@error{#1}%
598 % }{%
599 % \letcs##1{#1\the\Expand{\csname#1M\endcsname}}%
600 % \ExpandIfT{\forest@temp}\global\advance\Expand{\csname#1M\endcsname 1}%
601 % }%
602 % }%
603 % \csedef{#1bottompop}##1{% ##1 = receiving cs
604 % \expandonce{\csname#1ifempty\endcsname}{%
605 % \noexpand\forest@array@empty@error{#1}%
606 % }{%
607 % \noexpand\letcs\noexpand##1{#1\noexpand\the\expandonce{\csname#1M\endcsname}}%
608 % \ifforest@temp\global\fi\advance\expandonce{\csname#1M\endcsname}1
609 % }%
610 % }%
611 \csedef{#1setappend}##1{% ##1 = definition
612 \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi
613 {#1\noexpand\the\expandonce{\csname#1N\endcsname}}%
614 {\noexpand\unexpanded{##1}}%
615 \ifforest@temp\global\fi\advance\expandonce{\csname#1N\endcsname}1
616 }%
617 \csedef{#1setappend@}##1##2{% ##1 = continue by, ##2 = definition
618 \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi
619 {#1\noexpand\the\expandonce{\csname#1N\endcsname}}%
620 {\noexpand\unexpanded{##2}}%
621 \ifforest@temp\global\fi\advance\expandonce{\csname#1N\endcsname}1
622 ##1%
623 }%
624 \csedef{#1setprepend}##1{% ##1 = definition
625 \ifforest@temp\global\fi\advance\expandonce{\csname#1M\endcsname}-1
626 \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi
627 {#1\noexpand\the\expandonce{\csname#1M\endcsname}}%
628 {\noexpand\unexpanded{##1}}%
629 }%
630 \csedef{#1esetappend}##1{% ##1 = definition
631 \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi{#1\noexpand\the\expandonce{\csname#1N\endcsname}}-1
632 \ifforest@temp\global\fi\advance\expandonce{\csname#1N\endcsname}1
633 }%
634 \csedef{#1esetprepend}##1{% ##1 = definition
635 \ifforest@temp\global\fi\advance\expandonce{\csname#1M\endcsname}-1
636 \ifforest@temp\noexpand\csxdef\else\noexpand\csedef\fi{#1\noexpand\the\expandonce{\csname#1M\endcsname}}-1
637 }%
638 \csedef{#1letappend}##1{% ##1 = cs
639 \ifforest@temp\noexpand\expandafter\noexpand\global\fi\noexpand\expandafter\noexpand\let
640 \noexpand\csname#1\noexpand\the\expandonce{\csname#1N\endcsname}\noexpand\endcsname
641 ##1%
642 \ifforest@temp\global\fi\advance\expandonce{\csname#1N\endcsname}1
643 }%
644 \csedef{#1letprepend}##1{% ##1 = cs
645 \ifforest@temp\global\fi\advance\expandonce{\csname#1M\endcsname}-1
646 \ifforest@temp\noexpand\expandafter\noexpand\global\fi\noexpand\expandafter\noexpand\let
647 \noexpand\csname#1\noexpand\the\expandonce{\csname#1M\endcsname}\noexpand\endcsname
648 ##1%
649 }%

```

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

```

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

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

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

```

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

```

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

```

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

```

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

```

```

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

```

4.2 Testing for numbers and dimensions

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

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

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

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

```

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

```

Expand in advance, like `pgfmath` does.

```

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

```

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

```

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

```

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

```

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

```

```

730     \fi
731     \fi
732     \fi
733     \forest@next
734 }

```

Eat + and -.

```

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

```

Eat the space!

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

772     \expandafter\forest@isnum@a\forest@isnum@temp\forest@eon\forest@eon\forest@eon\forest@eon\forest@END
773     \ifforest@isdim
774         \expandafter\@firstoftwo

```

```

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

```

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

```

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

```



```

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

```

4.3 forestmath

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

```

882 \def\forestmathtype@generic{_} % generic (token list)
883 \def\forestmathtype@count{n} % integer
884 \def\forestmathtype@dimen{d} % a dimension: <decimal> pt
885 \def\forestmathtype@unitless{P} % <decimal> (a unitless dimension) (P because pgfmath returns such numbers)
886 \def\forestmathtype@textasc{t} % text (ascending)
887 \def\forestmathtype@textdesc{T} % text (descending)

```

```

888 \def\forestmathtype@none{} % internal (for requests - means whatever)
889 \def\forestmathresult{}
890 \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.

```

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

```

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

```

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

```

```

940 }{%
941   \forestmathtruncatemacro@#1{#2}%
942 }%
943 }
944 \def\forestmathtruncatemacro@#1#2{%
945   \forest@isnum{#2}{%
946     \edef#1{\the\forest@isnum@count}%
947   }{%
948     \pgfmathtruncatemacro#1{#2}%
949   }%
950 }
951 \def\forestmathsetlengthmacro@#1#2{%
952   \forest@tryprocess{false}#2\forest@spacegen{%
953     \forest@temp@dimen=\forest@process@result\relax
954     \edef#1{\the\forest@temp@dimen}%
955   }{%
956     \forestmathsetlengthmacro@#1{#2}%
957   }%
958 }
959 \def\forestmathsetlengthmacro@#1#2{%
960   \forest@isdim{#2}{%
961     \edef#1{\the\forest@isdim@dimen}%
962   }{%
963     \pgfmathsetlengthmacro#1{#2}%
964   }%
965 }
966 \def\forestmathsetmacro@#1#2{%
967   \forest@tryprocess{false}#2\forest@spacegen{%
968     \let#1\forest@process@result
969     \let\forestmathresulttype\forest@process@result@type
970   }{%
971     \forestmathsetmacro@#1{#2}%
972     \let\forestmathresulttype\forestmath@type@unitless
973   }%
974 }
975 \def\forestmathsetmacro@#1#2{%
976   \forest@isdim{#2}{%
977     \edef#1{\expandafter\Pgfg@geT\the\forest@isdim@dimen}%
978   }{%
979     \pgfmathsetmacro#1{#2}%
980   }%
981 }
982 \def\forestmathparse#1{%
983   \forest@tryprocess{false}#1\forest@spacegen{%
984     \let\forestmathresult\forest@process@result
985     \let\forestmathresulttype\forest@process@result@type
986   }{%
987     \forestmathparse@{#1}%
988     \let\forestmathresulttype\forestmath@type@unitless
989   }%
990 }
991 \def\forestmathparse@#1{%
992   \forest@isdim{#1}{%
993     \edef\forestmathresult{\expandafter\Pgfg@geT\the\forest@isdim@dimen}%
994   }{%
995     \pgfmathsetmacro\forestmathresult{#1}%
996   }%
997 }

```

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.

```

998 %\def\forestmathparse@returnarray#1{% same as above, but returns the result as an array (used only internal
999 % \forest@tryprocess{true}#1\forest@spacegen{}-%
1000 % \forestmathparse@{#1}%
1001 % \let\forest@process@result@type\forestmathtype@unitless
1002 % \forest@process@result@clear
1003 % \forest@process@result@letappend\forestmathresult
1004 % }%
1005 %}

```

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

```

1006 \csdef{forest@bh@0}{0}
1007 \csdef{forest@bh@false}{0}
1008 \csdef{forest@bh@1}{1}
1009 \csdef{forest@bh@true}{1}
1010 \def\forestmath@if#1{%
1011 \ifcsdef{forest@bh@\detokenize{#1}}{%
1012 \let\forest@next\forestmath@if@fast
1013 }{%
1014 \let\forest@next\forestmath@if@slow
1015 }%
1016 \forest@next{#1}%
1017 \ifnum\forest@temp=0
1018 \expandafter\@secondoftwo
1019 \else
1020 \expandafter\@firstoftwo
1021 \fi
1022 }
1023 \def\forestmath@if@fast#1{\letcs\forest@temp{forest@bh@\detokenize{#1}}}
1024 \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).

```

1025 \def\forestmath@convert@fromto#1#2#3{%
1026 \edef\forestmathresult{\csname forestmath@convert@from@#1@to@#2\endcsname{#3}}
1027 \def\forestmath@convert@from#1{\forestmath@convert@fromto{#1}{\forestmathresulttype}}
1028 \def\forestmath@convert@to{\forestmath@convert@fromto{\forestmathresulttype}}
1029 \def\forestmath@convert@from@n@to@n#1{#1}
1030 \def\forestmath@convert@from@n@to@d#1{#1pgfm@pt}
1031 \def\forestmath@convert@from@n@to@P#1{#1}
1032 \def\forestmath@convert@from@d@to@n#1{%
1033 \expandafter\forestmath@convert@uptodot\Pgf@geT#1.\forest@eov}
1034 \def\forestmath@convert@from@d@to@d#1{#1}
1035 \def\forestmath@convert@from@d@to@P#1{\Pgf@geT#1}
1036 \def\forestmath@convert@from@P@to@n#1{%
1037 \forestmath@convert@uptodot#1.\forest@eov}
1038 \def\forestmath@convert@from@P@to@d#1{#1pgfm@pt}
1039 \def\forestmath@convert@from@P@to@P#1{#1}
1040 \def\forestmath@convert@uptodot#1.#2\forest@eov{#1}
1041 \def\forestmathzero{\forestmath@convert@from\forestmathtype@count{0}}

```

These defer conversion (see aggregates).

```

1042 \csdef{forestmath@convert@from@n@to@_}#1{\unexpanded{#1}}
1043 \csdef{forestmath@convert@from@d@to@_}#1{\unexpanded{#1}}
1044 \csdef{forestmath@convert@from@P@to@_}#1{\unexpanded{#1}}

```

Sets \pgfm@resulttype to the type of #1.

```

1045 \def\forestmathsettypefrom#1{%
1046 \forest@isnumdim{%
1047 \let\forestmathresulttype\forestmathtype@count
1048 }{%

```

```

1049 \let\forestmathresulttype\forestmathtype@dimen
1050 }{%
1051 \let\forestmathresulttype\forestmathtype@unitless
1052 }%
1053 }

```

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.

```

1054 \def\forestmathadd#1#2{\edef\forestmathresult{%
1055 \csname forestmathadd@\forestmathresulttype\endcsname{#1}{#2}}}
1056 \def\forestmathadd@#1#2#3{\edef\forestmathresult{%
1057 \csname forestmathadd@#1\endcsname{#2}{#3}}}
1058 \def\forestmathadd@n#1#2{\the\numexpr#1+#2\relax}
1059 \def\forestmathadd@d#1#2{\the\dimexpr#1+#2\relax}
1060 \def\forestmathadd@P#1#2{\expandafter\Pgf@geT\the\dimexpr#1pt+#2pt\relax}
1061 \def\forestmathmultiply#1#2{%
1062 \csname forestmathmultiply@\forestmathresulttype\endcsname{#1}{#2}}
1063 \def\forestmathmultiply@#1#2#3{%
1064 \csname forestmathmultiply@#1\endcsname{#2}{#3}}
1065 \def\forestmathmultiply@n#1#2{\edef\forestmathresult{%
1066 \the\numexpr#1*#2\relax}}
1067 \def\forestmathmultiply@d#1#2{%
1068 \edef\forestmath@marshal{\forestmathmultiply@d@{#1}{#2}}\forestmath@marshal
1069 }
1070 \def\forestmathmultiply@d@#1#2{%
1071 \edef\forestmath@marshal{%
1072 \noexpand\pgfmathmultiply@{\Pgf@geT#1}{\Pgf@geT#2}%
1073 }\forestmath@marshal
1074 \edef\forestmathresult{\pgfmathresult\pgfmath@pt}%
1075 }
1076 \def\forestmathmultiply@P#1#2{%
1077 \pgfmathmultiply@{#1}{#2}%
1078 \let\forestmathresult\pgfmathresult
1079 }

```

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.

```

1080 \def\forestmathdivide#1#2{%
1081 \csname forestmathdivide@\forestmathresulttype\endcsname{#1}{#2}}
1082 \def\forestmathdivide@#1#2#3{%
1083 \csname forestmathdivide@#1\endcsname{#2}{#3}}
1084 \def\forestmathdivide@n#1#2{\edef\forestmathresult{%
1085 \the\numexpr#1/#2\relax}}
1086 \def\forestmathdivide@d#1#2{\edef\forestmathresult{%
1087 \the\dimexpr#1/#2\relax}}
1088 \def\forestmathdivide@P#1#2{%
1089 \edef\forest@marshal{%
1090 \noexpand\pgfmathdivide{+#1}{+#2}%
1091 }\forest@marshal
1092 \let\forestmathresult\pgfmathresult
1093 }

```

Booleans.

```

1094 \def\forestmathtrue{%
1095 \def\forestmathresult{1}%
1096 \let\forestmathresulttype\forestmathtype@count}
1097 \def\forestmathfalse{%
1098 \def\forestmathresult{0}%
1099 \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 <.

```

1100 \def\forestmath@if@lt@n#1#2{\ifnum#1<#2\relax
1101   \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1102 \def\forestmath@if@lt@d#1#2{\ifdim#1<#2\relax
1103   \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1104 \def\forestmath@if@lt@P#1#2{\ifdim#1pt<#2pt
1105   \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1106 \def\forestmath@if@lt@t#1#2{\ifnum\pdfstrcmp{#1}{#2}<0
1107   \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1108 \def\forestmath@if@lt@T#1#2{\ifnum\pdfstrcmp{#1}{#2}>0
1109   \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1110 \def\forest@cmp@error#1#2{\PackageError{forest}{Comparison
1111   ("<" or ">") of generic type arguments "#1" and "#2"
1112   makes no sense}{Use one of argument processor instructions
1113   "n", "d", "P" or "t" to change the type. Use package option
1114   "debug=process" to see what's happening here.}}
1115 \cslet{forestmath@if@lt@_}\forest@cmp@error

```

Low level =.

```

1116 \def\forestmath@if@eq@n#1#2{\ifnum#1=#2\relax
1117   \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1118 \def\forestmath@if@eq@d#1#2{\ifdim#1=#2\relax
1119   \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1120 \def\forestmath@if@eq@P#1#2{\ifdim#1pt=#2pt
1121   \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1122 \def\forestmath@if@eq@t#1#2{\ifnum\pdfstrcmp{#1}{#2}=0
1123   \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
1124 \let\forestmath@if@eq@T\forestmath@if@eq@t
1125 \csdef{forestmath@if@eq@_}#1#2{%
1126   \def\forestmath@tempa{#1}%
1127   \def\forestmath@tempb{#2}%
1128   \ifx\forestmath@tempa\forestmath@tempb
1129     \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}

```

High level <, > and =.

```

1130 \def\forestmathlt#1#2{%
1131   \csname forestmath@if@lt@\forestmathresulttype\endcsname{#1}{#2}%
1132   \forestmathtrue
1133   \forestmathfalse}
1134 \def\forestmathlt@#1#2#3{%
1135   \csname forestmath@if@lt@#1\endcsname{#2}{#3}%
1136   \forestmathtrue
1137   \forestmathfalse}
1138 \def\forestmathgt#1#2{%
1139   \csname forestmath@if@lt@\forestmathresulttype\endcsname{#2}{#1}%
1140   \forestmathtrue
1141   \forestmathfalse}
1142 \def\forestmathgt@#1#2#3{%
1143   \csname forestmath@if@lt@#1\endcsname{#3}{#2}%
1144   \forestmathtrue
1145   \forestmathfalse}
1146 \def\forestmatheq#1#2{%
1147   \csname forestmath@if@eq@\forestmathresulttype\endcsname{#1}{#2}%
1148   \forestmathtrue
1149   \forestmathfalse}
1150 \def\forestmatheq@#1#2#3{%
1151   \csname forestmath@if@eq@#1\endcsname{#2}{#3}%
1152   \forestmathtrue
1153   \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.

```

1154 \def\forestmathmin{\forestmath@minmax{min}{\forestmathresulttype}}
1155 \def\forestmathmax{\forestmath@minmax{max}{\forestmathresulttype}}
1156 \def\forestmathmin@{\forestmath@minmax{min}}
1157 \def\forestmathmax@{\forestmath@minmax{max}}
1158 \def\forestmath@minmax#1#2#3#4{% #1=min/max, #2=type, #3,#4=args
1159   \edef\forestmath@tempa{#3}%
1160   \edef\forestmath@tempb{#4}%
1161   \if\relax\detokenize\expandafter{\forestmath@tempa}\relax
1162     \forestmath@minmax@one{#1}{#2}\forestmath@tempb
1163   \else
1164     \if\relax\detokenize\expandafter{\forestmath@tempb}\relax
1165       \forestmath@minmax@one{#1}{#2}\forestmath@tempa
1166     \else
1167       \csname forestmath@#1\endcsname{#2}%
1168     \fi
1169   \fi
1170 }
1171 \def\forestmath@minmax@one#1#2#3{% #1=min/max, #2=type, #3 = the (possibly) non-empty arg
1172   \ifcsname forestmath@#1@one@#2\endcsname
1173     \csname forestmath@#1@one@#2\endcsname#3%
1174   \else
1175     \let\forestmathresult#3%
1176   \fi
1177 }
1178 \def\forestmath@min@one@t#1{\let\forestmathresult\forest@empty}
1179 \def\forestmath@max@one@t#1{\let\forestmathresult#1}
1180 \def\forestmath@min@one@T#1{\let\forestmathresult#1}
1181 \def\forestmath@max@one@T#1{\let\forestmathresult\forest@empty}
1182
1183 \def\forestmath@min#1{% #1 = type
1184   \csname forestmath@if@lt@#1\endcsname\forestmath@tempa\forestmath@tempb
1185   {\let\forestmathresult\forestmath@tempa}%
1186   {\let\forestmathresult\forestmath@tempb}%
1187 }
1188 \def\forestmath@max#1{% #1 = type
1189   \csname forestmath@if@lt@#1\endcsname\forestmath@tempa\forestmath@tempb
1190   {\let\forestmathresult\forestmath@tempb}%
1191   {\let\forestmathresult\forestmath@tempa}%
1192 }

```

4.4 Sorting

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

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

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

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

`\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.

```
1193 \def\forest@sort#1#2#3#4#5{%
1194   \let\forest@sort@cmp#1\relax
1195   \let\forest@sort@let#2\relax
1196   \let\forest@sort@direction#3\relax
1197   \forest@@sort{#4}{#5}%
1198 }
```

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.

```
1199 \def\forest@quicksort@minarraylength{10000}
1200 \def\forest@@sort#1#2{%
1201   \ifnum#1<#2\relax\@escapeif{%
1202     \forest@sort@m=#2
1203     \advance\forest@sort@m -#1
1204     \ifnum\forest@sort@m>\forest@quicksort@minarraylength\relax\@escapeif{%
1205       \forest@quicksort{#1}{#2}%
1206     }\else\@escapeif{%
1207       \forest@insertionsort{#1}{#2}%
1208     }\fi
1209   }\fi
1210 }
```

Various counters and macros needed by the sorting algorithms.

```
1211 \newcount\forest@sort@m\newcount\forest@sort@k\newcount\forest@sort@p
1212 \def\forest@sort@ascending{>}
1213 \def\forest@sort@descending{<}
1214 \def\forest@sort@cmp{%
1215   \PackageError{sort}{You must define forest@sort@cmp function before calling
1216     sort}{The macro must take two arguments, indices of the array
1217     elements to be compared, and return '=' if the elements are equal
1218     and '>'/ '<' if the first is greater /less than the secong element.}%
1219 }
1220 \def\forest@sort@cmp@gt{\def\forest@sort@cmp@result{>}}
1221 \def\forest@sort@cmp@lt{\def\forest@sort@cmp@result{<}}
1222 \def\forest@sort@cmp@eq{\def\forest@sort@cmp@result{=}}
1223 \def\forest@sort@let{%
1224   \PackageError{sort}{You must define forest@sort@let function before calling
1225     sort}{The macro must take two arguments, indices of the array:
1226     element 2 must be copied onto element 1.}%
1227 }
```

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

```
1228 \newloop\forest@sort@loop
1229 \newloop\forest@sort@loopA
1230 \def\forest@quicksort#1#2{%
```

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

```
1231   \forest@sort@m=#2
1232   \advance\forest@sort@m -#1
1233   \ifodd\forest@sort@m\relax\advance\forest@sort@m1 \fi
1234   \divide\forest@sort@m 2
1235   \advance\forest@sort@m #1
```

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

```
1236   \forest@sort@cmp{#1}{#2}%
1237   \if\forest@sort@cmp@result=%
1238     \forest@sort@p=#1
1239   \else
1240     \if\forest@sort@cmp@result>%
```



```

1241     \forest@sort@p=#1\relax
1242     \else
1243     \forest@sort@p=#2\relax
1244     \fi
1245     \forest@sort@cmp{\the\forest@sort@p}{\the\forest@sort@m}%
1246     \if\forest@sort@cmp@result<%
1247     \else
1248     \forest@sort@p=\the\forest@sort@m
1249     \fi
1250 \fi

Exchange the pivot and the first element.
1251 \forest@sort@xch{#1}{\the\forest@sort@p}%

Counter \forest@sort@m will hold the final location of the pivot element.
1252 \forest@sort@m=#1\relax

Loop through the list.
1253 \forest@sort@k=#1\relax
1254 \forest@sort@loop
1255 \ifnum\forest@sort@k<#2\relax
1256     \advance\forest@sort@k 1

Compare the pivot and the current element.
1257 \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.
1258 \ifx\forest@sort@direction\forest@sort@cmp@result
1259     \advance\forest@sort@m 1
1260     \forest@sort@xch{\the\forest@sort@m}{\the\forest@sort@k}
1261     \fi
1262 \forest@sort@repeat

Move the pivot element into its final position.
1263 \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.
1264 \forest@sort@k=\forest@sort@m
1265 \advance\forest@sort@k -1
1266 \advance\forest@sort@m 1
1267 \edef\forest@sort@marshal{%
1268     \noexpand\forest@sort@{#1}{\the\forest@sort@k}%
1269     \noexpand\forest@sort@{\the\forest@sort@m}{#2}%
1270 }%
1271 \forest@sort@marshal
1272 }

1273 % We defines the item-exchange macro in terms of the (user-provided)
1274 % array let macro.
1275 % \begin{macrocode}
1276 \def\forest@sort@aux{aux}
1277 \def\forest@sort@xch#1#2{%
1278     \forest@sort@let{\forest@sort@aux}{#1}%
1279     \forest@sort@let{#1}{#2}%
1280     \forest@sort@let{#2}{\forest@sort@aux}%
1281 }

Insertion sort.
1282 \def\forest@insertionsort#1#2{%
1283     \forest@sort@m=#1
1284     \edef\forest@insertionsort@low{#1}%
1285     \forest@sort@loopA

```

```

1286 \ifnum\forest@sort@m<#2
1287   \advance\forest@sort@m 1
1288   \forest@insertionsort@qbody
1289   \forest@sort@repeatA
1290 }
1291 \newif\ifforest@insertionsort@loop
1292 \def\forest@insertionsort@qbody{%
1293   \forest@sort@let{\forest@sort@aux}{\the\forest@sort@m}%
1294   \forest@sort@k\forest@sort@m
1295   \advance\forest@sort@k -1
1296   \forest@insertionsort@looptrue
1297   \forest@sort@loop
1298   \ifforest@insertionsort@loop
1299     \forest@insertionsort@qbody
1300   \forest@sort@repeat
1301   \advance\forest@sort@k 1
1302   \forest@sort@let{\the\forest@sort@k}{\forest@sort@aux}%
1303 }
1304 \def\forest@insertionsort@qbody{%
1305   \forest@sort@cmp{\the\forest@sort@k}{\forest@sort@aux}%
1306   \ifx\forest@sort@direction\forest@sort@cmp@result\relax
1307     \forest@sort@p=\forest@sort@k
1308     \advance\forest@sort@p 1
1309     \forest@sort@let{\the\forest@sort@p}{\the\forest@sort@k}%
1310     \advance\forest@sort@k -1
1311     \ifnum\forest@sort@k<\forest@insertionsort@low\relax
1312       \forest@insertionsort@loopfalse
1313     \fi
1314   \else
1315     \forest@insertionsort@loopfalse
1316   \fi
1317 }

```

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

Compare numbers.

```

1318 \def\forest@sort@cmpnumcs#1#2{%
1319   \ifnum\csname#1\endcsname>\csname#2\endcsname\relax
1320   \forest@sort@cmp@gt
1321 \else
1322   \ifnum\csname#1\endcsname<\csname#2\endcsname\relax
1323     \forest@sort@cmp@lt
1324   \else
1325     \forest@sort@cmp@eq
1326   \fi
1327 \fi
1328 }

```

Compare dimensions.

```

1329 \def\forest@sort@cmpdimcs#1#2{%
1330   \ifdim\csname#1\endcsname>\csname#2\endcsname\relax
1331   \forest@sort@cmp@gt
1332 \else
1333   \ifdim\csname#1\endcsname<\csname#2\endcsname\relax
1334     \forest@sort@cmp@lt
1335   \else
1336     \forest@sort@cmp@eq
1337   \fi
1338 \fi
1339 }

```

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

```

1340 \def\forest@sort@cmptwodimcs#1#2#3#4{%
1341   \ifdim\csname#1\endcsname>\csname#3\endcsname\relax
1342     \forest@sort@cmp@gt
1343   \else
1344     \ifdim\csname#1\endcsname<\csname#3\endcsname\relax
1345       \forest@sort@cmp@lt
1346     \else
1347       \ifdim\csname#2\endcsname>\csname#4\endcsname\relax
1348         \forest@sort@cmp@gt
1349       \else
1350         \ifdim\csname#2\endcsname<\csname#4\endcsname\relax
1351           \forest@sort@cmp@lt
1352         \else
1353           \forest@sort@cmp@eq
1354         \fi
1355       \fi
1356     \fi
1357 \fi
1358 }

```

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).

```

1359 \def\forest@reversearray#1#2#3{%
1360   \let\forest@sort@let#1%
1361   \c@pgf@countc=#2
1362   \c@pgf@countd=#3
1363   \advance\c@pgf@countd -1
1364   \safeloop
1365   \ifnum\c@pgf@countc<\c@pgf@countd\relax
1366     \forest@sort@exch{\the\c@pgf@countc}{\the\c@pgf@countd}%
1367     \advance\c@pgf@countc 1
1368     \advance\c@pgf@countd -1
1369   \saferepeat
1370 }

```

5 The bracket representation parser

5.1 The user interface macros

Settings.

```

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

```

1391 }

`\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.)

```
1392 \newtoks\bracket@content
1393 \newtoks\bracket@afterthought
1394 \def\bracketParse#1#2={%
1395   \def\bracketEndParsingHook{#1}%
1396   \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.)

```
1397   \bracket@content={}%
1398   \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.

```
1399   \let\bracket@state\bracket@state@starting
1400   \bracket@ignorepacestrue
```

By default, don’t expand anything.

```
1401   \bracket@expandtokensfalse
```

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

```
1402   \def\bracket@parentNode{0}%
1403   \def\bracket@rootNode{0}%
1404   \def\bracket@newNode{0}%
1405   \def\bracket@afterthoughtNode{0}%
```

Finally, we start the parser.

```
1406   \bracket@Parse
1407 }
```

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.

```
1408 \def\bracketResume{\bracket@Parse}%
```

5.2 Parsing

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

```
1409 \def\bracket@Parse{%
1410   \futurelet\bracket@next@token\bracket@Parse@checkForSpace
1411 }
1412 \def\bracket@Parse@checkForSpace{%
1413   \expandafter\ifx\space\bracket@next@token\@escapeif{%
1414     \ifbracket@ignorepaces\else
1415       \bracket@haveSpacetrue
1416     \fi
1417     \expandafter\bracket@Parse\romannumeral-‘0%
1418   }\else\@escapeif{%
1419     \bracket@Parse@maybeexpand
1420   }\fi
1421 }
```

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

```
1422 \newif\ifbracket@expandtokens
1423 \def\bracket@Parse@maybeexpand{%
```

```

1424 \ifbracket@expandtokens\@escapeif{%
1425   \expandafter\bracket@Parse@peekAhead\romannumeral-‘0%
1426 }\else\@escapeif{%
1427   \bracket@Parse@peekAhead
1428 }\fi
1429 }

```

We then look ahead to see what’s coming.

```

1430 \def\bracket@Parse@peekAhead{%
1431   \futurelet\bracket@next@token\bracket@Parse@checkForTeXGroup
1432 }

```

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

```

1433 \def\bracket@Parse@checkForTeXGroup{%
1434   \ifx\bracket@next@token\bgroup%
1435     \@escapeif{\bracket@Parse@appendGroup}%
1436   \else
1437     \@escapeif{\bracket@Parse@token}%
1438   \fi
1439 }

```

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.

```

1440 \long\def\bracket@Parse@token#1{%
1441   \ifx#1\bracket@openingBracket
1442     \@escapeif{\bracket@Parse@openingBracketFound}%
1443   \else
1444     \@escapeif{%
1445       \ifx#1\bracket@closingBracket
1446         \@escapeif{\bracket@Parse@closingBracketFound}%
1447       \else
1448         \@escapeif{%
1449           \ifx#1\bracket@actionCharacter
1450             \@escapeif{\futurelet\bracket@next@token\bracket@Parse@actionCharacterFound}%
1451           \else
1452             \@escapeif{\bracket@Parse@appendToken#1}%
1453           \fi
1454         }%
1455       \fi
1456     }%
1457   \fi
1458 }

```

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

```

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

```

```

1475 \ifcase\bracket@state\relax
1476   \lapptotoks\bracket@content{#1}%
1477 \or
1478   \lapptotoks\bracket@afterthought{#1}%
1479 \or
1480   \lapptotoks\bracket@afterthought{#1}%
1481 \fi
1482 \bracket@ignorespacesfalse
1483 \bracket@Parse
1484 }
1485 \def\bracket@Parse@appendGroup#1{%
1486   \bracket@Parse@appendSpace
1487   \ifcase\bracket@state\relax
1488     \apptotoks\bracket@content{{#1}}%
1489   \or
1490     \apptotoks\bracket@afterthought{{#1}}%
1491   \or
1492     \apptotoks\bracket@afterthought{{#1}}%
1493   \fi
1494   \bracket@ignorespacesfalse
1495   \bracket@Parse
1496 }

```

Declare states.

```

1497 \def\bracket@state@inContent{0}
1498 \def\bracket@state@inAfterthought{1}
1499 \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.)

```

1500 \def\bracket@Parse@openingBracketFound{%
1501   \bracket@haveSpacefalse
1502   \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.

```

1503     \@escapeif{%
1504       \bracket@createNode
1505       \ifnum\bracket@parentNode=0 \else
1506         \forest@node@Append{\bracket@parentNode}{\bracket@newNode}%
1507       \fi
1508       \let\bracket@parentNode\bracket@newNode
1509       \bracket@Parse
1510     }%
1511   \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.

```

1512     \@escapeif{%
1513       \bracket@addAfterthought
1514       \let\bracket@state\bracket@state@inContent
1515       \bracket@Parse
1516     }%
1517   \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.

```

1518 \escapeif{%
1519 \let\bracket@state\bracket@state@inContent
1520 \bracket@Parse
1521 }%
1522 \fi
1523 }
1524 \def\bracket@Parse@closingBracketFound{%
1525 \bracket@haveSpacefalse
1526 \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.

```

1527 \escapeif{%
1528 \bracket@createNode
1529 \ifnum\bracket@parentNode=0
1530 \escapeif\bracket@endParsingHook
1531 \else
1532 \escapeif{%
1533 \let\bracket@afterthoughtNode\bracket@newNode
1534 \let\bracket@state\bracket@state@inAfterthought
1535 \forest@node@Append{\bracket@parentNode}{\bracket@newNode}%
1536 \bracket@Parse
1537 }%
1538 \fi
1539 }%
1540 \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.

```

1541 \escapeif{%
1542 \bracket@addAfterthought
1543 \let\bracket@afterthoughtNode\bracket@parentNode
1544 \edef\bracket@parentNode{\forest@ve{\bracket@parentNode}{@parent}}%
1545 \ifnum\bracket@parentNode=0
1546 \expandafter\bracket@endParsingHook
1547 \else
1548 \expandafter\bracket@Parse
1549 \fi
1550 }%
1551 \else % starting

```

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

```

1552 \PackageError{forest}{You're attempting to start a bracket representation
1553 with a closing bracket}{}%
1554 \fi
1555 }

```

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

```

1556 \def\bracket@Parse@actionCharacterFound{%

```

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

```

1557 \ifx\bracket@next@token\bgroup\escapeif{%
1558 \bracket@Parse@action@expandgroup
1559 }\else\escapeif{%
1560 \bracket@Parse@action@notagroup
1561 }\fi
1562 }
1563 \def\bracket@Parse@action@expandgroup#1{%

```

```

1564 \edef\bracket@Parse@action@expandgroup@macro{#1}%
1565 \expandafter\bracket@Parse\bracket@Parse@action@expandgroup@macro
1566 }
1567 \let\bracket@action@fullyexpandCharacter+
1568 \let\bracket@action@dontexpandCharacter-
1569 \let\bracket@action@executeCharacter!
1570 \def\bracket@Parse@action@notagroup#1{%

```

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

```

1571 \ifx#1\bracket@action@fullyexpandCharacter\@escapeif{%
1572   \bracket@expandtokenstrue\bracket@Parse
1573 }\else\@escapeif{%

```

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

```

1574 \ifx#1\bracket@action@dontexpandCharacter\@escapeif{%
1575   \bracket@expandtokensfalse\bracket@Parse
1576 }\else\@escapeif{%

```

Inhibit expansion of the next token.

```

1577 \ifx#10\@escapeif{%
1578   \bracket@Parse@appendToken
1579 }\else\@escapeif{%

```

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

```

1580 \ifx#1\bracket@actionCharacter
1581 \else\@escapeif{%

```

Anything else will be expanded once.

```

1582 \expandafter\bracket@Parse#1%
1583 }\fi
1584 }\fi
1585 }\fi
1586 }\fi
1587 }

```

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.

```

1588 \def\bracket@createNode{%
1589   \ifnum\bracket@rootNode=0
1590     % root node
1591     \bracketset{new node/.expanded=%
1592       {\the\bracket@afterthought}%
1593       {\the\bracket@content}%
1594       \noexpand\bracket@newNode
1595     }%
1596     \bracket@afterthought={}%
1597     \let\bracket@rootNode\bracket@newNode
1598     \expandafter\let\bracket@saveRootNodeTo\bracket@newNode
1599   \else
1600     % other nodes
1601     \bracketset{new node/.expanded=%
1602       {}%
1603       {\the\bracket@content}%
1604       \noexpand\bracket@newNode
1605     }%
1606   \fi
1607   \bracket@content={}%
1608 }

```


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

```

1609 \def\bracket@addAfterthought{%
1610   \bracketset{%
1611     set afterthought/.expanded={\bracket@afterthoughtNode}{\the\bracket@afterthought}%
1612   }%
1613   \bracket@afterthought={}%
1614 }

```

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.

```

1615 % full expansion expands precisely to the value
1616 \def\forestov#1{\expandafter\expandonce\csname fRST\forest@cn/#1\endcsname}
1617 % full expansion expands all the way
1618 \def\forestove#1{\csname fRST\forest@cn/#1\endcsname}
1619 % full expansion expands to the cs holding the value
1620 \def\forestom#1{\expandonce{\csname fRST\forest@cn/#1\endcsname}}
1621 \def\forestoget#1#2{\expandafter\let\expandafter#2\csname fRST\forest@cn/#1\endcsname}
1622 \def\forestolet#1#2{\expandafter\let\csname fRST\forest@cn/#1\endcsname#2}
1623 % \def\forestocslet#1#2{%
1624 %   \edef\forest@marshal{%
1625 %     \noexpand\pgfkeyslet{/forest/@node/\forest@cn/#1}{\expandonce{\csname#2\endcsname}}%
1626 %   }\forest@marshal
1627 % }
1628 \def\forestoset#1#2{\expandafter\edef\csname fRST\forest@cn/#1\endcsname{\unexpanded{#2}}}
1629 \def\forestoeset#1#2
1630   {\expandafter\edef\csname fRST\forest@cn/#1\endcsname
1631     {%#2}
1632   }
1633 \def\forestoaappto#1#2{%
1634   \forestoeset{#1}{\forestov{#1}\unexpanded{#2}}%
1635 }
1636 \def\forestoiifdefined#1#2#3
1637 {%
1638   \ifcsdef{fRST\forest@cn/#1}%{#2}{#3}%
1639 }

```

User macros for retrieving node options of the current node.

```

1640 \let\forestoption\forestov
1641 \let\foresteooption\forestove

```

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

```

1642 \def\forest0v#1#2{\expandafter\expandonce\csname fRST#1/#2\endcsname}
1643 \def\forest0ve#1#2{\csname fRST#1/#2\endcsname}
1644 % full expansion expands to the cs holding the value
1645 \def\forest0m#1#2{\expandonce{\csname fRST#1/#2\endcsname}}
1646 \def\forest0get#1#2#3{\expandafter\let\expandafter#3\csname fRST#1/#2\endcsname}
1647 \def\forest0let#1#2#3{\expandafter\let\csname fRST#1/#2\endcsname#3}
1648 % \def\forest0cslet#1#2#3{%
1649 %   \edef\forest@marshal{%
1650 %     \noexpand\pgfkeyslet{/forest/@node/#1/#2}{\expandonce{\csname#3\endcsname}}%
1651 %   }\forest@marshal
1652 % }
1653 \def\forest0set#1#2#3{\expandafter\edef\csname fRST#1/#2\endcsname{\unexpanded{#3}}}

```

```

1654 \def\forestOeset#1#2#3
1655 {\expandafter\edef\csname fRsT#1/#2\endcsname
1656   % {#3}
1657 }
1658 \def\forestOappto#1#2#3{%
1659   \forestOeset{#1}{#2}{\forestOv{#1}{#2}\unexpanded{#3}}%
1660 }
1661 \def\forestOeappto#1#2#3{%
1662   \forestOeset{#1}{#2}{\forestOv{#1}{#2}#3}%
1663 }
1664 \def\forestOpreto#1#2#3{%
1665   \forestOeset{#1}{#2}{\unexpanded{#3}\forestOv{#1}{#2}}%
1666 }
1667 \def\forestOepreto#1#2#3{%
1668   \forestOeset{#1}{#2}{#3\forestOv{#1}{#2}}%
1669 }
1670 \def\forestOifdefined#1#2#3#4
1671 {%
1672   \ifcsdef{fRsT#1/#2}%{#3}{#4}%
1673 }
1674 \def\forestOlet0#1#2#3#4{% option #2 of node #1 <-- option #4 of node #3
1675   \forestOget{#3}{#4}\forestoption@temp
1676   \forestOlet{#1}{#2}\forestoption@temp}
1677 \def\forestOleto#1#2#3{%
1678   \forestoget{#3}\forestoption@temp
1679   \forestOlet{#1}{#2}\forestoption@temp}
1680 \def\forestolet0#1#2#3{%
1681   \forestOget{#2}{#3}\forestoption@temp
1682   \forestolet{#1}\forestoption@temp}
1683 \def\forestoletto#1#2{%
1684   \forestoget{#2}\forestoption@temp
1685   \forestolet{#1}\forestoption@temp}

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

```

```

1713 \forestoget{#4}#1%
1714 }%
1715 \pgfkeysgetvalue{/forest/#4/@type}#2%
1716 }

Macros for retrieving/setting registers.

1717 % full expansion expands precisely to the value
1718 \def\forestrv#1{\expandafter\expandonce\csname fRsT/#1\endcsname}
1719 % full expansion expands all the way
1720 \def\forestrve#1{\csname fRsT/#1\endcsname}
1721 % full expansion expands to the cs holding the value
1722 \def\forestrm#1{\expandonce{\csname fRsT/#1\endcsname}}
1723 \def\forestrget#1#2{\expandafter\let\expandafter#2\csname fRsT/#1\endcsname}
1724 \def\forestrlet#1#2{\expandafter\let\csname fRsT/#1\endcsname#2}
1725 % \def\forestrcslet#1#2{%
1726 % \edef\forest@marshal{%
1727 % \noexpand\pgfkeyslet{/forest/@node/register/#1}{\expandonce{\csname#2\endcsname}}%
1728 % }\forest@marshal
1729 % }
1730 \def\forestrset#1#2{\expandafter\edef\csname fRsT/#1\endcsname{\unexpanded{#2}}}
1731 \def\forestreset#1#2
1732 {\expandafter\edef\csname fRsT/#1\endcsname}%{#2}
1733 \def\forestrappto#1#2{%
1734 \forestreset{#1}{\forestrv{#1}\unexpanded{#2}}%
1735 }
1736 \def\forestrpreto#1#2{%
1737 \forestreset{#1}{\unexpanded{#2}\forestrv{#1}}%
1738 }
1739 \def\forestrifdefined#1#2#3
1740 {%
1741 \ifcsdef{fRsT/#1}%{#2}{#3}%
1742 }

```

User macros for retrieving node options of the current node.

```

1743 \def\forestregister#1{\forestrv{#1}}
1744 \def\foresteregister#1{\forestrve{#1}}

```

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

```

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

```

```

1769 \advance\forest@node@maxid1
1770 \def\forest@init##1{\ifstrequal{##1}{name}{\forest@set{name}{node@\forest@cn}}{\forest@let0{##1}{#1}{##1}}}
1771 \forest@for@node{\the\forest@node@maxid}{%
1772   \forest@node@init
1773   \forest@set{id}{\forest@cn}%
1774   \forest@node@setname{\forest@copy@name@template{\forest@ve{#1}{name}}}%
1775   \edef#2{\forest@cn}%
1776 }%
1777 \let\forest@init\forest@init@orig
1778 }
1779 \forest@set{
1780   copy name template/.code={\def\forest@copy@name@template##1{#1}},
1781   copy name template/.default={node@\the\forest@node@maxid},
1782   copy name template
1783 }
1784 \def\forest@tree@copy#1#2{% #1=from node id, #2=cs receiving the new node id
1785   \forest@node@copy{#1}\forest@node@copy@temp@id
1786   \forest@for@node{\forest@node@copy@temp@id}{%
1787     \expandafter\forest@tree@copy\expandafter{\forest@node@copy@temp@id}{#1}%
1788     \edef#2{\forest@cn}%
1789   }%
1790 }
1791 \def\forest@tree@copy@#1#2{%
1792   \forest@node@Foreachchild{#2}{%
1793     \expandafter\forest@tree@copy\expandafter{\forest@cn}\forest@node@copy@temp@childid
1794     \forest@node@Append{#1}{\forest@node@copy@temp@childid}%
1795   }%
1796 }

```

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.

```

1797 \def\forest@cn{0}
1798 \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.

```

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

```

```

1819 \PackageError{forest}{Insertafter(#1,#2,#3):
1820 node #2 already has a parent (\forestOve{#2}{@parent})}{}%
1821 \fi
1822 \ifnum#3=0
1823 \else
1824 \ifnum#1=\forestOve{#3}{@parent}
1825 \else
1826 \PackageError{forest}{Insertafter(#1,#2,#3): node #1 is not the parent of the
1827 intended sibling #3 (with parent \forestOve{#3}{@parent})}{}%
1828 \fi
1829 \fi
1830 \forestOset{#2}{@parent}{#1}%
1831 \forestOset{#2}{@previous}{#3}%
1832 \ifnum#3=0
1833 \forestOget{#1}{@first}\forest@node@temp
1834 \forestOset{#1}{@first}{#2}%
1835 \else
1836 \forestOget{#3}{@next}\forest@node@temp
1837 \forestOset{#3}{@next}{#2}%
1838 \fi
1839 \forestOset{#2}{@next}\forest@node@temp}%
1840 \ifnum\forest@node@temp=0
1841 \forestOset{#1}{@last}{#2}%
1842 \else
1843 \forestOset{\forest@node@temp}{@previous}{#2}%
1844 \fi
1845 }
1846 \def\forest@node@Append@#1#2{%
1847 \ifnum\forestOve{#2}{@parent}=0
1848 \else
1849 \PackageError{forest}{Append(#1,#2):
1850 node #2 already has a parent (\forestOve{#2}{@parent})}{}%
1851 \fi
1852 \forestOset{#2}{@parent}{#1}%
1853 \forestOget{#1}{@last}\forest@node@temp
1854 \forestOset{#1}{@last}{#2}%
1855 \forestOset{#2}{@previous}{\forest@node@temp}%
1856 \ifnum\forest@node@temp=0
1857 \forestOset{#1}{@first}{#2}%
1858 \else
1859 \forestOset{\forest@node@temp}{@next}{#2}%
1860 \fi
1861 }
1862 \def\forest@node@Remove@#1{%
1863 \forestOget{#1}{@parent}\forest@node@temp@parent
1864 \ifnum\forest@node@temp@parent=0
1865 \else
1866 \forestOget{#1}{@previous}\forest@node@temp@previous
1867 \forestOget{#1}{@next}\forest@node@temp@next
1868 \ifnum\forest@node@temp@previous=0
1869 \forestOset{\forest@node@temp@parent}{@first}{\forest@node@temp@next}%
1870 \else
1871 \forestOset{\forest@node@temp@previous}{@next}{\forest@node@temp@next}%
1872 \fi
1873 \ifnum\forest@node@temp@next=0
1874 \forestOset{\forest@node@temp@parent}{@last}{\forest@node@temp@previous}%
1875 \else
1876 \forestOset{\forest@node@temp@next}{@previous}{\forest@node@temp@previous}%
1877 \fi
1878 \forestOset{#1}{@parent}{0}%
1879 \forestOset{#1}{@previous}{0}%

```

```

1880 \forest0set{#1}{@next}{0}%
1881 \fi
1882 }

Do some stuff and return to the current node.

1883 \def\forest@forthis#1{%
1884 \edef\forest@node@marshal{\unexpanded{#1}\def\noexpand\forest@cn}%
1885 \expandafter\forest@node@marshal\expandafter{\forest@cn}%
1886 }
1887 \def\forest@fornode#1#2{%
1888 \edef\forest@node@marshal{\edef\noexpand\forest@cn{#1}\unexpanded{#2}\def\noexpand\forest@cn}%
1889 \expandafter\forest@node@marshal\expandafter{\forest@cn}%
1890 }

Looping methods: children.

1891 \def\forest@node@foreachchild#1{\forest@node@Foreachchild{\forest@cn}{#1}}
1892 \def\forest@node@Foreachchild#1#2{%
1893 \forest@fornode{\forest0ve{#1}{@first}}{\forest@node@@forselfandfollowingsiblings{#2}}%
1894 }
1895 \def\forest@node@@forselfandfollowingsiblings#1{%
1896 \ifnum\forest@cn=0
1897 \else
1898 \forest@forthis{#1}%
1899 \@escapeif{%
1900 \edef\forest@cn{\forestove{@next}}%
1901 \forest@node@@forselfandfollowingsiblings{#1}%
1902 }%
1903 \fi
1904 }
1905 \def\forest@node@@forselfandfollowingsiblings@reversed#1{%
1906 \ifnum\forest@cn=0
1907 \else
1908 \@escapeif{%
1909 \edef\forest@marshal{%
1910 \noexpand\def\noexpand\forest@cn{\forestove{@next}}%
1911 \noexpand\forest@node@@forselfandfollowingsiblings@reversed{\unexpanded{#1}}%
1912 \noexpand\forest@fornode{\forest@cn}{\unexpanded{#1}}%
1913 }\forest@marshal
1914 }%
1915 \fi
1916 }
1917 \def\forest@node@foreachchild@reversed#1{\forest@node@Foreachchild@reversed{\forest@cn}{#1}}
1918 \def\forest@node@Foreachchild@reversed#1#2{%
1919 \forest@fornode{\forest0ve{#1}{@last}}{\forest@node@@forselfandprecedingsiblings@reversed{#2}}%
1920 }
1921 \def\forest@node@@forselfandprecedingsiblings@reversed#1{%
1922 \ifnum\forest@cn=0
1923 \else
1924 \forest@forthis{#1}%
1925 \@escapeif{%
1926 \edef\forest@cn{\forestove{@previous}}%
1927 \forest@node@@forselfandprecedingsiblings@reversed{#1}%
1928 }%
1929 \fi
1930 }
1931 \def\forest@node@@forselfandprecedingsiblings#1{%
1932 \ifnum\forest@cn=0
1933 \else
1934 \@escapeif{%
1935 \edef\forest@marshal{%
1936 \noexpand\def\noexpand\forest@cn{\forestove{@previous}}%
1937 \noexpand\forest@node@@forselfandprecedingsiblings{\unexpanded{#1}}%

```

```

1938      \noexpand\forest@fornode{\forest@cn}{\unexpanded{#1}}%
1939    }\forest@marshal
1940  }%
1941  \fi
1942 }

```

Looping methods: (sub)tree and descendants.

```

1943 \def\forest@node@@foreach#1#2#3#4{%
1944   % #1 = do what
1945   % #2 = do that -1=before,1=after processing children
1946   % #3 & #4: normal or reversed order of children?
1947   %   #3 = @first/@last
1948   %   #4 = \forest@node@@forselfandfollowingsiblings / \forest@node@@forselfandprecedingsiblings@reversed
1949   \ifnum#2<0 \forest@forthis{#1}\fi
1950   \ifnum\forest@tove{#3}=0
1951     \else\@escapeif{%
1952       \forest@forthis{%
1953         \edef\forest@cn{\forest@tove{#3}}%
1954         #4{\forest@node@@foreach{#1}{#2}{#3}{#4}}%
1955       }%
1956     }\fi
1957   \ifnum#2>0 \forest@forthis{#1}\fi
1958 }
1959 \def\forest@node@foreach#1{%
1960   \forest@node@@foreach{#1}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1961 \def\forest@node@Foreach#1#2{%
1962   \forest@fornode{#1}{\forest@node@@foreach{#2}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1963 \def\forest@node@foreach@reversed#1{%
1964   \forest@node@@foreach{#1}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1965 \def\forest@node@Foreach@reversed#1#2{%
1966   \forest@fornode{#1}{\forest@node@@foreach{#2}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1967 \def\forest@node@foreach@childrenfirst#1{%
1968   \forest@node@@foreach{#1}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1969 \def\forest@node@Foreach@childrenfirst#1#2{%
1970   \forest@fornode{#1}{\forest@node@@foreach{#2}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1971 \def\forest@node@foreach@childrenfirst@reversed#1{%
1972   \forest@node@@foreach{#1}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1973 \def\forest@node@Foreach@childrenfirst@reversed#1#2{%
1974   \forest@fornode{#1}{\forest@node@@foreach{#2}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1975 \def\forest@node@foreach@descendant#1{%
1976   \forest@node@foreachchild{\forest@node@@foreach{#1}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1977 \def\forest@node@Foreach@descendant#1#2{%
1978   \forest@node@Foreachchild{#1}{\forest@node@@foreach{#2}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1979 \def\forest@node@foreach@descendant@reversed#1{%
1980   \forest@node@foreachchild@reversed{\forest@node@@foreach{#1}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1981 \def\forest@node@Foreach@descendant@reversed#1#2{%
1982   \forest@node@Foreachchild@reversed{#1}{\forest@node@@foreach{#2}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1983 \def\forest@node@foreach@descendant@childrenfirst#1{%
1984   \forest@node@foreachchild{\forest@node@@foreach{#1}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1985 \def\forest@node@Foreach@descendant@childrenfirst#1#2{%
1986   \forest@node@Foreachchild{#1}{\forest@node@@foreach{#2}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1987 \def\forest@node@foreach@descendant@childrenfirst@reversed#1{%
1988   \forest@node@foreachchild@reversed{\forest@node@@foreach{#1}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1989 \def\forest@node@Foreach@descendant@childrenfirst@reversed#1#2{%
1990   \forest@node@Foreachchild@reversed{#1}{\forest@node@@foreach{#2}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}

```

Looping methods: breadth-first.

```

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

```

```

1996 \forest@node@Foreach@breadthfirst@{#1}@{first}@{next}@{#2}@{#3}}
1997 \def\forest@node@Foreach@breadthfirst@reversed#1#2#3{% #1 = node id, #2 = max level, #3 = code
1998 \forest@node@Foreach@breadthfirst@{#1}@{last}@{previous}@{#2}@{#3}}
1999 \def\forest@node@Foreach@breadthfirst@#1#2#3#4#5{%
2000 % #1 = root node,
2001 % #2 = @first/@last, #3 = @next/@previous (must be in sync with #2),
2002 % #4 = max level (< 0 means infinite)
2003 % #5 = code to execute at each node
2004 \forest@node@Foreach@breadthfirst@processqueue{#1},{#2}@{#3}@{#4}@{#5}%
2005 }
2006 \def\forest@node@Foreach@breadthfirst@processqueue#1#2#3#4#5{%
2007 % #1 = queue,
2008 % #2 = @first/@last, #3 = @next/@previous (must be in sync with #2),
2009 % #4 = max level (< 0 means infinite)
2010 % #5 = code to execute at each node
2011 \ifstrempy{#1}@{}{%
2012 \forest@node@Foreach@breadthfirst@processqueue@#1\forest@node@Foreach@breadthfirst@processqueue@
2013 {#2}@{#3}@{#4}@{#5}%
2014 }%
2015 }
2016 \def\forest@node@Foreach@breadthfirst@processqueue@#1,#2\forest@node@Foreach@breadthfirst@processqueue@#3#4#5%
2017 % #1 = first,
2018 % #2 = rest,
2019 % #3 = @first/@last, #4 = next/previous (must be in sync with #2),
2020 % #5 = max level (< 0 means infinite)
2021 % #6 = code to execute at each node
2022 \forest@for@node{#1}@{%
2023 #6%
2024 \ifnum#5<0
2025 \forest@node@getlistofchildren\forest@temp{#3}@{#4}%
2026 \else
2027 \ifnum\forest@level>#5\relax
2028 \def\forest@temp{}%
2029 \else
2030 \forest@node@getlistofchildren\forest@temp{#3}@{#4}%
2031 \fi
2032 \fi
2033 \edef\forest@marshal{%
2034 \noexpand\forest@node@Foreach@breadthfirst@processqueue{\unexpanded{#2}\forest@temp}%
2035 {#3}@{#4}@{#5}@{\unexpanded{#6}}}%
2036 }\forest@marshal
2037 }%
2038 }
2039 \def\forest@node@getlistofchildren#1#2#3{% #1 = list cs, #2 = @first/@last, #3 = @next/@previous
2040 \forest@node@Getlistofchildren{\forest@cn}{#1}@{#2}@{#3}%
2041 }
2042 \def\forest@node@Getlistofchildren#1#2#3#4{% #1 = node, #2 = list cs, #3 = @first/@last, #4 = @next/@previous
2043 \def#2{}%
2044 \ifnum\forest@level=0
2045 \else
2046 \eappto#2{\forest@level{#1}@{#3}},}%
2047 \@escapeif{%
2048 \edef\forest@marshal{%
2049 \noexpand\forest@node@Getlistofchildren@{\forest@level{#1}@{#3}}\noexpand#2{#4}%
2050 }\forest@marshal
2051 }%
2052 \fi
2053 }
2054 \def\forest@node@Getlistofchildren@#1#2#3{% #1 = node, #2 = list cs, #3 = @next/@previous
2055 \ifnum\forest@level=0
2056 \else

```



```

2057 \eappto#2{\forestOve{#1}{#3}},}%
2058 \@escapeif{%
2059 \edef\forest@marshal{%
2060 \noexpand\forest@node@Getlistofchildren@{\forestOve{#1}{#3}}\noexpand#2{#3}%
2061 }\forest@marshal
2062 }%
2063 \fi
2064 }

    Compute n, n', n children and level.
2065 \def\forest@node@Compute@numeric@ts@info@#1{%
2066 \forest@node@Foreach{#1}{\forest@node@@compute@numeric@ts@info}%
2067 \ifnum\forestOve{#1}{@parent}=0
2068 \else
2069 \forest@for node{#1}{\forest@node@@compute@numeric@ts@info@nbar}%
2070 % hack: the parent of the node we called the update for gets +1 for n_children
2071 \edef\forest@node@temp{\forestOve{#1}{@parent}}%
2072 \forest@set{\forest@node@temp}{n children}{%
2073 \number\numexpr\forestOve{\forest@node@temp}{n children}-1%
2074 }%
2075 \fi
2076 \forest@node@Foreachdescendant{#1}{\forest@node@@compute@numeric@ts@info@nbar}%
2077 }
2078 \def\forest@node@@compute@numeric@ts@info{%
2079 \forest@set{n children}{0}%
2080 %
2081 \edef\forest@node@temp{\forestove{@previous}}%
2082 \ifnum\forest@node@temp=0
2083 \forest@set{n}{1}%
2084 \else
2085 \forest@set{n}{\number\numexpr\forestOve{\forest@node@temp}{n}+1}%
2086 \fi
2087 %
2088 \edef\forest@node@temp{\forestove{@parent}}%
2089 \ifnum\forest@node@temp=0
2090 \forest@set{n}{0}%
2091 \forest@set{n'}{0}%
2092 \forest@set{level}{0}%
2093 \else
2094 \forest@set{\forest@node@temp}{n children}{%
2095 \number\numexpr\forestOve{\forest@node@temp}{n children}+1%
2096 }%
2097 \forest@set{level}{%
2098 \number\numexpr\forestOve{\forest@node@temp}{level}+1%
2099 }%
2100 \fi
2101 }
2102 \def\forest@node@@compute@numeric@ts@info@nbar{%
2103 \forest@set{n'}{\number\numexpr\forestOve{\forestove{@parent}}{n children}-\forestove{n}+1}%
2104 }
2105 \def\forest@node@compute@numeric@ts@info#1{%
2106 \expandnumberarg\forest@node@Compute@numeric@ts@info@{\forest@cn}%
2107 }
2108 \def\forest@node@Compute@numeric@ts@info#1{%
2109 \expandnumberarg\forest@node@Compute@numeric@ts@info@{#1}%
2110 }

    Tree structure queries.
2111 \def\forest@node@rootid{%
2112 \expandnumberarg\forest@node@Rootid{\forest@cn}%
2113 }
2114 \def\forest@node@Rootid#1{% #1=node

```

```

2115 \ifnum\forestOve{#1}{@parent}=0
2116   #1%
2117 \else
2118   \@escapeif{\expandnumberarg\forest@node@Rootid{\forestOve{#1}{@parent}}}%
2119 \fi
2120 }
2121 \def\forest@node@nthchildid#1{% #1=n
2122   \ifnum#1<1
2123     0%
2124   \else
2125     \expandnumberarg\forest@node@nthchildid@{\number\forestove{@first}}{#1}%
2126   \fi
2127 }
2128 \def\forest@node@nthchildid@#1#2{%
2129   \ifnum#1=0
2130     0%
2131   \else
2132     \ifnum#2>1
2133       \@escapeifif{\expandtwonumberargs
2134         \forest@node@nthchildid@{\forestOve{#1}{@next}}{\numexpr#2-1}}%
2135     \else
2136       #1%
2137     \fi
2138   \fi
2139 }
2140 \def\forest@node@nbarthchildid#1{% #1=n
2141   \expandnumberarg\forest@node@nbarthchildid@{\number\forestove{@last}}{#1}%
2142 }
2143 \def\forest@node@nbarthchildid@#1#2{%
2144   \ifnum#1=0
2145     0%
2146   \else
2147     \ifnum#2>1
2148       \@escapeifif{\expandtwonumberargs
2149         \forest@node@nbarthchildid@{\forestOve{#1}{@previous}}{\numexpr#2-1}}%
2150     \else
2151       #1%
2152     \fi
2153   \fi
2154 }
2155 \def\forest@node@nornbarthchildid#1{%
2156   \ifnum#1>0
2157     \forest@node@nthchildid{#1}%
2158   \else
2159     \ifnum#1<0
2160       \forest@node@nbarthchildid{-#1}%
2161     \else
2162       \forest@node@nornbarthchildid@error
2163     \fi
2164   \fi
2165 }
2166 \def\forest@node@nornbarthchildid@error{%
2167   \PackageError{forest}{In \string\forest@node@nornbarthchildid, n should !=0}{}%
2168 }
2169 \def\forest@node@previousleafid{%
2170   \expandnumberarg\forest@node@Previousleafid{\forest@cn}%
2171 }
2172 \def\forest@node@Previousleafid#1{%
2173   \ifnum\forestOve{#1}{@previous}=0
2174     \@escapeif{\expandnumberarg\forest@node@previousleafid@Goup{#1}}%
2175   \else

```

```

2176 \expandnumberarg\forest@node@previousleafid@Gdown{\forestOve{#1}{@previous}}%
2177 \fi
2178 }
2179 \def\forest@node@previousleafid@Goup#1{%
2180 \ifnum\forestOve{#1}{@parent}=0
2181 \PackageError{forest}{get previous leaf: this is the first leaf}{}%
2182 \else
2183 \@escapeif{\expandnumberarg\forest@node@Previousleafid{\forestOve{#1}{@parent}}}%
2184 \fi
2185 }
2186 \def\forest@node@previousleafid@Gdown#1{%
2187 \ifnum\forestOve{#1}{@last}=0
2188 #1%
2189 \else
2190 \@escapeif{\expandnumberarg\forest@node@previousleafid@Gdown{\forestOve{#1}{@last}}}%
2191 \fi
2192 }
2193 \def\forest@node@nextleafid{%
2194 \expandnumberarg\forest@node@Nextleafid{\forest@cn}%
2195 }
2196 \def\forest@node@Nextleafid#1{%
2197 \ifnum\forestOve{#1}{@next}=0
2198 \@escapeif{\expandnumberarg\forest@node@nextleafid@Goup{#1}}%
2199 \else
2200 \expandnumberarg\forest@node@nextleafid@Gdown{\forestOve{#1}{@next}}%
2201 \fi
2202 }
2203 \def\forest@node@nextleafid@Goup#1{%
2204 \ifnum\forestOve{#1}{@parent}=0
2205 \PackageError{forest}{get next leaf: this is the last leaf}{}%
2206 \else
2207 \@escapeif{\expandnumberarg\forest@node@Nextleafid{\forestOve{#1}{@parent}}}%
2208 \fi
2209 }
2210 \def\forest@node@nextleafid@Gdown#1{%
2211 \ifnum\forestOve{#1}{@first}=0
2212 #1%
2213 \else
2214 \@escapeif{\expandnumberarg\forest@node@nextleafid@Gdown{\forestOve{#1}{@first}}}%
2215 \fi
2216 }
2217
2218
2219
2220 \def\forest@node@linearnextid{%
2221 \ifnum\forestove{@first}=0
2222 \expandafter\forest@node@linearnextnotdescendantid
2223 \else
2224 \forestove{@first}%
2225 \fi
2226 }
2227 \def\forest@node@linearnextnotdescendantid{%
2228 \expandnumberarg\forest@node@Linearnextnotdescendantid{\forest@cn}%
2229 }
2230 \def\forest@node@Linearnextnotdescendantid#1{%
2231 \ifnum\forestOve{#1}{@next}=0
2232 \ifnum\forestOve{#1}{@parent}=0
2233 0%
2234 \else
2235 \@escapeif{\expandnumberarg\forest@node@Linearnextnotdescendantid{\forestOve{#1}{@parent}}}%
2236 \fi

```

```

2237 \else
2238   \forestOve{#1}{@next}%
2239 \fi
2240 }
2241 \def\forest@node@linearpreviousid{%
2242   \ifnum\forestove{@previous}=0
2243     \forestove{@parent}%
2244   \else
2245     \forest@node@previousleafid
2246   \fi
2247 }

```

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.

```

2248
2249 \def\forest@ifancestorof#1{% is the current node an ancestor of #1? Yes: #2, no: #3
2250   \begingroup
2251   \expandnumberarg\forest@ifancestorof@{\forestOve{#1}{@parent}}%
2252 }
2253 \def\forest@ifancestorof@#1{%
2254   \ifnum#1=0
2255     \def\forest@ifancestorof@next{\expandafter\endgroup\@secondoftwo}%
2256   \else
2257     \ifnum\forest@cn=#1
2258       \def\forest@ifancestorof@next{\expandafter\endgroup\@firstoftwo}%
2259     \else
2260       \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.

```

2261       \def\forest@ifancestorof@next{\expandafter\endgroup\@secondoftwo}%
2262     }{%
2263       \csdef{forest@circularity@used#1}{}%
2264       \def\forest@ifancestorof@next{\expandnumberarg\forest@ifancestorof@{\forestOve{#1}{@parent}}}%
2265     }%
2266   \fi
2267 \fi
2268 \forest@ifancestorof@next
2269 }

```

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

```

2270 \NewDocumentCommand\forestdebugtypeouttrees{o}{%
2271   \forestdebug@typeouttrees\forest@temp
2272   \typeout{%
2273     \forestdebugtypeouttreesprefix
2274     \IfValueTF{#1}{#1: }{}%
2275     \detokenize\expandafter{\forest@temp}%
2276     \forestdebugtypeouttreessuffix
2277   }%
2278 }
2279 \def\forestdebug@typeouttrees#1{% #1 = cs to store the result
2280   \begingroup
2281   \edef\forest@temp@message{}%
2282   \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.

```

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

```

```

2287 \forest@fornode{\forestdebug@typeouttrees@n}{%
After finding the root, we need to restore our notes about visited nodes.
2288 \begingroup
2289 \forestdebug@typeouttrees@findroot
2290 \expandafter\endgroup
2291 \expandafter\edef\expandafter\forest@cn\expandafter{\forest@cn}%
2292 \forestdebug@typeouttree@build
2293 \appto\forest@temp@message{ }%
2294 }%
2295 }%
2296 \repeat
2297 \expandafter\endgroup
2298 \expandafter\def\expandafter#1\expandafter{\forest@temp@message}%
2299 }
2300 \def\forestdebug@typeouttrees@findroot{%
2301 \let\forestdebug@typeouttrees@next\relax
2302 \edef\forestdebug@typeouttrees@parent{\forest@ve{\forest@cn}{@parent}}%
2303 \ifnum\forestdebug@typeouttrees@parent=0
2304 \else
2305 \ifcsdef\forestdebug@typeouttree@used@\forest@cn}{-}{%
2306 \csdef\forestdebug@typeouttree@used@\forest@cn}{-}%
2307 \edef\forest@cn{\forestdebug@typeouttrees@parent}%
2308 \let\forestdebug@typeouttrees@next\forestdebug@typeouttrees@findroot
2309 }%
2310 \fi
2311 \forestdebug@typeouttrees@next
2312 }
2313 \def\forestdebug@typeouttree#1#2{% #1=root id, #2=cs to receive result
2314 \begingroup
2315 \edef\forest@temp@message{}%
2316 \forest@fornode{#1}{\forestdebug@typeouttree@build}%
2317 \expandafter\endgroup
2318 \expandafter\edef\expandafter#2\expandafter{\forest@temp@message}%
2319 }
2320 \NewDocumentCommand\forestdebugtypeouttree{o m}{%
2321 \forestdebug@typeouttree{#1}\forest@temp
2322 \typeout{\IfValueTF{#1}{#1: }{}\forest@temp}%
2323 }

```

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

```

2324 \def\forestdebug@typeouttree@build{%
2325 \eappto\forest@temp@message{\forestdebugtypeouttreenodeinfo%]
2326 \ifcsdef\forestdebug@typeouttree@used@\forest@cn}{*}{-}%
2327 }%
2328 \ifcsdef\forestdebug@typeouttree@used@\forest@cn}{-}{%
2329 \csdef\forestdebug@typeouttree@used@\forest@cn}{-}%
2330 \forest@node@foreachchild{\forestdebug@typeouttree@build}%
2331 }%
2332 \eappto\forest@temp@message{[%
2333 ]}%
2334 }
2335 \def\forestdebugtypeouttreenodeinfo{\forest@cn}
2336 \def\forestdebugtypeouttreesprefix{}
2337 \def\forestdebugtypeouttreessuffix{}

```

6.3 Node options

6.3.1 Option-declaration mechanism

Common code for declaring options.

```

2338 \def\forest@declarehandler#1#2#3{% #1=handler for specific type,#2=option name,#3=default value
2339   \pgfkeyssetvalue{/forest/#2}{#3}%
2340   \appto\forest@node@init{\forest@init{#2}}%
2341   \pgfkeyssetvalue{/forest/#2/node@or@reg}{\forest@cn}%
2342   \forest@convert@others@to@underscores{#2}\forest@pgfmathoptionname
2343   \edef\forest@marshal{%
2344     \noexpand#1{/forest/#2}{/forest}{#2}{\forest@pgfmathoptionname}%
2345   }\forest@marshal
2346 }
2347 \def\forest@def@with@pgfeov#1#2{% \pgfeov mustn't occur in the arg of the .code handler!!!
2348   \long\def#1##1\pgfeov{#2}%
2349 }

```

Option-declaration handlers.

```

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

```

```

2397 \forestOget{\forest@setter@node}{#3}\forest@temp
2398 \ifdefempty{\forest@temp}{%
2399 \forestOset{\forest@setter@node}{#3}{##1}%
2400 }{%
2401 \forestOappto{\forest@setter@node}{#3}{#5##1}%
2402 }%
2403 },
2404 #2/+ #3/.code={%
2405 \forestOget{\forest@setter@node}{#3}\forest@temp
2406 \ifdefempty{\forest@temp}{%
2407 \forestOset{\forest@setter@node}{#3}{##1}%
2408 }{%
2409 \forestOpreto{\forest@setter@node}{#3}{##1#5}%
2410 }%
2411 }%
2412 }%
2413 }%
2414 \pgfkeyssetvalue{#1/option@name}{#3}%
2415 \pgfkeyssetvalue{#1+/option@name}{#3}%
2416 \pgfkeyssetvalue{#2/+ #3/option@name}{#3}%
2417 \pgfkeyslet{#1/@type}\forestmathtype@generic % for .process & co
2418 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@toks{##1}{#3}}%
2419 }
2420 \def\forest@declareautowrappedtoks@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
2421 \forest@declaretoks@handler{#1}{#2}{#3}{#4}%
2422 \forest@copycommandkey{#1}{#1'}%
2423 \pgfkeysalso{#1/.style={#1'/.wrap value={##1}}}%
2424 \pgfkeyssetvalue{#1'/option@name}{#3}%
2425 \forest@copycommandkey{#1+}{#1+'}%
2426 \pgfkeysalso{#1+/.style={#1+'/.wrap value={##1}}}%
2427 \pgfkeyssetvalue{#1+'/option@name}{#3}%
2428 \forest@copycommandkey{#2/+ #3}{#2/+ #3'}%
2429 \pgfkeysalso{#2/+ #3/.style={#2/+ #3'/.wrap value={##1}}}%
2430 \pgfkeyssetvalue{#2/+ #3'/option@name}{#3}%
2431 }
2432 \def\forest@declarereadonlydimen@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2433 % this is to have 'pt' with the correct category code
2434 \pgfutil@tempdima=\pgfkeysvalueof{/forest/#3}\relax
2435 \edef\forest@marshal{%
2436 \noexpand\pgfkeyssetvalue{/forest/#3}{\the\pgfutil@tempdima}%
2437 }\forest@marshal
2438 \pgfkeysalso{%
2439 #2/if #3/.code n args={3}{%
2440 \forestOget{#3}\forest@temp@option@value
2441 \ifdim\forest@temp@option@value=##1\relax
2442 \pgfkeysalso{##2}%
2443 \else
2444 \pgfkeysalso{##3}%
2445 \fi
2446 },
2447 #2/if #3</.code n args={3}{%
2448 \forestOget{#3}\forest@temp@option@value
2449 \ifdim\forest@temp@option@value>##1\relax
2450 \pgfkeysalso{##3}%
2451 \else
2452 \pgfkeysalso{##2}%
2453 \fi
2454 },
2455 #2/if #3>/.code n args={3}{%
2456 \forestOget{#3}\forest@temp@option@value
2457 \ifdim\forest@temp@option@value<##1\relax

```

```

2458     \pgfkeysalso{##3}%
2459     \else
2460     \pgfkeysalso{##2}%
2461     \fi
2462 },
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 #2/where #3>/.style n args={3}{for tree={#2/if #3>={##1}{##2}{##3}}},
2466 }%
2467 \pgfkeyslet{#1/@type}\forestmathtype@dimen % for .process & co
2468 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@dimen{##1}{#3}}%
2469 }
2470 \def\forest@declaredimen@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2471 \forest@declarereadonlydimen@handler{#1}{#2}{#3}{#4}%
2472 \pgfkeysalso{%
2473 #1/.code={%
2474 \forestmathsetlengthmacro\forest@temp{##1}%
2475 \forest@let{\forest@setter@node}{#3}\forest@temp
2476 },
2477 #1+/.code={%
2478 \forestmathsetlengthmacro\forest@temp{##1}%
2479 \pgfutil@tempdima=\forestove{#3}
2480 \advance\pgfutil@tempdima\forest@temp\relax
2481 \forest@set{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2482 },
2483 #1-/.code={%
2484 \forestmathsetlengthmacro\forest@temp{##1}%
2485 \pgfutil@tempdima=\forestove{#3}
2486 \advance\pgfutil@tempdima-\forest@temp\relax
2487 \forest@set{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2488 },
2489 #1*/.style={%
2490 #1={#4()}*(##1)}%
2491 },
2492 #1:/ .style={%
2493 #1={#4()/ (##1)}%
2494 },
2495 #1'/.code={%
2496 \pgfutil@tempdima=##1\relax
2497 \forest@set{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2498 },
2499 #1'+/.code={%
2500 \pgfutil@tempdima=\forestove{#3}\relax
2501 \advance\pgfutil@tempdima##1\relax
2502 \forest@set{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2503 },
2504 #1'-/.code={%
2505 \pgfutil@tempdima=\forestove{#3}\relax
2506 \advance\pgfutil@tempdima-##1\relax
2507 \forest@set{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2508 },
2509 #1'*/.style={%
2510 \pgfutil@tempdima=\forestove{#3}\relax
2511 \multiply\pgfutil@tempdima##1\relax
2512 \forest@set{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2513 },
2514 #1':/.style={%
2515 \pgfutil@tempdima=\forestove{#3}\relax
2516 \divide\pgfutil@tempdima##1\relax
2517 \forest@set{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
2518 },

```



```

2519 }%
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 \pgfkeyssetvalue{#1':/option@name}{#3}%
2530 }
2531 \def\forest@declarereadonlycount@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2532 \pgfkeysalso{
2533 #2/if #3/.code n args={3}{%
2534 \foresttoget{#3}\forest@temp@option@value
2535 \ifnum\forest@temp@option@value=##1\relax
2536 \pgfkeysalso{##2}%
2537 \else
2538 \pgfkeysalso{##3}%
2539 \fi
2540 },
2541 #2/if #3</.code n args={3}{%
2542 \foresttoget{#3}\forest@temp@option@value
2543 \ifnum\forest@temp@option@value>##1\relax
2544 \pgfkeysalso{##3}%
2545 \else
2546 \pgfkeysalso{##2}%
2547 \fi
2548 },
2549 #2/if #3>/.code n args={3}{%
2550 \foresttoget{#3}\forest@temp@option@value
2551 \ifnum\forest@temp@option@value<##1\relax
2552 \pgfkeysalso{##3}%
2553 \else
2554 \pgfkeysalso{##2}%
2555 \fi
2556 },
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 #2/where #3>/.style n args={3}{for tree={#2/if #3>={##1}{##2}{##3}}},
2560 }%
2561 \pgfkeyslet{#1/@type}\forestmathtype@count % for .process & co
2562 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@count{##1}{#3}}%
2563 }
2564 \def\forest@declarecount@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2565 \forest@declarereadonlycount@handler{#1}{#2}{#3}{#4}%
2566 \pgfkeysalso{
2567 #1/.code={%
2568 \forestmathtruncatemacro\forest@temp{##1}%
2569 \forestOlet{\forest@setter@node}{#3}\forest@temp
2570 },
2571 #1+/.code={%
2572 \forestmathtruncatemacro\forest@temp{##1}%
2573 \c@pgf@counta=\forestove{#3}\relax
2574 \advance\c@pgf@counta\forest@temp\relax
2575 \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2576 },
2577 #1-/.code={%
2578 \forestmathtruncatemacro\forest@temp{##1}%
2579 \c@pgf@counta=\forestove{#3}\relax

```

```

2580     \advance\c@pgf@counta-\forest@temp\relax
2581     \forestOreset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2582 },
2583 #1*/.code={%
2584     \forestmathtruncatemacro\forest@temp{##1}%
2585     \c@pgf@counta=\forestove{#3}\relax
2586     \multiply\c@pgf@counta\forest@temp\relax
2587     \forestOreset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2588 },
2589 #1:/.code={%
2590     \forestmathtruncatemacro\forest@temp{##1}%
2591     \c@pgf@counta=\forestove{#3}\relax
2592     \divide\c@pgf@counta\forest@temp\relax
2593     \forestOreset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2594 },
2595 #1'/.code={%
2596     \c@pgf@counta=##1\relax
2597     \forestOreset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2598 },
2599 #1'+/.code={%
2600     \c@pgf@counta=\forestove{#3}\relax
2601     \advance\c@pgf@counta##1\relax
2602     \forestOreset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2603 },
2604 #1'-/.code={%
2605     \c@pgf@counta=\forestove{#3}\relax
2606     \advance\c@pgf@counta-##1\relax
2607     \forestOreset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2608 },
2609 #1'*/.style={%
2610     \c@pgf@counta=\forestove{#3}\relax
2611     \multiply\c@pgf@counta##1\relax
2612     \forestOreset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2613 },
2614 #1':/.style={%
2615     \c@pgf@counta=\forestove{#3}\relax
2616     \divide\c@pgf@counta##1\relax
2617     \forestOreset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
2618 },
2619 }%
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 \pgfkeyssetvalue{#1':/option@name}{#3}%
2630 }

```

Nothing else should be defined in this namespace.

```

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

```

```

2639     \forestOlet{\forest@setter@node}{#3}\forest@temp
2640 },
2641 #1/.default=1,
2642 #2/not #3/.code={\forestOset{\forest@setter@node}{#3}{0}},
2643 #2/if #3/.code 2 args={%
2644     \forestoget{#3}\forest@temp@option@value
2645     \ifnum\forest@temp@option@value=0
2646         \pgfkeysalso{##2}%
2647     \else
2648         \pgfkeysalso{##1}%
2649     \fi
2650 },
2651 #2/where #3/.style 2 args={for tree={#2/if #3={##1}{##2}}}
2652 }%
2653 \pgfkeyssetvalue{#1/option@name}{#3}%
2654 \pgfkeyslet{#1/@type}\forest@math@type@count % for .process & co
2655 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@count{##1}{#3}}%
2656 }
2657 \forestset{
2658     declare toks/.code 2 args={%
2659         \forest@declarehandler\forest@declaretoks@handler{#1}{#2}%
2660     },
2661     declare autowrapped toks/.code 2 args={%
2662         \forest@declarehandler\forest@declareautowrappedtoks@handler{#1}{#2}%
2663     },
2664     declare keylist/.code 2 args={%
2665         \forest@declarehandler\forest@declarekeylist@handler{#1}{#2}%
2666     },
2667     declare readonly dimen/.code 2 args={%
2668         \forestmathsetlengthmacro\forest@temp{#2}%
2669         \edef\forest@marshal{%
2670             \unexpanded{\forest@declarehandler\forest@declarereadonlydimen@handler{#1}}{\forest@temp}%
2671         }\forest@marshal
2672     },
2673     declare dimen/.code 2 args={%
2674         \forestmathsetlengthmacro\forest@temp{#2}%
2675         \edef\forest@marshal{%
2676             \unexpanded{\forest@declarehandler\forest@declaredimen@handler{#1}}{\forest@temp}%
2677         }\forest@marshal
2678     },
2679     declare readonly count/.code 2 args={%
2680         \forestmathtruncatemacro\forest@temp{#2}%
2681         \edef\forest@marshal{%
2682             \unexpanded{\forest@declarehandler\forest@declarereadonlycount@handler{#1}}{\forest@temp}%
2683         }\forest@marshal
2684     },
2685     declare count/.code 2 args={%
2686         \forestmathtruncatemacro\forest@temp{#2}%
2687         \edef\forest@marshal{%
2688             \unexpanded{\forest@declarehandler\forest@declarecount@handler{#1}}{\forest@temp}%
2689         }\forest@marshal
2690     },
2691     declare boolean/.code 2 args={%
2692         \forestmath@if{#2}{%
2693             \def\forest@temp{1}%
2694         }{%
2695             \def\forest@temp{0}%
2696         }%
2697         \edef\forest@marshal{%
2698             \unexpanded{\forest@declarehandler\forest@declareboolean@handler{#1}}{\forest@temp}%
2699         }\forest@marshal

```

2700 } ,

7 Handlers

[illegible]

```

2757 \forest@wrap@n@pgfm@do{#1}{5}},
2758 /handlers/.wrap 6 pgfm@args/.code n args={7}{%
2759 \forest@wrap@n@pgfm@args{#2}{#3}{#4}{#5}{#6}{#7}{-}{-}{6}%
2760 \forest@wrap@n@pgfm@do{#1}{6}},
2761 /handlers/.wrap 7 pgfm@args/.code n args={8}{%
2762 \forest@wrap@n@pgfm@args{#2}{#3}{#4}{#5}{#6}{#7}{#8}{-}{-}{7}%
2763 \forest@wrap@n@pgfm@do{#1}{7}},
2764 /handlers/.wrap 8 pgfm@args/.code n args={9}{%
2765 \forest@wrap@n@pgfm@args{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}{8}%
2766 \forest@wrap@n@pgfm@do{#1}{8}},
2767 }
2768 \def\forest@wrap@n@pgfm@args#1#2#3#4#5#6#7#8#9{%
2769 \forestmathparse{#1}\let\forest@wrap@arg@i\forestmathresult
2770 \ifnum#9>1 \forestmathparse{#2}\let\forest@wrap@arg@ii\forestmathresult\fi
2771 \ifnum#9>2 \forestmathparse{#3}\let\forest@wrap@arg@iii\forestmathresult\fi
2772 \ifnum#9>3 \forestmathparse{#4}\let\forest@wrap@arg@iv\forestmathresult\fi
2773 \ifnum#9>4 \forestmathparse{#5}\let\forest@wrap@arg@v\forestmathresult\fi
2774 \ifnum#9>5 \forestmathparse{#6}\let\forest@wrap@arg@vi\forestmathresult\fi
2775 \ifnum#9>6 \forestmathparse{#7}\let\forest@wrap@arg@vii\forestmathresult\fi
2776 \ifnum#9>7 \forestmathparse{#8}\let\forest@wrap@arg@viii\forestmathresult\fi
2777 \edef\forest@wrap@args{%
2778 {\expandonce\forest@wrap@arg@i}
2779 \ifnum#9>1 {\expandonce\forest@wrap@arg@ii}\fi
2780 \ifnum#9>2 {\expandonce\forest@wrap@arg@iii}\fi
2781 \ifnum#9>3 {\expandonce\forest@wrap@arg@iv}\fi
2782 \ifnum#9>4 {\expandonce\forest@wrap@arg@v}\fi
2783 \ifnum#9>5 {\expandonce\forest@wrap@arg@vi}\fi
2784 \ifnum#9>6 {\expandonce\forest@wrap@arg@vii}\fi
2785 \ifnum#9>7 {\expandonce\forest@wrap@arg@viii}\fi
2786 }%
2787 }
2788 \def\forest@wrap@n@pgfm@do#1#2{%
2789 \ifcase#2\relax
2790 \or\def\forest@wrap@code##1{#1}%
2791 \or\def\forest@wrap@code##1#2{#1}%
2792 \or\def\forest@wrap@code##1#2#3{#1}%
2793 \or\def\forest@wrap@code##1#2#3#4{#1}%
2794 \or\def\forest@wrap@code##1#2#3#4#5{#1}%
2795 \or\def\forest@wrap@code##1#2#3#4#5#6{#1}%
2796 \or\def\forest@wrap@code##1#2#3#4#5#6#7{#1}%
2797 \or\def\forest@wrap@code##1#2#3#4#5#6#7#8{#1}%
2798 \fi
2799 \forest@wrap@pgfm@args@@@wrapandpasson
2800 }

```

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

```

2801 \def\forest@wrap@pgfm@args@@@wrapandpasson{%
2802 \expandafter\expandafter\expandafter\forest@temp@toks
2803 \expandafter\expandafter\expandafter{%
2804 \expandafter\forest@wrap@code\forest@wrap@args}%
2805 \expandafter\pgfkeysalso\expandafter{%
2806 \expandafter\pgfkeyscurrentpath\expandafter=\expandafter{%
2807 \the\forest@temp@toks}}%
2808 }

```

7.1 .process

```

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

```

```

2814 \forest@newglobalarray\forest@process@result@
2815 \newif\ifforest@process@returnarray@

Processing instruction need not (but may) be enclosed in braces.
2816 \def\forest@process#1#2#{% #1 = true/false (should we return an array?)
2817 % #2 = processing instructions (if non-empty),
2818 % (initial) args follow
2819 \ifblank{#2}{\forest@process@a{#1}}{\forest@process@a{#1}{#2}}%
2820 }
2821 \Inline\def\forest@process@a#1#2{%
2822 \begingroup
2823 \forest@process@left@clear
2824 \forest@process@right@clear
2825 \forest@process@sav@clear
2826 \let\forest@process@sav@type\forest@math@type@generic
2827 \csname forest@process@returnarray@#1\endcsname
2828 \def\forest@top@extend@next{%
2829 \ExpandIfT{forest@debug}{%
2830 \edef\forest@process@debug@args{\unexpanded{#2}}%
2831 \forest@processor@debug@info@template{Start "\unexpanded{#2}}%
2832 }%
2833 \forest@process@cat@regime
2834 \endlinechar=-1
2835 \scantokens{#2}%
2836 \forest@process@finish
2837 }%
2838 \forest@process@right@top@extend
2839 }
2840 \pgfkeys{%
2841 /handlers/.process/.code={%
2842 \forest@process{true}#1\forest@eov
2843 \edef\forest@marshal{%
2844 \noexpand\pgfkeysalso{\noexpand\pgfkeyscurrentpath=\forest@process@result@values}%
2845 }\forest@marshal
2846 },
2847 /forest/copy command key={/handlers/.process}{/handlers/.process args},
2848 }
2849 \def\forest@process@finish{%
2850 \ifforest@process@returnarray@
2851 \forest@process@finish@array
2852 \else
2853 \forest@process@finish@single
2854 \fi
2855 \global\let\forest@process@result@type\forest@math@result@type
2856 \ifforest@debug@process\forest@process@debug@end\fi
2857 \endgroup
2858 }
2859 \def\forest@process@finish@single{%
2860 \edef\forest@temp{\forest@process@finish@single@%
2861 \the\numexpr\forest@process@left@N-\forest@process@left@M\relax
2862 \the\numexpr\forest@process@right@N-\forest@process@right@M\relax
2863 }%
2864 \ifcsname\forest@temp\endcsname
2865 \csname\forest@temp\endcsname
2866 \global\let\forest@process@result\forest@temp
2867 \else
2868 \forest@process@length@error
2869 \fi
2870 }
2871 \csdef{forest@process@finish@single@10}{\forest@process@left@toppop\forest@temp}
2872 \csdef{forest@process@finish@single@01}{\forest@process@right@toppop\forest@temp}

```

```

2873 \def\forest@process@finish@array{%
2874   \forest@process@result@clear
2875   \forest@temp@count\forest@process@left@M\relax
2876   \forest@loop
2877   \ifnum\forest@temp@count<\forest@process@left@N\relax
2878     \forest@process@left@get@{\the\forest@temp@count}\forest@temp
2879     \forest@process@result@letappend\forest@temp
2880     \advance\forest@temp@count1
2881   \forest@repeat
2882   \forest@temp@count\forest@process@right@M\relax
2883   \forest@loop
2884   \ifnum\forest@temp@count<\forest@process@right@N\relax
2885     \forest@process@right@get@{\the\forest@temp@count}\forest@temp
2886     \forest@process@result@letappend\forest@temp
2887     \advance\forest@temp@count1
2888   \forest@repeat
2889 }

```

Debugging and error messages.

```

2890 \ifforestdebug
2891   \let\forest@process@d\forest@process@b
2892   \def\forest@process@b#1\forest@eov{% save and print initial arguments
2893     \edef\forest@process@debug@args{\unexpanded{#1}}%
2894     \typeout{[[forest .process] Start "\unexpanded{#1}"}%
2895     \forest@process@d#1\forest@eov
2896   }
2897 \fi
2898 \def\forest@process@debug@end{%
2899   \typeout{[[forest .process] End "\expandonce{\forest@process@debug@args}" -> "\forest@process@left@values\fo
2900 }
2901 \def\forest@process@lengtherror{%
2902   \PackageError{forest}{%
2903     The ".process" expression was expected to evaluate to a single argument,
2904     but the result is \the\forest@process@result@N
2905     \space items long.}{}%
2906 }

```

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

```

2907 \def\forest@def@processor#1{%
2908   {%
2909     \def\forest@dp@double##1{%
2910       \gdef\forest@global@temp{\forest@def@processor@{#1}{##1}}%
2911     }%
2912     \let\\forest@dp@double
2913     \catcode'#1=13
2914     \scantokens{\#1}%
2915   }%
2916   \forest@global@temp
2917 }
2918 \def\forest@def@processor@#1#2{%
2919   % #1 = instruction char (normal catcode), #2 = instruction char (active)
2920   % #3 = default n (optional numeric arg, which precedes any other args;
2921   %           if the default is empty, this means no optional n)
2922   % #4 = args spec,
2923   % #5 = code
2924   \eappto\forest@process@catregime{%
2925     \unexpanded{\let#2}\expandonce{\csname forest@processor@#1\endcsname}%
2926     \unexpanded{\catcode'#1=13 }%
2927   }%
2928   \def\forest@def@processor@inschar{#1}%
2929   \forest@def@processor@@
2930 }

```

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

```

2931 \def\forest@def@processor@@#1{%
2932   \ifstrepty{#1}{%
2933     \forest@def@processor@@non
2934   }{%
2935     \def\forest@def@processor@@default@n{#1}%
2936     \forest@def@processor@@n
2937   }%
2938 }

```

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:

```

2939 \def\forest@def@processor@@non#1#2{% #1=args spec, #2=code
2940   \csedef{forest@processor@\forest@def@processor@inschar}#1{%
2941     \relax %% we need this (see above)
2942     \unexpanded{#2}%
2943     \expandafter\forest@def@processor@debuginfo\expandafter{%
2944       \expandafter"\forest@def@processor@inschar"\ifstrepty{#1}{(#{1})}%
2945     \ignorespaces
2946   }%
2947 }

```

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

```

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

```



```

3040 \csname forest@processor@-@\forestmathresulttype\endcsname
3041 \forest@process@left@letappend\forestmathresult
3042 }
3043 \cslet{forest@processor@-@\forestmathtype@generic}\relax
3044 \csdef{forest@processor@-@\forestmathtype@count}{%
3045 \forestmathadd{\forestmathzero}{-\forestmathresult}}
3046 \csletcs{forest@processor@-@\forestmathtype@dimen}
3047 {forest@processor@-@\forestmathtype@count}
3048 \csletcs{forest@processor@-@\forestmathtype@unitless}
3049 {forest@processor@-@\forestmathtype@count}
3050 \csdef{forest@processor@-@\forestmathtype@textasc}{%
3051 \let\forestmathresulttype\forestmathtype@textdesc}
3052 \csdef{forest@processor@-@\forestmathtype@textdesc}{%
3053 \let\forestmathresulttype\forestmathtype@textasc}
3054
3055 \forest@def@processor{c}{-}{-}{% to lowercase
3056 \forest@process@right@bottompop\forest@temp
3057 \expandafter\lowercase\expandafter{\expandafter\def\expandafter\forest@temp\expandafter{\forest@temp}}}%
3058 \forest@process@left@letappend\forest@temp
3059 }
3060 \forest@def@processor{C}{-}{-}{% to uppercase
3061 \forest@process@right@bottompop\forest@temp
3062 \expandafter\uppercase\expandafter{\expandafter\def\expandafter\forest@temp\expandafter{\forest@temp}}}%
3063 \forest@process@left@letappend\forest@temp
3064 }

```

Expansions:

```

3065 \forest@def@processor{x}{-}{-}{% expand
3066 \forest@process@right@bottompop\forest@temp
3067 \forest@process@left@setappend{\forest@temp}%
3068 \let\forestmathresulttype\forestmathtype@generic
3069 }
3070 \forest@def@processor{o}{1}{-}{% expand once (actually, \forest@count@n times)
3071 \forest@process@right@bottompop\forest@temp
3072 \forest@repeat@n@times{\forest@process@n}{%
3073 \expandafter\expandafter\expandafter\def
3074 \expandafter\expandafter\expandafter\forest@temp
3075 \expandafter\expandafter\expandafter{\forest@temp}%
3076 }%
3077 \expandafter\forest@process@left@setappend\expandafter{\forest@temp}%
3078 \let\forestmathresulttype\forestmathtype@generic
3079 }

```

Access to FOREST data.

```

3080 \forest@def@processor{0}{1}*{}{% option
3081 \forest@process@right@bottompop\forest@temp
3082 \expandafter\forestRNO@getvalueandtype\expandafter{\forest@temp}\forest@tempvalue\forest@temp@type
3083 \let\forestmathresulttype\forest@temp@type
3084 \forest@process@left@letappend\forest@tempvalue
3085 }
3086 \forest@def@processor{R}{1}*{}{% register
3087 \forest@process@right@bottompop\forest@temp
3088 \forest@rget{\forest@temp}\forest@tempvalue
3089 \forest@process@left@letappend\forest@tempvalue
3090 \pgfkeysgetvalue{/forest/\forest@temp/@type}\forest@temp@type
3091 \let\forestmathresulttype\forest@temp@type
3092 }

```

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

```

3093 \forest@def@processor{+}{1}*{}{% join processors = pop one from result
3094 \forest@process@left@toppop\forest@temp
3095 \forest@process@right@letprepend\forest@temp
3096 }

```

```

3097 \forest@def@processor{u}{-}{% ungroup: remove braces and leave in the argument list
3098   \forest@process@right@bottompop\forest@temp
3099   \forest@temparray@clear
3100   \let\forestmathresulttype\forestmathtype@generic
3101   \let\forest@topextend@next\forest@processor@u@
3102   \expandafter\forest@temparray@topextend\forest@temp\forest@eov
3103 }
3104 \def\forest@processor@u@{%
3105   \forest@loop
3106   \ifnum\forest@temparray@N>0
3107     \forest@temparray@toppop\forest@temp
3108     \expandafter\forest@process@right@setprepend\expandafter{\forest@temp}%
3109   \forest@repeat
3110 }
3111 \def\forest@process@check@mn#1#2#3#4{%
3112   % #1 = processor, #2 = given n, #3/#4 = lower/upper bound (inclusive)
3113   \ifnum#3>#2\relax
3114     \forest@process@check@n@error{#1}{#2}{#3<=}{<=#4}%
3115   \else
3116     \ifnum#4<#2\relax
3117       \forest@process@check@n@error{#1}{#2}{#3<=}{<=#4}%
3118     \fi
3119   \fi
3120 }
3121 \def\forest@process@check@m#1#2#3{%
3122   % #1 = processor, #2 = given n, #3 = lower bound (inclusive)
3123   \ifnum#2<#3\relax
3124     \forest@process@check@n@error{#1}{#2}{#3<=}{}%
3125   \fi
3126 }
3127 \def\forest@process@check@n@error#1#2#3#4{%
3128   \PackageError{forest}{'.process' instruction '#1' requires a numeric modifier #3n#4, but n="#2" was given.}
3129 }
3130 \newif\ifforest@process@W
3131 \forest@def@processor{w}{1}{-}{% consuming wrap: first test 1<=#1<=9
3132   \forest@process@Wtrue
3133   \forest@process@check@mn{w}{0\the\forest@process@n}{1}{9}%
3134   \expandafter\forest@processor@wW@\expandafter{\the\forest@process@n}%
3135 }
3136 \forest@def@processor{W}{1}{-}{% nonconsuming wrap: first test 1<=#1<=9
3137   \forest@process@Wfalse
3138   \forest@process@check@mn{W}{0\the\forest@process@n}{1}{9}%
3139   \expandafter\forest@processor@wW@\expandafter{\the\forest@process@n}%
3140 }
3141 \def\forest@processor@wW@#1{%
3142   \forest@process@left@checkindex{\forest@process@left@N-#1}%
3143   \edef\forest@marshal{%
3144     \edef\noexpand\forest@temp@args{%
3145       \noexpand\forest@process@left@valuesfromrange
3146       {\number\numexpr\forest@process@left@N-#1}%
3147       {\the\forest@process@left@N}%
3148     }%
3149   }\forest@marshal
3150   \ifforest@process@W
3151     \advance\forest@process@left@N-#1\relax
3152   \fi
3153   \forest@process@right@bottompop\forest@temp@macrobody
3154   \expandafter\forest@def@n\expandafter\forest@process@temp@macro\expandafter{\expandafter#1\expandafter}\exp
3155   \expandafter\expandafter\expandafter\forest@process@left@setappend\expandafter\expandafter\expandafter{\exp
3156   \let\forestmathresulttype\forestmathtype@generic
3157 }

```

```

3158 \def\forest@def@n#1#2{\csname forest@def@n@#2\endcsname#1}
3159 \csdef{forest@def@n@1}#1{\def#1##1}
3160 \csdef{forest@def@n@2}#1{\def#1##1##2}
3161 \csdef{forest@def@n@3}#1{\def#1##1##2##3}
3162 \csdef{forest@def@n@4}#1{\def#1##1##2##3##4}
3163 \csdef{forest@def@n@5}#1{\def#1##1##2##3##4##5}
3164 \csdef{forest@def@n@6}#1{\def#1##1##2##3##4##5##6}
3165 \csdef{forest@def@n@7}#1{\def#1##1##2##3##4##5##6##7}
3166 \csdef{forest@def@n@8}#1{\def#1##1##2##3##4##5##6##7##8}
3167 \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.

```

3168 \forest@def@processor{s}{1}{-}{%
3169   \forest@temptrue   % delete the originals
3170   \expandafter\forest@processor@save\expandafter{%
3171     \the\numexpr\forest@process@left@N-\forest@process@n}}
3172 \forest@def@processor{S}{1}{-}{%
3173   \forest@tempfalse  % keep the originals
3174   \expandafter\forest@processor@save\expandafter{%
3175     \the\numexpr\forest@process@left@N-\forest@process@n}}
3176 \def\forest@processor@save#1{%
3177   \forest@process@left@checkindex{#1}%
3178   \forest@temp@count#1
3179   \forest@loop
3180   \ifnum\forest@temp@count<\forest@process@left@N\relax
3181     \forest@process@left@get@{\the\forest@temp@count}\forest@temp
3182     \forest@process@saved@letappend\forest@temp
3183     \advance\forest@temp@count+1
3184   \forest@repeat
3185   \let\forest@process@savedtype\forestmathresulttype
3186   \ifforest@temp
3187     \forest@process@left@N=#1
3188   \fi
3189 }

```

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.

```

3190 \forest@def@processor{l}{0}{-}{%
3191   \forest@temptrue
3192   \forest@processor@U@@
3193 }
3194 \forest@def@processor{L}{0}{-}{%
3195   \forest@tempfalse
3196   \forest@processor@U@@
3197 }
3198
3199 \def\forest@processor@U@@{%
3200   \ifnum\forest@process@n=0
3201     \forest@process@n\forest@process@saved@N\relax
3202   \fi
3203   \expandafter\forest@processor@U@@@%
3204   \the\numexpr\forest@process@saved@N-\forest@process@n}%
3205 }
3206 \def\forest@processor@U@@@#1{%
3207   \forest@temp@count#1
3208   \forest@loop
3209   \ifnum\forest@temp@count<\forest@process@saved@N\relax
3210     \forest@process@saved@get@{\the\forest@temp@count}\forest@temp
3211     \forest@process@left@letappend\forest@temp
3212     \advance\forest@temp@count1
3213   \forest@repeat

```

```

3214 \let\forestmathresulttype\forest@process@savetype
3215 \ifforest@temp
3216 \let\forest@process@savetype\forestmathtype@none
3217 \forest@process@saved@N#1
3218 \fi
3219 }

Boolean operations:
3220 \forest@def@processor{&}{2}{-}{% and
3221 \def\forest@tempa{1}%
3222 \forest@repeat@n@times{\forest@process@n}{%
3223 \forest@process@left@toppop\forest@tempb
3224 \edef\forest@tempa{\ifnum10<\forest@tempa\forest@tempb\space 1\else0\fi}%
3225 }%
3226 \forest@process@left@esetappend{\forest@tempa}%
3227 \let\forestmathresulttype\forestmathtype@count
3228 }
3229 \forest@def@processor{||}{2}{-}{% or
3230 \def\forest@tempa{0}%
3231 \forest@repeat@n@times{\forest@process@n}{%
3232 \forest@process@left@toppop\forest@tempb
3233 \edef\forest@tempa{\ifnum0=\forest@tempa\forest@tempb\space 0\else1\fi}%
3234 }%
3235 \forest@process@left@esetappend{\forest@tempa}%
3236 \let\forestmathresulttype\forestmathtype@count
3237 }
3238 \forest@def@processor{!}{-}{-}{% not
3239 \forest@process@left@toppop\forest@temp
3240 \forest@process@left@esetappend{\ifnum0=\forest@temp\space 1\else0\fi}%
3241 \let\forestmathresulttype\forestmathtype@count
3242 }
3243 \forest@def@processor{?}{-}{-}{%
3244 \forest@process@left@toppop\forest@temp
3245 \forest@process@right@bottompop\forest@tempa
3246 \forest@process@right@bottompop\forest@tempb
3247 \ifnum\forest@temp=0
3248 \forest@process@right@letprepend\forest@tempb
3249 \else
3250 \forest@process@right@letprepend\forest@tempa
3251 \fi
3252 \let\forestmathresulttype\forestmathtype@generic
3253 }

```

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

```

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

```

```

3271 \forest@process@left@toppop\forest@tempb
3272 \forest@process@left@toppop\forest@tempa
3273 \ifx\forestmathresulttype\forestmathtype@generic
3274 \forest@cmp@error\forest@tempa\forest@tempb
3275 \else
3276 \forestmathgt{\forest@tempa}{\forest@tempb}%
3277 \forest@process@left@esetappend{\forestmathresult}%
3278 \fi
3279 }

Various.
3280 \forest@def@processor{r}{\}{\}{% reverse keylist
3281 \forest@process@right@bottompop\forest@temp
3282 \expandafter\forest@processor@r@\expandafter{\forest@temp}%
3283 }
3284 \def\forest@processor@r@#1{%
3285 \forest@temp@toks{}%
3286 \def\forest@tempcomma{,%}%
3287 \pgfkeys{/forest}{split={#1}{,}{process@rk}}%
3288 \forest@process@left@esetappend{\the\forest@temp@toks}%
3289 \let\forestmathresulttype\forestmathtype@generic
3290 }
3291 \forestset{%
3292 process@rk/.code={%
3293 \epretotoks\forest@temp@toks{#1\forest@tempcomma}%
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@for@node{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=pgfmname,#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\forest@temp@option@value}
3344       \ifpgfutil@in@
3345         \pgfkeysalso{##2}%
3346       \else
3347         \pgfkeysalso{##3}%
3348       \fi
3349     },
3350   }%
3351   \ifstrempy{#5}{%
3352     \pgfkeysalso{%
3353       #1+/.code={\foresttrappto{#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           \foresttrappto{#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}\forestmathtype@generic % for .process & co
3380   \pgfmathdeclarefunction{#4}{1}{\forest@pgfmnamehelper@register@toks{##1}{#3}}%
3381 }
3382 \def\forest@declareautowrappedtoksregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmname,#5=infix
3383   \forest@declaretoksregister@handler{#1}{#2}{#3}{#4}%
3384   \forest@copycommandkey{#1}{#1'}%
3385   \pgfkeysalso{#1/.style={#1'/.wrap value={##1}}}%
3386   \pgfkeyssetvalue{#1'/option@name}{#3}%
3387   \forest@copycommandkey{#1+}{#1+'}%

```

```

3388 \pgfkeysalso{#1+/.style={#1+'/.wrap value={##1}}}%
3389 \pgfkeyssetvalue{#1+'/.option@name}{#3}%
3390 \forest@copycommandkey{#2/+#3}{#2/+#3'}%
3391 \pgfkeysalso{#2/+#3/.style={#2/+#3'/.wrap value={##1}}}%
3392 \pgfkeyssetvalue{#2/+#3'/.option@name}{#3}%
3393 }
3394 \def\forest@declarereadonlydimenregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
3395 \pgfkeysalso{%
3396 #2/if #3/.code n args={3}{%
3397 \forest@trget{#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 \forest@trget{#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 \forest@trget{#3}\forest@temp@option@value
3414 \ifdim\forest@temp@option@value<##1\relax
3415 \pgfkeysalso{##3}%
3416 \else
3417 \pgfkeysalso{##2}%
3418 \fi
3419 },
3420 }%
3421 \pgfkeyslet{#1/@type}\forest@math@type@dimen % for .process & co
3422 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@dimen{##1}{#3}}%
3423 }
3424 \def\forest@declaredimenregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
3425 \forest@declarereadonlydimenregister@handler{#1}{#2}{#3}{#4}%
3426 \pgfkeysalso{%
3427 #1/.code={%
3428 \forest@math@setlengthmacro\forest@temp{##1}%
3429 \forest@trlet{#3}\forest@temp
3430 },
3431 #1+/.code={%
3432 \forest@math@setlengthmacro\forest@temp{##1}%
3433 \pgfutil@tempdima=\forest@trve{#3}
3434 \advance\pgfutil@tempdima\forest@temp\relax
3435 \forest@reset{#3}{\the\pgfutil@tempdima}%
3436 },
3437 #1-/.code={%
3438 \forest@math@setlengthmacro\forest@temp{##1}%
3439 \pgfutil@tempdima=\forest@trve{#3}
3440 \advance\pgfutil@tempdima-\forest@temp\relax
3441 \forest@reset{#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=\forestrve{#3}\relax
3525       \advance\c@pgf@counta\forest@temp\relax
3526       \forestreset{#3}{\the\c@pgf@counta}%
3527     },
3528     #1-/.code={%
3529       \forestmathtruncatemacro\forest@temp{##1}%
3530       \c@pgf@counta=\forestrve{#3}\relax
3531       \advance\c@pgf@counta-\forest@temp\relax
3532       \forestreset{#3}{\the\c@pgf@counta}%
3533     },
3534     #1*/.code={%
3535       \forestmathtruncatemacro\forest@temp{##1}%
3536       \c@pgf@counta=\forestrve{#3}\relax
3537       \multiply\c@pgf@counta\forest@temp\relax
3538       \forestreset{#3}{\the\c@pgf@counta}%
3539     },
3540     #1:/.code={%
3541       \forestmathtruncatemacro\forest@temp{##1}%
3542       \c@pgf@counta=\forestrve{#3}\relax
3543       \divide\c@pgf@counta\forest@temp\relax
3544       \forestreset{#3}{\the\c@pgf@counta}%
3545     },
3546     #1'/.code={%
3547       \c@pgf@counta=##1\relax
3548       \forestreset{#3}{\the\c@pgf@counta}%
3549     },
3550     #1'+/.code={%
3551       \c@pgf@counta=\forestrve{#3}\relax
3552       \advance\c@pgf@counta##1\relax
3553       \forestreset{#3}{\the\c@pgf@counta}%
3554     },
3555     #1'-/.code={%
3556       \c@pgf@counta=\forestrve{#3}\relax
3557       \advance\c@pgf@counta-##1\relax
3558       \forestreset{#3}{\the\c@pgf@counta}%
3559     },
3560     #1'*/.style={%
3561       \c@pgf@counta=\forestrve{#3}\relax
3562       \multiply\c@pgf@counta##1\relax
3563       \forestreset{#3}{\the\c@pgf@counta}%
3564     },
3565     #1':/.style={%
3566       \c@pgf@counta=\forestrve{#3}\relax
3567       \divide\c@pgf@counta##1\relax
3568       \forestreset{#3}{\the\c@pgf@counta}%
3569     },
3570   }%

```

```

3571 \pgfkeyssetvalue{#1/option@name}{#3}%
3572 \pgfkeyssetvalue{#1+/option@name}{#3}%
3573 \pgfkeyssetvalue{#1-/option@name}{#3}%
3574 \pgfkeyssetvalue{#1*/option@name}{#3}%
3575 \pgfkeyssetvalue{#1:/option@name}{#3}%
3576 \pgfkeyssetvalue{#1'/option@name}{#3}%
3577 \pgfkeyssetvalue{#1'+/option@name}{#3}%
3578 \pgfkeyssetvalue{#1'-/option@name}{#3}%
3579 \pgfkeyssetvalue{#1'*/option@name}{#3}%
3580 \pgfkeyssetvalue{#1':/option@name}{#3}%
3581 }
3582 \def\forest@declarebooleanregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
3583 \pgfkeysalso{%
3584 #1/.code={%
3585 \ifcsdef{forest@bh@\detokenize{##1}}{%
3586 \letcs\forest@temp{forest@bh@\detokenize{##1}}%
3587 }{%
3588 \forestmathtruncatemacro\forest@temp{##1}%
3589 \ifx\forest@temp0\else\def\forest@temp{1}\fi
3590 }%
3591 \forestrllet{#3}\forest@temp
3592 },
3593 #1/.default=1,
3594 #2/not #3/.code={\forestset{#3}{0}},
3595 #2/if #3/.code 2 args={%
3596 \forestget{#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'={Opt},
3644   declare dimen register=tempdimb,tempdimb'={Opt},
3645   declare dimen register=tempdimc,tempdimc'={Opt},
3646   declare dimen register=tempdimd,tempdimd'={Opt},
3647   declare dimen register=tempdimx,tempdimx'={Opt},
3648   declare dimen register=tempdimxa,tempdimxa'={Opt},
3649   declare dimen register=tempdimxb,tempdimxb'={Opt},
3650   declare dimen register=tempdimy,tempdimy'={Opt},
3651   declare dimen register=tempdimya,tempdimya'={Opt},
3652   declare dimen register=tempdimyb,tempdimyb'={Opt},
3653   declare dimen register=tempdiml,tempdiml'={Opt},
3654   declare dimen register=tempdimla,tempdimla'={Opt},
3655   declare dimen register=tempdimlb,tempdimlb'={Opt},
3656   declare dimen register=tempdims,tempdims'={Opt},
3657   declare dimen register=tempdimsa,tempdimsa'={Opt},
3658   declare dimen register=tempdimsb,tempdimsb'={Opt},
3659   declare count register=tempcounta,tempcounta'={0},
3660   declare count register=tempcountb,tempcountb'={0},
3661   declare count register=tempcountc,tempcountc'={0},
3662   declare count register=tempcountd,tempcountd'={0},
3663   declare boolean register=tempboola,tempboola={0},
3664   declare boolean register=tempboolb,tempboolb={0},
3665   declare boolean register=tempboolc,tempboolc={0},
3666   declare boolean register=tempboold,tempboold={0},
3667 }

```

7.1.2 Declaring options

```

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

```

```

3688 \def\forest@temp@silent{n}%
3689 \def\forest@temp@propagating{n}%
3690 \forest@node@setnameoralias{#1}%
3691 }
3692 \def\forest@node@setalias@silent#1{%
3693 \def\forest@temp@setname{n}%
3694 \def\forest@temp@silent{y}%
3695 \def\forest@temp@propagating{n}%
3696 \forest@node@setnameoralias{#1}%
3697 }
3698 \def\forest@node@setnameoralias#1{%
3699 \ifstrempy{#1}{%
3700 \forest@node@setnameoralias{node@\forest@cn}%
3701 }{%
3702 \forest@node@ifnamedefined{#1}{%
3703 \if y\forest@temp@propagating
3704 % this will find a unique name, eventually:
3705 \@escapeif{\forest@node@setnameoralias{#1@\forest@cn}}%
3706 \else\@escapeif{%
3707 \if y\forest@temp@setname
3708 \edef\forest@marshal{%
3709 \ifstrequal{\forest@ve{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@for node{\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@ve{name}}{}%
3743 \forest@set{name}{#1}%
3744 \fi
3745 \csedef{forest@id@of@#1}{\forest@cn}%
3746 }
3747 \forestset{
3748 TeX/.code={#1},

```

```

3749 TeX'/.code={\appto\forest@externalize@loadimages{#1}#1},
3750 TeX'/.code={\appto\forest@externalize@loadimages{#1}},
3751 options/.code={\forestset{#1}},
3752 typeout/.style={TeX={\typeout{#1}}},
3753 declare toks={name}{},
3754 name/.code={% override the default setter
3755   \forest@for@node{\forest@setter@node}{\forest@node@setname{#1}}%
3756 },
3757 name/.default={},
3758 name'/.code={% override the default setter
3759   \forest@for@node{\forest@setter@node}{\forest@node@setname@silent{#1}}%
3760 },
3761 name'/.default={},
3762 alias/.code={\forest@for@node{\forest@setter@node}{\forest@node@setalias{#1}}},
3763 alias'/.code={\forest@for@node{\forest@setter@node}{\forest@node@setalias@silent{#1}}},
3764 begin draw/.code={\begin{tikzpicture}},
3765 end draw/.code={\end{tikzpicture}},
3766 declare keylist register=default preamble,
3767 default preamble'={},
3768 declare keylist register=preamble,
3769 preamble'={},
3770 declare autowrapped toks={content}{},
3771 % #1 = which option to split, #2 = separator (one char!), #3 = receiving options
3772 split option/.code n args=3{%
3773   \forestRNOget{#1}\forest@temp
3774   \edef\forest@marshal{%
3775     \noexpand\pgfkeysalso{split={\expandonce{\forest@temp}}\unexpanded{{#2}{#3}}}%
3776   }\forest@marshal
3777 },
3778 split register/.code n args=3{% #1 = which register to split, #2 = separator (one char!), #3 = receiving options
3779   \forestRget{#1}\forest@temp
3780   \edef\forest@marshal{%
3781     \noexpand\pgfkeysalso{split={\expandonce{\forest@temp}}\unexpanded{{#2}{#3}}}%
3782   }\forest@marshal
3783 },
3784 TeX={%
3785   \def\forest@split@sourcevalues{%
3786     \def\forest@split@sourcevalue{%
3787       \def\forest@split@receivingoptions{%
3788         \def\forest@split@receivingoption{%
3789         },
3790 split/.code n args=3{% #1 = string to split, #2 = separator (one char!), #3 = receiving options
3791   \forest@saveandrestoremacro\forest@split@sourcevalues{%
3792     \forest@saveandrestoremacro\forest@split@sourcevalue{%
3793       \forest@saveandrestoremacro\forest@split@receivingoptions{%
3794         \forest@saveandrestoremacro\forest@split@receivingoption{%
3795           \def\forest@split@sourcevalues{#1#2}%
3796           \edef\forest@split@receivingoptions{#3,%
3797           \def\forest@split@receivingoption{%
3798             \safeloop
3799               \expandafter\forest@split\expandafter{\forest@split@sourcevalues}{#2}\forest@split@sourcevalue\
3800             \ifdefempty\forest@split@receivingoptions{}%
3801               \expandafter\forest@split\expandafter{\forest@split@receivingoptions}{,}\forest@temp\forest@split@receivingoption\
3802             \ifdefempty\forest@temp{ }\let\forest@split@receivingoption\forest@temp\def\forest@temp{ }\
3803             }%
3804           \edef\forest@marshal{%
3805             \noexpand\pgfkeysalso{\forest@split@receivingoption={\expandonce{\forest@split@sourcevalue}}}%
3806           }\forest@marshal
3807           \ifdefempty\forest@split@sourcevalues{\forest@tempfalse}{\forest@temprtrue}%
3808           \ifforest@temp
3809             \saferepeat

```

```

3810     }}}}
3811 },
3812 declare count={grow}{270},
3813 TeX={% a hack for grow-reversed connection, and compass-based grow specification
3814     \forest@copycommandkey{/forest/grow}{/forest/grow@}%
3815     %\pgfkeysgetvalue{/forest/grow/.@cmd}\forest@temp
3816     %\pgfkeyslet{/forest/grow@/.@cmd}\forest@temp
3817 },
3818 grow/.style={grow@={#1},reversed=0},
3819 grow'/.style={grow@={#1},reversed=1},
3820 grow''/.style={grow@={#1}},
3821 grow@/.is choice,
3822 grow@/east/.style={/forest/grow@@=0},
3823 grow@/north east/.style={/forest/grow@@=45},
3824 grow@/north/.style={/forest/grow@@=90},
3825 grow@/north west/.style={/forest/grow@@=135},
3826 grow@/west/.style={/forest/grow@@=180},
3827 grow@/south west/.style={/forest/grow@@=225},
3828 grow@/south/.style={/forest/grow@@=270},
3829 grow@/south east/.style={/forest/grow@@=315},
3830 grow@/.unknown/.code={\let\forest@temp@grow\pgfkeyscurrentname
3831     \pgfkeysalso{/forest/grow@@/.expand once=\forest@temp@grow}},
3832 declare boolean={reversed}{0},
3833 declare toks={parent anchor}{},
3834 declare toks={child anchor}{},
3835 declare toks={anchor}{base},
3836 Autoforward={anchor}{
3837     node options-=anchor,
3838     node options+= {anchor={#1}}
3839 },
3840 anchor'/.style={anchor@no@compass=true,anchor=#1},
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 % /tikz/forest anchor/.style={
3850 %     /forest/TeX={\forestanchortotikzanchor{#1}\forest@temp@anchor},
3851 %     anchor/.expand once=\forest@temp@anchor
3852 % },
3853 declare toks={calign}{midpoint},
3854 TeX={%
3855     \forest@copycommandkey{/forest/calign}{/forest/calign'}%
3856 },
3857 calign/.is choice,
3858 calign/child/.style={calign'=child},
3859 calign/first/.style={calign'=child,calign primary child=1},
3860 calign/last/.style={calign'=child,calign primary child=-1},
3861 calign with current/.style={for parent/.wrap pgfmath arg={calign=child,calign primary child=##1}{n}},
3862 calign with current edge/.style={for parent/.wrap pgfmath arg={calign=child edge,calign primary child=##1}{n}},
3863 calign/child edge/.style={calign'=child edge},
3864 calign/midpoint/.style={calign'=midpoint},
3865 calign/center/.style={calign'=midpoint,calign primary child=1,calign secondary child=-1},
3866 calign/edge midpoint/.style={calign'=edge midpoint},
3867 calign/fixed angles/.style={calign'=fixed angles},
3868 calign/fixed edge angles/.style={calign'=fixed edge angles},
3869 calign/.unknown/.code={\PackageError{forest}{unknown calign '\pgfkeyscurrentname'}{}},
3870 declare count={calign primary child}{1},

```

```

3871 declare count={calign secondary child}{-1},
3872 declare count={calign primary angle}{-35},
3873 declare count={calign secondary angle}{35},
3874 calign child/.style={calign primary child={#1}},
3875 calign angle/.style={calign primary angle={-#1},calign secondary angle={#1}},
3876 declare toks={tier}{},
3877 declare toks={fit}{tight},
3878 declare boolean={ignore}{0},
3879 declare boolean={ignore edge}{0},
3880 no edge/.style={edge'={},ignore edge},
3881 declare keylist={edge}{draw},
3882 declare toks={edge path}{%
3883   \noexpand\path[\forestoption{edge}]{%
3884     (\forestOve{\forestove{@parent}}{name}.parent anchor)--(\forestove{name}.child anchor)
3885     % =
3886     % (!u.parent anchor)--(.child anchor)\forestoption{edge label};
3887     \forestoption{edge label}};%
3888 },
3889 edge path'/.style={
3890   edge path={%
3891     \noexpand\path[\forestoption{edge}]{%
3892       #1%
3893       \forestoption{edge label};
3894     }
3895   },
3896   declare toks={edge label}{},
3897   declare boolean={phantom}{0},
3898   baseline/.style={alias={forest@baseline@node}},
3899   declare readonly count={id}{0},
3900   declare readonly count={n}{0},
3901   declare readonly count={n'}{0},
3902   declare readonly count={n children}{-1},
3903   declare readonly count={level}{-1},
3904   declare dimen=x{0pt},
3905   declare dimen=y{0pt},
3906   declare dimen={s}{0pt},
3907   declare dimen={l}{6ex}, % just in case: should be set by the calibration
3908   declare dimen={s sep}{0.6666em},
3909   declare dimen={l sep}{1ex}, % just in case: calibration!
3910   declare keylist={node options}{anchor=base},
3911   declare toks={tikz}{},
3912   afterthought/.style={tikz+=#{#1}},
3913   label/.style={tikz+={\path[late options={%
3914     name=\forestoption{name},label={#1}}];}},
3915   pin/.style={tikz+={\path[late options={%
3916     name=\forestoption{name},pin={#1}}];}},
3917   declare toks={content format}{\forestoption{content}},
3918   plain content/.style={content format={\forestoption{content}}},
3919   math content/.style={content format={\noexpand\ensuremath{\forestoption{content}}}},
3920   declare toks={node format}{%
3921     \noexpand\node
3922     (\forestoption{name})%
3923     [\forestoption{node options}]%
3924     {\forestoption{content format}};%
3925   },
3926   node format'/.style={
3927     node format={\noexpand\node(\forestoption{name})#1;}
3928   },
3929   tabular@environment/.style={content format={%
3930     \noexpand\begin{tabular}[\forestoption{base}]{\forestoption{align}}%
3931     \forestoption{content}%

```



```

3932 \noexpand\end{tabular}%
3933 }},
3934 declare toks={align}{},
3935 TeX={%
3936 \forest@copycommandkey{/forest/align}{/forest/align'}%
3937 %\pgfkeysgetvalue{/forest/align/.@cmd}\forest@temp
3938 %\pgfkeyslet{/forest/align'/.@cmd}\forest@temp
3939 },
3940 align/.is choice,
3941 align/.unknown/.code={%
3942 \edef\forest@marshal{%
3943 \noexpand\pgfkeysalso{%
3944 align'={\pgfkeyscurrentname},%
3945 tabular@environment
3946 }%
3947 }\forest@marshal
3948 },
3949 align/center/.style={align'={@{}c@{}}},tabular@environment},
3950 align/left/.style={align'={@{}l@{}}},tabular@environment},
3951 align/right/.style={align'={@{}r@{}}},tabular@environment},
3952 declare toks={base}{t},
3953 TeX={%
3954 \forest@copycommandkey{/forest/base}{/forest/base'}%
3955 %\pgfkeysgetvalue{/forest/base/.@cmd}\forest@temp
3956 %\pgfkeyslet{/forest/base'/.@cmd}\forest@temp
3957 },
3958 base/.is choice,
3959 base/top/.style={base'=t},
3960 base/bottom/.style={base'=b},
3961 base/.unknown/.style={base'/.expand once=\pgfkeyscurrentname},
3962 unknown to/.store in=\forest@unknownto,
3963 unknown to=node options,
3964 unknown key error/.code={\PackageError{forest}{Unknown keyval: \detokenize{#1}}{}}},
3965 content to/.store in=\forest@contentto,
3966 content to=content,
3967 .unknown/.code={%
3968 \expandafter\pgfutil@in@\expandafter.\expandafter{\pgfkeyscurrentname}%
3969 \ifpgfutil@in@
3970 \expandafter\forest@relatednode@option@setter\pgfkeyscurrentname=#1\forest@END
3971 \else
3972 \edef\forest@marshal{%
3973 \noexpand\pgfkeysalso{\forest@unknownto={\pgfkeyscurrentname=\unexpanded{#1}}}%
3974 }\forest@marshal
3975 \fi
3976 },
3977 get node boundary/.code={%
3978 \foresttoget{@boundary}\forest@node@boundary
3979 \def#1{%
3980 \forest@extendpath#1\forest@node@boundary{\pgfqpoint{\forestove{x}}{\forestove{y}}}%
3981 },
3982 % get min l tree boundary/.code={%
3983 % \forest@get@tree@boundary{negative}{\the\numexpr\forestove{grow}-90\relax}#1},
3984 % get max l tree boundary/.code={%
3985 % \forest@get@tree@boundary{positive}{\the\numexpr\forestove{grow}-90\relax}#1},
3986 get min s tree boundary/.code={%
3987 \forest@get@tree@boundary{negative}{\forestove{grow}}#1},
3988 get max s tree boundary/.code={%
3989 \forest@get@tree@boundary{positive}{\forestove{grow}}#1},
3990 use as bounding box/.style={%
3991 before drawing tree={
3992 tikz+/.expanded={%

```

```

3993     \noexpand\pgfresetboundingbox
3994     \noexpand\useasboundingbox
3995     ($(.anchor)+(\forestoption{min x},\forestoption{min y}))$)
3996     rectangle
3997     ($(.anchor)+(\forestoption{max x},\forestoption{max y}))$)
3998     ;
3999   }
4000 }
4001 },
4002 use as bounding box'/.style={%
4003   before drawing tree={
4004     tikz+/.expanded={%
4005       \noexpand\pgfresetboundingbox
4006       \noexpand\useasboundingbox
4007       ($(.anchor)+(\forestoption{min x}+\pgfkeysvalueof{/pgf/outer xsep}/2+\pgfkeysvalueof{/pgf/inner xsep}
4008       rectangle
4009       ($(.anchor)+(\forestoption{max x}-\pgfkeysvalueof{/pgf/outer xsep}/2-\pgfkeysvalueof{/pgf/inner xsep}
4010       ;
4011     }
4012   }
4013 },
4014 }%
4015 \def\forest@iftikzkey#1#2#3{% #1 = key name, #2 = true code, #3 = false code
4016   \forest@temptrue
4017   \pgfkeysifdefined{/tikz/\pgfkeyscurrentname}{}%
4018   \pgfkeysifdefined{/tikz/\pgfkeyscurrentname/.@cmd}{}%
4019   \pgfkeysifdefined{/pgf/\pgfkeyscurrentname}{}%
4020   \pgfkeysifdefined{/pgf/\pgfkeyscurrentname/.@cmd}{}%
4021   \forest@tempfalse
4022   }%%
4023   \ifforest@temp\@escapeif{#2}\else\@escapeif{#3}\fi
4024 }
4025 \def\forest@ifoptionortikzkey#1#2#3{% #1 = key name, #2 = true code, #3 = false code
4026   \forest@temptrue
4027   \pgfkeysifdefined{/forest/\pgfkeyscurrentname}{}%
4028   \pgfkeysifdefined{/forest/\pgfkeyscurrentname/.@cmd}{}%
4029   \forest@iftikzkey{#1}{}%
4030   }%%
4031   \ifforest@temp\@escapeif{#2}\else\@escapeif{#3}\fi
4032 }
4033 \def\forest@get@tree@boundary#1#2#3{% #1=pos/neg,#2=grow,#3=receiving cs
4034   \def#3{}%
4035   \forest@node@getedge{#1}{#2}\forest@temp@boundary
4036   \forest@extendpath#3\forest@temp@boundary{\pgfpoint{\forestove{x}}{\forestove{y}}}%
4037 }
4038 \def\forest@setter@node{\forest@cn}%
4039 \def\forest@relatednode@option@compat@ignoreinvalidsteps#1{#1}
4040 \def\forest@relatednode@option@setter#1.#2=#3\forest@END{%
4041   \forest@forthis{%
4042     \forest@relatednode@option@compat@ignoreinvalidsteps{%
4043       \forest@nameandgo{#1}%
4044       \let\forest@setter@node\forest@cn
4045     }%
4046   }%
4047   \ifnum\forest@setter@node=0
4048   \else
4049     \forestset{#2={#3}}%
4050   \fi
4051   \def\forest@setter@node{\forest@cn}%
4052 }%
4053 \def\forest@split#1#2#3#4{% #1=list (assuming that the list is nonempty and finishes with the separator), #2

```

```

4054 \def\forest@split@@##1#2##2\forest@split@@##3##4{\def##3{##1}\def##4{##2}}%
4055 \forest@split@@#1\forest@split@@{#3}{#4}}

```

7.1.3 Option propagation

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

```

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

```

```

4111     \forest@untilkey{\forestmath@if{#1}{\forestloopbreak}}{#2}%
4112 }%
4113 },
4114 while/.code 2 args={%
4115     \ifstrempy{#1}{%
4116         \forest@untilkey{\ifnum\forest@cn=0\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
4117     }{%
4118         \forest@untilkey{\forestmath@if{#1}{\forestloopbreak}}{#2}%
4119     }%
4120 },
4121 do until/.code 2 args={%
4122     \ifstrempy{#1}{%
4123         \forest@dountilkey{\ifnum\forest@cn=0\else\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
4124     }{%
4125         \forest@dountilkey{\forestmath@if{#1}{\forestloopbreak}}{#2}%
4126     }%
4127 },
4128 do while/.code 2 args={%
4129     \ifstrempy{#1}{%
4130         \forest@dountilkey{\ifnum\forest@cn=0\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
4131     }{%
4132         \forest@dountilkey{\forestmath@if{#1}{\forestloopbreak}}{#2}%
4133     }%
4134 },
4135 until nodewalk valid/.code 2 args={%
4136     \forest@untilkey{\forest@forthis{%
4137         \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\else\forestloopbreak\fi}}{}}{#2}
4138     },
4139 while nodewalk valid/.code 2 args={%
4140     \forest@untilkey{\forest@forthis{%
4141         \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\forestloopbreak\fi}}{}}{#2}%
4142     },
4143 do until nodewalk valid/.code 2 args={%
4144     \forest@dountilkey{\forest@forthis{%
4145         \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\else\forestloopbreak\fi}}{}}{#2}
4146     },
4147 do while nodewalk valid/.code 2 args={%
4148     \forest@dountilkey{\forest@forthis{%
4149         \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\forestloopbreak\fi}}{}}{#2}%
4150     },
4151 until nodewalk empty/.code 2 args={%
4152     \forest@untilkey{\forest@forthis{%
4153         \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\forestloopbreak\fi}}{}}
4154     },
4155 while nodewalk empty/.code 2 args={%
4156     \forest@untilkey{\forest@forthis{%
4157         \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\else\forestloopbreak\fi}}
4158     },
4159 do until nodewalk empty/.code 2 args={%
4160     \forest@dountilkey{\forest@forthis{%
4161         \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\forestloopbreak\fi}}{}}
4162     },
4163 do while nodewalk empty/.code 2 args={%
4164     \forest@dountilkey{\forest@forthis{%
4165         \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\else\forestloopbreak\fi}}
4166     },
4167 break/.code={\forestloopBreak{#1}},
4168 break/.default=0,
4169 }
4170 \def\forest@repeatkey#1#2{%
4171     \safeRKloop

```

```

4172 \ifnum\safeRKloopn>#1\relax
4173 \csuse{safeRKbreak@\the\safeRKloop@depth true}%
4174 \fi
4175 \expandafter\unless\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
4176 \pgfkeysalso{#2}%
4177 \safeRKrepeat
4178 }
4179 \def\forest@untilkey#1#2{% #1 = condition, #2 = keys
4180 \safeRKloop
4181 #1%
4182 \expandafter\unless\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
4183 \pgfkeysalso{#2}%
4184 \safeRKrepeat
4185 }
4186 \def\forest@dountilkey#1#2{% #1 = condition, #2 = keys
4187 \safeRKloop
4188 \pgfkeysalso{#2}%
4189 #1%
4190 \expandafter\unless\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
4191 \safeRKrepeat
4192 }
4193 \def\forestloopbreak{%
4194 \csname safeRKbreak@\the\safeRKloop@depth true\endcsname
4195 }
4196 \def\forestloopBreak#1{%
4197 \csname safeRKbreak@\number\numexpr\the\safeRKloop@depth-#1\relax true\endcsname
4198 }
4199 \def\forestloopcount{%
4200 \csname safeRKloopn@\number\numexpr\the\safeRKloop@depth\endcsname
4201 }
4202 \def\forestloopCount#1{%
4203 \csname safeRKloopn@\number\numexpr\the\safeRKloop@depth-#1\endcsname
4204 }
4205 \pgfmathdeclarefunction{forestloopcount}{1}{%
4206 \edef\pgfmathresult{\forestloopCount{\ifstrempy{#1}{0}{#1}}}%
4207 }
4208 \forest@copycommandkey{/forest/repeat}{/forest/nodewalk/repeat}
4209 \forest@copycommandkey{/forest/while}{/forest/nodewalk/while}
4210 \forest@copycommandkey{/forest/do while}{/forest/nodewalk/do while}
4211 \forest@copycommandkey{/forest/until}{/forest/nodewalk/until}
4212 \forest@copycommandkey{/forest/do until}{/forest/nodewalk/do until}
4213 \forest@copycommandkey{/forest/if}{/forest/nodewalk/if}
4214 \forest@copycommandkey{/forest/if nodewalk valid}{/forest/nodewalk/if nodewalk valid}
4215 %

```

7.2 Aggregate functions

```

4216 \forestset{
4217 aggregate postparse/.is choice,
4218 aggregate postparse/int/.code={%
4219 \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@toint},
4220 aggregate postparse/none/.code={%
4221 \let\forest@aggregate@pgfmathpostparse\relax},
4222 aggregate postparse/print/.code={%
4223 \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@print},
4224 aggregate postparse/macro/.code={%
4225 \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@usemacro},
4226 aggregate postparse macro/.store in=\forest@aggregate@pgfmathpostparse@macro,
4227 }
4228 \def\forest@aggregate@pgfmathpostparse@print{%

```

```

4229 \pgfmathprintnumberto{\pgfmathresult}{\pgfmathresult}%
4230 }
4231 \def\forest@aggregate@pgfmathpostparse@toint{%
4232 \expandafter\forest@split\expandafter{\pgfmathresult.}{.}\pgfmathresult\forest@temp
4233 }
4234 \def\forest@aggregate@pgfmathpostparse@usemacro{%
4235 \forest@aggregate@pgfmathpostparse@macro
4236 }
4237 \let\forest@aggregate@pgfmathpostparse\relax
4238 \forestset{
4239 /handlers/.aggregate/.code n args=4{%
4240 % #1 = start value (forestmath)
4241 % #2 = forestmath expression that calculates "aggregate result" at each step
4242 % #3 = forestmath expression that calculates "aggregate result" at the end of the nodewalk
4243 % #4 = nodewalk
4244 \forest@aggregate@handler{\forest@aggregate@generic{#1}{#2}{#3}{#4}}%
4245 },
4246 /handlers/.sum/.code 2 args={% #1=forestmath, #2=nodewalk
4247 \forest@aggregate@handler{\forest@aggregate@sum{#1}{#2}}%
4248 },
4249 /handlers/.count/.code={% #1=nodewalk
4250 \forest@aggregate@handler{\forest@aggregate@count{#1}}%
4251 },
4252 /handlers/.average/.code 2 args={% #1=forestmath, #2=nodewalk
4253 \forest@aggregate@handler{\forest@aggregate@average{#1}{#2}}%
4254 },
4255 /handlers/.product/.code 2 args={% #1=forestmath, #2=nodewalk
4256 \forest@aggregate@handler{\forest@aggregate@product{#1}{#2}}%
4257 },
4258 /handlers/.min/.code 2 args={% #1=forestmath, #2=nodewalk
4259 \forest@aggregate@handler{\forest@aggregate@min{#1}{#2}}%
4260 },
4261 /handlers/.max/.code 2 args={% #1=forestmath, #2=nodewalk
4262 \forest@aggregate@handler{\forest@aggregate@max{#1}{#2}}%
4263 },
4264 declare count register={aggregate n},
4265 declare toks register={aggregate value},
4266 declare toks register={aggregate result},
4267 aggregate result={},
4268 }
4269 \def\forest@aggregate@handler#1{%
4270 \edef\forest@marshal{%
4271 \unexpanded{%
4272 #1%
4273 }{%
4274 \noexpand\pgfkeysalso{\pgfkeyscurrentpath/.register=aggregate result}}%
4275 }%
4276 }\forest@marshal
4277 }
4278 \def\forest@aggregate@pgfmathfunction@finish{%
4279 \forestget{aggregate result}\pgfmathresult
4280 }
4281 \pgfmathdeclarefunction{aggregate}{4}{%
4282 \forest@aggregate@generic{#1}{#2}{#3}{#4}%
4283 \forest@aggregate@pgfmathfunction@finish
4284 }
4285 \pgfmathdeclarefunction{aggregate_count}{1}{%
4286 \forest@aggregate@sum{#1}%
4287 \forest@aggregate@pgfmathfunction@finish
4288 }
4289 \pgfmathdeclarefunction{aggregate_sum}{2}{%

```

```

4290 \forest@aggregate@sum{#1}{#2}%
4291 \forest@aggregate@pgfmathfunction@finish
4292 }
4293 \pgfmathdeclarefunction{aggregate_product}{2}{%
4294 \forest@aggregate@product{#1}{#2}%
4295 \forest@aggregate@pgfmathfunction@finish
4296 }
4297 \pgfmathdeclarefunction{aggregate_average}{2}{%
4298 \forest@aggregate@average{#1}{#2}%
4299 \forest@aggregate@pgfmathfunction@finish
4300 }
4301 \pgfmathdeclarefunction{aggregate_min}{2}{%
4302 \forest@aggregate@min{#1}{#2}%
4303 \forest@aggregate@pgfmathfunction@finish
4304 }
4305 \pgfmathdeclarefunction{aggregate_max}{2}{%
4306 \forest@aggregate@max{#1}{#2}%
4307 \forest@aggregate@pgfmathfunction@finish
4308 }

```

Define particular aggregate functions.

```

4309 \def\forest@aggregate#1#2#3#4#5#6{% #1...#5=real args,
4310                                     % #6=what to do with |aggregate result| register
4311     % #1 = start value (forestmath)
4312     % #2 = forestmath expression that calculates "aggregate current" at each step
4313     % #3 = forestmath expression that calculates "aggregate result" at each step
4314     % #4 = forestmath expression that calculates "aggregate result" at the end of the nodewalk
4315     % #5 = nodewalk
4316     \forest@saveandrestorereregister{aggregate result}{%
4317     \forest@saveandrestorereregister{aggregate n}{%
4318     \forest@aggregate@{#1}{#2}{#3}{#4}{#5}%
4319     #6%
4320     }%
4321     }%
4322 }
4323 \def\forest@aggregate@generic#1#2#3#4{\forest@aggregate
4324 {\forestmathparse{#1}}%
4325 {}%
4326 {\forestmathparse{#2}}%
4327 {\forestmathparse{#3}}%
4328 {#4}%
4329 }
4330 \def\forest@aggregate@sum#1#2{\forest@aggregate
4331 {\forestmath@convert@fromto\forestmathtype@count\forestmathtype@generic{0}}%
4332 {\forestmathparse{#1}}%
4333 {\forestmathadd{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
4334 {\forestregister{aggregate result}\forestmathresult}%
4335 {#2}%
4336 }
4337 \def\forest@aggregate@count#1{\forest@aggregate
4338 {\def\forestmathresult{1}}%
4339 {\def\forestmathresult{1}}%
4340 {\edef\forestmathresult{\the\numexpr\forestregister{aggregate result}+1}}%
4341 {\forestregister{aggregate result}\forestmathresult}%
4342 {#1}%
4343 }
4344 \def\forest@aggregate@average#1#2{\forest@aggregate
4345 {\forestmath@convert@fromto\forestmathtype@count\forestmathtype@generic{0}}%
4346 {\forestmathparse{#1}}%
4347 {\forestmathadd{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
4348 {\forestmathdivideP{\forestregister{aggregate result}}{\forestregister{aggregate n}}}%

```

```

4349   {#2}%
4350 }
4351 \def\forest@aggregate@product#1#2{\forest@aggregate
4352   {\forestmath@convert@fromto\forestmath@type@count\forestmath@type@generic{1}}%
4353   {\forestmath@parse{#1}}%
4354   {\forestmath@multiply{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
4355   {\forestregister{aggregate result}\forestmathresult}%
4356   {#2}%
4357 }
4358 \def\forest@aggregate@min#1#2{\forest@aggregate
4359   {\def\forestmathresult{}}%
4360   {\forestmath@parse{#1}}%
4361   {\forestmath@min{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
4362   {\forestregister{aggregate result}\forestmathresult}%
4363   {#2}%
4364 }
4365 \def\forest@aggregate@max#1#2{\forest@aggregate
4366   {\def\forestmathresult{}}%
4367   {\forestmath@parse{#1}}%
4368   {\forestmath@max{\forestregister{aggregate value}}{\forestregister{aggregate result}}}%
4369   {\forestregister{aggregate result}\forestmathresult}%
4370   {#2}%
4371 }

```

Actual computation.

```

4372 \def\forest@aggregate@#1#2#3#4#5{%
4373   % #1 = start value (forestmath)
4374   % #2 = forestmath expression that calculates "aggregate current" at each step
4375   % #3 = forestmath expression that calculates "aggregate result" at each step
4376   % #4 = forestmath expression that calculates "aggregate result" at the end of the nodewalk
4377   % #5 = nodewalk
4378   #1%
4379   \forestregister{aggregate result}\forestmathresult
4380   \forestregister{aggregate value}{}%
4381   \forestregister{aggregate n}{0}%
4382   \forest@forthis{%
4383     \forest@nodewalk{#5}{%
4384       TeX={%
4385         \forestreset{aggregate n}{\number\numexpr\forestregister{aggregate n}+1}%
4386         #2%
4387         \forestregister{aggregate value}\forestmathresult
4388         #3%
4389         \forestregister{aggregate result}\forestmathresult
4390       }%
4391     }%
4392   }%
4393   #4%
4394   \let\forest@temp@pgfmathpostparse\pgfmathpostparse
4395   \let\pgfmathpostparse\forest@aggregate@pgfmathpostparse
4396   \forestmath@convert@to\forestmath@type@dimen{\forestmathresult}
4397   \pgfmathqparse{\forestmathresult}%
4398   \let\pgfmathpostparse\forest@temp@pgfmathpostparse
4399   \forestregister{aggregate result}\pgfmathresult
4400 }

```

7.2.1 pgfmath extensions

```

4401 \pgfmathdeclarefunction{strequal}{2}{%
4402   \ifstrequal{#1}{#2}{\def\pgfmathresult{1}}{\def\pgfmathresult{0}}%
4403 }
4404 \pgfmathdeclarefunction{instr}{2}{%
4405   \pgfutil@in@{#1}{#2}%

```



```

4406 \ifpgfutil@in@def\pgfmathresult{1}\else\def\pgfmathresult{0}\fi
4407 }
4408 \pgfmathdeclarefunction{strcat}{...}{%
4409 \edef\pgfmathresult{\forest@strip@braces{#1}}%
4410 }
4411 \pgfmathdeclarefunction{min_s}{2}{% #1 = node, #2 = context node (for growth rotation)
4412 \forest@forthis{%
4413 \forest@nameandgo{#1}%
4414 \forest@compute@minmax@ls{#2}%
4415 \edef\forest@temp{\forestove{min@s}}%
4416 \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
4417 }%
4418 }
4419 \pgfmathdeclarefunction{min_l}{2}{% #1 = node, #2 = context node (for growth rotation)
4420 \forest@forthis{%
4421 \forest@nameandgo{#1}%
4422 \forest@compute@minmax@ls{#2}%
4423 \edef\forest@temp{\forestove{min@l}}%
4424 \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
4425 }%
4426 }
4427 \pgfmathdeclarefunction{max_s}{2}{% #1 = node, #2 = context node (for growth rotation)
4428 \forest@forthis{%
4429 \forest@nameandgo{#1}%
4430 \forest@compute@minmax@ls{#2}%
4431 \edef\forest@temp{\forestove{max@s}}%
4432 \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
4433 }%
4434 }
4435 \pgfmathdeclarefunction{max_l}{2}{% #1 = node, #2 = context node (for growth rotation)
4436 \forest@forthis{%
4437 \forest@nameandgo{#1}%
4438 \forest@compute@minmax@ls{#2}%
4439 \edef\forest@temp{\forestove{max@l}}%
4440 \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
4441 }%
4442 }
4443 \def\forest@compute@minmax@ls#1{% #1 = nodewalk; in the context of which node?
4444 {%
4445 \pgftransformreset
4446 \forest@forthis{%
4447 \forest@nameandgo{#1}%
4448 \forest@pgfqtransformrotate{-\forestove{grow}}%
4449 }%
4450 \forestoget{min x}\forest@temp@minx
4451 \forestoget{min y}\forest@temp@miny
4452 \forestoget{max x}\forest@temp@maxx
4453 \forestoget{max y}\forest@temp@maxy
4454 \pgfpointrtransformed{\pgfqpoint{\forest@temp@minx}{\forest@temp@miny}}%
4455 \forestoeset{min@l}{\the\pgf@x}%
4456 \forestoeset{min@s}{\the\pgf@y}%
4457 \forestoeset{max@l}{\the\pgf@x}%
4458 \forestoeset{max@s}{\the\pgf@y}%
4459 \pgfpointrtransformed{\pgfqpoint{\forest@temp@minx}{\forest@temp@maxy}}%
4460 \ifdim\pgf@x<\forestove{min@l}\relax\forestoeset{min@l}{\the\pgf@x}\fi
4461 \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
4462 \ifdim\pgf@x>\forestove{max@l}\relax\forestoeset{max@l}{\the\pgf@x}\fi
4463 \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
4464 \pgfpointrtransformed{\pgfqpoint{\forest@temp@maxx}{\forest@temp@miny}}%
4465 \ifdim\pgf@x<\forestove{min@l}\relax\forestoeset{min@l}{\the\pgf@x}\fi
4466 \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi

```

```

4467 \ifdim\pgf@x>\forestove{max@1}\relax\forestoeset{max@1}{\the\pgf@x}\fi
4468 \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
4469 \pgfpointrtransformed{\pgfqpoint{\forest@temp@maxx}{\forest@temp@maxy}}%
4470 \ifdim\pgf@x<\forestove{min@1}\relax\forestoeset{min@1}{\the\pgf@x}\fi
4471 \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
4472 \ifdim\pgf@x>\forestove{max@1}\relax\forestoeset{max@1}{\the\pgf@x}\fi
4473 \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
4474 % smuggle out
4475 \edef\forest@marshal{%
4476 \noexpand\forestoeset{min@1}{\forestove{min@1}}%
4477 \noexpand\forestoeset{min@s}{\forestove{min@s}}%
4478 \noexpand\forestoeset{max@1}{\forestove{max@1}}%
4479 \noexpand\forestoeset{max@s}{\forestove{max@s}}%
4480 }\expandafter
4481 }\forest@marshal
4482 }
4483 \def\forest@pgfmathhelper@attribute@toks#1#2{%
4484 \forest@forthis{%
4485 \forest@nameandgo{#1}%
4486 \ifnum\forest@cn=0
4487 \def\pgfmathresult{}%
4488 \else
4489 \forestoget{#2}\pgfmathresult
4490 \fi
4491 }%
4492 }
4493 \def\forest@pgfmathhelper@attribute@dimen#1#2{%
4494 \forest@forthis{%
4495 \forest@nameandgo{#1}%
4496 \ifnum\forest@cn=0
4497 \def\pgfmathresult{0}%
4498 \else
4499 \forestoget{#2}\forest@temp
4500 \edef\pgfmathresult{\expandafter\Pgf@geT\forest@temp}%
4501 \fi
4502 }%
4503 }
4504 \def\forest@pgfmathhelper@attribute@count#1#2{%
4505 \forest@forthis{%
4506 \forest@nameandgo{#1}%
4507 \ifnum\forest@cn=0
4508 \def\pgfmathresult{0}%
4509 \else
4510 \forestoget{#2}\pgfmathresult
4511 \fi
4512 }%
4513 }
4514 \pgfmathdeclarefunction*{id}{1}{%
4515 \forest@forthis{%
4516 \forest@nameandgo{#1}%
4517 \let\pgfmathresult\forest@cn
4518 }%
4519 }

```

7.3 Nodewalk

Setup machinery.

```

4520 \def\forest@nodewalk@n{0}
4521 \def\forest@nodewalk@historyback{0,}
4522 \def\forest@nodewalk@historyforward{0,}
4523 \def\forest@nodewalk@origin{0}

```

```

4524 \def\forest@nodewalk@config@everystep@independent@before#1{% #1 = every step keylist
4525   \forestset{every step}{#1}%
4526 }
4527 \def\forest@nodewalk@config@everystep@independent@after{%
4528   \noexpand\forestset{every step}{\forestrv{every step}}%
4529 }
4530 \def\forest@nodewalk@config@history@independent@before{%
4531   \def\forest@nodewalk@n{0}%
4532   \edef\forest@nodewalk@origin{\forest@cn}%
4533   \def\forest@nodewalk@historyback{0,}%
4534   \def\forest@nodewalk@historyforward{0,}%
4535 }
4536 \def\forest@nodewalk@config@history@independent@after{%
4537   \edef\noexpand\forest@nodewalk@n{\expandonce{\forest@nodewalk@n}}%
4538   \edef\noexpand\forest@nodewalk@origin{\expandonce{\forest@nodewalk@origin}}%
4539   \edef\noexpand\forest@nodewalk@historyback{\expandonce{\forest@nodewalk@historyback}}%
4540   \edef\noexpand\forest@nodewalk@historyforward{\expandonce{\forest@nodewalk@historyforward}}%
4541 }
4542 \def\forest@nodewalk@config@everystep@shared@before#1{% #1 = every step keylist
4543 \def\forest@nodewalk@config@everystep@shared@after{}
4544 \def\forest@nodewalk@config@history@shared@before{}
4545 \def\forest@nodewalk@config@history@shared@after{}
4546 \def\forest@nodewalk@config@everystep@inherited@before#1{% #1 = every step keylist
4547 \let\forest@nodewalk@config@everystep@inherited@after\forest@nodewalk@config@everystep@independent@after
4548 \def\forest@nodewalk@config@history@inherited@before{}
4549 \let\forest@nodewalk@config@history@inherited@after\forest@nodewalk@config@history@independent@after
4550 \def\forest@nodewalk#1#2{% #1 = nodewalk, #2 = every step keylist
4551   \forest@configured@nodewalk{independent}{independent}{inherited}{#1}{#2}%
4552 }
4553 \def\forest@configured@nodewalk#1#2#3#4#5{%
4554   % #1 = every step method, #2 = history method, #3 = on invalid
4555   % #4 = nodewalk, #5 = every step keylist
4556   \def\forest@nodewalk@config@everystep@method{#1}%
4557   \def\forest@nodewalk@config@history@method{#2}%
4558   \def\forest@nodewalk@config@oninvalid{#3}%
4559   \forest@nodewalk{#4}{#5}%
4560 }
4561 \def\forest@nodewalk@oninvalid@inherited@text{inherited}
4562 \def\forest@nodewalk#1#2{% #1 = nodewalk, #2 = every step keylist
4563   \ifx\forest@nodewalk@config@oninvalid\forest@nodewalk@oninvalid@inherited@text
4564     \edef\forest@nodewalk@config@oninvalid{\forest@nodewalk@oninvalid}%
4565   \fi
4566   \edef\forest@marshal{%
4567     \noexpand\pgfqkeys{/forest/nodewalk}{\unexpanded{#1}}%
4568     \csname forest@nodewalk@config@everystep\forest@nodewalk@config@everystep@method @after\endcsname
4569     \csname forest@nodewalk@config@history\forest@nodewalk@config@history@method @after\endcsname
4570     \edef\noexpand\forest@nodewalk@oninvalid{\forest@nodewalk@oninvalid}%
4571   }%
4572   \csname forest@nodewalk@config@everystep\forest@nodewalk@config@everystep@method @before\endcsname{#2}%
4573   \csname forest@nodewalk@config@history\forest@nodewalk@config@history@method @before\endcsname
4574   \edef\forest@nodewalk@oninvalid{\forest@nodewalk@config@oninvalid}%
4575   \forest@nodewalk@fakefalse
4576   \forest@marshal
4577 }
4578 \pgfmathdeclarefunction{valid}{1}{%
4579   \forest@forthis{%
4580     \forest@nameandgo{#1}%
4581     \edef\pgfmathresult{\ifnum\forest@cn=0 0\else 1\fi}%
4582   }%
4583 }
4584 \pgfmathdeclarefunction{invalid}{1}{%

```

```

4585 \forest@forthis{%
4586 \forest@nameandgo{#1}%
4587 \edef\pgfmathresult{\ifnum\forest@cn=0 1\else 0\fi}%
4588 }%
4589 }
4590 \newif\ifforest@nodewalk@fake
4591 \def\forest@nodewalk@oninvalid{error}
4592 \def\forest@nodewalk@makestep{%
4593 \ifnum\forest@cn=0
4594 \csname forest@nodewalk@makestep@oninvalid\forest@nodewalk@oninvalid\endcsname
4595 \else
4596 \forest@nodewalk@makestep@
4597 \fi
4598 }
4599 \csdef{forest@nodewalk@makestep@oninvalid@error if real}{\ifforest@nodewalk@fake\expandafter\forest@nodewalk@
4600 \csdef{forest@nodewalk@makestep@oninvalid@last valid}{%
4601 \forest@nodewalk@tolastvalid
4602 \ifforestdebugnodewalks\forest@nodewalk@makestep@invalidtolastvalid@debug\fi}%
4603 \def\forest@nodewalk@makestep@oninvalid@error{\PackageError{forest}{nodewalk stepped to the invalid node\Mess
4604 \let\forest@nodewalk@makestep@oninvalid@fake\relax
4605 \def\forest@nodewalk@makestep@oninvalid@compatfake{%
4606 \forest@deprecated{last step in stack "\forest@nodewalk@currentstepname", which stepped on an invalid node;
4607 }%
4608 \def\forest@nodewalk@makestep@{%
4609 \ifforestdebugnodewalks\forest@nodewalk@makestep@debug\fi
4610 \ifforest@nodewalk@fake
4611 \else
4612 \edef\forest@nodewalk@n{\number\numexpr\forest@nodewalk@n+1}%
4613 \epreto\forest@nodewalk@historyback{\forest@cn,}%
4614 \def\forest@nodewalk@historyforward{0,}%
4615 \forest@process@keylist@register{every step}%
4616 \fi
4617 }
4618 \def\forest@nodewalk@makestep@debug{%
4619 \edef\forest@marshal{%
4620 \noexpand\typeout{\ifforest@nodewalk@fake fake \fi "\forest@nodewalk@currentstepname" step to node id=\fo
4621 }\forest@marshal
4622 }%
4623 \def\forest@nodewalk@makestep@invalidtolastvalid@debug{%
4624 \edef\forest@marshal{%
4625 \noexpand\typeout{\ifforest@nodewalk@fake fake \fi "\forest@nodewalk@currentstepname" step to invalid nod
4626 }\forest@marshal
4627 }%
4628 \def\forest@handlers@savecurrentpath{%
4629 \edef\pgfkeyscurrentkey{\pgfkeyscurrentpath}%
4630 \let\forest@currentkey\pgfkeyscurrentkey
4631 \pgfkeys@split@path
4632 \edef\forest@currentpath{\pgfkeyscurrentpath}%
4633 \let\forest@currentname\pgfkeyscurrentname
4634 }
4635 \pgfkeys{/handlers/save current path/.code={\forest@handlers@savecurrentpath}}
4636 \newif\ifforest@nodewalkstephandler@style
4637 \newif\ifforest@nodewalkstephandler@autostep
4638 \newif\ifforest@nodewalkstephandler@stripfakesteps
4639 \newif\ifforest@nodewalkstephandler@muststartatvalidnode
4640 \newif\ifforest@nodewalkstephandler@makefor
4641 \let\forest@nodewalkstephandler@styletrueorfalse\forest@nodewalkstephandler@stylefalse
4642 \def\forest@nodewalk@currentstepname{}
4643 \forestset{
4644 /forest/define@step/style/.is if=forest@nodewalkstephandler@style,
4645 /forest/define@step/autostep/.is if=forest@nodewalkstephandler@autostep,

```

```

4646 % the following is useful because some macros use grouping (by \forest@forthis or similar) and therefore, a
4647 /forest/define@step/strip fake steps/.is if=forest@nodewalkstephandler@stripfakesteps,
4648 % this can never happen with autosteps ...
4649 /forest/define@step/autostep/.append code={%
4650   \ifforest@nodewalkstephandler@autostep
4651     \forest@nodewalkstephandler@stripfakestepsfalse
4652   \fi
4653 },
4654 /forest/define@step/must start at valid node/.is if=forest@nodewalkstephandler@muststartatvalidnode,
4655 /forest/define@step/n args/.store in=\forest@nodewalkstephandler@nargs,
4656 /forest/define@step/make for/.is if=forest@nodewalkstephandler@makefor,
4657 /forest/define@step/@bare/.style={strip fake steps=false,must start at valid node=false,make for=false},
4658 define long step/.code n args=3{%
4659   \forest@nodewalkstephandler@styletrueorfalse % true for end users; but in the package, most of steps are
4660   \forest@nodewalkstephandler@autostepfalse
4661   \forest@nodewalkstephandler@stripfakestepstrue
4662   \forest@nodewalkstephandler@muststartatvalidnodetrue % most steps can only start at a valid node
4663   \forest@nodewalkstephandler@makefortrue % make for prefix?
4664   \def\forest@nodewalkstephandler@nargs{0}%
4665   \pgfqkeys{/forest/define@step}{#2}%
4666   \forest@temp@toks{#3}% handler code
4667   \ifforest@nodewalkstephandler@style
4668     \expandafter\forest@temp@toks\expandafter{%
4669       \expandafter\pgfqkeysalso\expandafter{\the\forest@temp@toks}%
4670     }%
4671   \fi
4672   \ifforest@nodewalkstephandler@autostep
4673     \apptotoks\forest@temp@toks{\forest@nodewalk@makestep}%
4674   \fi
4675   \ifforest@nodewalkstephandler@stripfakesteps
4676     \expandafter\forest@temp@toks\expandafter{\expandafter\forest@nodewalk@stripfakesteps\expandafter{\the\
4677   \fi
4678   \ifforest@nodewalkstephandler@muststartatvalidnode
4679     \edef\forest@marshal{%
4680       \noexpand\forest@temp@toks{%
4681         \unexpanded{%
4682           \ifnum\forest@cn=0
4683             \csname forest@nodewalk@start@oninvalid\forest@nodewalk@oninvalid\endcsname{#1}%
4684           \else
4685             }%
4686             \noexpand\@escapeif{\the\forest@temp@toks}%
4687             \noexpand\fi
4688           }%
4689         }\forest@marshal
4690   \fi
4691   \pretotoks\forest@temp@toks{\appto\forest@nodewalk@currentstepname{,#1}}%
4692   \expandafter\forest@temp@toks\expandafter{\expandafter\forest@saveandrestoremacro\expandafter\forest@node
4693   \ifforestdebugnodewalks
4694     \epretotoks\forest@temp@toks{\noexpand\typeout{Starting step "#1" from id=\noexpand\forest@cn
4695       \ifnum\forest@nodewalkstephandler@nargs>0 \space with args ####1\fi
4696       \ifnum\forest@nodewalkstephandler@nargs>1 ,####2\fi
4697       \ifnum\forest@nodewalkstephandler@nargs>2 ,####3\fi
4698       \ifnum\forest@nodewalkstephandler@nargs>3 ,####4\fi
4699       \ifnum\forest@nodewalkstephandler@nargs>4 ,####5\fi
4700       \ifnum\forest@nodewalkstephandler@nargs>5 ,####6\fi
4701       \ifnum\forest@nodewalkstephandler@nargs>6 ,####7\fi
4702       \ifnum\forest@nodewalkstephandler@nargs>7 ,####8\fi
4703       \ifnum\forest@nodewalkstephandler@nargs>8 ,####9\fi
4704     }}%
4705   \fi
4706   \def\forest@temp{/forest/nodewalk/#1/.code}%

```

```

4707 \ifnum\forest@nodewalkstephandler@nargs<2
4708 \eappto\forest@temp{=}%
4709 \else\ifnum\forest@nodewalkstephandler@nargs=2
4710 \eappto\forest@temp{ 2 args=}%
4711 \else
4712 \eappto\forest@temp{ n args={\forest@nodewalkstephandler@nargs}}%
4713 \fi\fi
4714 \eappto\forest@temp{{\the\forest@temp@toks}}%
4715 \expandafter\pgfkeysalso\expandafter{\forest@temp}%
4716 \ifforest@nodewalkstephandler@makefor
4717 \ifnum\forest@nodewalkstephandler@nargs=0
4718 \forestset{%
4719 for #1/.code={\forest@forstepwrapper{#1}{##1}},
4720 }%
4721 \else\ifnum\forest@nodewalkstephandler@nargs=1
4722 \forestset{%
4723 for #1/.code 2 args={\forest@forstepwrapper{#1={##1}}{##2}},
4724 }%
4725 \else
4726 \forestset{%
4727 for #1/.code n args/.expanded=%
4728 {\number\numexpr\forest@nodewalkstephandler@nargs+1}%
4729 {\noexpand\forest@forstepwrapper{#1\ifnum\forest@nodewalkstephandler@nargs>0=\fi\forest@util@narg
4730 }%
4731 \fi\fi
4732 \fi
4733 },
4734 }
4735 {\csname forest@@doc@@hook@bigbadforlist\endcsname}%
4736 \pgfkeys{/handlers}{
4737 .nodewalk style/.code={\forest@handlers@savecurrentpath\pgfkeysalso{
4738 \forest@currentpath/nodewalk/\forest@currentname/.style={#1}%
4739 }},
4740 }

```

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

```

4741 \def\forest@forstepwrapper#1#2{\forest@forthis{\forest@nodewalk{#1}{#2}}}
4742 \def\forest@util@nargs#1#2#3{% #1 = prefix (#, ##, ...), #2 = n args, #3=start; returns {#start+1}...{#start+
4743 \ifnum#2>0 {#1\number\numexpr#3+1}\fi
4744 \ifnum#2>1 {#1\number\numexpr#3+2}\fi
4745 \ifnum#2>2 {#1\number\numexpr#3+3}\fi
4746 \ifnum#2>3 {#1\number\numexpr#3+4}\fi
4747 \ifnum#2>4 {#1\number\numexpr#3+5}\fi
4748 \ifnum#2>5 {#1\number\numexpr#3+6}\fi
4749 \ifnum#2>6 {#1\number\numexpr#3+7}\fi
4750 \ifnum#2>7 {#1\number\numexpr#3+8}\fi
4751 \ifnum#2>8 {#1\number\numexpr#3+9}\fi
4752 }
4753 \def\forest@nodewalk@start@oninvalid@fake#1{}
4754 \def\forest@nodewalk@start@oninvalid@compatfake#1{%
4755 \forest@deprecated{last step in stack "\forest@nodewalk@currentstepname", which started from an invalid nod
4756 }%
4757 \let\forest@nodewalk@start@oninvalid@errorifreal\forest@nodewalk@start@oninvalid@fake % the step will be to a
4758 \let\forest@nodewalk@start@oninvalid@lastvalid\forest@nodewalk@start@oninvalid@fake
4759 \def\forest@nodewalk@start@oninvalid@error#1{\PackageError{forest}{nodewalk step "#1" cannot start at the inv

```

Define long-form single-step walks.

```

4760 \forestset{
4761 define long step={current}{autostep}{},
4762 define long step={next}{autostep}{\edef\forest@cn{\forest@next}},

```

```

4763 define long step={previous}{autostep}{\edef\forest@cn{\forestove{@previous}}},
4764 define long step={parent}{autostep}{\edef\forest@cn{\forestove{@parent}}},
4765 define long step={first}{autostep}{\edef\forest@cn{\forestove{@first}}},
4766 define long step={last}{autostep}{\edef\forest@cn{\forestove{@last}}},
4767 define long step={sibling}{autostep}{%
4768   \edef\forest@cn{%
4769     \ifnum\forestove{@previous}=0
4770       \forestove{@next}%
4771     \else
4772       \forestove{@previous}%
4773     \fi
4774   }%
4775 },
4776 define long step={next node}{autostep}{\edef\forest@cn{\forest@node@linearnextid}},
4777 define long step={previous node}{autostep}{\edef\forest@cn{\forest@node@linearpreviousid}},
4778 define long step={first leaf}{autostep}{%
4779   \safeloop
4780   \edef\forest@cn{\forestove{@first}}%
4781   \unless\ifnum\forestove{@first}=0
4782     \saferepeat
4783 },
4784 define long step={first leaf'}{autostep}{%
4785   \safeloop
4786   \unless\ifnum\forestove{@first}=0
4787     \edef\forest@cn{\forestove{@first}}%
4788   \saferepeat
4789 },
4790 define long step={last leaf}{autostep}{%
4791   \safeloop
4792   \edef\forest@cn{\forestove{@last}}%
4793   \unless\ifnum\forestove{@last}=0
4794     \saferepeat
4795 },
4796 define long step={last leaf'}{autostep}{%
4797   \safeloop
4798   \unless\ifnum\forestove{@last}=0
4799     \edef\forest@cn{\forestove{@last}}%
4800   \saferepeat
4801 },
4802 define long step={next leaf}{style,strip fake steps=false}{group={do until={n_children()==0}{next node}}},
4803 define long step={previous leaf}{style,strip fake steps=false}{group={do until={n_children()==0}{previous n
4804 define long step={next on tier}{autostep,n args=1}{%
4805   \def\forest@temp{#1}%
4806   \ifx\forest@temp\pgfkeysnovalue@text
4807     \foresttoget{tier}\forest@nodewalk@giventier
4808   \else
4809     \def\forest@nodewalk@giventier{#1}%
4810   \fi
4811   \edef\forest@cn{\forest@node@linearnextid}%
4812   \safeloop
4813     \forest@nodewalk@gettier
4814   \ifforest@temp
4815     \edef\forest@cn{\forest@node@linearnextid}%
4816   \saferepeat
4817 },
4818 define long step={previous on tier}{autostep,n args=1}{%
4819   \def\forest@temp{#1}%
4820   \ifx\forest@temp\pgfkeysnovalue@text
4821     \foresttoget{tier}\forest@nodewalk@giventier
4822   \else
4823     \def\forest@nodewalk@giventier{#1}%

```

```

4824 \fi
4825 \safeloop
4826 \edef\forest@cn{\forest@node@linearpreviousid}%
4827 \forest@nodewalk@gettier
4828 \ifforest@temp
4829 \saferepeat
4830 },
4831 TeX={%
4832 \def\forest@nodewalk@gettier{%
4833 \ifnum\forest@cn=0
4834 \forest@tempfalse
4835 \else
4836 \forest@toget{tier}\forest@temp
4837 \ifx\forest@temp\forest@nodewalk@giventier
4838 \forest@tempfalse
4839 \else
4840 \forest@temptrue
4841 \fi
4842 \fi
4843 }%
4844 },
4845 %
4846 define long step={root}{autostep,must start at valid node=false}{%
4847 \edef\forest@cn{\forest@node@rootid}},
4848 define long step={root'}{autostep,must start at valid node=false}{%
4849 \forest@ifdefined{\forest@root}{@parent}{\edef\forest@cn{\forest@root}}{\edef\forest@cn{0}}}%
4850 },
4851 define long step={origin}{autostep,must start at valid node=false}{\edef\forest@cn{\forest@nodewalk@origin}}%
4852 %
4853 define long step={n}{autostep,n args=1}{%
4854 \forest@mathtruncatemacro\forest@temp@n{#1}%
4855 \edef\forest@cn{\forest@node@nthchildid{\forest@temp@n}}}%
4856 },
4857 define long step={n}{autostep,make for=false,n args=1}{%
4858 % Yes, twice. ;- )
4859 % n=1 and n(ext)
4860 \def\forest@nodewalk@temp{#1}%
4861 \ifx\forest@nodewalk@temp\pgfkeysnovalue@text
4862 \edef\forest@cn{\forest@toe{@next}}}%
4863 \else
4864 \forest@mathtruncatemacro\forest@temp@n{#1}%
4865 \edef\forest@cn{\forest@node@nthchildid{\forest@temp@n}}}%
4866 \fi
4867 },
4868 define long step={n'}{autostep,n args=1}{%
4869 \forest@mathtruncatemacro\forest@temp@n{#1}%
4870 \edef\forest@cn{\forest@node@nbarthchildid{\forest@temp@n}}}%
4871 },
4872 define long step={to tier}{autostep,n args=1}{%
4873 \def\forest@nodewalk@giventier{#1}%
4874 \safeloop
4875 \forest@nodewalk@gettier
4876 \ifforest@temp
4877 \forest@toget{@parent}\forest@cn
4878 \saferepeat
4879 },
4880 %
4881 define long step={name}{autostep,n args=1,must start at valid node=false}{%
4882 \edef\forest@cn{%
4883 \forest@node@ifnamedefined{#1}{\forest@node@Nametoid{#1}}{0}}}%
4884 }%

```



```

4885 },
4886 define long step={id}{autostep,n args=1,must start at valid node=false}{%
4887   \forestOifdefined{#1}{@parent}{\edef\forest@cn{#1}}{\edef\forest@cn{0}}%
4888 },
4889 define long step={Nodewalk}{n args=3,@bare}{% #1 = config, #2 = nodewalk
4890   \def\forest@nodewalk@config@everystep@method{independent}%
4891   \def\forest@nodewalk@config@history@method{shared}%
4892   \def\forest@nodewalk@config@oninvalid{inherited}%
4893   \pgfqkeys{/forest/nodewalk@config}{#1}%
4894   \forest@Nodewalk{#2}{#3}%
4895 },
4896 define long step={nodewalk}{n args=2,@bare}{% #1 = nodewalk, #2 = every step
4897   \forest@nodewalk{#1}{#2}%
4898 },
4899 define long step={nodewalk'}{n args=1,@bare}{% #1 = nodewalk
4900   \forest@configured@nodewalk{inherited}{independent}{inherited}{#1}{}%
4901 },
4902 % these "for ..." keys must be defined explicitly
4903 % (and copied into node keyspace manually),
4904 % as prefix "for" normally introduces the every-step keylist
4905 define long step={for nodewalk}{n args=2,@bare}{% #1 = nodewalk, #2 = every step
4906   \forest@forthis{\forest@nodewalk{#1}{#2}},
4907 define long step={for nodewalk'}{n args=1,@bare}{% #1 = nodewalk
4908   \forest@forthis{%
4909     \forest@configured@nodewalk{inherited}{independent}{inherited}{#1}{}%
4910   }%
4911 },
4912 define long step={for Nodewalk}{n args=3,@bare}{% #1 = config, #2 = nodewalk, #3 = every-step
4913   \def\forest@nodewalk@config@everystep@method{independent}%
4914   \def\forest@nodewalk@config@history@method{shared}%
4915   \def\forest@nodewalk@config@oninvalid{inherited}%
4916   \pgfqkeys{/forest/nodewalk@config}{#1}%
4917   \forest@forthis{\forest@Nodewalk{#2}{#3}}%
4918 },
4919 copy command key={/forest/nodewalk/Nodewalk}{/forest/Nodewalk},
4920 copy command key={/forest/nodewalk/for nodewalk}{/forest/for nodewalk},
4921 copy command key={/forest/nodewalk/for Nodewalk}{/forest/for Nodewalk},
4922 declare keylist register=every step,
4923 every step'={},
4924 %% begin nodewalk config
4925 nodewalk@config/.cd,
4926 every@step/.is choice,
4927 every@step/independent/.code={},
4928 every@step/inherited/.code={},
4929 every@step/shared/.code={},
4930 every step/.store in=\forest@nodewalk@config@everystep@method,
4931 every step/.prefix style={every@step=#1},
4932 @history/.is choice,
4933 @history/independent/.code={},
4934 @history/inherited/.code={},
4935 @history/shared/.code={},
4936 history/.store in=\forest@nodewalk@config@history@method,
4937 history/.prefix style={@history=#1},
4938 on@invalid/.is choice,
4939 on@invalid/error/.code={},
4940 on@invalid/fake/.code={},
4941 on@invalid/error if real/.code={},
4942 on@invalid/last valid/.code={},
4943 on@invalid/inherited/.code={},
4944 on invalid/.store in=\forest@nodewalk@config@oninvalid,
4945 on invalid/.prefix style={on@invalid=#1},

```

```

4946   %% end nodewalk config
4947 }
4948 \newtoks\forest@nodewalk@branch@toks
4949 \forestset{
4950   declare toks register=branch@temp@toks,
4951   branch@temp@toks={},
4952   declare keylist register=branched@nodewalk,
4953   branched@nodewalk={},
4954   define long step={branch}{n args=1,@bare,make for,style}{@branch={#1}{branch@build@realstep,branch@build@fa
4955   define long step={branch'}{n args=1,@bare,make for,style}{@branch={#1}{branch@build@realstep}},
4956   @branch/.style 2 args={%
4957     save and restore register={branched@nodewalk}{
4958       branch@temp@toks={},
4959       split/.process={r}{#1}{,}{#2},
4960       branch@temp@style/.style/.register=branch@temp@toks,
4961       branch@temp@style,
4962       branch@temp@style/.style/.register=branched@nodewalk,
4963       branch@temp@style,
4964     }
4965   },
4966   nodewalk/branch@build@realstep/.style={% #1 = nodewalk for this branch
4967     branch@temp@toks/.expanded={for nodewalk={\unexpanded{#1}}{
4968       branched@nodewalk+/.expanded={id=\noexpand\forestoption{id}},
4969       \forestregister{branch@temp@toks}}},
4970   },
4971   nodewalk/branch@build@fakestep/.style={% #1 = nodewalk for this branch
4972     branch@temp@toks/.expanded={for nodewalk={\unexpanded{#1}}{
4973       \forestregister{branch@temp@toks}}},
4974   },
4975   define long step={group}{autostep,n args=1}{\forest@go{#1}},
4976   nodewalk/fake/.code={%
4977     \forest@saveandrestoreifcs{forest@nodewalk@fake}{%
4978       \forest@nodewalk@faketrue
4979       \pgfkeysalso{#1}%
4980     }%
4981   },
4982   nodewalk/real/.code={%
4983     \forest@saveandrestoreifcs{forest@nodewalk@fake}{%
4984       \forest@nodewalk@fakefalse
4985       \pgfkeysalso{#1}%
4986     }%
4987   },
4988   declare keylist register=filtered@nodewalk,
4989   filtered@nodewalk={},
4990   define long step={filter}{n args=2,@bare,make for,style}{% #1 = nodewalk, #2 = condition
4991     save and restore register={filtered@nodewalk}{
4992       filtered@nodewalk'={},
4993       Nodewalk=%
4994       {history=inherited}%
4995       {#1}%
4996       {if={#2}{filtered@nodewalk+/.expanded={id=\forestoption{id}}}{}},
4997       filtered@nodewalk@style/.style/.register=filtered@nodewalk,
4998       filtered@nodewalk@style
4999     },
5000   },
5001   on@invalid/.is choice,
5002   on@invalid/error/.code={},
5003   on@invalid/fake/.code={},
5004   on@invalid/error if real/.code={},
5005   on@invalid/last valid/.code={},
5006   on invalid/.code 2 args={%

```

```

5007 \pgfkeysalso{/forest/on@invalid={#1}}%
5008 \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
5009 \def\forest@nodewalk@oninvalid{#1}%
5010 \pgfkeysalso{#2}%
5011 }%
5012 },
5013 define long step={strip fake steps}{n args=1,@bare}{%
5014 \forest@nodewalk@stripfakesteps{\pgfkeysalso{#1}}},
5015 define long step={unique}{n args=1}{%
5016 \beginngroup
5017 \def\forest@nodewalk@unique@temp{%
5018 \forest@nodewalk{#1}{%
5019 TeX={%
5020 \forestoget{unique@visited}\forest@temp
5021 \ifx\forest@temp\relax
5022 \forestoset{unique@visited}{1}%
5023 \eappto\forest@nodewalk@unique@temp{id=\forest@cn}%
5024 \fi
5025 }%
5026 }%
5027 \global\let\forest@global@temp\forest@nodewalk@unique@temp
5028 \endgroup
5029 \pgfkeysalsofrom{\forest@global@temp}%
5030 },
5031 define long step={walk back}{n args=1,@bare}{%
5032 \forestmathtruncatemacro\forest@temp@n{#1}%
5033 \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\ifnum\forest@cn=0\relax\forest@temp@n-1\fi}%
5034 \forest@nodewalk@back@updatehistory
5035 },
5036 nodewalk/walk back/.default=1,
5037 define long step={jump back}{n args=1,@bare}{%
5038 \forestmathtruncatemacro\forest@temp@n{(#1)+\ifnum\forest@cn=0\relax\forest@temp@n-1\fi}%
5039 \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\forest@temp@n-1}%
5040 \forest@nodewalk@back@updatehistory
5041 },
5042 nodewalk/jump back/.default=1,
5043 define long step={back}{n args=1,@bare}{%
5044 \forestmathtruncatemacro\forest@temp@n{#1}%
5045 \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\ifnum\forest@cn=0\relax\forest@temp@n-1\fi}%
5046 \forest@nodewalk@back@updatehistory
5047 },
5048 nodewalk/back/.default=1,
5049 define long step={walk forward}{n args=1,@bare}{%
5050 \forestmathtruncatemacro\forest@temp@n{#1}%
5051 \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{0}{\forest@temp@n-1}%
5052 \forest@nodewalk@forward@updatehistory
5053 },
5054 nodewalk/walk forward/.default=1,
5055 define long step={jump forward}{n args=1,@bare}{%
5056 \forestmathtruncatemacro\forest@temp@n{#1}%
5057 \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{\forest@temp@n-1}%
5058 \forest@nodewalk@forward@updatehistory
5059 },
5060 nodewalk/jump forward/.default=1,
5061 define long step={forward}{n args=1,@bare}{%
5062 \forestmathtruncatemacro\forest@temp@n{#1}%
5063 \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{0}{\forest@temp@n-1}%
5064 \forest@nodewalk@forward@updatehistory
5065 },
5066 nodewalk/forward/.default=1,
5067 define long step={last valid'}{@bare}{%

```

```

5068 \ifnum\forest@cn=0
5069 \forest@nodewalk@tolastvalid
5070 \forest@nodewalk@makestep
5071 \fi
5072 },
5073 define long step={last valid}{@bare}{%
5074 \forest@nodewalk@tolastvalid
5075 },
5076 define long step={reverse}{n args=1,@bare,make for}{%
5077 \forest@nodewalk{#1,TeX={%
5078 \global\let\forest@global@temp\forest@nodewalk@historyback
5079 \global\let\forest@global@tempn\forest@nodewalk@n
5080 }}{%
5081 \forest@nodewalk@walklist}{\forest@global@temp}{0}{\forest@global@tempn}{\let\forest@cn\forest@nodewalk@cn}
5082 },
5083 define long step={walk and reverse}{n args=1,@bare,make for}{%
5084 \edef\forest@marshal{%
5085 \noexpand\pgfkeysalso{\unexpanded{#1}}%
5086 \noexpand\forest@nodewalk@walklist}{\noexpand\forest@nodewalk@historyback}{0}{\noexpand\forest@nodewalk@n}
5087 }\forest@marshal
5088 },
5089 define long step={sort}{n args=1,@bare,make for}{%
5090 \forest@nodewalk{#1,TeX={%
5091 \global\let\forest@global@temp\forest@nodewalk@historyback
5092 \global\let\forest@global@tempn\forest@nodewalk@n
5093 }}{%
5094 \forest@nodewalk@sortlist{\forest@global@temp}{\forest@global@tempn}\forest@sort@ascending
5095 },
5096 define long step={sort'}{n args=1,@bare,make for}{%
5097 \forest@nodewalk{#1,TeX={%
5098 \global\let\forest@global@temp\forest@nodewalk@historyback
5099 \global\let\forest@global@tempn\forest@nodewalk@n
5100 }}{%
5101 \forest@nodewalk@sortlist{\forest@global@temp}{\forest@global@tempn}\forest@sort@descending
5102 },
5103 define long step={walk and sort}{n args=1,@bare,make for}{% walk as given, then walk sorted
5104 \edef\forest@marshal{%
5105 \noexpand\pgfkeysalso{\unexpanded{#1}}%
5106 \noexpand\forest@nodewalk@sortlist{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n}
5107 }\forest@marshal
5108 },
5109 define long step={walk and sort'}{n args=1,@bare,make for}{%
5110 \edef\forest@marshal{%
5111 \noexpand\pgfkeysalso{\unexpanded{#1}}%
5112 \noexpand\forest@nodewalk@sortlist{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n}
5113 }\forest@marshal
5114 },
5115 declare keylist register=sort by,
5116 copy command key={/forest/sort by'}{/forest/sort by},
5117 sort by={},
5118 define long step={save}{n args=2,@bare,make for}{% #1 = name, #2 = nodewalk
5119 \forest@forthis{%
5120 \forest@nodewalk{#2,TeX={%
5121 \global\let\forest@global@temp\forest@nodewalk@historyback
5122 \global\let\forest@global@tempn\forest@nodewalk@n
5123 }}{%
5124 }%
5125 \forest@nodewalk@walklist}{\forest@global@temp}{0}{\forest@global@tempn}\relax
5126 \csedef\forest@nodewalk@sav#1{\forest@nodewalk@walklist@walked}%
5127 },
5128 define long step={walk and save}{n args=2,@bare,make for}{% #1 = name, #2 = nodewalk

```

```

5129 \edef\forest@marshal{%
5130 \noexpand\pgfkeysalso{\unexpanded{#2}}%
5131 \noexpand\forest@nodewalk@walklist}{\noexpand\forest@nodewalk@historyback}{0}{\noexpand\forest@nodewalk@
5132 }\forest@marshal
5133 \csedef{forest@nodewalk@samed@#1}{\forest@nodewalk@walklist@walked}%
5134 },
5135 define long step={save append}{style,n args=2,@bare,make for}{% #1 = nodewalk name, #2 = nodewalk
5136 save@append@prepend={#1}{#2}{save}{\cseappto}},
5137 define long step={save prepend}{style,n args=2,@bare,make for}{% #1 = nodewalk name, #2 = nodewalk
5138 save@append@prepend={#1}{#2}{save}{\csepreto}},
5139 define long step={walk and save append}{style,n args=2,@bare,make for}{% #1 = nodewalk name, #2 = nodewalk
5140 save@append@prepend={#1}{#2}{walk and save}{\cseappto}},
5141 define long step={walk and save prepend}{style,n args=2,@bare,make for}{% #1 = nodewalk name, #2 = nodewalk
5142 save@append@prepend={#1}{#2}{walk and save}{\csepreto}},
5143 nodewalk/save@append@prepend/.code n args=4{%
5144 % #1 = nodewalk name, #2 = nodewalk
5145 % #3 = "(walk and) save" #4 = \cseappto/\csepreto
5146 \pgfkeysalso{#3={@temp}{#2}}%
5147 \letcs\forest@temp{forest@nodewalk@samed@temp}%
5148 #4{forest@nodewalk@samed@#1}{\expandonce{\forest@temp}}%
5149 },
5150 nodewalk/save history/.code 2 args={% #1 = back, forward
5151 \csedef{forest@nodewalk@samed@#1}{\forest@nodewalk@historyback}%
5152 \csedef{forest@nodewalk@samed@#2}{\forest@nodewalk@historyforward}%
5153 },
5154 define long step={load}{n args=1,@bare,make for}{%
5155 \forest@nodewalk@walklist}{\csuse{forest@nodewalk@samed@#1}0,}{0}{-1}{\ifnum\forest@nodewalk@cn=0 \else\
5156 },
5157 if in saved nodewalk/.code n args=4{% is node #1 in nodewalk #2; yes: #3, no: #4
5158 \forest@forthis{%
5159 \forest@go{#1}%
5160 \edef\forest@marshal{%
5161 \noexpand\pgfutil@in@{\forest@cn,}{\csuse{forest@nodewalk@samed@#2}},}%
5162 }\forest@marshal
5163 }%
5164 \ifpgfutil@in@
5165 \@escapeif{\pgfkeysalso{#3}}%
5166 \else
5167 \@escapeif{\pgfkeysalso{#4}}%
5168 \fi
5169 },
5170 where in saved nodewalk/.style n args=4{
5171 for tree={if in saved nodewalk={#1}{#2}{#3}{#4}}
5172 },
5173 nodewalk/options/.code={\forestset{#1}},
5174 nodewalk/TeX/.code={#1},
5175 nodewalk/TeX'/.code={\appto\forest@externalize@loadimages{#1}#1},
5176 nodewalk/TeX''/.code={\appto\forest@externalize@loadimages{#1}},
5177 nodewalk/typeout/.style={TeX={\typeout{#1}}},
5178 % repeat is taken later from /forest/repeat
5179 }
5180 \def\forest@nodewalk@walklist#1#2#3#4#5{%
5181 % #1 = list of preceding, #2 = list to walk
5182 % #3 = from, #4 = to
5183 % #5 = every step code
5184 \let\forest@nodewalk@cn\forest@cn
5185 \edef\forest@marshal{%
5186 \noexpand\forest@nodewalk@walklist@{#1}{#2}{\number\numexpr#3}{\number\numexpr#4}{1}{0}{\unexpanded{#5}}%
5187 }\forest@marshal
5188 }
5189 \def\forest@nodewalk@walklist@#1#2#3#4#5#6#7{%

```

```

5190 % #1 = list of walked, #2 = list to walk
5191 % #3 = from, #4 = to
5192 % #5 = current step n, #6 = steps made
5193 % #7 = every step code
5194 \def\forest@nodewalk@walklist@walked{#1}%
5195 \def\forest@nodewalk@walklist@rest{#2}%
5196 \edef\forest@nodewalk@walklist@stepsmade{#6}%
5197 \ifnum#4<0
5198   \forest@temptrue
5199 \else
5200   \ifnum#5>#4\relax
5201     \forest@tempfalse
5202   \else
5203     \forest@temptrue
5204   \fi
5205 \fi
5206 \ifforest@temp
5207   \edef\forest@nodewalk@cn{\forest@csvlist@getfirst@{#2}}%
5208   \ifnum\forest@nodewalk@cn=0
5209     #7%
5210   \else
5211     \ifnum#5>#3\relax
5212       #7%
5213       \edef\forest@nodewalk@walklist@stepsmade{\number\numexpr#6+1}%
5214     \fi
5215     \forest@csvlist@getfirstrest@{#2}\forest@nodewalk@cn\forest@nodewalk@walklist@rest
5216     \@escapeiffif{%
5217       \edef\forest@marshal{%
5218         \noexpand\forest@nodewalk@walklist@
5219         {\forest@nodewalk@cn,#1}{\forest@nodewalk@walklist@rest}{#3}{#4}{\number\numexpr#5+1}{\forest@nod
5220         }\forest@marshal
5221       }%
5222     \fi
5223   \fi
5224 }
5225
5226 \def\forest@nodewalk@back@updatehistory{%
5227   \ifnum\forest@cn=0
5228     \let\forest@nodewalk@historyback\forest@nodewalk@walklist@rest
5229     \let\forest@nodewalk@historyforward\forest@nodewalk@walklist@walked
5230   \else
5231     \expandafter\forest@csvlist@getfirstrest@\expandafter{\forest@nodewalk@walklist@walked}\forest@temp\forest@
5232     \edef\forest@nodewalk@historyback{\forest@temp,\forest@nodewalk@walklist@rest}%
5233   \fi
5234 }
5235 \def\forest@nodewalk@forward@updatehistory{%
5236   \let\forest@nodewalk@historyforward\forest@nodewalk@walklist@rest
5237   \let\forest@nodewalk@historyback\forest@nodewalk@walklist@walked
5238 }
5239 \def\forest@go#1{%
5240   \forest@configured@nodewalk{independent}{inherited}{inherited}{#1}{}%
5241 }
5242 \def\forest@csvlist@getfirst@#1{% assuming that the list is nonempty and finishes with a comma
5243   \forest@csvlist@getfirst@@#1\forest@csvlist@getfirst@@}
5244 \def\forest@csvlist@getfirst@@#1,#2\forest@csvlist@getfirst@@{#1}
5245 \def\forest@csvlist@getrest@#1{% assuming that the list is nonempty and finishes with a comma
5246   \forest@csvlist@getrest@@#1\forest@csvlist@getrest@@}
5247 \def\forest@csvlist@getrest@@#1,#2\forest@csvlist@getrest@@{#2}
5248 \def\forest@csvlist@getfirstrest@#1#2#3{% assuming that the list is nonempty and finishes with a comma
5249   % #1 = list, #2 = cs receiving first, #3 = cs receiving rest
5250   \forest@csvlist@getfirstrest@@#1\forest@csvlist@getfirstrest@@{#2}{#3}}

```

```

5251 \def\forest@csvlist@getfirstrest@@#1,#2\forest@csvlist@getfirstrest@@#3#4{%
5252   \def#3{#1}%
5253   \def#4{#2}%
5254 }
5255 \def\forest@nodewalk@stripfakesteps#1{%
5256   % go to the last valid node if the walk contained any nodes, otherwise restore the current node
5257   \edef\forest@marshal{%
5258     \unexpanded{#1}%
5259     \noexpand\ifnum\noexpand\forest@nodewalk@n=\forest@nodewalk@n\relax
5260     \def\noexpand\forest@cn{\forest@cn}%
5261     \noexpand\else
5262     \unexpanded{%
5263       \edef\forest@cn{%
5264         \expandafter\forest@csvlist@getfirst@\expandafter{\forest@nodewalk@historyback}%
5265       }%
5266     }%
5267     \noexpand\fi
5268   }\forest@marshal
5269 }
5270 \def\forest@nodewalk@tolastvalid{%
5271   \ifnum\forest@cn=0
5272     \edef\forest@cn{\expandafter\forest@csvlist@getfirst@\expandafter{\forest@nodewalk@historyback}}%
5273     \ifnum\forest@cn=0
5274       \let\forest@cn\forest@nodewalk@origin
5275     \fi
5276   \fi
5277 }
5278 \def\forest@nodewalk@sortlist#1#2#3{%#1=list,#2=to,#3=asc/desc
5279   \edef\forest@nodewalksort@list{#1}%
5280   \expandafter\forest@nodewalk@sortlist@\expandafter{\number\numexpr#2-#3}%
5281 }
5282 \def\forest@nodewalk@sortlist@#1#2{%#1=to,#2=asc/desc
5283   \safeloop
5284     \unless\ifnum\safeloopn>#1\relax
5285       \expandafter\forest@csvlist@getfirstrest@\expandafter{\forest@nodewalksort@list}\forest@nodewalksort@cn\fi
5286       \csedef\forest@nodesort@\safeloopn{\forest@nodewalksort@cn}%
5287     \saferepeat
5288     \foreststrget{sort by}\forest@nodesort@sortkey
5289     \forest@sort\forest@nodesort@cmpnodes\forest@nodesort@let#2{1}{#1}%
5290     \def\forest@nodewalksort@sorted{}%
5291     \safeloop
5292       \unless\ifnum\safeloopn>#1\relax
5293         \edef\forest@cn{\csname forest@nodesort@\safeloopn\endcsname}%
5294         \forest@nodewalk@makestep
5295       \saferepeat
5296 }

```

Find minimal/maximal node in a walk.

```

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

```

```

5310 define long step={walk and min}{n args=1,@bare}{%
5311     \edef\forest@marshal{%
5312         \noexpand\pgfkeysalso{\unexpanded{#1}}%
5313         \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
5314     }\forest@marshal
5315 },
5316 define long step={walk and mins}{n args=1,@bare}{%
5317     \edef\forest@marshal{%
5318         \noexpand\pgfkeysalso{\unexpanded{#1}}%
5319         \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
5320     }\forest@marshal
5321 },
5322 define long step={min in nodewalk}{@bare}{% find the first min in the preceding nodewalk, step to it
5323     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{\forest@nodewalk@minmax@node,}%
5324 },
5325 define long step={mins in nodewalk}{@bare}{% find mins in the preceding nodewalk, step to mins
5326     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{\forest@nodewalk@minmax@nodes,}%
5327 },
5328 define long step={min in nodewalk'}{@bare}{% find the first min in the preceding nodewalk, step to min in h
5329     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{}%
5330 },
5331 %
5332 define long step={max}{n args=1,@bare,make for}{% the first max in the argument nodewalk
5333     \forest@nodewalk{#1,TeX={%
5334         \global\let\forest@global@temp\forest@nodewalk@historyback
5335     }}{%
5336         \forest@nodewalk@minmax{\forest@global@temp}{-1}{>}{\forest@nodewalk@minmax@node,}%
5337     },
5338 define long step={maxs}{n args=1,@bare,make for}{% all maxs in the argument nodewalk
5339     \forest@nodewalk{#1,TeX={%
5340         \global\let\forest@global@temp\forest@nodewalk@historyback
5341     }}{%
5342         \forest@nodewalk@minmax{\forest@global@temp}{-1}{>}{\forest@nodewalk@minmax@nodes}%
5343     },
5344 define long step={walk and max}{n args=1,@bare}{%
5345     \edef\forest@marshal{%
5346         \noexpand\pgfkeysalso{\unexpanded{#1}}%
5347         \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
5348     }\forest@marshal
5349 },
5350 define long step={walk and maxs}{n args=1,@bare}{%
5351     \edef\forest@marshal{%
5352         \noexpand\pgfkeysalso{\unexpanded{#1}}%
5353         \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
5354     }\forest@marshal
5355 },
5356 define long step={max in nodewalk}{@bare}{% find the first max in the preceding nodewalk, step to it
5357     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{\forest@nodewalk@minmax@node,}%
5358 },
5359 define long step={maxs in nodewalk}{@bare}{% find maxs in the preceding nodewalk, step to maxs
5360     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{\forest@nodewalk@minmax@nodes,}%
5361 },
5362 define long step={max in nodewalk'}{@bare}{% find the first max in the preceding nodewalk, step to max in h
5363     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{}%
5364 },
5365 }
5366
5367 \def\forest@nodewalk@minmax#1#2#3#4{%
5368     % #1 = list of nodes
5369     % #2 = max index in list (start with 1)
5370     % #3 = min/max = ascending/descending = </>

```



```

5371 % #4 = how many results? 1 = {\forest@nodewalk@minmax@node,}, all={\forest@nodewalk@minmax@nodes}, walk in
5372 \forestget{sort by}\forest@nodesort@sortkey
5373 \edef\forest@nodewalk@minmax@N{\number\numexpr#2}%
5374 \edef\forest@nodewalk@minmax@n{%
5375 \edef\forest@nodewalk@minmax@list{#1}%
5376 \def\forest@nodewalk@minmax@nodes{%
5377 \def\forest@nodewalk@minmax@node{%
5378 \ifdefempty{\forest@nodewalk@minmax@list}{%
5379 }{%
5380 \safeloop
5381 \expandafter\forest@csvlist@getfirstrest@\expandafter{\forest@nodewalk@minmax@list}\forest@nodewalk@min
5382 \ifnum\forest@nodewalk@minmax@cn=0 \else
5383 \ifdefempty{\forest@nodewalk@minmax@node}{%
5384 \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
5385 \edef\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,}%
5386 \edef\forest@nodewalk@minmax@n{\safeloopn}%
5387 }{%
5388 \csedef{forest@nodesort@1}{\forest@nodewalk@minmax@node}%
5389 \csedef{forest@nodesort@2}{\forest@nodewalk@minmax@cn}%
5390 \forest@nodesort@cmpnodes{2}{1}%
5391 \if=\forest@sort@cmp@result
5392 \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
5393 \epreto\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,}%
5394 \edef\forest@nodewalk@minmax@n{\safeloopn}%
5395 \else
5396 \if#3\forest@sort@cmp@result
5397 \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
5398 \edef\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,}%
5399 \edef\forest@nodewalk@minmax@n{\safeloopn}%
5400 \fi
5401 \fi
5402 }%
5403 \fi
5404 \ifdefempty{\forest@nodewalk@minmax@list}{\forest@tempfalse}{\forest@temptrue}%
5405 \ifnum\safeloopn=\forest@nodewalk@minmax@N\relax\forest@temptrue\fi
5406 \ifforest@temp
5407 \saferepeat
5408 \edef\forest@nodewalk@minmax@list{#4}%
5409 \ifdefempty\forest@nodewalk@minmax@list{%
5410 \forestset{nodewalk/jump back=\forest@nodewalk@minmax@n-1}% CHECK
5411 }{%
5412 \safeloop
5413 \expandafter\forest@csvlist@getfirstrest@\expandafter{\forest@nodewalk@minmax@list}\forest@cn\forest@
5414 \forest@nodewalk@makestep
5415 \ifdefempty{\forest@nodewalk@minmax@list}{\forest@tempfalse}{\forest@temptrue}%
5416 \ifforest@temp
5417 \saferepeat
5418 }%
5419 }%
5420 }

```

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.

```

5421 \newtoks\forest@nodewalk@shortsteps@resolution
5422 \newif\ifforest@nodewalk@areshortsteps
5423 \pgfqkeys{/forest/nodewalk}{
5424 .unknown/.code={%
5425 \forest@nodewalk@areshortstepsfalse
5426 \pgfkeysifdefined{/forest/\pgfkeyscurrentname/.cmd}{%
5427 }{%

```

```

5428 \ifx\pgfkeyscurrentvalue\pgfkeysnovalue@text % no value, so possibly short steps
5429 \forest@nodewalk@shortsteps@resolution{}}%
5430 \forest@nodewalk@aeshortstepstrue
5431 \expandafter\forest@nodewalk@shortsteps\pgfkeyscurrentname=====,% "=" and "," cannot be short st
5432 \fi
5433 }%
5434 \ifforest@nodewalk@aeshortsteps
5435 \@escapeif{\expandafter\pgfkeysalso\expandafter{\the\forest@nodewalk@shortsteps@resolution}}%
5436 \else
5437 \@escapeif{\pgfkeysalso{/forest/\pgfkeyscurrentname={#1}}}%
5438 \fi
5439 },
5440 }
5441 \def\forest@nodewalk@shortsteps{%
5442 \futurelet\forest@nodewalk@nexttoken\forest@nodewalk@shortsteps@
5443 }
5444 \def\forest@nodewalk@shortsteps@{%
5445 \ifx\forest@nodewalk@nexttoken=%
5446 \let\forest@nodewalk@nextop\forest@nodewalk@shortsteps@end
5447 \else
5448 \ifx\forest@nodewalk@nexttoken\bgroup
5449 \letcs\forest@nodewalk@nextop{forest@shortstep@group}%
5450 \else
5451 \let\forest@nodewalk@nextop\forest@nodewalk@shortsteps@@
5452 \fi
5453 \fi
5454 \forest@nodewalk@nextop
5455 }
5456 \def\forest@nodewalk@shortsteps@@#1{%
5457 \ifcsdef{forest@shortstep@#1}{%
5458 \csname forest@shortstep@#1\endcsname
5459 }{%
5460 \forest@nodewalk@aeshortstepsfalse
5461 \forest@nodewalk@shortsteps@end
5462 }%
5463 }
5464 % in the following definitions:
5465 % #1 = short step
5466 % #2 = (long) step, or a style in /forest/nodewalk (taking n args)
5467 \csdef{forest@nodewalk@defshortstep@0@args}#1#2{%
5468 \csdef{forest@shortstep@#1}{%
5469 \apptotoks\forest@nodewalk@shortsteps@resolution{,#2}%
5470 \forest@nodewalk@shortsteps}}
5471 \csdef{forest@nodewalk@defshortstep@1@args}#1#2{%
5472 \csdef{forest@shortstep@#1}##1{%
5473 \edef\forest@marshal####1{#2}%
5474 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}}%
5475 \forest@nodewalk@shortsteps}}
5476 \csdef{forest@nodewalk@defshortstep@2@args}#1#2{%
5477 \csdef{forest@shortstep@#1}##1##2{%
5478 \edef\forest@marshal####1####2{#2}%
5479 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}}%
5480 \forest@nodewalk@shortsteps}}
5481 \csdef{forest@nodewalk@defshortstep@3@args}#1#2{%
5482 \csdef{forest@shortstep@#1}##1##2##3{%
5483 \edef\forest@marshal####1####2####3{#2}%
5484 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}}%
5485 \forest@nodewalk@shortsteps}}
5486 \csdef{forest@nodewalk@defshortstep@4@args}#1#2{%
5487 \csdef{forest@shortstep@#1}##1##2##3##4{%
5488 \edef\forest@marshal####1####2####3####4{#2}%

```

```

5489 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}}%
5490 \forest@nodewalk@shortsteps}}
5491 \csdef{forest@nodewalk@defshortstep@5@args}#1#2{%
5492 \csdef{forest@shortstep@#1}##1##2##3##4##5{%
5493 \edef\forest@marshal####1#####2#####3#####4#####5{#2}%
5494 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}}%
5495 \forest@nodewalk@shortsteps}}
5496 \csdef{forest@nodewalk@defshortstep@6@args}#1#2{%
5497 \csdef{forest@shortstep@#1}##1##2##3##4##5##6{%
5498 \edef\forest@marshal####1#####2#####3#####4#####5#####6{#2}%
5499 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}}%
5500 \forest@nodewalk@shortsteps}}
5501 \csdef{forest@nodewalk@defshortstep@7@args}#1#2{%
5502 \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7{%
5503 \edef\forest@marshal####1#####2#####3#####4#####5#####6#####7{#2}%
5504 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}}%
5505 \forest@nodewalk@shortsteps}}
5506 \csdef{forest@nodewalk@defshortstep@8@args}#1#2{%
5507 \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7##8{%
5508 \edef\forest@marshal####1#####2#####3#####4#####5#####6#####7#####8{#2}%
5509 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}{##8}}%
5510 \forest@nodewalk@shortsteps}}
5511 \csdef{forest@nodewalk@defshortstep@9@args}#1#2{%
5512 \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7##8##9{%
5513 \edef\forest@marshal####1#####2#####3#####4#####5#####6#####7#####8#####9{#2}%
5514 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}{##8}{##9}}%
5515 \forest@nodewalk@shortsteps}}
5516 \forestset{
5517   define short step/.code n args=3{% #1 = short step, #2 = n args, #3 = long step
5518     \csname forest@nodewalk@defshortstep@#2@args\endcsname{##1}{##3}%
5519   },
5520 }
5521 \def\forest@nodewalk@shortsteps@end#1,{
  Define short-form steps.

5522 \forestset{
5523   define short step={group}{1}{group={#1}}, % {braces} are special
5524   define short step={p}{0}{previous},
5525   define short step={n}{0}{next},
5526   define short step={u}{0}{parent},
5527   define short step={s}{0}{sibling},
5528   define short step={c}{0}{current},
5529   define short step={o}{0}{origin},
5530   define short step={r}{0}{root},
5531   define short step={R}{0}{root'},
5532   define short step={P}{0}{previous leaf},
5533   define short step={N}{0}{next leaf},
5534   define short step={F}{0}{first leaf},
5535   define short step={L}{0}{last leaf},
5536   define short step={>}{0}{next on tier},
5537   define short step={<}{0}{previous on tier},
5538   define short step={1}{0}{n=1},
5539   define short step={2}{0}{n=2},
5540   define short step={3}{0}{n=3},
5541   define short step={4}{0}{n=4},
5542   define short step={5}{0}{n=5},
5543   define short step={6}{0}{n=6},
5544   define short step={7}{0}{n=7},
5545   define short step={8}{0}{n=8},
5546   define short step={9}{0}{n=9},
5547   define short step={l}{0}{last},

```

```

5548 define short step={b}{0}{back},
5549 define short step={f}{0}{forward},
5550 define short step={v}{0}{last valid},
5551 define short step={*}{2}{repeat={#1}{#2}},
5552 for 1/.style={for nodewalk={n=1}{#1}},
5553 for 2/.style={for nodewalk={n=2}{#1}},
5554 for 3/.style={for nodewalk={n=3}{#1}},
5555 for 4/.style={for nodewalk={n=4}{#1}},
5556 for 5/.style={for nodewalk={n=5}{#1}},
5557 for 6/.style={for nodewalk={n=6}{#1}},
5558 for 7/.style={for nodewalk={n=7}{#1}},
5559 for 8/.style={for nodewalk={n=8}{#1}},
5560 for 9/.style={for nodewalk={n=9}{#1}},
5561 for -1/.style={for nodewalk={n'=1}{#1}},
5562 for -2/.style={for nodewalk={n'=2}{#1}},
5563 for -3/.style={for nodewalk={n'=3}{#1}},
5564 for -4/.style={for nodewalk={n'=4}{#1}},
5565 for -5/.style={for nodewalk={n'=5}{#1}},
5566 for -6/.style={for nodewalk={n'=6}{#1}},
5567 for -7/.style={for nodewalk={n'=7}{#1}},
5568 for -8/.style={for nodewalk={n'=8}{#1}},
5569 for -9/.style={for nodewalk={n'=9}{#1}},
5570 }

```

Define multiple-step walks.

```

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

```

```

5607 \else
5608 \edef\forest@marshal{%
5609 \noexpand\forest@node@foreach@breadthfirst
5610 {\forest@temp}%
5611 {\noexpand\forest@nodewalk@makestep}%
5612 }\forest@marshal
5613 \fi
5614 },
5615 define long step={level reversed}{n args=1}{%
5616 \forestmathtruncatemacro\forest@temp{#1}%
5617 \edef\forest@marshal{%
5618 \noexpand\forest@node@foreach@breadthfirst@reversed
5619 {\forest@temp}%
5620 {\noexpand\ifnum\noexpand\forestove{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexp
5621 }\forest@marshal
5622 },
5623 define long step={level reversed>}{n args=1}{%
5624 \forestmathtruncatemacro\forest@temp{#1}%
5625 \edef\forest@marshal{%
5626 \noexpand\forest@node@foreach@breadthfirst@reversed
5627 {-1}%
5628 {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
5629 }\forest@marshal
5630 },
5631 define long step={level reversed<}{n args=1}{%
5632 \forestmathtruncatemacro\forest@temp{(#1)-1}%
5633 \edef\forest@marshal{%
5634 \noexpand\forest@node@foreach@breadthfirst@reversed
5635 {\forest@temp}%
5636 {\noexpand\forest@nodewalk@makestep}%
5637 }\forest@marshal
5638 },
5639 %
5640 define long step={relative level}{n args=1}{%
5641 \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
5642 \edef\forest@marshal{%
5643 \noexpand\forest@node@foreach@breadthfirst
5644 {\forest@temp}%
5645 {\noexpand\ifnum\noexpand\forestove{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexp
5646 }\forest@marshal
5647 },
5648 define long step={relative level>}{n args=1}{%
5649 \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
5650 \edef\forest@marshal{%
5651 \noexpand\forest@node@foreach@breadthfirst
5652 {-1}%
5653 {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
5654 }\forest@marshal
5655 },
5656 define long step={relative level<}{n args=1}{%
5657 \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}-1}%
5658 \edef\forest@marshal{%
5659 \noexpand\forest@node@foreach@breadthfirst
5660 {\forest@temp}%
5661 {\noexpand\forest@nodewalk@makestep}%
5662 }\forest@marshal
5663 },
5664 define long step={relative level reversed}{n args=1}{%
5665 \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
5666 \edef\forest@marshal{%
5667 \noexpand\forest@node@foreach@breadthfirst@reversed

```

```

5668     {\forest@temp}%
5669     {\noexpand\ifnum\noexpand\forestove{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexp
5670 } \forest@marshal
5671 },
5672 define long step={relative level reversed>}{n args=1}{%
5673   \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
5674   \edef\forest@marshal{%
5675     \noexpand\forest@node@foreach@breadthfirst@reversed
5676     {-1}%
5677     {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
5678 } \forest@marshal
5679 },
5680 define long step={relative level reversed<}{n args=1}{%
5681   \forestmathtruncatemacro\forest@temp{(#1)+\forestove{level}-1}%
5682   \edef\forest@marshal{%
5683     \noexpand\forest@node@foreach@breadthfirst@reversed
5684     {\forest@temp}%
5685     {\noexpand\forest@nodewalk@makestep}%
5686   } \forest@marshal
5687 },
5688 define long step={leaves}{}{%
5689   \forest@node@foreach{%
5690     \ifnum\forestove{n children}=0
5691       \forest@nodewalk@makestep
5692     \fi
5693   }%
5694 },
5695 define long step={-level}{n args=1,style}{%
5696   unique={branch={leaves,{group={repeat={#1}{parent}}}}}}
5697 },
5698 define long step={-level'}{n args=1,style}{%
5699   unique={on invalid={fake}{branch={leaves,{group={repeat={#1}{parent}}}}}}
5700 },
5701 define long step={children}{}{\forest@node@foreachchild{\forest@nodewalk@makestep}},
5702 define long step={children reversed}{}{\forest@node@foreachchild@reversed{\forest@nodewalk@makestep}},
5703 define long step={current and following siblings}{}{\forest@node@@forselfandfollowingsiblings{\forest@nodewalk@makestep}},
5704 define long step={following siblings}{style}{if nodewalk valid={next}{fake=next,current and following siblings}}
5705 define long step={current and preceding siblings}{}{\forest@node@@forselfandprecedingsiblings{\forest@nodewalk@makestep}},
5706 define long step={preceding siblings}{style}{if nodewalk valid={previous}{fake=previous,current and preceding siblings}}
5707 define long step={current and following siblings reversed}{}{\forest@node@@forselfandfollowingsiblings@reversed{\forest@nodewalk@makestep}},
5708 define long step={following siblings reversed}{style}{fake=next,current and following siblings reversed},
5709 define long step={current and preceding siblings reversed}{}{\forest@node@@forselfandprecedingsiblings@reversed{\forest@nodewalk@makestep}},
5710 define long step={preceding siblings reversed}{style}{fake=previous,current and preceding siblings reversed},
5711 define long step={siblings}{style}{for nodewalk'={preceding siblings},following siblings},
5712 define long step={siblings reversed}{style}{for nodewalk'={following siblings reversed},preceding siblings},
5713 define long step={current and siblings}{style}{for nodewalk'={preceding siblings},current and following siblings},
5714 define long step={current and siblings reversed}{style}{for nodewalk'={current and following siblings reversed},current and preceding siblings},
5715 define long step={ancestors}{style}{while={}{parent},last valid},
5716 define long step={current and ancestors}{style}{current,ancestors},
5717 define long step={following nodes}{style}{while={}{next node},last valid},
5718 define long step={preceding nodes}{style}{while={}{previous node},last valid},
5719 define long step={current and following nodes}{style}{current,following nodes},
5720 define long step={current and preceding nodes}{style}{current,preceding nodes},
5721 }
5722 \let\forest@nodewalkstephandler@styletrueorfalse\forest@nodewalkstephandler@styletrue

```

7.4 Dynamic tree

```

5723 \def\forest@last@node{0}
5724 \csdef{forest@nodewalk@saved@dynamic nodes}{}

```

```

5725 \def\forest@nodehandleby@name@nodewalk@or@bracket#1{%
5726   \ifx\pgfkeysnovalue#1%
5727     \edef\forest@last@node{\forest@node@Nametoid{forest@last@node}}%
5728   \else
5729     \forest@nodehandleby@nnb@checkfirst#1\forest@END
5730   \fi
5731 }
5732 \def\forest@nodehandleby@nnb@checkfirst#1#2\forest@END{%
5733   \ifx[#1%
5734     \forest@create@node{#1#2}%
5735     \cseappto{forest@nodewalk@saved@dynamic nodes}{\forest@last@node,}%
5736   \else
5737     \forest@forthis{%
5738       \forest@nameandgo{#1#2}%
5739       \ifnum\forest@cn=0
5740         \PackageError{forest}{Cannot use a dynamic key on the invalid node}{}%
5741       \fi
5742       \let\forest@last@node\forest@cn
5743     }%
5744   \fi
5745 }
5746 \def\forest@create@node#1{% #1=bracket representation
5747   \bracketParse{\forest@create@collectafterthought}%
5748     \forest@last@node=#1\forest@end@create@node
5749 }
5750 \def\forest@create@collectafterthought#1\forest@end@create@node{%
5751   \forest@node@Foreach{\forest@last@node}{%
5752     \forest@toletto{delay}{given options}%
5753     \forest@setto{given options}{}%
5754   }%
5755   \forest@eappto{\forest@last@node}{delay}{,\unexpanded{#1}}%
5756   \forest@set{\forest@last@node}{given options}{delay={}}%
5757 }
5758 \def\forest@create@node@and@process@given@options#1{% #1=bracket representation
5759   \bracketParse{\forest@createandprocess@collectafterthought}%
5760     \forest@last@node=#1\forest@end@create@node
5761 }
5762 \def\forest@createandprocess@collectafterthought#1\forest@end@create@node{%
5763   \forest@node@Compute@numeric@ts@info{\forest@last@node}%
5764   \forest@saveandrestoremacro\forest@root{%
5765     \let\forest@root\forest@last@node
5766     \forest@set{process keylist=given options}%
5767   }%
5768 }
5769 \def\forest@saveandrestoremacro#1#2{% #1 = the (zero-arg) macro to save before and restore after processing c
5770   \edef\forest@marshal{%
5771     \unexpanded{#2}%
5772     \noexpand\def\noexpand#1{\expandonce{#1}}%
5773   }\forest@marshal
5774 }
5775 \def\forest@saveandrestoreifcs#1#2{% #1 = the if cs to save before and restore after processing code in #2
5776   \edef\forest@marshal{%
5777     \unexpanded{#2}%
5778     \ifbool{#1}{\noexpand\setbool{#1}{true}}{\noexpand\setbool{#1}{false}}%
5779   }\forest@marshal
5780 }
5781 \def\forest@globalsaveandrestoreifcs#1#2{% #1 = the if cs to save before and restore after processing code in
5782   \edef\forest@marshal{%
5783     \unexpanded{#2}%
5784     \ifbool{#1}{\global\noexpand\setbool{#1}{true}}{\global\noexpand\setbool{#1}{false}}%
5785   }\forest@marshal

```

```

5786 }
5787 \def\forest@saveandrestoretoks#1#2{% #1 = the toks to save before and restore after processing code in #2
5788   \edef\forest@marshal{%
5789     \unexpanded{#2}%
5790     \noexpand#1{\the#1}%
5791   }\forest@marshal
5792 }
5793 \def\forest@saveandrestoreregister#1#2{% #1 = the register to save before and restore after processing code i
5794   \edef\forest@marshal{%
5795     \unexpanded{#2}%
5796     \noexpand\forestset{#1}{\forestregister{#1}}%
5797   }\forest@marshal
5798 }
5799 \forestset{
5800   save and restore register/.code 2 args={%
5801     \forest@saveandrestoreregister{#1}{%
5802       \pgfkeysalso{#2}%
5803     }%
5804   },
5805 }
5806 \def\forest@remove@node#1{%
5807   \ifforestdebugdynamics\forestdebug@dynamics{before removing #1}\fi
5808   \forest@node@Remove{#1}%
5809 }
5810 \def\forest@append@node#1#2{%
5811   \ifforestdebugdynamics\forestdebug@dynamics{before appending #2 to #1}\fi
5812   \forest@dynamic@circularitytest{#2}{#1}{append}%
5813   \forest@node@Remove{#2}%
5814   \forest@node@Append{#1}{#2}%
5815 }
5816 \def\forest@prepend@node#1#2{%
5817   \ifforestdebugdynamics\forestdebug@dynamics{before prepending #2 to #1}\fi
5818   \forest@dynamic@circularitytest{#2}{#1}{prepend}%
5819   \forest@node@Remove{#2}%
5820   \forest@node@Prepend{#1}{#2}%
5821 }
5822 \def\forest@insertafter@node#1#2{%
5823   \ifforestdebugdynamics\forestdebug@dynamics{before inserting #2 after #1}\fi
5824   \forest@node@Remove{#2}%
5825   \forest@node@Insertafter{\forestOve{#1}{@parent}}{#2}{#1}%
5826 }
5827 \def\forest@insertbefore@node#1#2{%
5828   \ifforestdebugdynamics\forestdebug@dynamics{before inserting #2 before #1}\fi
5829   \forest@node@Remove{#2}%
5830   \forest@node@Insertbefore{\forestOve{#1}{@parent}}{#2}{#1}%
5831 }
5832 \def\forest@set@root#1#2{%
5833   \ifforestdebugdynamics\forestdebug@dynamics{before setting #1 as root}\fi
5834   \def\forest@root{#2}%
5835 }
5836 \def\forest@dynamic@circularitytest#1#2#3{%
5837   % #1=potential ancestor, #2=potential descendant, #3=message prefix
5838   \ifnum#1=#2
5839     \forest@circularityerror{#1}{#2}{#3}%
5840   \else
5841     \forest@fornode{#1}{%
5842       \forest@ifancestorof{#2}{\forest@circularityerror{#1}{#2}{#3}}{%
5843       }%
5844     }%
5845   }
5846 \def\forest@circularityerror#1#2#3{%

```



```

5847 \forestdebug@typeouttrees{\forest@temp}%
5848 \PackageError{forest}{#3ing node id=#1 to id=#2 would result in a circular tree\MessageBreak forest of ids:
5849 }%
5850 \def\forestdebug@dynamics#1{%
5851 \forestdebug@typeouttrees\forest@temp
5852 \typeout{#1: \forest@temp}%
5853 }
5854 \def\forest@appto@do@dynamics#1#2{%
5855 \forest@nodehandleby@name@nodewalk@or@bracket{#2}%
5856 \ifcase\forest@dynamics@copyhow\relax\or
5857 \forest@tree@copy{\forest@last@node}\forest@last@node
5858 \or
5859 \forest@node@copy{\forest@last@node}\forest@last@node
5860 \fi
5861 \forest@node@ifnamedefined{\forest@last@node}{%
5862 \forest@depreto{\forest@last@node}{delay}
5863 {for id={\forest@node@Nametoid{\forest@last@node}}{alias=\forest@last@node},}%
5864 }{}%
5865 \edef\forest@marshal{%
5866 \noexpand\apptotoks\noexpand\forest@do@dynamics{%
5867 \noexpand#1{\forest@cn}{\forest@last@node}}%
5868 }\forest@marshal
5869 }
5870 \forestset{%
5871 create/.code={%
5872 \forest@create@node{#1}%
5873 \forest@for@node{\forest@last@node}{%
5874 \forest@node@setalias{\forest@last@node}%
5875 \cseappto{\forest@nodewalk@saved@dynamic nodes}{\forest@last@node},}%
5876 }%
5877 },
5878 create'/.code={%
5879 \forest@create@node@and@process@given@options{#1}%
5880 \forest@for@node{\forest@last@node}{%
5881 \forest@node@setalias{\forest@last@node}%
5882 \cseappto{\forest@nodewalk@saved@dynamic nodes}{\forest@last@node},}%
5883 }%
5884 },
5885 append/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@append@node{#1}},
5886 prepend/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@prepend@node{#1}},
5887 insert after/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@insertafter@node{#1}},
5888 insert before/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@insertbefore@node{#1}},
5889 append'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@append@node{#1}},
5890 prepend'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@prepend@node{#1}},
5891 insert after'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@insertafter@node{#1}},
5892 insert before'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@insertbefore@node{#1}},
5893 append''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@append@node{#1}},
5894 prepend''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@prepend@node{#1}},
5895 insert after''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@insertafter@node{#1}},
5896 insert before''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@insertbefore@node{#1}},
5897 remove/.code={%
5898 \pgfkeysalso{alias=\forest@last@node}%
5899 \cseappto{\forest@nodewalk@saved@dynamic nodes}{\forest@cn},}%
5900 \expandafter\apptotoks\expandafter\forest@do@dynamics\expandafter{%
5901 \expandafter\forest@remove@node\expandafter{\forest@cn}}%
5902 },
5903 set root/.code={%
5904 \def\forest@dynamics@copyhow{0}%
5905 \forest@appto@do@dynamics\forest@set@root{#1}%
5906 },
5907 replace by/.code={\forest@replaceby@code{#1}{insert after}},

```

```

5908 replace by'/.code={\forest@replaceby@code{#1}{insert after'}},
5909 replace by'/.code={\forest@replaceby@code{#1}{insert after''}},
5910 sort/.code={%
5911   \eapptotoks\forest@do@ynamics{%
5912     \def\noexpand\forest@nodesort@sortkey{\forestrv{sort by}}%
5913     \noexpand\forest@nodesort\noexpand\forest@sort@ascending{\forest@cn}
5914   }%
5915 },
5916 sort'/.code={%
5917   \eapptotoks\forest@do@ynamics{%
5918     \def\noexpand\forest@nodesort@sortkey{\forestrv{sort by}}%
5919     \noexpand\forest@nodesort\noexpand\forest@sort@descending{\forest@cn}
5920   }%
5921 },
5922 }
5923 \def\forest@replaceby@code#1#2{%#1=node spec,#2=insert after['']}]
5924 \ifnum\forestove{@parent}=0
5925   \cseappto{\forest@nodewalk@saved@dynamic nodes}{\forest@cn,}%
5926   \pgfkeysalso{alias=forest@last@node,set root={#1}}%
5927 \else
5928   \cseappto{\forest@nodewalk@saved@dynamic nodes}{\forest@cn,}%
5929   \pgfkeysalso{alias=forest@last@node,#2={#1}}%
5930   \eapptotoks\forest@do@ynamics{%
5931     \noexpand\ifnum\noexpand\forestOve{\forest@cn}{@parent}=\forestove{@parent}
5932     \noexpand\forest@remove@node{\forest@cn}%
5933     \noexpand\fi
5934   }%
5935 \fi
5936 }
5937 \def\forest@nodesort#1#2{% #1 = direction, #2 = parent node
5938   \ifforestdebugdynamics\forestdebug@dynamics{before sorting children of #2}\fi
5939   \forest@for@node{#2}{\forest@nodesort@#1}%
5940   \ifforestdebugdynamics\forestdebug@dynamics{after sorting children of #2}\fi
5941 }
5942 \def\forest@nodesort@#1{%
5943   % prepare the array of child ids
5944   \c@pgf@counta=0
5945   \forest@toget{@first}\forest@nodesort@id
5946   \forest@loop
5947   \ifnum\forest@nodesort@id>0
5948     \advance\c@pgf@counta 1
5949     \csedef{\forest@nodesort@the\c@pgf@counta}{\forest@nodesort@id}%
5950     \forest@get{\forest@nodesort@id}{@next}\forest@nodesort@id
5951   \forest@repeat
5952   % sort
5953   \forest@toget{n children}\forest@nodesort@n
5954   \forest@sort\forest@nodesort@cmpnodes\forest@nodesort@let#1{1}{\forest@nodesort@n}%
5955   % remove all children
5956   \forest@toget{@first}\forest@nodesort@id
5957   \forest@loop
5958   \ifnum\forest@nodesort@id>0
5959     \forest@node@Remove{\forest@nodesort@id}%
5960     \forest@toget{@first}\forest@nodesort@id
5961   \forest@repeat
5962   % insert the children in new order
5963   \c@pgf@counta=0
5964   \forest@loop
5965   \ifnum\c@pgf@counta<\forest@nodesort@n\relax
5966     \advance\c@pgf@counta 1
5967     \forest@node@append{\csname forest@nodesort@the\c@pgf@counta\endcsname}%
5968   \forest@repeat

```

```

5969 }
5970 \def\forest@nodesort@cmpnodes#1#2{%
5971   \expandafter\forest@nodesort@cmpnodes@\forest@nodesort@sortkey,\forest@END{#1}{#2}%
5972 }
5973 \def\forest@nodesort@cmpnodes@#1,#2\forest@END#3#4{%
5974   % #1=process ins+arg for this dimension, #2=for next dimensions
5975   % #3, #4 = node ids
5976   {%
5977     \forest@fornode{\csname forest@nodesort@#3\endcsname}{%
5978       \forestmathsetmacro\forest@nodesort@resulta{#1}%
5979     }%
5980     \forest@fornode{\csname forest@nodesort@#4\endcsname}{%
5981       \forestmathsetmacro\forest@nodesort@resultb{#1}%
5982     }%
5983     \ifx\forestmathresulttype\forestmathtype@generic
5984       \forest@cmp@error{\forest@nodesort@resulta}{\forest@nodesort@resultb}%
5985     \fi
5986     \edef\forest@temp{%
5987       \noexpand\forest@nodesort@cmp
5988       {\expandonce{\forest@nodesort@resulta}}%
5989       {\expandonce{\forest@nodesort@resultb}}%
5990     }%
5991     \xdef\forest@global@temp{\forest@temp}%
5992   }%
5993   \if=\forest@global@temp
5994     \let\forest@next\forest@nodesort@cmpnodes@
5995   \else
5996     \let\forest@next\forest@nodesort@cmpnodes@finish
5997   \fi
5998   \ifstrempy{#2}{\let\forest@next\forest@nodesort@cmpnodes@finish}{}%
5999   \forest@next#2\forest@END{#3}{#4}%
6000 }
6001 \def\forest@nodesort@cmpnodes@finish#1\forest@END#2#3{%
6002   \let\forest@sort@cmp@result\forest@global@temp
6003 }

```

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

```

6004 \def\forest@nodesort@cmp{\csname fRsT@nsc@\forestmathresulttype\endcsname}
6005 \def\fRsT@nsc@#1{\csname fRsT@nsc@#1\endcsname}
6006 \def\fRsT@nsc@n#1#2{\ifnum#1<#2 <\else\ifnum#1=#2 =\else>\fi\fi}
6007 \def\fRsT@nsc@d#1#2{\ifdim#1<#2 <\else\ifdim#1=#2 =\else>\fi\fi}
6008 \def\fRsT@nsc@P#1#2{\ifdim#1pt<#2pt <\else\ifdim#1pt=#2pt =\else>\fi\fi}
6009 \def\fRsT@nsc@t#1#2{\csname fRsT@nsc@\pdfstrcmp{#1}{#2}\endcsname}
6010 \def\fRsT@nsc@T#1#2{\csname fRsT@nsc@\pdfstrcmp{#2}{#1}\endcsname}
6011 \csdef{fRsT@nsc@-1}{<}
6012 \csdef{fRsT@nsc@0}{=}
6013 \csdef{fRsT@nsc@1}{>}
6014 \def\forest@nodesort@let#1#2{%
6015   \csletcs{forest@nodesort@#1}{forest@nodesort@#2}%
6016 }
6017 \forestset{
6018   define long step={last dynamic node}{style,must start at valid node=false}{%
6019     name=forest@last@node
6020   }
6021 }

```

8 Stages

```

6022 \def\forest@root{0}
6023 %%% begin listing region: stages

```

```

6024 \forestset{
6025   stages/.style={
6026     for root'={
6027       process keylist register=default preamble,
6028       process keylist register=preamble
6029     },
6030     process keylist=given options,
6031     process keylist=before typesetting nodes,
6032     typeset nodes stage,
6033     process keylist=before packing,
6034     pack stage,
6035     process keylist=before computing xy,
6036     compute xy stage,
6037     process keylist=before drawing tree,
6038     draw tree stage
6039   },
6040   typeset nodes stage/.style={for root'=typeset nodes},
6041   pack stage/.style={for root'=pack},
6042   compute xy stage/.style={for root'=compute xy},
6043   draw tree stage/.style={for root'=draw tree},
6044 }
6045 %%% end listing region: stages
6046 \forestset{
6047   process keylist/.code={%
6048     \forest@process@hook@keylist{#1}{#1 processing order/.try,processing order/.lastretry}},
6049   process keylist'/.code 2 args={\forest@process@hook@keylist@nodynamics{#1}{#2}},
6050   process keylist''/.code 2 args={\forest@process@hook@keylist@{#1}{#2}},
6051   process keylist register/.code={\forest@process@keylist@register{#1}},
6052   process delayed/.code={%
6053     \forest@havedelayedoptions{@delay}{#1}%
6054     \ifforest@havedelayedoptions
6055       \forest@process@hook@keylist@nodynamics{@delay}{#1}%
6056     \fi
6057   },
6058   do dynamics/.code={%
6059     \the\forest@do@dynamics
6060     \forest@do@dynamics{}%
6061     \forest@node@Compute@numeric@ts@info{\forest@root}%
6062   },
6063   declare keylist={given options}{},
6064   declare keylist={before typesetting nodes}{},
6065   declare keylist={before packing}{},
6066   declare keylist={before packing node}{},
6067   declare keylist={after packing node}{},
6068   declare keylist={before computing xy}{},
6069   declare keylist={before drawing tree}{},
6070   declare keylist={delay}{},
6071   delay n/.code 2 args={%
6072     \forestmathsetcount\forest@temp@count{#1}%
6073     \pgfkeysalso{delay n'={\forest@temp@count}{#2}}%
6074   },
6075   delay n'/.code 2 args={
6076     \ifnum#1=0
6077       \pgfkeysalso{#2}%
6078     \else
6079       \pgfkeysalso{delay={delay n'/.expand once=\expandafter{\number\numexpr#1-1\relax}{#2}}}%
6080     \fi
6081   },
6082   if have delayed/.style 2 args={if have delayed'={processing order}{#1}{#2}},
6083   if have delayed'/.code n args=3{%
6084     \forest@havedelayedoptionsfalse

```

```

6085 \forest@forthis{%
6086 \forest@nodewalk{#1}{%
6087 TeX={%
6088 \forest@toget{delay}\forest@temp@delayed
6089 \ifdefempty\forest@temp@delayed{}\forest@havedelayedoptionstrue}%
6090 }%
6091 }%
6092 }%
6093 \ifforest@havedelayedoptions\pgfkeysalso{#2}\else\pgfkeysalso{#3}\fi
6094 },
6095 typeset nodes/.code={%
6096 \forest@drawtree@preservenodeboxes@false
6097 \forest@nodewalk
6098 {typeset nodes processing order/.try,processing order/.lastretry}%
6099 {TeX={\forest@node@typeset}}}%
6100 },
6101 typeset nodes'/.code={%
6102 \forest@drawtree@preservenodeboxes@true
6103 \forest@nodewalk
6104 {typeset nodes processing order/.try,processing order/.lastretry}%
6105 {TeX={\forest@node@typeset}}}%
6106 },
6107 typeset node/.code={%
6108 \forest@drawtree@preservenodeboxes@false
6109 \forest@node@typeset
6110 },
6111 pack/.code={\forest@pack},
6112 pack'/.code={\forest@pack@onlythisnode},
6113 compute xy/.code={\forest@node@computeabsolute positions},
6114 draw tree box/.store in=\forest@drawtreebox,
6115 draw tree box,
6116 draw tree/.code={%
6117 \forest@drawtree@preservenodeboxes@false
6118 \forest@node@drawtree
6119 },
6120 draw tree'/.code={%
6121 \forest@drawtree@preservenodeboxes@true
6122 \forest@node@drawtree
6123 },
6124 %%% begin listing region: draw_tree_method
6125 draw tree method/.style={
6126 for nodewalk={
6127 draw tree nodes processing order/.try,
6128 draw tree processing order/.retry,
6129 processing order/.lastretry
6130 }{draw tree node},
6131 for nodewalk={
6132 draw tree edges processing order/.try,
6133 draw tree processing order/.retry,
6134 processing order/.lastretry
6135 }{draw tree edge},
6136 for nodewalk={
6137 draw tree tikz processing order/.try,
6138 draw tree processing order/.retry,
6139 processing order/.lastretry
6140 }{draw tree tikz}
6141 },
6142 %%% end listing region: draw_tree_method
6143 draw tree node/.code={\forest@draw@node},
6144 draw tree node'/.code={\forest@draw@node@},
6145 if node drawn/.code n args={3}{%

```

```

6146 \forest@forthis{%
6147 \forest@configured@nodewalk{independent}{inherited}{fake}{#1}{}%
6148 \ifnum\forest@cn=0
6149 \forest@tempfalse
6150 \else
6151 \ifcsdef{forest@drawn@\forest@cn}{\forest@temptrue}{\forest@tempfalse}%
6152 \fi
6153 }%
6154 \ifforest@temp\pgfkeysalso{#2}\else\pgfkeysalso{#3}\fi
6155 },
6156 draw tree edge/.code={\forest@draw@edge},
6157 draw tree edge'/.code={\forest@draw@edge@},
6158 draw tree tikz/.code={\forest@draw@tikz@}, % always!
6159 draw tree tikz'/.code={\forest@draw@tikz@},
6160 processing order/.nodewalk style={tree},
6161 %given options processing order/.style={processing order},
6162 %before typesetting nodes processing order/.style={processing order},
6163 %before packing processing order/.style={processing order},
6164 %before computing xy processing order/.style={processing order},
6165 %before drawing tree processing order/.style={processing order},
6166 }
6167 \newtoks\forest@do@ynamics
6168 \newif\ifforest@havedelayedoptions
6169 \def\forest@process@hook@keylist#1#2{%#1=keylist,#2=processing order nodewalk
6170 \safeloop
6171 \forest@fornode{\forest@root}{\forest@process@hook@keylist@{#1}{#2}}%
6172 \expandafter\ifstremp\expandafter{\the\forest@do@ynamics}{}%
6173 \the\forest@do@ynamics
6174 \forest@do@ynamics={}%
6175 \forest@node@Compute@numeric@ts@info{\forest@root}%
6176 }%
6177 \forest@fornode{\forest@root}{\forest@havedelayedoptions{#1}{#2}}%
6178 \ifforest@havedelayedoptions
6179 \saferepeat
6180 }
6181 \def\forest@process@hook@keylist@nodynamics#1#2{%#1=keylist,#2=processing order nodewalk
6182 % note: this macro works on (nodewalk starting at) the current node
6183 \safeloop
6184 \forest@forthis{\forest@process@hook@keylist@{#1}{#2}}%
6185 \forest@havedelayedoptions{#1}{#2}%
6186 \ifforest@havedelayedoptions
6187 \saferepeat
6188 }
6189 \def\forest@process@hook@keylist@#1#2{%#1=keylist,#2=processing order nodewalk
6190 \forest@nodewalk{#2}{%
6191 TeX={%
6192 \forest@toget{#1}\forest@temp@keys
6193 \ifdefvoid\forest@temp@keys}{}%
6194 \forest@set{#1}{}%
6195 \expandafter\forest@set\expandafter{\forest@temp@keys}%
6196 }%
6197 }%
6198 }%
6199 }
6200 \def\forest@process@keylist@register#1{%
6201 \edef\forest@marshal{%
6202 \noexpand\forest@set{\forest@register{#1}}%
6203 }\forest@marshal
6204 }

```

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

```

6205 \def\forest@havedelayedoptions#1#2{%#1 = keylist, #2=nodewalk
6206   \forest@havedelayedoptionsfalse
6207   \forest@forthis{%
6208     \forest@nodewalk{#2}{%
6209       TeX={%
6210         \forestoget{delay}\forest@temp@delayed
6211         \ifdefempty\forest@temp@delayed{}\forest@havedelayedoptionstrue}%
6212         \forestolet{#1}\forest@temp@delayed
6213         \forestoset{delay}{}%
6214       }%
6215     }%
6216   }%
6217 }

```

8.1 Typesetting nodes

```

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

```

```

6264 \gdef\forest@smuggle{}%
6265 \setbox0=\hbox{%
6266   \begin{tikzpicture}[%
6267     /forest/copy command key={/tikz/anchor}/{/tikz/forest@orig@anchor},
6268     anchor/.style={%
6269       /forest/TeX={\forestanchortotikzanchor{##1}\forest@temp@anchor},
6270       forest@orig@anchor/.expand once=\forest@temp@anchor
6271     }%
6272     \pgfpositionnodelater{\forest@positionnodelater@save}%
6273     \forest@temp@nodeformat
6274     \pgfinterruptpath
6275     \pgfpntanchor{\forest@pgf@notyetpositioned\forest@nodename}{\forestcomputenodeboundary}%
6276     \endpgfinterruptpath
6277   \end{tikzpicture}}%
6278 }%
6279 \setbox\forestove{@box}=\box\forest@box % smuggle the box
6280 \forestolet{@boundary}\forest@global@boundary
6281 \forest@smuggle % ... and the rest
6282 }
6283
6284
6285 \forestset{
6286   declare readonly dimen={min x}{0pt},
6287   declare readonly dimen={min y}{0pt},
6288   declare readonly dimen={max x}{0pt},
6289   declare readonly dimen={max y}{0pt},
6290 }
6291 \def\forest@patch@enormouscoordinateboxbounds@plus#1{%
6292   \expandafter\ifstrequal\expandafter{#1}{16000.0pt}{\edef#1{0.0\pgfmath@pt}}{}}%
6293 }
6294 \def\forest@patch@enormouscoordinateboxbounds@minus#1{%
6295   \expandafter\ifstrequal\expandafter{#1}{-16000.0pt}{\edef#1{0.0\pgfmath@pt}}{}}%
6296 }
6297 \def\forest@positionnodelater@save{%
6298   \global\setbox\forest@box=\box\pgfpositionnodelaterbox
6299   \xappto\forest@smuggle{\noexpand\forestoset{later@name}{\pgfpositionnodelatername}}%
6300   % a bug in pgf? ---well, here's a patch
6301   \forest@patch@enormouscoordinateboxbounds@plus\pgfpositionnodelaterminx
6302   \forest@patch@enormouscoordinateboxbounds@plus\pgfpositionnodelaterminy
6303   \forest@patch@enormouscoordinateboxbounds@minus\pgfpositionnodelatermaxx
6304   \forest@patch@enormouscoordinateboxbounds@minus\pgfpositionnodelatermaxy
6305   % end of patch
6306   \xappto\forest@smuggle{\noexpand\forestoset{min x}{\pgfpositionnodelaterminx}}%
6307   \xappto\forest@smuggle{\noexpand\forestoset{min y}{\pgfpositionnodelaterminy}}%
6308   \xappto\forest@smuggle{\noexpand\forestoset{max x}{\pgfpositionnodelatermaxx}}%
6309   \xappto\forest@smuggle{\noexpand\forestoset{max y}{\pgfpositionnodelatermaxy}}%
6310 }
6311 \def\forest@node@forest@positionnodelater@restore{%
6312   \ifforest@drawtree@preservenodeboxes@
6313     \let\forest@boxorcopy\copy
6314   \else
6315     \let\forest@boxorcopy\box
6316   \fi
6317   \forestoget{@box}\forest@temp
6318   \setbox\pgfpositionnodelaterbox=\forest@boxorcopy\forest@temp
6319   \edef\pgfpositionnodelatername{\forestove{later@name}}%
6320   \edef\pgfpositionnodelaterminx{\forestove{min x}}%
6321   \edef\pgfpositionnodelaterminy{\forestove{min y}}%
6322   \edef\pgfpositionnodelatermaxx{\forestove{max x}}%
6323   \edef\pgfpositionnodelatermaxy{\forestove{max y}}%
6324   \ifforest@drawtree@preservenodeboxes@

```



```

6325 \else
6326   \forestset{@box}{}%
6327 \fi
6328 }

```

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.

```

6329 \def\forest@pack{%
6330   \pgfsyssoftpath@getcurrentpath\forest@pack@original@path
6331   \forest@pack@computetiers
6332   \forest@pack@computegrowthuniformity
6333   \forest@@pack
6334   \pgfsyssoftpath@setcurrentpath\forest@pack@original@path
6335 }
6336 \def\forest@@pack{%
6337   \ifnum\forest@ve{uniform growth}>0
6338     \ifnum\forest@ve{n children}>0
6339       \forest@pack@level@uniform
6340       \forest@pack@aligntiers@ofsubtree
6341       \forest@pack@sibling@uniform@recursive
6342     \fi
6343   \else
6344     \forest@node@foreachchild{\forest@@pack}%
6345     \forest@process@hook@keylist@nodynamics{before packing node}{current}%
6346     \ifnum\forest@ve{n children}>0
6347       \forest@pack@level@nonuniform
6348       \forest@pack@aligntiers
6349       \forest@pack@sibling@uniform@applyreversed
6350     \fi
6351     \forest@get{after packing node}\forest@temp@keys
6352     \forest@process@hook@keylist@nodynamics{after packing node}{current}%
6353   \fi
6354 }
6355 % \forestset{recalculate tree boundary/.code={\forest@node@recalculate@edges}}
6356 % \def\forest@node@recalculate@edges{%
6357 %   \edef\forest@marshal{%
6358 %     \noexpand\forest@forthis{\noexpand\forest@node@getedges{\forest@ve{grow}}}%
6359 %   }\forest@marshal
6360 % }
6361 \def\forest@pack@onlythisnode{%
6362   \ifnum\forest@ve{n children}>0
6363     \forest@pack@computetiers
6364     \forest@pack@level@nonuniform
6365     \forest@pack@aligntiers
6366     \forest@node@foreachchild{\forest@set{s}{0\pgfmath@pt}}%
6367     \forest@pack@sibling@uniform@applyreversed
6368   \fi
6369 }

```

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

```

6370 \def\forest@pack@computegrowthuniformity{%
6371   \forest@node@foreachchild{\forest@pack@computegrowthuniformity}%
6372   \edef\forest@pack@cgu@uniformity{%
6373     \ifnum\forest@ve{n children}=0
6374       2\else 1\fi
6375   }%
6376   \forest@get{grow}\forest@pack@cgu@parentgrow
6377   \forest@node@foreachchild{%

```

```

6378 \ifnum\forestove{uniform growth}=0
6379 \def\forest@pack@cgu@uniformity{0}%
6380 \else
6381 \ifnum\forestove{uniform growth}=1
6382 \ifnum\forestove{grow}=\forest@pack@cgu@parentgrow\relax\else
6383 \def\forest@pack@cgu@uniformity{0}%
6384 \fi
6385 \fi
6386 \fi
6387 }%
6388 \forestoget{before packing node}\forest@temp@a
6389 \forestoget{after packing node}\forest@temp@b
6390 \expandafter\expandafter\expandafter\ifstrempy\expandafter\expandafter\expandafter{\expandafter\forest@temp@b}
6391 \forestolet{uniform growth}\forest@pack@cgu@uniformity
6392 }{%
6393 \forestoset{uniform growth}{0}%
6394 }%
6395 }

```

Pack children in the level dimension in a uniform tree.

```

6396 \def\forest@pack@level@uniform{%
6397 \let\forest@plu@minchildl\relax
6398 \forestoget{grow}\forest@plu@grow
6399 \forest@node@foreachchild{%
6400 \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
6401 \advance\pgf@xa\forestove{1}\relax
6402 \ifx\forest@plu@minchildl\relax
6403 \edef\forest@plu@minchildl{\the\pgf@xa}%
6404 \else
6405 \ifdim\pgf@xa<\forest@plu@minchildl\relax
6406 \edef\forest@plu@minchildl{\the\pgf@xa}%
6407 \fi
6408 \fi
6409 }%
6410 \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
6411 \pgfutil@tempdima=\pgf@xb\relax
6412 \advance\pgfutil@tempdima-\forest@plu@minchildl\relax
6413 \advance\pgfutil@tempdima\forestove{1 sep}\relax
6414 \ifdim\pgfutil@tempdima>0pt
6415 \forest@node@foreachchild{%
6416 \forestoeset{1}{\the\dimexpr\forestove{1}+\the\pgfutil@tempdima}%
6417 }%
6418 \fi
6419 \forest@node@foreachchild{%
6420 \ifnum\forestove{n children}>0
6421 \forest@pack@level@uniform
6422 \fi
6423 }%
6424 }

```

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

```

6425 \def\forest@pack@level@nonuniform{%
6426 \let\forest@plu@minchildl\relax
6427 \forestoget{grow}\forest@plu@grow
6428 \forest@node@foreachchild{%
6429 \forest@node@getedge{negative}{\forest@plu@grow}{\forest@plnu@negativechildedge}%
6430 \forest@node@getedge{positive}{\forest@plu@grow}{\forest@plnu@positivechildedge}%
6431 \def\forest@plnu@childedge{\forest@plnu@negativechildedge\forest@plnu@positivechildedge}%
6432 \forest@path@getboundingrectangle@ls\forest@plnu@childedge{\forest@plu@grow}%
6433 \advance\pgf@xa\forestove{1}\relax
6434 \ifx\forest@plu@minchildl\relax
6435 \edef\forest@plu@minchildl{\the\pgf@xa}%

```

```

6436 \else
6437 \ifdim\pgf@xa<\forest@plu@minchildl\relax
6438 \edef\forest@plu@minchildl{\the\pgf@xa}%
6439 \fi
6440 \fi
6441 }%
6442 \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
6443 \pgfutil@tempdima=\pgf@xb\relax
6444 \advance\pgfutil@tempdima -\forest@plu@minchildl\relax
6445 \advance\pgfutil@tempdima \forestove{1 sep}\relax
6446 \ifdim\pgfutil@tempdima>0pt
6447 \forest@node@foreachchild{%
6448 \forestoeset{1}{\the\dimexpr\the\pgfutil@tempdima+\forestove{1}}%
6449 }%
6450 \fi
6451 }

Align tiers.
6452 \def\forest@pack@aligntiers{%
6453 \forestoget{grow}\forest@temp@parentgrow
6454 \forestoget{@tiers}\forest@temp@tiers
6455 \forlistloop\forest@pack@aligntier@\forest@temp@tiers
6456 }
6457 \def\forest@pack@aligntiers@ofsubtree{%
6458 \forest@node@foreach{\forest@pack@aligntiers}%
6459 }
6460 \def\forest@pack@aligntiers@computeabsl{%
6461 \forestoleto{abs@1}{1}%
6462 \forest@node@foreachdescendant{\forest@pack@aligntiers@computeabsl}%
6463 }
6464 \def\forest@pack@aligntiers@computeabsl@{%
6465 \forestoeset{abs@1}{\the\dimexpr\forestove{1}+\forestove{\forestove{@parent}}{abs@1}}%
6466 }
6467 \def\forest@pack@aligntier@#1{%
6468 \forest@pack@aligntiers@computeabsl
6469 \pgfutil@tempdima=-\maxdimen\relax
6470 \def\forest@temp@currenttier{#1}%
6471 \forest@node@foreach{%
6472 \forestoget{tier}\forest@temp@tier
6473 \ifx\forest@temp@currenttier\forest@temp@tier
6474 \ifdim\pgfutil@tempdima<\forestove{abs@1}\relax
6475 \pgfutil@tempdima=\forestove{abs@1}\relax
6476 \fi
6477 \fi
6478 }%
6479 \ifdim\pgfutil@tempdima=-\maxdimen\relax\else
6480 \forest@node@foreach{%
6481 \forestoget{tier}\forest@temp@tier
6482 \ifx\forest@temp@currenttier\forest@temp@tier
6483 \forestoeset{1}{\the\dimexpr\pgfutil@tempdima-\forestove{abs@1}+\forestove{1}}%
6484 \fi
6485 }%
6486 \fi
6487 }

```

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

```

6488 \def\forest@pack@sibling@uniform@recursive{%
6489 \forest@node@foreachchild{\forest@pack@sibling@uniform@recursive}%
6490 \forest@pack@sibling@uniform@applyreversed
6491 }

```

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

```

6492 \def\forest@pack@sibling@uniform@applyreversed{%
6493   \ifnum\forestove{n children}>1
6494     \ifnum\forestove{reversed}=0
6495       \forest@pack@sibling@uniform@main{first}{last}{next}{previous}%
6496     \else
6497       \forest@pack@sibling@uniform@main{last}{first}{previous}{next}%
6498     \fi
6499   \else
6500     \ifnum\forestove{n children}=1

```

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

```

6501     \csname forest@calign@\forestove{calign}\endcsname
6502   \fi
6503 \fi
6504 }

```

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

```

6505 \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.

```

6506   \forestoget{@#1}\forest@child
6507   \edef\forest@marshal{%
6508     \noexpand\forest@fornode{\forestove{@#1}}{%
6509       \noexpand\forest@node@getedge
6510         {positive}%
6511       {\forestove{grow}}}%
6512     \noexpand\forest@temp@edge
6513   }%
6514   }\forest@marshal
6515   \forest@pack@pgfpoint@childposition\forest@child
6516   \let\forest@previous@positive@edge\pgfutil@empty
6517   \forest@extendpath\forest@previous@positive@edge\forest@temp@edge{%
6518     \forestOget{\forest@child}{@#3}\forest@child

```

Loop until the current child is the null node.

```

6519   \edef\forest@previous@child@s{0\pgfmath@pt}%
6520   \safeloop
6521   \unless\ifnum\forest@child=0

```

Get the negative edge of the child.

```

6522     \edef\forest@temp{%
6523       \noexpand\forest@fornode{\forest@child}{%
6524         \noexpand\forest@node@getedge
6525           {negative}%
6526         {\forestove{grow}}}%
6527       \noexpand\forest@temp@edge
6528     }%
6529     }\forest@temp

```

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

```

6530     \forest@pack@pgfpoint@childposition\forest@child

```

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

```

6531     \let\forest@child@negative@edge\pgfutil@empty
6532     \forest@extendpath\forest@child@negative@edge\forest@temp@edge{%

```

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

```

6533     \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!)

```
6534 \forest@distance@between@edge@paths\forest@previous@positive@edge\forest@child@negative@edge\forest@csdis
```

If the distance is `\relax`, the projections of the edges onto the grow line don't overlap: do nothing.

Otherwise, shift the current child so that its distance to the block of previous children is `s_sep`.

```
6535 \ifx\forest@csdistance\relax
6536 %\forest@reset{\forest@child}{s}{\forest@previous@child@s}%
6537 \else
6538 \advance\pgfutil@tempdimb-\forest@csdistance\relax
6539 \advance\pgfutil@tempdimb\forestove{s_sep}\relax
6540 \forest@reset{\forest@child}{s}{\the\dimexpr\forestove{\forest@child}{s}-\forest@csdistance+\forestove{s}
6541 \fi
```

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

```
6542 \ifdim\forestove{\forest@child}{s}<\forest@previous@child@s\relax
6543 \forest@reset{\forest@child}{s}{\forest@previous@child@s}%
6544 \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.

```
6545 \forest@get{\forest@child}{s}\forest@child@s
6546 \edef\forest@previous@child@s{\forest@child@s}%
6547 \edef\forest@temp{%
6548 \noexpand\forest@for@node{\forest@child}{%
6549 \noexpand\forest@node@get@edge
6550 {positive}%
6551 {\forestove{grow}}}%
6552 \noexpand\forest@temp@edge
6553 }%
6554 }\forest@temp
6555 \forest@pack@pgfpoint@child@position\forest@child
6556 \forest@extend@path\forest@previous@positive@edge\forest@temp@edge{}%
6557 \forest@get@positive@tight@edge@of@path\forest@previous@positive@edge\forest@previous@positive@edge
6558 \forest@get{\forest@child}{#3}\forest@child
6559 \saferepeat
```

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

```
6560 \csname forest@calign@\forestove{calign}\endcsname
6561 }
```

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

```
6562 \def\forest@pack@pgfpoint@child@position#1{%
6563 {%
6564 \pgftransformreset
6565 \forest@pgfqtransformrotate{\forestove{grow}}%
6566 \forest@for@node{#1}{%
6567 \pgfpointransformed{\pgfqpoint{\forestove{1}}{\forestove{s}}}%
6568 }%
6569 }%
6570 }
```

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

```
6571 \def\forest@pack@pgfpoint@positioningrow#1{%
6572 {%
6573 \pgftransformreset
6574 \forest@pgfqtransformrotate{#1}%
6575 \pgfpointransformed{\pgfqpoint{\forestove{1}}{\forestove{s}}}%
6576 }%
6577 }
```

Child alignment.

```
6578 \def\forest@calign@s@shift#1{%
```

```

6579 \pgfutil@tempdima=#1\relax
6580 \forest@node@foreachchild{%
6581   \forestoeset{s}{\the\dimexpr\forestove{s}+\pgfutil@tempdima}%
6582 }%
6583 }
6584 \def\forest@calign@child{%
6585   \forest@calign@s@shift{-\forestOve{\forest@node@nornbarthchildid{\forestove{calign primary child}}}{s}}%
6586 }
6587 \csdef{forest@calign@child edge}{%
6588   {%
6589     \edef\forest@temp@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
6590     \pgftransformreset
6591     \forest@pgfqtransformrotate{\forestove{grow}}%
6592     \pgfpointrtransformed{\pgfqpoint{\forestOve{\forest@temp@child}{1}}{\forestOve{\forest@temp@child}{s}}}%
6593     \pgf@xa=\pgf@x\relax\pgf@ya=\pgf@y\relax
6594     \forest@Pointanchor{\forest@temp@child}{child anchor}%
6595     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
6596     \forest@pointanchor{parent anchor}%
6597     \advance\pgf@xa-\pgf@x\relax\advance\pgf@ya-\pgf@y\relax
6598     \edef\forest@marshal{%
6599       \noexpand\pgftransformreset
6600       \noexpand\forest@pgfqtransformrotate{-\forestove{grow}}%
6601       \noexpand\pgfpointrtransformed{\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}%
6602     }\forest@marshal
6603   }%
6604   \forest@calign@s@shift{\the\dimexpr-\the\pgf@y}%
6605 }
6606 \csdef{forest@calign@midpoint}{%
6607   \forest@calign@s@shift{\the\dimexpr Opt -%
6608     (\forestOve{\forest@node@nornbarthchildid{\forestove{calign primary child}}}{s}%
6609     +\forestOve{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}{s}%
6610     )/2\relax
6611   }%
6612 }
6613 \csdef{forest@calign@edge midpoint}{%
6614   {%
6615     \edef\forest@temp@firstchild{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
6616     \edef\forest@temp@secondchild{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
6617     \pgftransformreset
6618     \forest@pgfqtransformrotate{\forestove{grow}}%
6619     \pgfpointrtransformed{\pgfqpoint{\forestOve{\forest@temp@firstchild}{1}}{\forestOve{\forest@temp@firstchild}{s}}}%
6620     \pgf@xa=\pgf@x\relax\pgf@ya=\pgf@y\relax
6621     \forest@Pointanchor{\forest@temp@firstchild}{child anchor}%
6622     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
6623     \edef\forest@marshal{%
6624       \noexpand\pgfpointrtransformed{\noexpand\pgfqpoint{\forestOve{\forest@temp@secondchild}{1}}{\forestOve{\forest@temp@secondchild}{s}}}%
6625     }\forest@marshal
6626     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
6627     \forest@Pointanchor{\forest@temp@secondchild}{child anchor}%
6628     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
6629     \divide\pgf@xa2 \divide\pgf@ya2
6630     \forest@pointanchor{parent anchor}%
6631     \advance\pgf@xa-\pgf@x\relax\advance\pgf@ya-\pgf@y\relax
6632     \edef\forest@marshal{%
6633       \noexpand\pgftransformreset
6634       \noexpand\forest@pgfqtransformrotate{-\forestove{grow}}%
6635       \noexpand\pgfpointrtransformed{\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}%
6636     }\forest@marshal
6637   }%
6638   \forest@calign@s@shift{\the\dimexpr-\the\pgf@y}%
6639 }

```

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.

```

6640 \def\forest@edef@strippt#1#2{%
6641   \edef#1{#2}%
6642   \edef#1{\expandafter\Pgf@geT#1}%
6643 }
6644 \csdef{forest@calign@fixed angles}{%
6645   \ifnum\forest@ve{n children}>1
6646     \edef\forest@ca@first@child{\forest@node@nornbarthchildid{\forest@ve{calign primary child}}}%
6647     \edef\forest@ca@second@child{\forest@node@nornbarthchildid{\forest@ve{calign secondary child}}}%
6648     \ifnum\forest@ve{reversed}=1
6649       \let\forest@temp\forest@ca@first@child
6650       \let\forest@ca@first@child\forest@ca@second@child
6651       \let\forest@ca@second@child\forest@temp
6652     \fi
6653     \forest@get{\forest@ca@first@child}{l}\forest@ca@first@l
6654     \edef\forest@ca@first@l{\expandafter\Pgf@geT\forest@ca@first@l}%
6655     \forest@get{\forest@ca@second@child}{l}\forest@ca@second@l
6656     \edef\forest@ca@second@l{\expandafter\Pgf@geT\forest@ca@second@l}%
6657     \pgfmathatan@{\forest@ve{calign secondary angle}}%
6658     \pgfmathmultiply@{\pgfmathresult}{\forest@ca@second@l}%
6659     \let\forest@calign@temp\pgfmathresult
6660     \pgfmathatan@{\forest@ve{calign primary angle}}%
6661     \pgfmathmultiply@{\pgfmathresult}{\forest@ca@first@l}%
6662     \edef\forest@ca@desired@s@distance{\the\dimexpr
6663       \forest@calign@temp pt-\pgfmathresult pt}%
6664     % \pgfmathsetlengthmacro\forest@ca@desired@s@distance{%
6665     %   tan(\forest@ve{calign secondary angle})*\forest@ca@second@l
6666     %   -tan(\forest@ve{calign primary angle})*\forest@ca@first@l
6667     % }%
6668     \forest@get{\forest@ca@first@child}{s}\forest@ca@first@s
6669     \forest@get{\forest@ca@second@child}{s}\forest@ca@second@s
6670     \edef\forest@ca@actual@s@distance{\the\dimexpr
6671       \forest@ca@second@s-\forest@ca@first@s}%
6672     % \pgfmathsetlengthmacro\forest@ca@actual@s@distance{%
6673     %   \forest@ca@second@s-\forest@ca@first@s}%
6674     \ifdim\forest@ca@desired@s@distance>\forest@ca@actual@s@distance\relax
6675       \ifdim\forest@ca@actual@s@distance=0pt
6676         \pgfmathatan@{\forest@ve{calign primary angle}}%
6677         \pgfmathmultiply@{\pgfmathresult}{\forest@ca@second@l}%
6678         \pgfutil@tempdima=\pgfmathresult pt
6679         % \pgfmathsetlength\pgfutil@tempdima{tan(\forest@ve{calign primary angle})*\forest@ca@second@l}%
6680         \pgfutil@tempdimb=\dimexpr
6681         \forest@ca@desired@s@distance/(\forest@ve{n children}-1)\relax%
6682         %\pgfmathsetlength\pgfutil@tempdimb{\forest@ca@desired@s@distance/(\forest@ve{n children}-1)}%
6683         \forest@node@foreachchild{%
6684           \forest@set{s}{\the\pgfutil@tempdima}%
6685           \advance\pgfutil@tempdima\pgfutil@tempdimb
6686         }%
6687         \def\forest@calign@anchor{0pt}%
6688       \else
6689         \edef\forest@marshal{\noexpand\pgfmathdivide@
6690           {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
6691           {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
6692         }\forest@marshal
6693         \let\forest@ca@ratio\pgfmathresult
6694         %\pgfmathsetmacro\forest@ca@ratio{%
6695         %   \forest@ca@desired@s@distance/\forest@ca@actual@s@distance}%

```

```

6696 \forest@node@foreachchild{%
6697 \forest@edef@striipt\forest@temp{\forestove{s}}%
6698 \pgfmathmultiply@{\forest@ca@ratio}{\forest@temp}%
6699 \forestoeset{s}{\the\dimexpr\pgfmathresult pt}%
6700 %\pgfmathsetlengthmacro\forest@temp{\forest@ca@ratio*\forestove{s}}%
6701 %\forestolet{s}\forest@temp
6702 }%
6703 \pgfmathatan@{\forestove{calign primary angle}}%
6704 \pgfmathmultiply@{\pgfmathresult}{\forest@ca@first@l}%
6705 \edef\forest@calign@anchor{\the\dimexpr-\pgfmathresult pt}%
6706 %\pgfmathsetlengthmacro\forest@calign@anchor{%
6707 % -tan(\forestove{calign primary angle})*\forest@ca@first@l}%
6708 \fi
6709 \else
6710 \ifdim\forest@ca@desired@s@distance<\forest@ca@actual@s@distance\relax
6711 \edef\forest@marshal{\noexpand\pgfmathdivide@
6712 {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
6713 {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
6714 }\forest@marshal
6715 \let\forest@ca@ratio\pgfmathresult
6716 %\pgfmathsetlengthmacro\forest@ca@ratio{%
6717 % \forest@ca@actual@s@distance/\forest@ca@desired@s@distance}%
6718 \forest@node@foreachchild{%
6719 \forest@edef@striipt\forest@temp{\forestove{l}}%
6720 \pgfmathmultiply@{\forest@ca@ratio}{\forest@temp}%
6721 \forestoeset{l}{\the\dimexpr\pgfmathresult pt}%
6722 %\pgfmathsetlengthmacro\forest@temp{\forest@ca@ratio*\forestove{l}}%
6723 %\forestolet{l}\forest@temp
6724 }%
6725 \forest@get{\forest@ca@first@child}{l}\forest@ca@first@l
6726 \edef\forest@ca@first@l{\expandafter\Pgf@geT\forest@ca@first@l}%
6727 \pgfmathatan@{\forestove{calign primary angle}}%
6728 \pgfmathmultiply@{\pgfmathresult}{\forest@ca@first@l}%
6729 \edef\forest@calign@anchor{\the\dimexpr-\pgfmathresult pt}%
6730 %\pgfmathsetlengthmacro\forest@calign@anchor{%
6731 % -tan(\forestove{calign primary angle})*\forest@ca@first@l}%
6732 \fi
6733 \fi
6734 \forest@calign@s@shift{-\forest@calign@anchor}%
6735 \fi
6736 }
6737 \csdef\forest@calign@fixed edge angles{%
6738 \ifnum\forestove{n children}>1
6739 \edef\forest@ca@first@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
6740 \edef\forest@ca@second@child{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
6741 \ifnum\forestove{reversed}=1
6742 \let\forest@temp\forest@ca@first@child
6743 \let\forest@ca@first@child\forest@ca@second@child
6744 \let\forest@ca@second@child\forest@temp
6745 \fi
6746 \forest@get{\forest@ca@first@child}{l}\forest@ca@first@l
6747 \forest@get{\forest@ca@second@child}{l}\forest@ca@second@l
6748 \forest@pointanchor{parent anchor}%
6749 \edef\forest@ca@parent@anchor@s{\the\pgf@x}%
6750 \edef\forest@ca@parent@anchor@l{\the\pgf@y}%
6751 \forest@pointanchor{\forest@ca@first@child}{child anchor}%
6752 \edef\forest@ca@first@child@anchor@s{\the\pgf@x}%
6753 \edef\forest@ca@first@child@anchor@l{\the\pgf@y}%
6754 \forest@pointanchor{\forest@ca@second@child}{child anchor}%
6755 \edef\forest@ca@second@child@anchor@s{\the\pgf@x}%
6756 \edef\forest@ca@second@child@anchor@l{\the\pgf@y}%

```



```

6757 \pgfmathatan@{\forestove{calign secondary angle}}%
6758 \edef\forest@temp{\the\dimexpr
6759 \forest@ca@second@l-\forest@ca@second@child@anchor@l+\forest@ca@parent@anchor@l}%
6760 \pgfmathmultiply@{\pgfmathresult}{\expandafter\Pgf@geT\forest@temp}%
6761 \edef\forest@ca@desired@second@edge@s{\the\dimexpr\pgfmathresult pt}%
6762 %\pgfmathsetlengthmacro\forest@ca@desired@second@edge@s{%
6763 % \tan(\forestove{calign secondary angle})*%
6764 % (\forest@ca@second@l-\forest@ca@second@child@anchor@l+\forest@ca@parent@anchor@l)}%
6765 \pgfmathatan@{\forestove{calign primary angle}}%
6766 \edef\forest@temp{\the\dimexpr
6767 \forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@l}%
6768 \pgfmathmultiply@{\pgfmathresult}{\expandafter\Pgf@geT\forest@temp}%
6769 \edef\forest@ca@desired@first@edge@s{\the\dimexpr\pgfmathresult pt}%
6770 %\pgfmathsetlengthmacro\forest@ca@desired@first@edge@s{%
6771 % \tan(\forestove{calign primary angle})*%
6772 % (\forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@l)}%
6773 \edef\forest@ca@desired@s@distance{\the\dimexpr
6774 \forest@ca@desired@second@edge@s-\forest@ca@desired@first@edge@s}%
6775 %\pgfmathsetlengthmacro\forest@ca@desired@s@distance{\forest@ca@desired@second@edge@s-\forest@ca@desired@
6776 \forest@lget{\forest@ca@first@child}{s}\forest@ca@first@s
6777 \forest@lget{\forest@ca@second@child}{s}\forest@ca@second@s
6778 \edef\forest@ca@actual@s@distance{\the\dimexpr
6779 \forest@ca@second@s+\forest@ca@second@child@anchor@s
6780 -\forest@ca@first@s-\forest@ca@first@child@anchor@s}%
6781 %\pgfmathsetlengthmacro\forest@ca@actual@s@distance{%
6782 % \forest@ca@second@s+\forest@ca@second@child@anchor@s
6783 % -\forest@ca@first@s-\forest@ca@first@child@anchor@s}%
6784 \ifdim\forest@ca@desired@s@distance>\forest@ca@actual@s@distance\relax
6785 \ifdim\forest@ca@actual@s@distance=0pt
6786 \forest@toget{n children}\forest@temp@n@children
6787 \forest@node@foreachchild{%
6788 \forest@pointanchor{child anchor}%
6789 \edef\forest@temp@child@anchor@s{\the\pgf@x}%
6790 \forest@oaset{s}{\the\dimexpr
6791 \forest@ca@desired@first@edge@s+\forest@ca@desired@s@distance*(\forestove{n}-1)/(\forest@temp@n@
6792 %\pgfmathsetlengthmacro\forest@temp{%
6793 % \forest@ca@desired@first@edge@s+(\forestove{n}-1)*\forest@ca@desired@s@distance/(\forest@temp@n@
6794 %\forest@oaset{s}\forest@temp
6795 }%
6796 \def\forest@calign@anchor{0pt}%
6797 \else
6798 \edef\forest@marshal{\noexpand\pgfmathdivide@
6799 {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
6800 {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
6801 }\forest@marshal
6802 \let\forest@ca@ratio\pgfmathresult
6803 %\pgfmathsetmacro\forest@ca@ratio{%
6804 % \forest@ca@desired@s@distance/\forest@ca@actual@s@distance}%
6805 \forest@node@foreachchild{%
6806 \forest@pointanchor{child anchor}%
6807 \edef\forest@temp@child@anchor@s{\the\pgf@x}%
6808 \edef\forest@marshal{\noexpand\pgfmathmultiply@
6809 {\forest@ca@ratio}%
6810 {\expandafter\Pgf@geT\the\dimexpr
6811 \forestove{s}-\forest@ca@first@s+
6812 \forest@temp@child@anchor@s-\forest@ca@first@child@anchor@s}%
6813 }\forest@marshal
6814 \forest@oaset{s}{\the\dimexpr\pgfmathresult pt+\forest@ca@first@s
6815 +\forest@ca@first@child@anchor@s-\forest@temp@child@anchor@s}%
6816 % \pgfmathsetlengthmacro\forest@temp{%
6817 % \forest@ca@ratio*(%

```

```

6818 % \forestove{s}-\forest@ca@first@s
6819 % +\forest@temp@child@anchor@s-\forest@ca@first@child@anchor@s}%
6820 % +\forest@ca@first@s
6821 % +\forest@ca@first@child@anchor@s-\forest@temp@child@anchor@s}%
6822 % \forestolet{s}\forest@temp
6823 }%
6824 \pgfmathatan@{\forestove{calign primary angle}}%
6825 \edef\forest@marshal{\noexpand\pgfmathmultiply@
6826 {\pgfmathresult}%
6827 {\expandafter\Pgf@geT\the\dimexpr
6828 \forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@l}%
6829 }\forest@marshal
6830 \edef\forest@calign@anchor{\the\dimexpr
6831 -\pgfmathresult pt+\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s}%
6832 % \pgfmathsetlengthmacro\forest@calign@anchor{%
6833 % -tan(\forestove{calign primary angle})*(\forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@s)
6834 % +\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s
6835 % }%
6836 \fi
6837 \else
6838 \ifdim\forest@ca@desired@s@distance<\forest@ca@actual@s@distance\relax
6839 \edef\forest@marshal{\noexpand\pgfmathdivide@
6840 {\expandafter\Pgf@geT\forest@ca@actual@s@distance}%
6841 {\expandafter\Pgf@geT\forest@ca@desired@s@distance}%
6842 }\forest@marshal
6843 \let\forest@ca@ratio\pgfmathresult
6844 %\pgfmathsetlengthmacro\forest@ca@ratio{%
6845 % \forest@ca@actual@s@distance/\forest@ca@desired@s@distance}%
6846 \forest@node@foreachchild{%
6847 \forest@pointanchor{child anchor}%
6848 \edef\forest@temp@child@anchor@l{\the\pgf@y}%
6849 \edef\forest@marshal{\noexpand\pgfmathmultiply@
6850 {\forest@ca@ratio}%
6851 {\expandafter\Pgf@geT\the\dimexpr\forestove{l}+\forest@ca@parent@anchor@l-\forest@temp@child@anchor@l}%
6852 }\forest@marshal
6853 \forestoeset{l}{\the\dimexpr
6854 \pgfmathresult pt-\forest@ca@parent@anchor@l+\forest@temp@child@anchor@l}%
6855 % \pgfmathsetlengthmacro\forest@temp@l{%
6856 % \forest@ca@ratio*(%
6857 % \forestove{l}+\forest@ca@parent@anchor@l-\forest@temp@child@anchor@l)
6858 % -\forest@ca@parent@anchor@l+\forest@temp@child@anchor@l}%
6859 % \forestolet{l}\forest@temp
6860 }%
6861 \forest@get{\forest@ca@first@child}{l}\forest@ca@first@l
6862 \pgfmathatan@{\forestove{calign primary angle}}%
6863 \edef\forest@marshal{\noexpand\pgfmathmultiply@
6864 {\pgfmathresult}%
6865 {\expandafter\Pgf@geT\the\dimexpr
6866 \forest@ca@first@l+\forest@ca@parent@anchor@l-\forest@temp@child@anchor@l}%
6867 }\forest@marshal
6868 \edef\forest@calign@anchor{\the\dimexpr
6869 -\pgfmathresult pt+\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s}%
6870 % \pgfmathsetlengthmacro\forest@calign@anchor{%
6871 % -tan(\forestove{calign primary angle})*(\forest@ca@first@l+\forest@ca@parent@anchor@l-\forest@temp@child@anchor@l)
6872 % +\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s
6873 % }%
6874 \fi
6875 \fi
6876 \forest@calign@s@shift{-\forest@calign@anchor}%
6877 \fi
6878 }

```

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.

```

6879 \def\forest@node@getedge#1#2#3{%
6880   \forestoget{#1@edge@#2}#3%
6881   \ifx#3\relax
6882     \forest@node@foreachchild{%
6883       \forest@node@getedge{#1}{#2}{\forest@temp@edge}%
6884     }%
6885     \forest@forthis{\forest@node@getedges{#2}}%
6886     \forestoget{#1@edge@#2}#3%
6887   \fi
6888 }

```

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.

```

6889 \def\forest@node@getedges#1{%

```

Run the computation in a \TeX group.

```

6890   {%

```

Setup the grow line.

```

6891     \forest@setupgrowline{#1}%

```

Get the edge of the node itself.

```

6892     \ifnum\forestove{ignore}=0
6893       \forestoget{@boundary}\forest@node@boundary
6894     \else
6895       \def\forest@node@boundary{}%
6896     \fi
6897     \csname forest@getboth\forestove{fit}edgesofpath\endcsname
6898     \forest@node@boundary\forest@negative@node@edge\forest@positive@node@edge
6899     \forestolet{negative@edge@#1}\forest@negative@node@edge
6900     \forestolet{positive@edge@#1}\forest@positive@node@edge

```

Add the edges of the children.

```

6901     \forest@get@edges@merge{negative}{#1}%
6902     \forest@get@edges@merge{positive}{#1}%
6903   }%
6904 }

```

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

```

6905 \def\forest@get@edges@merge#1#2{%
6906   \ifnum\forestove{n children}>0
6907     \forestoget{#1@edge@#2}\forest@node@edge

```

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

```

6908     \forest@pointanchor{parent anchor}%
6909     \edef\forest@getedge@pa@l{\the\pgf@x}%
6910     \edef\forest@getedge@pa@s{\the\pgf@y}%
6911     \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).

```

6912     \pgfgettransform\forest@temp@transform
6913     \pgftransformreset
6914     \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.

```

6915     \def\forest@all@edges{}%

```

```

6916 \forest@node@foreachchild{%
6917 \forest@get{#1@edge@#2}\forest@temp@edge
6918 \pgfpointtransformed{\pgfpoint{\forest@ve{1}}{\forest@ve{s}}}%
6919 \forest@extendpath\forest@node@edge\forest@temp@edge}%
6920 \ifnum\forest@ve{ignore edge}=0
6921 \pgfpointadd
6922 {\pgfpointtransformed{\pgfpoint{\forest@ve{1}}{\forest@ve{s}}}}%
6923 {\forest@pointanchor{child anchor}}%
6924 \pgfgetlastxy{\forest@getedge@ca@1}\forest@getedge@ca@s}%
6925 \eappto\forest@all@edges{%
6926 \noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@pa@1}\forest@getedge@pa@s}%
6927 \noexpand\pgfsyssoftpath@linetotoken{\forest@getedge@ca@1}\forest@getedge@ca@s}%
6928 }%
6929 % this deals with potential overlap of the edges:
6930 \eappto\forest@node@edge{\noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@ca@1}\forest@getedge@ca@s}%
6931 \fi
6932 }%
6933 \ifdefempty{\forest@all@edges}{}%
6934 \pgfintersectionofpaths{\pgfsetpath\forest@all@edges}{\pgfsetpath\forest@node@edge}%
6935 \def\forest@edgenode@intersections{%
6936 \forest@merge@intersectionloop
6937 \eappto\forest@node@edge{\expandonce{\forest@all@edges}\expandonce{\forest@edgenode@intersections}}%
6938 }%
6939 \pgfsettransform\forest@temp@transform

```

Process the path into an edge and store the edge.

```

6940 \csname forest@get#1\forest@ve{fit}edgeofpath\endcsname\forest@node@edge\forest@node@edge
6941 \forest@let{#1@edge@#2}\forest@node@edge
6942 \fi
6943 }
6944 %\newloop\forest@merge@loop
6945 \def\forest@merge@intersectionloop{%
6946 \c@pgf@counta=0
6947 \forest@loop
6948 \ifnum\c@pgf@counta<\pgfintersectionofsolutions\relax
6949 \advance\c@pgf@counta1
6950 \pgfpointintersectionsolution{\the\c@pgf@counta}%
6951 \eappto\forest@edgenode@intersections{\noexpand\pgfsyssoftpath@movetotoken
6952 {\the\pgf@x}{\the\pgf@y}}%
6953 \forest@repeat
6954 }

```

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

```

6955 \def\forest@node@getboundingrectangle@ls#1{%
6956 \forest@get{@boundary}\forest@node@boundary
6957 \forest@path@getboundingrectangle@ls\forest@node@boundary{#1}%
6958 }

```

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).

```

6959 \def\forest@pgfpathtransformed#1{%
6960 \forest@save@pgfsyssoftpath@tokendefs
6961 \let\pgfsyssoftpath@movetotoken\forest@pgfpathtransformed@moveto
6962 \let\pgfsyssoftpath@linetotoken\forest@pgfpathtransformed@lineto
6963 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
6964 #1%
6965 \forest@restore@pgfsyssoftpath@tokendefs
6966 }
6967 \def\forest@pgfpathtransformed@moveto#1#2{%
6968 \forest@pgfpathtransformed@op\pgfsyssoftpath@moveto{#1}{#2}%
6969 }
6970 \def\forest@pgfpathtransformed@lineto#1#2{%

```

```

6971 \forest@pgfpathtransformed@op\pgfsyssoftpath@lineto{#1}{#2}%
6972 }
6973 \def\forest@pgfpathtransformed@op#1#2#3{%
6974 \pgfpointtransformed{\pgfpoint{#2}{#3}}%
6975 \edef\forest@temp{%
6976 \noexpand#1{\the\pgf@x}{\the\pgf@y}%
6977 }%
6978 \forest@temp
6979 }

```

8.2.1 Tiers

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

```

6980 \def\forest@pack@computetiers{%
6981 {%
6982 \forest@pack@tiers@getalltiersinsubtree
6983 \forest@pack@tiers@computetierhierarchy
6984 \forest@pack@tiers@findcontainers
6985 \forest@pack@tiers@raisecontainers
6986 \forest@pack@tiers@computeprocessingorder
6987 \gdef\forest@smuggle{%
6988 \forest@pack@tiers@write
6989 }%
6990 \forest@node@foreach{\forestoset{@tiers}{}}%
6991 \forest@smuggle
6992 }

```

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

```

6993 \def\forest@pack@tiers@getalltiersinsubtree{%
6994 \ifnum\forestove{n children}>0
6995 \forest@node@foreachchild{\forest@pack@tiers@getalltiersinsubtree}%
6996 \fi
6997 \forestoget{tier}\forest@temp@mytier
6998 \def\forest@temp@mytiers{%
6999 \ifdefempty\forest@temp@mytier{}{%
7000 \listead\forest@temp@mytiers\forest@temp@mytier
7001 }%
7002 \ifnum\forestove{n children}>0
7003 \forest@node@foreachchild{%
7004 \forestoget{tiers}\forest@temp@tiers
7005 \forlistloop\forest@pack@tiers@forhandlerA\forest@temp@tiers
7006 }%
7007 \fi
7008 \forestolet{tiers}\forest@temp@mytiers
7009 }
7010 \def\forest@pack@tiers@forhandlerA#1{%
7011 \ifinlist{#1}\forest@temp@mytiers{}{%
7012 \listead\forest@temp@mytiers{#1}%
7013 }%
7014 }

```

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.

```

7015 \def\forest@pack@tiers@computetierhierarchy{%
7016 \def\forest@tiers@ancestors{%
7017 \forestoget{tiers}\forest@temp@mytiers
7018 \forlistloop\forest@pack@tiers@cth@init\forest@temp@mytiers
7019 \forest@pack@tiers@computetierhierarchy@
7020 }
7021 \def\forest@pack@tiers@cth@init#1{%
7022 \csdef{forest@tiers@higher@#1}{}%

```

```

7023 \csdef{forest@tiers@lower@#1}{}%
7024 }
7025 \def\forest@pack@tiers@computetierhierarchy{%
7026   \forestoget{tier}\forest@temp@mytier
7027   \ifdefempty\forest@temp@mytier{}{%
7028     \forlistloop\forest@pack@tiers@forhandlerB\forest@tiers@ancestors
7029     \listseadd\forest@tiers@ancestors\forest@temp@mytier
7030   }%
7031   \forest@node@foreachchild{%
7032     \forest@pack@tiers@computetierhierarchy@
7033   }%
7034   \forestoget{tier}\forest@temp@mytier
7035   \ifdefempty\forest@temp@mytier{}{%
7036     \forest@listedel\forest@tiers@ancestors\forest@temp@mytier
7037   }%
7038 }
7039 \def\forest@pack@tiers@forhandlerB#1{%
7040   \def\forest@temp@tier{#1}%
7041   \ifx\forest@temp@tier\forest@temp@mytier
7042     \PackageError{forest}{Circular tier hierarchy (tier \forest@temp@mytier)}{ }%
7043   \fi
7044   \ifinlistcs{#1}\forest@tiers@higher@\forest@temp@mytier{}{%
7045     \listcsadd{forest@tiers@higher@\forest@temp@mytier}{#1}%
7046     \xifinlistcs\forest@temp@mytier\forest@tiers@lower@#1{}{%
7047       \listcseadd{forest@tiers@lower@#1}\forest@temp@mytier}%
7048 }
7049 \def\forest@pack@tiers@findcontainers{%
7050   \forestoget{tiers}\forest@temp@tiers
7051   \forlistloop\forest@pack@tiers@findcontainer\forest@temp@tiers
7052 }
7053 \def\forest@pack@tiers@findcontainer#1{%
7054   \def\forest@temp@tier{#1}%
7055   \forestoget{tier}\forest@temp@mytier
7056   \ifx\forest@temp@tier\forest@temp@mytier
7057     \csedef{forest@tiers@container@#1}\forest@cn}%
7058   \else\@escapeif{%
7059     \forest@pack@tiers@findcontainerA{#1}%
7060   }\fi%
7061 }
7062 \def\forest@pack@tiers@findcontainerA#1{%
7063   \c@pgf@counta=0
7064   \forest@node@foreachchild{%
7065     \forestoget{tiers}\forest@temp@tiers
7066     \ifinlist{#1}\forest@temp@tiers{%
7067       \advance\c@pgf@counta 1
7068       \let\forest@temp@child\forest@cn
7069     }{}%
7070   }%
7071   \ifnum\c@pgf@counta>1
7072     \csedef{forest@tiers@container@#1}\forest@cn}%
7073   \else\@escapeif{% surely =1
7074     \forest@fornode{\forest@temp@child}{%
7075       \forest@pack@tiers@findcontainer{#1}%
7076     }%
7077   }\fi
7078 }
7079 \def\forest@pack@tiers@raisecontainers{%
7080   \forestoget{tiers}\forest@temp@mytiers
7081   \forlistloop\forest@pack@tiers@rc@forhandlerA\forest@temp@mytiers
7082 }
7083 \def\forest@pack@tiers@rc@forhandlerA#1{%

```

```

7084 \edef\forest@tiers@temptier{#1}%
7085 \letcs\forest@tiers@containernodeoftier{forest@tiers@container@#1}%
7086 \letcs\forest@temp@lowertiers{forest@tiers@lower@#1}%
7087 \forlistloop\forest@pack@tiers@rc@forhandlerB\forest@temp@lowertiers
7088 }
7089 \def\forest@pack@tiers@rc@forhandlerB#1{%
7090 \letcs\forest@tiers@containernodeoflowertier{forest@tiers@container@#1}%
7091 \forest@get{\forest@tiers@containernodeoflowertier}{content}\lowercontent
7092 \forest@get{\forest@tiers@containernodeoftier}{content}\uppercontent
7093 \forest@fornode{\forest@tiers@containernodeoflowertier}{%
7094 \forest@ifancestorof
7095 {\forest@tiers@containernodeoftier}
7096 {\csletcs{forest@tiers@container@\forest@tiers@temptier}{forest@tiers@container@#1}}%
7097 {}}%
7098 }%
7099 }
7100 \def\forest@pack@tiers@computeprocessingorder{%
7101 \def\forest@tiers@processingorder{%
7102 \forest@toget{tiers}\forest@tiers@cpo@tierstodo
7103 \safeloop
7104 \ifdefempty\forest@tiers@cpo@tierstodo{\forest@tempfalse}{\forest@temptrue}%
7105 \ifforest@temp
7106 \def\forest@tiers@cpo@tiersremaining{%
7107 \def\forest@tiers@cpo@tiersindependent{%
7108 \forlistloop\forest@pack@tiers@cpo@forhandlerA\forest@tiers@cpo@tierstodo
7109 \ifdefempty\forest@tiers@cpo@tiersindependent{%
7110 \PackageError{forest}{Circular tiers!}{}}}%
7111 \forlistloop\forest@pack@tiers@cpo@forhandlerB\forest@tiers@cpo@tiersremaining
7112 \let\forest@tiers@cpo@tierstodo\forest@tiers@cpo@tiersremaining
7113 \saferepeat
7114 }
7115 \def\forest@pack@tiers@cpo@forhandlerA#1{%
7116 \ifcseempty{forest@tiers@higher@#1}%
7117 \listadd\forest@tiers@cpo@tiersindependent{#1}%
7118 \listadd\forest@tiers@processingorder{#1}%
7119 }{%
7120 \listadd\forest@tiers@cpo@tiersremaining{#1}%
7121 }%
7122 }
7123 \def\forest@pack@tiers@cpo@forhandlerB#1{%
7124 \def\forest@pack@tiers@cpo@aremainingtier{#1}%
7125 \forlistloop\forest@pack@tiers@cpo@forhandlerC\forest@tiers@cpo@tiersindependent
7126 }
7127 \def\forest@pack@tiers@cpo@forhandlerC#1{%
7128 \ifinlistcs{#1}{forest@tiers@higher@\forest@pack@tiers@cpo@aremainingtier}{%
7129 \forest@listcsdel{forest@tiers@higher@\forest@pack@tiers@cpo@aremainingtier}{#1}%
7130 }{}%
7131 }
7132 \def\forest@pack@tiers@write{%
7133 \forlistloop\forest@pack@tiers@write@forhandler\forest@tiers@processingorder
7134 }
7135 \def\forest@pack@tiers@write@forhandler#1{%
7136 \forest@fornode{\csname forest@tiers@container@#1\endcsname}{%
7137 \forest@pack@tiers@check{#1}%
7138 }%
7139 \xappto\forest@smuggle{%
7140 \noexpand\listadd
7141 \forest@om{\csname forest@tiers@container@#1\endcsname}{@tiers}%
7142 {#1}%
7143 }%
7144 }

```

```

7145 % checks if the tier is compatible with growth changes and calign=node/edge angle
7146 \def\forest@pack@tiers@check#1{%
7147   \def\forest@temp@currenttier{#1}%
7148   \forest@node@foreachdescendant{%
7149     \ifnum\forestove{grow}=\forestOve{\forestove{@parent}}{grow}
7150     \else
7151       \forest@pack@tiers@check@grow
7152     \fi
7153     \ifnum\forestove{n children}>1
7154       \foresttoget{calign}\forest@temp
7155       \ifx\forest@temp\forest@pack@tiers@check@nodeangle
7156         \forest@pack@tiers@check@calign
7157       \fi
7158       \ifx\forest@temp\forest@pack@tiers@check@edgeangle
7159         \forest@pack@tiers@check@calign
7160       \fi
7161     \fi
7162   }%
7163 }
7164 \def\forest@pack@tiers@check@nodeangle{node angle}%
7165 \def\forest@pack@tiers@check@edgeangle{edge angle}%
7166 \def\forest@pack@tiers@check@grow{%
7167   \foresttoget{content}\forest@temp@content
7168   \let\forest@temp@currentnode\forest@cn
7169   \forest@node@foreachdescendant{%
7170     \foresttoget{tier}\forest@temp
7171     \ifx\forest@temp@currenttier\forest@temp
7172       \forest@pack@tiers@check@grow@error
7173     \fi
7174   }%
7175 }
7176 \def\forest@pack@tiers@check@grow@error{%
7177   \PackageError{forest}{Tree growth direction changes in node \forest@temp@currentnode\space
7178     (content: \forest@temp@content), while tier '\forest@temp' is specified for nodes both
7179     out- and inside the subtree rooted in node \forest@temp@currentnode. This will not work.}{}%
7180 }
7181 \def\forest@pack@tiers@check@calign{%
7182   \forest@node@foreachchild{%
7183     \foresttoget{tier}\forest@temp
7184     \ifx\forest@temp@currenttier\forest@temp
7185       \forest@pack@tiers@check@calign@warning
7186     \fi
7187   }%
7188 }
7189 \def\forest@pack@tiers@check@calign@warning{%
7190   \PackageWarning{forest}{Potential option conflict: node \forestove{@parent} (content:
7191     '\forestOve{\forestove{@parent}}{content}') was given 'calign=\forestove{calign}', while its
7192     child \forest@cn\space (content: '\forestove{content}') was given 'tier=\forestove{tier}'.
7193     The parent's 'calign' will only work if the child was the lowest node on its tier before the
7194     alignment.}%
7195 }

```

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.

```

7196 \pgfdeclaregenericanchor{forestcomputenodeboundary}{%
7197   \letcs\forest@temp@boundary@macro\forest@compute@node@boundary@#1}%
7198   \ifcsname forest@compute@node@boundary@#1\endcsname
7199     \csname forest@compute@node@boundary@#1\endcsname

```



```

7200 \else
7201   \forest@compute@node@boundary@rectangle
7202 \fi
7203 \pgfsyssoftpath@getcurrentpath\forest@temp
7204 \global\let\forest@global@boundary\forest@temp
7205 }
7206 \def\forest@mt#1{%
7207   \expandafter\pgfpointanchor\expandafter{\pgfreferencednodename}{#1}%
7208   \pgfsyssoftpath@moveto{\the\pgf@x}{\the\pgf@y}%
7209 }%
7210 \def\forest@lt#1{%
7211   \expandafter\pgfpointanchor\expandafter{\pgfreferencednodename}{#1}%
7212   \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
7213 }%
7214 \def\forest@compute@node@boundary@coordinate{%
7215   \forest@mt{center}%
7216 }
7217 \def\forest@compute@node@boundary@circle{%
7218   \forest@mt{east}%
7219   \forest@lt{north east}%
7220   \forest@lt{north}%
7221   \forest@lt{north west}%
7222   \forest@lt{west}%
7223   \forest@lt{south west}%
7224   \forest@lt{south}%
7225   \forest@lt{south east}%
7226   \forest@lt{east}%
7227 }
7228 \def\forest@compute@node@boundary@rectangle{%
7229   \forest@mt{south west}%
7230   \forest@lt{south east}%
7231   \forest@lt{north east}%
7232   \forest@lt{north west}%
7233   \forest@lt{south west}%
7234 }
7235 \def\forest@compute@node@boundary@diamond{%
7236   \forest@mt{east}%
7237   \forest@lt{north}%
7238   \forest@lt{west}%
7239   \forest@lt{south}%
7240   \forest@lt{east}%
7241 }
7242 \let\forest@compute@node@boundary@ellipse\forest@compute@node@boundary@circle
7243 \def\forest@compute@node@boundary@trapezium{%
7244   \forest@mt{top right corner}%
7245   \forest@lt{top left corner}%
7246   \forest@lt{bottom left corner}%
7247   \forest@lt{bottom right corner}%
7248   \forest@lt{top right corner}%
7249 }
7250 \def\forest@compute@node@boundary@semicircle{%
7251   \forest@mt{arc start}%
7252   \forest@lt{north}%
7253   \forest@lt{east}%
7254   \forest@lt{north east}%
7255   \forest@lt{apex}%
7256   \forest@lt{north west}%
7257   \forest@lt{west}%
7258   \forest@lt{arc end}%
7259   \forest@lt{arc start}%
7260 }

```

```

7261 %\newloop\forest@computenodeboundary@loop
7262 \csdef{forest@compute@node@boundary@regular polygon}{%
7263   \forest@mt{corner 1}%
7264   \c@pgf@counta=\sides\relax
7265   \forest@loop
7266   \ifnum\c@pgf@counta>0
7267     \forest@lt{corner \the\c@pgf@counta}%
7268     \advance\c@pgf@counta-1
7269   \forest@repeat
7270 }%
7271 \def\forest@compute@node@boundary@star{%
7272   \forest@mt{outer point 1}%
7273   \c@pgf@counta=\totalstarpoints\relax
7274   \divide\c@pgf@counta2
7275   \forest@loop
7276   \ifnum\c@pgf@counta>0
7277     \forest@lt{inner point \the\c@pgf@counta}%
7278     \forest@lt{outer point \the\c@pgf@counta}%
7279     \advance\c@pgf@counta-1
7280   \forest@repeat
7281 }%
7282 \csdef{forest@compute@node@boundary@isosceles triangle}{%
7283   \forest@mt{apex}%
7284   \forest@lt{left corner}%
7285   \forest@lt{right corner}%
7286   \forest@lt{apex}%
7287 }
7288 \def\forest@compute@node@boundary@kite{%
7289   \forest@mt{upper vertex}%
7290   \forest@lt{left vertex}%
7291   \forest@lt{lower vertex}%
7292   \forest@lt{right vertex}%
7293   \forest@lt{upper vertex}%
7294 }
7295 \def\forest@compute@node@boundary@dart{%
7296   \forest@mt{tip}%
7297   \forest@lt{left tail}%
7298   \forest@lt{tail center}%
7299   \forest@lt{right tail}%
7300   \forest@lt{tip}%
7301 }
7302 \csdef{forest@compute@node@boundary@circular sector}{%
7303   \forest@mt{sector center}%
7304   \forest@lt{arc start}%
7305   \forest@lt{arc center}%
7306   \forest@lt{arc end}%
7307   \forest@lt{sector center}%
7308 }
7309 \def\forest@compute@node@boundary@cylinder{%
7310   \forest@mt{top}%
7311   \forest@lt{after top}%
7312   \forest@lt{before bottom}%
7313   \forest@lt{bottom}%
7314   \forest@lt{after bottom}%
7315   \forest@lt{before top}%
7316   \forest@lt{top}%
7317 }
7318 \cslet{forest@compute@node@boundary@forbidden sign}\forest@compute@node@boundary@circle
7319 \cslet{forest@compute@node@boundary@magnifying glass}\forest@compute@node@boundary@circle
7320 \def\forest@compute@node@boundary@cloud{%
7321   \getradii

```

```

7322 \forest@mt{puff 1}%
7323 \c@pgf@counta=\puffs\relax
7324 \forest@loop
7325 \ifnum\c@pgf@counta>0
7326 \forest@lt{puff \the\c@pgf@counta}%
7327 \advance\c@pgf@counta-1
7328 \forest@repeat
7329 }
7330 \def\forest@compute@node@boundary@starburst{
7331 \calculatedstarburstpoints
7332 \forest@mt{outer point 1}%
7333 \c@pgf@counta=\totalpoints\relax
7334 \divide\c@pgf@counta2
7335 \forest@loop
7336 \ifnum\c@pgf@counta>0
7337 \forest@lt{inner point \the\c@pgf@counta}%
7338 \forest@lt{outer point \the\c@pgf@counta}%
7339 \advance\c@pgf@counta-1
7340 \forest@repeat
7341 }%
7342 \def\forest@compute@node@boundary@signal{%
7343 \forest@mt{east}%
7344 \forest@lt{south east}%
7345 \forest@lt{south west}%
7346 \forest@lt{west}%
7347 \forest@lt{north west}%
7348 \forest@lt{north east}%
7349 \forest@lt{east}%
7350 }
7351 \def\forest@compute@node@boundary@tape{%
7352 \forest@mt{north east}%
7353 \forest@lt{60}%
7354 \forest@lt{north}%
7355 \forest@lt{120}%
7356 \forest@lt{north west}%
7357 \forest@lt{south west}%
7358 \forest@lt{240}%
7359 \forest@lt{south}%
7360 \forest@lt{310}%
7361 \forest@lt{south east}%
7362 \forest@lt{north east}%
7363 }
7364 \csdef{forest@compute@node@boundary@single arrow}{%
7365 \forest@mt{tip}%
7366 \forest@lt{after tip}%
7367 \forest@lt{after head}%
7368 \forest@lt{before tail}%
7369 \forest@lt{after tail}%
7370 \forest@lt{before head}%
7371 \forest@lt{before tip}%
7372 \forest@lt{tip}%
7373 }
7374 \csdef{forest@compute@node@boundary@double arrow}{%
7375 \forest@mt{tip 1}%
7376 \forest@lt{after tip 1}%
7377 \forest@lt{after head 1}%
7378 \forest@lt{before head 2}%
7379 \forest@lt{before tip 2}%
7380 \forest@mt{tip 2}%
7381 \forest@lt{after tip 2}%
7382 \forest@lt{after head 2}%

```

```

7383 \forest@lt{before head 1}%
7384 \forest@lt{before tip 1}%
7385 \forest@lt{tip 1}%
7386 }
7387 \csdef{forest@compute@node@boundary@arrow box}{%
7388 \forest@mt{before north arrow}%
7389 \forest@lt{before north arrow head}%
7390 \forest@lt{before north arrow tip}%
7391 \forest@lt{north arrow tip}%
7392 \forest@lt{after north arrow tip}%
7393 \forest@lt{after north arrow head}%
7394 \forest@lt{after north arrow}%
7395 \forest@lt{north east}%
7396 \forest@lt{before east arrow}%
7397 \forest@lt{before east arrow head}%
7398 \forest@lt{before east arrow tip}%
7399 \forest@lt{east arrow tip}%
7400 \forest@lt{after east arrow tip}%
7401 \forest@lt{after east arrow head}%
7402 \forest@lt{after east arrow}%
7403 \forest@lt{south east}%
7404 \forest@lt{before south arrow}%
7405 \forest@lt{before south arrow head}%
7406 \forest@lt{before south arrow tip}%
7407 \forest@lt{south arrow tip}%
7408 \forest@lt{after south arrow tip}%
7409 \forest@lt{after south arrow head}%
7410 \forest@lt{after south arrow}%
7411 \forest@lt{south west}%
7412 \forest@lt{before west arrow}%
7413 \forest@lt{before west arrow head}%
7414 \forest@lt{before west arrow tip}%
7415 \forest@lt{west arrow tip}%
7416 \forest@lt{after west arrow tip}%
7417 \forest@lt{after west arrow head}%
7418 \forest@lt{after west arrow}%
7419 \forest@lt{north west}%
7420 \forest@lt{before north arrow}%
7421 }
7422 \cslet{forest@compute@node@boundary@circle split}\forest@compute@node@boundary@circle
7423 \cslet{forest@compute@node@boundary@circle solidus}\forest@compute@node@boundary@circle
7424 \cslet{forest@compute@node@boundary@ellipse split}\forest@compute@node@boundary@ellipse
7425 \cslet{forest@compute@node@boundary@rectangle split}\forest@compute@node@boundary@rectangle
7426 \def\forest@compute@node@boundary@@callout{%
7427 \beforecalloutpointer
7428 \pgfsyssoftpath@moveto{\the\pgf@x}{\the\pgf@y}%
7429 \calloutpointeranchor
7430 \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
7431 \aftercalloutpointer
7432 \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
7433 }
7434 \csdef{forest@compute@node@boundary@rectangle callout}{%
7435 \forest@compute@node@boundary@rectangle
7436 \rectanglecalloutpoints
7437 \forest@compute@node@boundary@@callout
7438 }
7439 \csdef{forest@compute@node@boundary@ellipse callout}{%
7440 \forest@compute@node@boundary@ellipse
7441 \ellipsecalloutpoints
7442 \forest@compute@node@boundary@@callout
7443 }

```

```

7444 \csdef{forest@compute@node@boundary@cloud callout}{%
7445   \forest@compute@node@boundary@cloud
7446   % at least a first approx...
7447   \forest@mt{center}%
7448   \forest@lt{pointer}%
7449 }%
7450 \csdef{forest@compute@node@boundary@cross out}{%
7451   \forest@mt{south east}%
7452   \forest@lt{north west}%
7453   \forest@mt{south west}%
7454   \forest@lt{north east}%
7455 }%
7456 \csdef{forest@compute@node@boundary@strike out}{%
7457   \forest@mt{north east}%
7458   \forest@lt{south west}%
7459 }%
7460 \csdef{forest@compute@node@boundary@rounded rectangle}{%
7461   \forest@mt{east}%
7462   \forest@lt{north east}%
7463   \forest@lt{north}%
7464   \forest@lt{north west}%
7465   \forest@lt{west}%
7466   \forest@lt{south west}%
7467   \forest@lt{south}%
7468   \forest@lt{south east}%
7469   \forest@lt{east}%
7470 }%
7471 \csdef{forest@compute@node@boundary@chamfered rectangle}{%
7472   \forest@mt{before south west}%
7473   \forest@mt{after south west}%
7474   \forest@lt{before south east}%
7475   \forest@lt{after south east}%
7476   \forest@lt{before north east}%
7477   \forest@lt{after north east}%
7478   \forest@lt{before north west}%
7479   \forest@lt{after north west}%
7480   \forest@lt{before south west}%
7481 }%

```

8.3 Compute absolute positions

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

```

7482 \def\forest@node@computeabsolutepositions{%
7483   \edef\forest@marshal{%
7484     \noexpand\forest@node@foreachchild{%
7485       \noexpand\forest@node@computeabsolutepositions@{\forestove{x}}{\forestove{y}}{\forestove{grow}}}%
7486     }%
7487   }\forest@marshal
7488 }
7489 \def\forest@node@computeabsolutepositions@#1#2#3{%
7490   \pgfpointadd
7491     {\pgfqpoint{#1}{#2}}%
7492     {\pgfpointadd
7493       {\pgfqpointpolar{#3}{\forestove{1}}}%
7494       {\pgfqpointpolar{\numexpr 90+#3\relax}{\forestove{s}}}%
7495     }%
7496   \pgfgetlastxy\forest@temp@x\forest@temp@y
7497   \forestolet{x}\forest@temp@x
7498   \forestolet{y}\forest@temp@y

```

```

7499 \edef\forest@marshal{%
7500   \noexpand\forest@node@foreachchild{%
7501     \noexpand\forest@node@computeabsolute positions@{\forest@temp@x}{\forest@temp@y}{\forest@temp@z}{\forest@temp@grow}}%
7502   }%
7503 }\forest@marshal
7504 }

```

8.4 Drawing the tree

```

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

```

```

7556 \noexpand\forest@node@drawtree@
7557 \expandonce{\forest@temp@enddraw\pgfkeysnovalue\pgfeov}%
7558 \noexpand\forest@drawtree@endbox
7559 }\forest@marshal
7560 \ifforesttikzcshack
7561 \let\tikz@parse@node\forest@original\tikz@parse@node
7562 \fi
7563 %
7564 \ifforest@external@
7565 \ifforest@externalize@tree@
7566 \tikzexternaldisable
7567 \eappto\forest@externalize@checkimages{%
7568 \noexpand\forest@includeexternal@check{\forest@externalize@filename}%
7569 }%
7570 \expandafter\ifstrequal\expandafter{\forest@drawtreebox}{\pgfkeysnovalue}{%
7571 \eappto\forest@externalize@loadimages{%
7572 \noexpand\forest@includeexternal{\forest@externalize@filename}%
7573 }%
7574 }{%
7575 \eappto\forest@externalize@loadimages{%
7576 \noexpand\forest@includeexternal@box\forest@drawtreebox{\forest@externalize@filename}%
7577 }%
7578 }%
7579 \fi
7580 \fi
7581 }
7582 \def\forest@drawtree@root{0}
7583 \def\forest@node@drawtree{%
7584 \def\forest@clear@drawn{%
7585 \forest@forthis{%
7586 \forest@saveandrestoremacro\forest@drawtree@root{%
7587 \edef\forest@drawtree@root{\forest@cn}%
7588 \forestset{draw tree method}%
7589 }%
7590 }%
7591 \forest@node@ifnamedefined{forest@baseline@node}{%
7592 \edef\forest@baseline@id{\forest@node@Nametoid{forest@baseline@node}}%
7593 \ifcsdef{forest@drawn@\forest@baseline@id}{%
7594 \edef\forest@marshal{%
7595 \noexpand\pgfsetbaselinepointlater{%
7596 \noexpand\pgfpointanchor
7597 {\forest@ve{\forest@baseline@id}{name}}%
7598 {\forest@ve{\forest@baseline@id}{anchor}}}%
7599 }%
7600 }\forest@marshal
7601 }{}%
7602 }{}%
7603 \forest@clear@drawn
7604 }
7605 \def\forest@draw@node{%
7606 \ifnum\forest@ve{phantom}=0
7607 \forest@draw@node@
7608 \fi
7609 }
7610 \def\forest@draw@node@{%
7611 \forest@node@forest@position@node@later@restore
7612 \ifforest@drawtree@preservenodeboxes@
7613 \pgfnodealias{forest@temp}{\forest@ve{later@name}}%
7614 \fi
7615 \pgfpositionnodenow{\pgfqpoint{\forest@ve{x}}{\forest@ve{y}}}%
7616 \ifforest@drawtree@preservenodeboxes@

```

```

7617 \pgfnodelalias{\forestove{later@name}}{\forest@temp}%
7618 \fi
7619 \csdef{forest@drawn@{\forest@cn}}{}%
7620 \eappto\forest@clear@drawn{\noexpand\csundef{forest@drawn@{\forest@cn}}}%
7621 }
7622 \def\forest@draw@edge{%
7623 \ifcsdef{forest@drawn@{\forest@cn}}{% was the current node drawn?
7624 \ifnum\forestove{@parent}=0 % do we have a parent?
7625 \else
7626 \ifcsdef{forest@drawn@{\forestove{@parent}}}{% was the parent drawn?
7627 \forest@draw@edge@
7628 }{}%
7629 \fi
7630 }{}%
7631 }
7632 \def\forest@draw@edge@{%
7633 \edef\forest@temp{\forestove{edge path}}\forest@temp
7634 }
7635 \def\forest@draw@tikz{%
7636 \ifnum\forestove{phantom}=0
7637 \forest@draw@tikz@
7638 \fi
7639 }
7640 \def\forest@draw@tikz@{%
7641 \forestove{tikz}%
7642 }

```

9 Geometry

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

```

7643 \newdimen\forest@xg
7644 \newdimen\forest@yg
7645 \newdimen\forest@xs
7646 \newdimen\forest@ys
7647 \def\forest@setupgrowline#1{%
7648 \edef\forest@grow{#1}%
7649 \pgfqpointpolar{\forest@grow}{1pt}%
7650 \forest@xg=\pgf@x
7651 \forest@yg=\pgf@y
7652 \forest@xs=-\pgf@y
7653 \forest@ys=\pgf@x
7654 }

```

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`.

```

7655 \def\forest@pgfpointprojectiontogrowline#1{%
7656 \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.

```

7657 \pgfutil@tempdima=\pgf@sys@tonumber{\pgf@x}\forest@xg%
7658 \advance\pgfutil@tempdima by\pgf@sys@tonumber{\pgf@y}\forest@yg%

```

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

```

7659 \global\pgf@x=\pgf@sys@tonumber{\pgfutil@tempdima}\forest@xg%

```



```

7660 \global\pgf@y=\pgf@sys@tonumber{\pgfutil@tempdima}\forest@yg%
7661 }}

```

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

```

7662 \def\forest@distancetogrowline#1#2{%
7663   \pgf@process{#2}%
7664   #1=\pgf@sys@tonumber{\pgf@x}\forest@xs\relax
7665   \advance#1 by\pgf@sys@tonumber{\pgf@y}\forest@ys\relax
7666 }

```

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

The following macro projects all segment edges (“points”) of a simple² path #1 onto the grow line. The result is an array of tuples (xo, yo, xp, yp), where xo and yo stand for the *original* point, and xp and yp stand for its *projection*. The prefix of the array is given by #2. If the array already exists, the new items are appended to it. The array is not sorted: the order of original points in the array is their order in the path. The computation does not destroy the current path. All result-macros have local scope.

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

```

7667 \let\forest@pp@n\relax
7668 \def\forest@projectpath@togrowline#1#2{%
7669   \edef\forest@pp@prefix{#2}%
7670   \forest@save@pgfsyssoftpath@tokendefs
7671   \let\pgfsyssoftpath@movetotoken\forest@projectpath@processpoint
7672   \let\pgfsyssoftpath@linetotoken\forest@projectpath@processpoint
7673   \c@pgf@counta=0
7674   #1%
7675   \csedef{#2n}{\the\c@pgf@counta}%
7676   \forest@restore@pgfsyssoftpath@tokendefs
7677 }

```

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

```

7678 \def\forest@projectpath@processpoint#1#2{%
7679   \pgfqpoint{#1}{#2}%
7680   \expandafter\edef\c@pgf@counta\forest@pp@prefix\the\c@pgf@counta xo\endcsname{\the\pgf@x}%
7681   \expandafter\edef\c@pgf@counta\forest@pp@prefix\the\c@pgf@counta yo\endcsname{\the\pgf@y}%
7682   \forest@pgfpointprojectiontogrowline{}%
7683   \expandafter\edef\c@pgf@counta\forest@pp@prefix\the\c@pgf@counta xp\endcsname{\the\pgf@x}%
7684   \expandafter\edef\c@pgf@counta\forest@pp@prefix\the\c@pgf@counta yp\endcsname{\the\pgf@y}%
7685   \advance\c@pgf@counta 1\relax
7686 }

```

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

```

7687 \def\forest@sortprojections#1{%
7688   % todo: optimize in cases when we know that the array is actually a
7689   % merger of sorted arrays; when does this happen? in
7690   % distance_between_paths, and when merging the edges of the parent
7691   % and its children in a uniform growth tree
7692   \edef\forest@ppi@inputprefix{#1}%
7693   \c@pgf@counta=\c@pgf@counta#1n\endcsname\relax
7694   \advance\c@pgf@counta -1
7695   \forest@sort\forest@ppiraw@cmp\forest@ppiraw@let\forest@sort@ascending{0}{\the\c@pgf@counta}%
7696 }

```

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)

²A 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 #2: keys are the coordinates (x,y) of the original points; a value is the index of the original point's projection in array #2.³

```
7697 \def\forest@processprojectioninfo#1#2{%
7698   \edef\forest@ppi@inputprefix{#1}%
```

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

```
7699   \c@pgf@counta=0
7700   \c@pgf@countb=-1
7701   \safeloop
7702   \ifnum\c@pgf@counta<\csname#1n\endcsname\relax
```

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

```
7703   \letcs\forest@xo{#1\the\c@pgf@counta xo}%
7704   \letcs\forest@yo{#1\the\c@pgf@counta yo}%
7705   \letcs\forest@xp{#1\the\c@pgf@counta xp}%
7706   \letcs\forest@yp{#1\the\c@pgf@counta yp}%
7707   \ifnum\c@pgf@countb<0
7708     \forest@equaltotolerancefalse
7709   \else
7710     \forest@equaltotolerance
7711     {\pgfqpoint\forest@xp\forest@yp}%
7712     {\pgfqpoint
7713       {\csname#2\the\c@pgf@countb x\endcsname}%
7714       {\csname#2\the\c@pgf@countb y\endcsname}%
7715     }%
7716   \fi
7717   \ifforest@equaltotolerance\else
```

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

```
7718   \advance\c@pgf@countb 1
7719   \cslet{#2\the\c@pgf@countb x}\forest@xp
7720   \cslet{#2\the\c@pgf@countb y}\forest@yp
7721   \csdef{#2\the\c@pgf@countb @n}{0}%
7722   \fi
```

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

```
7723   % todo: this is ugly!
7724   \ifdefined\forest@xo\ifx\forest@xo\relax\else
7725     \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.

```
7726     \forest@append@point@to@inner@array
7727     \forest@xo\forest@yo
7728     {#2\the\c@pgf@countb @}%
```

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

```
7729     \csedef{#2(\forest@xo,\forest@yo)}{\the\c@pgf@countb}%
7730     \fi\fi
7731   \fi\fi
```

Clean-up the raw array item.

```
7732   \cslet{#1\the\c@pgf@counta xo}\relax
7733   \cslet{#1\the\c@pgf@counta yo}\relax
7734   \cslet{#1\the\c@pgf@counta xp}\relax
```

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

```

7735 \cslet{#1\the\c@pgf@counta yp}\relax
7736 \advance\c@pgf@counta 1
7737 \saferepeat

Clean up the raw array length.
7738 \cslet{#1n}\relax

Store the length of the outer result array.
7739 \advance\c@pgf@countb 1
7740 \csedef{#2n}{\the\c@pgf@countb}%
7741 }

Item-exchange macro for quicksorting the raw projection data. (#1 is copied into #2.)
7742 \def\forest@ppiraw@let#1#2{%
7743 \csletcs{\forest@ppi@inputprefix#1xo}{\forest@ppi@inputprefix#2xo}%
7744 \csletcs{\forest@ppi@inputprefix#1yo}{\forest@ppi@inputprefix#2yo}%
7745 \csletcs{\forest@ppi@inputprefix#1xp}{\forest@ppi@inputprefix#2xp}%
7746 \csletcs{\forest@ppi@inputprefix#1yp}{\forest@ppi@inputprefix#2yp}%
7747 }

Item comparison macro for quicksorting the raw projection data.
7748 \def\forest@ppiraw@cmp#1#2{%
7749 \forest@sort@cmptwodimcs
7750 {\forest@ppi@inputprefix#1xp}{\forest@ppi@inputprefix#1yp}%
7751 {\forest@ppi@inputprefix#2xp}{\forest@ppi@inputprefix#2yp}%
7752 }

Append the point (#1,#2) to the (inner) array of points (prefix #3).
7753 \def\forest@append@point@to@inner@array#1#2#3{%
7754 \c@pgf@countc=\csname#3n\endcsname\relax
7755 \csedef{#3\the\c@pgf@countc x}{#1}%
7756 \csedef{#3\the\c@pgf@countc y}{#2}%
7757 \forest@distancetogrowline\pgfutil@tempdima{\pgfqpoint#1#2}%
7758 \csedef{#3\the\c@pgf@countc d}{\the\pgfutil@tempdima}%
7759 \advance\c@pgf@countc 1
7760 \csedef{#3n}{\the\c@pgf@countc}%
7761 }

```

9.2 Break path

The following macro computes from the given path (#1) a “broken” path (#3) that contains the same points of the plane, but has potentially more segments, so that, for every point from a given set of points on the grow line, a line through this point perpendicular to the grow line intersects the broken path only at its edge segments (i.e. not between them).

The macro works only for *simple* paths, i.e. paths built using only move-to and line-to operations. Furthermore, `\forest@processprojectioninfo` must be called before calling `\forest@breakpath`: we expect information with prefix #2. The macro updates the information compiled by `\forest@processprojectioninfo` with information about points added by path-breaking.

```

7762 \def\forest@breakpath#1#2#3{%

Store the current path in a macro and empty it, then process the stored path. The processing creates a new current path.
7763 \edef\forest@bp@prefix{#2}%
7764 \forest@save@pgfsyssoftpath@tokendefs
7765 \let\pgfsyssoftpath@movetotoken\forest@breakpath@processfirstpoint
7766 \let\pgfsyssoftpath@linetotoken\forest@breakpath@processfirstpoint
7767 %\pgfusepath{}% empty the current path. ok?
7768 #1%
7769 \forest@restore@pgfsyssoftpath@tokendefs
7770 \pgfsyssoftpath@getcurrentpath#3%
7771 }

```

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.)

```

7772 \def\forest@breakpath@processfirstpoint#1#2{%
7773   \forest@breakpath@processmoveto{#1}{#2}%
7774   \let\pgfsyssoftpath@movetotoken\forest@breakpath@processmoveto
7775   \let\pgfsyssoftpath@linetotoken\forest@breakpath@processlineto
7776 }

```

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.

```

7777 \def\forest@breakpath@processmoveto#1#2{%
7778   \pgfsyssoftpath@moveto{#1}{#2}%
7779   \def\forest@previous@x{#1}%
7780   \def\forest@previous@y{#2}%
7781   \expandafter\let\expandafter\forest@previous@i
7782   \csname\forest@bp@prefix(#1,#2)\endcsname
7783   \expandafter\let\expandafter\forest@previous@px
7784   \csname\forest@bp@prefix\forest@previous@i x\endcsname
7785   \expandafter\let\expandafter\forest@previous@py
7786   \csname\forest@bp@prefix\forest@previous@i y\endcsname
7787 }

```

This is the heart of the path-breaking procedure.

```

7788 \def\forest@breakpath@processlineto#1#2{%

```

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

```

7789   \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.)

```

7790   \expandafter\let\expandafter\forest@i
7791   \csname\forest@bp@prefix(#1,#2)\endcsname
7792   \expandafter\let\expandafter\forest@px
7793   \csname\forest@bp@prefix\forest@i x\endcsname
7794   \expandafter\let\expandafter\forest@py
7795   \csname\forest@bp@prefix\forest@i y\endcsname

```

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

```

7796   \forest@equaltotolerance
7797   {\pgfqpoint{\forest@previous@px}{\forest@previous@py}}%
7798   {\pgfqpoint{\forest@px}{\forest@py}}%
7799   \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.

```

7800   \let\forest@breakpath@op\pgfsyssoftpath@moveto
7801   \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.

```

7802   \forest@temp@count=\forest@previous@i\relax
7803   \ifnum\forest@previous@i<\forest@i\relax
7804     \def\forest@breakpath@step{1}%
7805     \def\forest@breakpath@test{\forest@temp@count<\forest@i\relax}%
7806   \else
7807     \def\forest@breakpath@step{-1}%
7808     \def\forest@breakpath@test{\forest@temp@count>\forest@i\relax}%
7809   \fi

```

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

```

7810   \safeloop

```

```

7811 \advance\forest@temp@count\forest@breakpath@step\relax
7812 \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.)
7813 \pgfpointintersectionoflines
7814 {\pgfpoint
7815 {\csname\forest@bp@prefix\the\forest@temp@count x\endcsname}%
7816 {\csname\forest@bp@prefix\the\forest@temp@count y\endcsname}%
7817 }%
7818 {\pgfpointadd
7819 {\pgfpoint
7820 {\csname\forest@bp@prefix\the\forest@temp@count x\endcsname}%
7821 {\csname\forest@bp@prefix\the\forest@temp@count y\endcsname}%
7822 }%
7823 {\pgfpoint{\forest@xs}{\forest@ys}}%
7824 }%
7825 {\pgfpoint{\forest@previous@x}{\forest@previous@y}}%
7826 {\pgfpoint{#1}{#2}}%

Break the segment at the intersection.
7827 \pgfgetlastxy\forest@last@x\forest@last@y
7828 \pgfsyssoftpath@lineto\forest@last@x\forest@last@y

Append the breaking point to the inner array for the projection.
7829 \forest@append@point@to@inner@array
7830 \forest@last@x\forest@last@y
7831 {\forest@bp@prefix\the\forest@temp@count @}%

Cache the projection of the new segment edge.
7832 \csedef{\forest@bp@prefix(\the\pgf@x,\the\pgf@y)}{\the\forest@temp@count}%
7833 \saferepeat
7834 \fi

Add the current point.
7835 \forest@breakpath@op{#1}{#2}%

Setup new “previous” info: the segment edge, its projection’s index, and the projection.
7836 \def\forest@previous@x{#1}%
7837 \def\forest@previous@y{#2}%
7838 \let\forest@previous@i\forest@i
7839 \let\forest@previous@px\forest@px
7840 \let\forest@previous@py\forest@py
7841 }

Patch for speed: no need to call \pgfmathparse here.
7842 \patchcmd{\pgfpointintersectionoflines}{\pgfpoint}{\pgfpoint}{-}{-}

```

9.3 Get tight edge of path

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

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

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

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

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

```

7843 \def\forest@getnegativetightedgeofpath#1#2{%
7844   \forest@get@onetightedgeofpath#1\forest@sort@ascending#2}
7845 \def\forest@getpositivetightedgeofpath#1#2{%
7846   \forest@get@onetightedgeofpath#1\forest@sort@descending#2}
7847 \def\forest@get@onetightedgeofpath#1#2#3{%
7848   {%
7849     \forest@get@one@tightedgeofpath#1#2\forest@gep@edge
7850     \global\let\forest@gep@global@edge\forest@gep@edge
7851   }%
7852   \let#3\forest@gep@global@edge
7853 }
7854 \def\forest@get@one@tightedgeofpath#1#2#3{%

```

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

```

7855 \forest@projectpathtogrowline#1{forest@pp@}%
7856 \forest@sortprojections{forest@pp@}%
7857 \forest@processprojectioninfo{forest@pp@}{forest@pi@}%

```

Break the path.

```

7858 \forest@breakpath#1{forest@pi@}\forest@brokenpath

```

Compile some more useful information.

```

7859 \forest@sort@inner@arrays{forest@pi@}#2%
7860 \forest@pathtodict\forest@brokenpath{forest@pi@}%

```

The auxiliary data is set up: do the work!

```

7861 \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).

```

7862 \forest@simplifypath\forest@edge#3%
7863 }

```

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

```

7864 \def\forest@getbothtightedgesofpath#1#2#3{%
7865   {%
7866     \forest@get@one@tightedgeofpath#1\forest@sort@ascending\forest@gep@firstedge

```

Reverse the order of items in the inner arrays.

```

7867   \c@pgf@counta=0
7868   \forest@loop
7869   \ifnum\c@pgf@counta<\forest@pi@n\relax
7870     \forest@ppi@deflet{forest@pi@}\the\c@pgf@counta @}%
7871     \forest@reversearray\forest@ppi@let
7872     {0}%
7873     {\csname forest@pi@\the\c@pgf@counta @n\endcsname}%
7874     \advance\c@pgf@counta 1
7875   \forest@repeat

```

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

```

7876   \forest@gettightedgeofpath@getedge\forest@edge
7877   \forest@simplifypath\forest@edge\forest@gep@secondedge

```

Smuggle the results out of the enclosing T_EX group.

```

7878   \global\let\forest@gep@global@firstedge\forest@gep@firstedge
7879   \global\let\forest@gep@global@secondedge\forest@gep@secondedge
7880 }%
7881 \let#2\forest@gep@global@firstedge
7882 \let#3\forest@gep@global@secondedge
7883 }

```

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

```

7884 \def\forest@sort@inner@arrays#1#2{%

```

```

7885 \c@pgf@counta=0
7886 \safeloop
7887 \ifnum\c@pgf@counta<\csname#1n\endcsname
7888 \c@pgf@countb=\csname#1\the\c@pgf@counta @n\endcsname\relax
7889 \ifnum\c@pgf@countb>1
7890 \advance\c@pgf@countb -1
7891 \forest@ppi@deflet{#1\the\c@pgf@counta @}%
7892 \forest@ppi@defcmp{#1\the\c@pgf@counta @}%
7893 \forest@sort\forest@ppi@cmp\forest@ppi@let#2{0}{\the\c@pgf@countb}%
7894 \fi
7895 \advance\c@pgf@counta 1
7896 \saferepeat
7897 }

```

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.

```

7898 \def\forest@ppi@deflet#1{%
7899 \edef\forest@ppi@let##1##2{%
7900 \noexpand\csletcs{#1##1x}{#1##2x}%
7901 \noexpand\csletcs{#1##1y}{#1##2y}%
7902 \noexpand\csletcs{#1##1d}{#1##2d}%
7903 }%
7904 }

```

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.

```

7905 \def\forest@ppi@defcmp#1{%
7906 \edef\forest@ppi@cmp##1##2{%
7907 \noexpand\forest@sort@cmpdimcs{#1##1d}{#1##2d}%
7908 }%
7909 }

```

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

```

7910 \let\forest@inpath\advance

```

This macro is just a wrapper to process the path.

```

7911 \def\forest@pathtodict#1#2{%
7912 \edef\forest@pathtodict@prefix{#2}%
7913 \forest@save@pgfsyssoftpath@tokendefs
7914 \let\pgfsyssoftpath@movetotoken\forest@pathtodict@movetooop
7915 \let\pgfsyssoftpath@linetotoken\forest@pathtodict@linetooop
7916 \def\forest@pathtodict@subpathstart{}%
7917 #1%
7918 \forest@restore@pgfsyssoftpath@tokendefs
7919 }

```

When a move-to operation is encountered:

```

7920 \def\forest@pathtodict@movetooop#1#2{%

```

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

```

7921 \def\forest@pathtodict@subpathstart{(#1,#2)-}%
7922 }

```

When a line-to operation is encountered:

```

7923 \def\forest@pathtodict@linetooop#1#2{%

```

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

```

7924 \if\relax\forest@pathtodict@subpathstart\relax\else
7925 \let\forest@pathtodict@from\forest@pathtodict@subpathstart
7926 \fi

```

Mark the segment as existing.

```
7927 \expandafter\let\csname\forest@pathtodict@prefix\forest@pathtodict@from-(#1,#2)\endcsname\forest@inpath
```

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

```
7928 \def\forest@pathtodict@from{(#1,#2)-}%
7929 \def\forest@pathtodict@subpathstart{}%
7930 }
```

In this macro, the edge is actually computed.

```
7931 \def\forest@gettightedgeofpath@getedge#1{% cs to store the edge into
```

Clear the path and the last projection.

```
7932 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
7933 \let\forest@last@x\relax
7934 \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.)

```
7935 \c@pgf@counta=0
7936 \forest@temp@count=\forest@pi@n\relax
7937 \advance\forest@temp@count -1
7938 \edef\forest@nminusone{\the\forest@temp@count}%
7939 \safeloop
7940 \ifnum\c@pgf@counta<\forest@nminusone\relax
7941 \forest@gettightedgeofpath@getedge@loopa
7942 \saferepeat
```

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

```
7943 \ifnum\forest@nminusone<\forest@n\relax\else
7944 \ifnum\csname forest@pi@\forest@nminusone @n\endcsname>0
7945 \forest@gettightedgeofpath@maybemoveto{\forest@nminusone}{0}%
7946 \fi
7947 \fi
7948 \pgfsyssoftpath@getcurrentpath#1%
7949 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
7950 }
```

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.

```
7951 \def\forest@gettightedgeofpath@getedge@loopa{%
7952 \ifnum\csname forest@pi@\the\c@pgf@counta @n\endcsname>0
```

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

```
7953 \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.

```
7954 \c@pgf@countb=0
7955 \safeloop
7956 \ifnum\c@pgf@countb<\csname forest@pi@\the\c@pgf@counta @n\endcsname\relax
7957 \forest@gettightedgeofpath@getedge@loopb
7958 \saferepeat
7959 \fi
7960 \advance\c@pgf@counta 1
7961 }
```

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

```
7962 \def\forest@gettightedgeofpath@getedge@loopb{%
7963 \c@pgf@countc=0
7964 \advance\c@pgf@counta 1
```



```

7965 \edef\forest@aplusone{\the\c@pgf@counta}%
7966 \advance\c@pgf@counta -1
7967 \safeloop
7968 \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!

```

7969 \forest@tempfalse
7970 \expandafter\ifx\csname forest@pi@(%
7971 \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb x\endcsname,%
7972 \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb y\endcsname)--(%
7973 \csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname,%
7974 \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname)%
7975 \endcsname\forest@inpath
7976 \forest@temptrue
7977 \else
7978 \expandafter\ifx\csname forest@pi@(%
7979 \csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname,%
7980 \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname)--(%
7981 \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb x\endcsname,%
7982 \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb y\endcsname)%
7983 \endcsname\forest@inpath
7984 \forest@temptrue
7985 \fi
7986 \fi
7987 \ifforest@temp

```

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

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

```

7988 \forest@gettightedgeofpath@maybemoveto{\the\c@pgf@counta}{\the\c@pgf@countb}%

```

Second, create a line to the end point.

```

7989 \edef\forest@last@x{%
7990 \csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname}%
7991 \edef\forest@last@y{%
7992 \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname}%
7993 \pgfsyssoftpath@lineto\forest@last@x\forest@last@y

```

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

```

7994 \c@pgf@countc=\csname forest@pi@\forest@aplusone @n\endcsname
7995 \c@pgf@countb=\csname forest@pi@\the\c@pgf@counta @n\endcsname
7996 \fi
7997 \advance\c@pgf@countc 1
7998 \saferepeat
7999 \advance\c@pgf@countb 1
8000 }

```

`\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.

```

8001 \def\forest@gettightedgeofpath@maybemoveto#1#2{%

```

```

8002 \forest@temptrue
8003 \ifx\forest@last@x\relax\else
8004   \ifdim\forest@last@x=\csname forest@pi@#1@#2x\endcsname\relax
8005     \ifdim\forest@last@y=\csname forest@pi@#1@#2y\endcsname\relax
8006       \forest@tempfalse
8007     \fi
8008   \fi
8009 \fi
8010 \ifforest@temp
8011   \edef\forest@last@x{\csname forest@pi@#1@#2x\endcsname}%
8012   \edef\forest@last@y{\csname forest@pi@#1@#2y\endcsname}%
8013   \pgfsyssoftpath@moveto\forest@last@x\forest@last@y
8014 \fi
8015 }

```

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

```

8016 \def\forest@simplifypath#1#2{%
8017   \pgfsyssoftpath@setcurrentpath\pgfutil@empty
8018   \forest@save\pgfsyssoftpath@tokendefs
8019   \let\pgfsyssoftpath@movetotoken\forest@simplifypath@moveto
8020   \let\pgfsyssoftpath@linetotoken\forest@simplifypath@lineto
8021   \let\forest@last@x\relax
8022   \let\forest@last@y\relax
8023   \let\forest@last@atan\relax
8024   #1%
8025   \ifx\forest@last@x\relax\else
8026     \ifx\forest@last@atan\relax\else
8027       \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
8028     \fi
8029   \fi
8030   \forest@restore\pgfsyssoftpath@tokendefs
8031   \pgfsyssoftpath@getcurrentpath#2%
8032   \pgfsyssoftpath@setcurrentpath\pgfutil@empty
8033 }

```

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.

```

8034 \def\forest@simplifypath@moveto#1#2{%
8035   \ifx\forest@last@x\relax\else
8036     \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
8037   \fi
8038   \pgfsyssoftpath@moveto{#1}{#2}%
8039   \def\forest@last@x{#1}%
8040   \def\forest@last@y{#2}%
8041   \let\forest@last@atan\relax
8042 }

```

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

```

8043 \def\forest@getedgeofpath@precision{1pt}

```

When a line-to is encountered...

```

8044 \def\forest@simplifypath@lineto#1#2{%
8045   \ifx\forest@last@x\relax

```

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

```

8046     \def\forest@last@x{#1}%
8047     \def\forest@last@y{#2}%
8048     \let\forest@last@atan\relax
8049   \else

```

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

```

8050 \pgfpointdiff{\pgfqpoint{#1}{#2}}{\pgfqpoint{\forest@last@x}{\forest@last@y}}%
8051 \ifdim\pgf@x<\pgfintersectiontolerance
8052 \ifdim-\pgf@x<\pgfintersectiontolerance
8053 \pgf@x=0pt
8054 \fi
8055 \fi
8056 \edef\forest@marshal{%
8057 \noexpand\pgfmthatantwo@
8058 {\expandafter\pgf@geT\the\pgf@x}%
8059 {\expandafter\pgf@geT\the\pgf@y}%
8060 }\forest@marshal
8061 \let\forest@current@atan\pgfmathresult
8062 \ifx\forest@last@atan\relax

```

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

```

8063 \def\forest@last@x{#1}%
8064 \def\forest@last@y{#2}%
8065 \let\forest@last@atan\forest@current@atan
8066 \else

```

Otherwise, compare the first and the current slope.

```

8067 \pgfutil@tempdima=\forest@current@atan pt
8068 \advance\pgfutil@tempdima -\forest@last@atan pt
8069 \ifdim\pgfutil@tempdima<0pt\relax
8070 \multiply\pgfutil@tempdima -1
8071 \fi
8072 \ifdim\pgfutil@tempdima<\forest@getedgeofpath@precision\relax
8073 \else

```

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

```

8074 \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
8075 \let\forest@last@atan\forest@current@atan
8076 \fi

```

In any event, update the last point.

```

8077 \def\forest@last@x{#1}%
8078 \def\forest@last@y{#2}%
8079 \fi
8080 \fi
8081 }

```

9.4 Get rectangle/band edge

```

8082 \def\forest@getnegativerectangleedgeofpath#1#2{%
8083 \forest@getnegativerectangleorbandededgeofpath{#1}{#2}{\the\pgf@xb}}
8084 \def\forest@getpositiverectangleedgeofpath#1#2{%
8085 \forest@getpositiverectangleorbandededgeofpath{#1}{#2}{\the\pgf@xb}}
8086 \def\forest@getbothrectangleedgesofpath#1#2#3{%
8087 \forest@getbothrectangleorbandededgesofpath{#1}{#2}{#3}{\the\pgf@xb}}
8088 \def\forest@bandlength{5000pt} % something large (ca. 180cm), but still manageable for TeX without producing
8089 \def\forest@getnegativebandededgeofpath#1#2{%
8090 \forest@getnegativerectangleorbandededgeofpath{#1}{#2}{\forest@bandlength}}
8091 \def\forest@getpositivebandededgeofpath#1#2{%
8092 \forest@getpositiverectangleorbandededgeofpath{#1}{#2}{\forest@bandlength}}
8093 \def\forest@getbothbandededgesofpath#1#2#3{%
8094 \forest@getbothrectangleorbandededgesofpath{#1}{#2}{#3}{\forest@bandlength}}
8095 \def\forest@getnegativerectangleorbandededgeofpath#1#2#3{%
8096 \forest@path@getboundingrectangle@ls#1{\forest@grow}%
8097 \edef\forest@gre@path{%
8098 \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@ya}%

```

```

8099 \noexpand\pgfsyssoftpath@linetotoken{#3}{\the\pgf@ya}%
8100 }%
8101 {%
8102 \pgftransformreset
8103 \forest@pgfqtransformrotate{\forest@grow}%
8104 \forest@pgfpathtransformed\forest@gre@path
8105 }%
8106 \pgfsyssoftpath@getcurrentpath#2%
8107 }
8108 \def\forest@getpositiverectangleorbandedgeofpath#1#2#3{%
8109 \forest@path@getboundingrectangle@ls#1{\forest@grow}%
8110 \edef\forest@gre@path{%
8111 \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@yb}%
8112 \noexpand\pgfsyssoftpath@linetotoken{#3}{\the\pgf@yb}%
8113 }%
8114 {%
8115 \pgftransformreset
8116 \forest@pgfqtransformrotate{\forest@grow}%
8117 \forest@pgfpathtransformed\forest@gre@path
8118 }%
8119 \pgfsyssoftpath@getcurrentpath#2%
8120 }
8121 \def\forest@getbothrectangleorbandedgesofpath#1#2#3#4{%
8122 \forest@path@getboundingrectangle@ls#1{\forest@grow}%
8123 \edef\forest@gre@negpath{%
8124 \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@ya}%
8125 \noexpand\pgfsyssoftpath@linetotoken{#4}{\the\pgf@ya}%
8126 }%
8127 \edef\forest@gre@pospath{%
8128 \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@yb}%
8129 \noexpand\pgfsyssoftpath@linetotoken{#4}{\the\pgf@yb}%
8130 }%
8131 {%
8132 \pgftransformreset
8133 \forest@pgfqtransformrotate{\forest@grow}%
8134 \forest@pgfpathtransformed\forest@gre@negpath
8135 }%
8136 \pgfsyssoftpath@getcurrentpath#2%
8137 {%
8138 \pgftransformreset
8139 \forest@pgfqtransformrotate{\forest@grow}%
8140 \forest@pgfpathtransformed\forest@gre@pospath
8141 }%
8142 \pgfsyssoftpath@getcurrentpath#3%
8143 }

```

9.5 Distance between paths

Another crucial part of the package.

```

8144 \def\forest@distance@between@edge@paths#1#2#3{%
8145 % #1, #2 = (edge) paths
8146 %
8147 % project paths
8148 \forest@projectpathtogrowline#1{\forest@p1@}%
8149 \forest@projectpathtogrowline#2{\forest@p2@}%
8150 % merge projections (the lists are sorted already, because edge
8151 % paths are |sorted|)
8152 \forest@dbep@mergeprojections
8153 {\forest@p1@}{\forest@p2@}%
8154 {\forest@P1@}{\forest@P2@}%
8155 % process projections

```

```

8156 \forest@processprojectioninfo{forest@P1@}{forest@PI1@}%
8157 \forest@processprojectioninfo{forest@P2@}{forest@PI2@}%
8158 % break paths
8159 \forest@breakpath#1{forest@PI1@}\forest@broken@one
8160 \forest@breakpath#2{forest@PI2@}\forest@broken@two
8161 % sort inner arrays ---optimize: it's enough to find max and min
8162 \forest@sort@inner@arrays{forest@PI1@}\forest@sort@descending
8163 \forest@sort@inner@arrays{forest@PI2@}\forest@sort@ascending
8164 % compute the distance
8165 \let\forest@distance\relax
8166 \c@pgf@countc=0
8167 \forest@loop
8168 \ifnum\c@pgf@countc<\csname forest@PI1@n\endcsname\relax
8169 \ifnum\csname forest@PI1@the\c@pgf@countc @n\endcsname=0 \else
8170 \ifnum\csname forest@PI2@the\c@pgf@countc @n\endcsname=0 \else
8171 \pgfutil@tempdima=\csname forest@PI2@the\c@pgf@countc @0d\endcsname\relax
8172 \advance\pgfutil@tempdima -\csname forest@PI1@the\c@pgf@countc @0d\endcsname\relax
8173 \ifx\forest@distance\relax
8174 \edef\forest@distance{\the\pgfutil@tempdima}%
8175 \else
8176 \ifdim\pgfutil@tempdima<\forest@distance\relax
8177 \edef\forest@distance{\the\pgfutil@tempdima}%
8178 \fi
8179 \fi
8180 \fi
8181 \fi
8182 \advance\c@pgf@countc 1
8183 \forest@repeat
8184 \let#3\forest@distance
8185 }
8186 % merge projections: we need two projection arrays, both containing
8187 % projection points from both paths, but each with the original
8188 % points from only one path
8189 \def\forest@dbep@mergeprojections#1#2#3#4{%
8190 % TODO: optimize: v bistvu ni treba sortirat, ker je edge path e sortiran
8191 \forest@sortprojections{#1}%
8192 \forest@sortprojections{#2}%
8193 \c@pgf@counta=0
8194 \c@pgf@countb=0
8195 \c@pgf@countc=0
8196 \edef\forest@input@prefix@one{#1}%
8197 \edef\forest@input@prefix@two{#2}%
8198 \edef\forest@output@prefix@one{#3}%
8199 \edef\forest@output@prefix@two{#4}%
8200 \forest@dbep@mp@iterate
8201 \csedef{#3n}{\the\c@pgf@countc}%
8202 \csedef{#4n}{\the\c@pgf@countc}%
8203 }
8204 \def\forest@dbep@mp@iterate{%
8205 \let\forest@dbep@mp@next\forest@dbep@mp@iterate
8206 \ifnum\c@pgf@counta<\csname\forest@input@prefix@one n\endcsname\relax
8207 \ifnum\c@pgf@countb<\csname\forest@input@prefix@two n\endcsname\relax
8208 \let\forest@dbep@mp@next\forest@dbep@mp@do
8209 \else
8210 \let\forest@dbep@mp@next\forest@dbep@mp@iteratefirst
8211 \fi
8212 \else
8213 \ifnum\c@pgf@countb<\csname\forest@input@prefix@two n\endcsname\relax
8214 \let\forest@dbep@mp@next\forest@dbep@mp@iteratesecond
8215 \else
8216 \let\forest@dbep@mp@next\relax

```

```

8217 \fi
8218 \fi
8219 \forest@dbep@mp@next
8220 }
8221 \def\forest@dbep@mp@do{%
8222 \forest@sort@cmtwodimcs%
8223 {\forest@input@prefix@one\the\c@pgf@counta xp}%
8224 {\forest@input@prefix@one\the\c@pgf@counta yp}%
8225 {\forest@input@prefix@two\the\c@pgf@countb xp}%
8226 {\forest@input@prefix@two\the\c@pgf@countb yp}%
8227 \if\forest@sort@cmp@result=%
8228 \forest@dbep@mp@@store@p\forest@input@prefix@one\c@pgf@counta
8229 \forest@dbep@mp@@store@o\forest@input@prefix@one
8230 \c@pgf@counta\forest@output@prefix@one
8231 \forest@dbep@mp@@store@o\forest@input@prefix@two
8232 \c@pgf@countb\forest@output@prefix@two
8233 \advance\c@pgf@counta 1
8234 \advance\c@pgf@countb 1
8235 \else
8236 \if\forest@sort@cmp@result>%
8237 \forest@dbep@mp@@store@p\forest@input@prefix@two\c@pgf@countb
8238 \forest@dbep@mp@@store@o\forest@input@prefix@two
8239 \c@pgf@countb\forest@output@prefix@two
8240 \advance\c@pgf@countb 1
8241 \else%<
8242 \forest@dbep@mp@@store@p\forest@input@prefix@one\c@pgf@counta
8243 \forest@dbep@mp@@store@o\forest@input@prefix@one
8244 \c@pgf@counta\forest@output@prefix@one
8245 \advance\c@pgf@counta 1
8246 \fi
8247 \fi
8248 \advance\c@pgf@countc 1
8249 \forest@dbep@mp@iterate
8250 }
8251 \def\forest@dbep@mp@@store@p#1#2{%
8252 \csletcs
8253 {\forest@output@prefix@one\the\c@pgf@countc xp}%
8254 {#1\the#2xp}%
8255 \csletcs
8256 {\forest@output@prefix@one\the\c@pgf@countc yp}%
8257 {#1\the#2yp}%
8258 \csletcs
8259 {\forest@output@prefix@two\the\c@pgf@countc xp}%
8260 {#1\the#2xp}%
8261 \csletcs
8262 {\forest@output@prefix@two\the\c@pgf@countc yp}%
8263 {#1\the#2yp}%
8264 }
8265 \def\forest@dbep@mp@@store@o#1#2#3{%
8266 \csletcs{#3\the\c@pgf@countc xo}{#1\the#2xo}%
8267 \csletcs{#3\the\c@pgf@countc yo}{#1\the#2yo}%
8268 }
8269 \def\forest@dbep@mp@iteratefirst{%
8270 \forest@dbep@mp@iterateone\forest@input@prefix@one\c@pgf@counta\forest@output@prefix@one
8271 }
8272 \def\forest@dbep@mp@iteratesecond{%
8273 \forest@dbep@mp@iterateone\forest@input@prefix@two\c@pgf@countb\forest@output@prefix@two
8274 }
8275 \def\forest@dbep@mp@iterateone#1#2#3{%
8276 \forest@loop
8277 \ifnum#2<\csname#1n\endcsname\relax

```

```

8278 \forest@dbep@mp@@store@p#1#2%
8279 \forest@dbep@mp@@store@o#1#2#3%
8280 \advance\c@pgf@countc 1
8281 \advance#21
8282 \forest@repeat
8283 }

```

9.6 Utilities

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

```

8284 \newif\ifforest@equaltotolerance
8285 \def\forest@equaltotolerance#1#2{%
8286 \pgfpointdiff{#1}{#2}%
8287 \ifdim\pgf@x<Opt \multiply\pgf@x -1 \fi
8288 \ifdim\pgf@y<Opt \multiply\pgf@y -1 \fi
8289 \global\forest@equaltotolerancefalse
8290 \ifdim\pgf@x<\pgfintersectiontolerance\relax
8291 \ifdim\pgf@y<\pgfintersectiontolerance\relax
8292 \global\forest@equaltolerancetrue
8293 \fi
8294 \fi
8295 }}

```

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

```

8296 \def\forest@save@pgfsyssoftpath@tokendefs{%
8297 \let\forest@origmovetotoken\pgfsyssoftpath@movetotoken
8298 \let\forest@origlinetotoken\pgfsyssoftpath@linetotoken
8299 \let\forest@origcurvetosupportatoken\pgfsyssoftpath@curvetosupportatoken
8300 \let\forest@origcurvetosupportbtoken\pgfsyssoftpath@curvetosupportbtoken
8301 \let\forest@origcurvetotoken\pgfsyssoftpath@curvetototoken
8302 \let\forest@origrectcornertoken\pgfsyssoftpath@rectcornertoken
8303 \let\forest@origrectsizetoken\pgfsyssoftpath@rectsizetoken
8304 \let\forest@origclosepathtoken\pgfsyssoftpath@closepathtoken
8305 \let\pgfsyssoftpath@movetotoken\forest@badtoken
8306 \let\pgfsyssoftpath@linetotoken\forest@badtoken
8307 \let\pgfsyssoftpath@curvetosupportatoken\forest@badtoken
8308 \let\pgfsyssoftpath@curvetosupportbtoken\forest@badtoken
8309 \let\pgfsyssoftpath@curvetototoken\forest@badtoken
8310 \let\pgfsyssoftpath@rectcornertoken\forest@badtoken
8311 \let\pgfsyssoftpath@rectsizetoken\forest@badtoken
8312 \let\pgfsyssoftpath@closepathtoken\forest@badtoken
8313 }
8314 \def\forest@badtoken{%
8315 \PackageError{forest}{This token should not be in this path}{}%
8316 }
8317 \def\forest@restore@pgfsyssoftpath@tokendefs{%
8318 \let\pgfsyssoftpath@movetotoken\forest@origmovetotoken
8319 \let\pgfsyssoftpath@linetotoken\forest@origlinetotoken
8320 \let\pgfsyssoftpath@curvetosupportatoken\forest@origcurvetosupportatoken
8321 \let\pgfsyssoftpath@curvetosupportbtoken\forest@origcurvetosupportbtoken
8322 \let\pgfsyssoftpath@curvetototoken\forest@origcurvetotoken
8323 \let\pgfsyssoftpath@rectcornertoken\forest@origrectcornertoken
8324 \let\pgfsyssoftpath@rectsizetoken\forest@origrectsizetoken
8325 \let\pgfsyssoftpath@closepathtoken\forest@origclosepathtoken
8326 }

```

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

```

8327 \def\forest@extendpath#1#2#3{%
8328 \pgf@process{#3}%
8329 \pgfsyssoftpath@setcurrentpath#1%

```

```

8330 \forest@save@pgfsyssoftpath@tokendefs
8331 \let\pgfsyssoftpath@movetotoken\forest@extendpath@moveto
8332 \let\pgfsyssoftpath@linetotoken\forest@extendpath@lineto
8333 #2%
8334 \forest@restore@pgfsyssoftpath@tokendefs
8335 \pgfsyssoftpath@getcurrentpath#1%
8336 }
8337 \def\forest@extendpath@moveto#1#2{%
8338 \forest@extendpath@do{#1}{#2}\pgfsyssoftpath@moveto
8339 }
8340 \def\forest@extendpath@lineto#1#2{%
8341 \forest@extendpath@do{#1}{#2}\pgfsyssoftpath@lineto
8342 }
8343 \def\forest@extendpath@do#1#2#3{%
8344 {%
8345 \advance\pgf@x #1
8346 \advance\pgf@y #2
8347 #3{\the\pgf@x}{\the\pgf@y}%
8348 }%
8349 }

```

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.)

```

8350 \def\forest@path@getboundingrectangle@ls#1#2{%
8351 {%
8352 \pgftransformreset
8353 \forest@pgfqtransformrotate{-#2}%
8354 \forest@pgfpathtransformed#1%
8355 }%
8356 \pgfsyssoftpath@getcurrentpath\forest@gbr@rotatedpath
8357 \forest@path@getboundingrectangle@xy\forest@gbr@rotatedpath
8358 }
8359 \def\forest@path@getboundingrectangle@xy#1{%
8360 \forest@save@pgfsyssoftpath@tokendefs
8361 \let\pgfsyssoftpath@movetotoken\forest@gbr@firstpoint
8362 \let\pgfsyssoftpath@linetotoken\forest@gbr@firstpoint
8363 #1%
8364 \forest@restore@pgfsyssoftpath@tokendefs
8365 }
8366 \def\forest@gbr@firstpoint#1#2{%
8367 \pgf@xa=#1 \pgf@xb=#1 \pgf@ya=#2 \pgf@yb=#2
8368 \let\pgfsyssoftpath@movetotoken\forest@gbr@point
8369 \let\pgfsyssoftpath@linetotoken\forest@gbr@point
8370 }
8371 \def\forest@gbr@point#1#2{%
8372 \ifdim#1<\pgf@xa\relax\pgf@xa=#1 \fi
8373 \ifdim#1>\pgf@xb\relax\pgf@xb=#1 \fi
8374 \ifdim#2<\pgf@ya\relax\pgf@ya=#2 \fi
8375 \ifdim#2>\pgf@yb\relax\pgf@yb=#2 \fi
8376 }

```

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.

```

8377 \let\forest@pgfqtransformrotate\pgftransformrotate
8378 \let\forest@pgftransformcm\pgftransformcm
8379 \let\forest@pgf@transformcm\pgf@transformcm
8380 \patchcmd{\forest@pgfqtransformrotate}{\pgfmathparse{#1}}{\edef\pgfmathresult{\number\numexpr#1}}{}{}
8381 \patchcmd{\forest@pgfqtransformrotate}{\pgftransformcm}{\forest@pgftransformcm}{}{}
8382 \patchcmd{\forest@pgftransformcm}{\pgf@transformcm}{\forest@pgf@transformcm}{}{}
8383 \patchcmd{\forest@pgf@transformcm}{\pgfmathsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x
8384 \patchcmd{\forest@pgf@transformcm}{\pgfmathsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x

```



```

8385 \patchcmd{\forest@pgf@transformcm}{\pgfmithsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x
8386 \patchcmd{\forest@pgf@transformcm}{\pgfmithsetlength}{\forest@pgf@transformcm@setlength}{}{} % 4x
8387 \def\forest@pgf@transformcm@setlength#1#2{#1=#2pt}

```

10 The outer UI

10.1 Externalization

```

8388 \pgfkeys{/forest/external/.cd,
8389 %copy command/.initial={cp "\source" "\target"},
8390 copy command/.initial={},
8391 optimize/.is if=forest@external@optimize@,
8392 context/.initial={%
8393   \forestOve{csname forest@id@of@standard node\endcsname}{environment@formula}},
8394 depends on macro/.style={context/.append/.expanded={%
8395   \expandafter\detokenize\expandafter{#1}}},
8396 }
8397 \def\forest@file@copy#1#2{%
8398   \IfFileExists{#1}{%
8399     \pgfkeysgetvalue{/forest/external/copy command}\forest@copy@command
8400     \ifdefempty\forest@copy@command{%
8401       \forest@file@copy@{#1}{#2}%
8402     }{ % copy by external command
8403       \def\source{#1}%
8404       \def\target{#2}%
8405       \immediate\write18{\forest@copy@command}%
8406     }%
8407   }{}%
8408 }
8409 \newread\forest@copy@in
8410 \newwrite\forest@copy@out
8411 \def\forest@file@copy@#1#2{%
8412   \begingroup
8413   \openin\forest@copy@in=#1
8414   \immediate\openout\forest@copy@out#2
8415   \endlinechar-1
8416   \loop
8417   \unless\ifeof\forest@copy@in
8418     \readline\forest@copy@in to\forest@temp
8419     \immediate\write\forest@copy@out{\forest@temp}%
8420   \repeat
8421   \immediate\closeout\forest@copy@out
8422   \closein\forest@copy@in
8423   \endgroup
8424 }
8425 \newif\ifforest@external@optimize@
8426 \forest@external@optimize@true
8427 \ifforest@install@keys@to@tikz@path@
8428 \tikzset{
8429   fit to/.style={
8430     /forest/for nodewalk=%
8431     {TeX={\def\forest@fitto{}}},#1}%
8432     {TeX={\eappto\forest@fitto{(\forestove{name})}}},
8433   fit/.expanded={\forest@fitto}
8434 },
8435 }
8436 \fi
8437 \ifforest@external@
8438   \ifdefined\tikzexternal@tikz@replacement\else
8439     \usetikzlibrary{external}%

```

```

8440 \fi
8441 \pgfkeys{%
8442   /tikz/external/failed ref warnings for={},
8443   /pgf/images/aux in dpth=false,
8444 }%
8445 \tikzifexternalizing{}{%
8446   \forest@file@copy{\jobname.aux}{\jobname.aux.copy}%
8447 }%
8448 \AtBeginDocument{%
8449   \tikzifexternalizing{%
8450     \IfFileExists{\tikzexternalrealjob.aux.copy}{%
8451       \makeatletter
8452       \input\tikzexternalrealjob.aux.copy\relax
8453       \makeatother
8454     }{}%
8455   }{%
8456     \newwrite\forest@auxout
8457     \immediate\openout\forest@auxout=\tikzexternalrealjob.for.tmp
8458   }%
8459   \IfFileExists{\tikzexternalrealjob.for}{%
8460     {%
8461       \makehashother\makeatletter
8462       \input\tikzexternalrealjob.for\relax
8463     }%
8464   }{}%
8465 }%
8466 \AtEndDocument{%
8467   \tikzifexternalizing{}{%
8468     \immediate\closeout\forest@auxout
8469     \forest@file@copy{\jobname.for.tmp}{\jobname.for}%
8470   }%
8471 }%
8472 \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.

```

8473 \NewDocumentEnvironment{forest}{D(){} }{%
8474   \forest@config{#1}%
8475   \Collect@Body
8476   \forest@env
8477 }{}
8478 \NewDocumentCommand{\Forest}{s D(){} m}{%
8479   \forest@config{#2}%
8480   \IfBooleanTF{#1}{\let\forest@next\forest@env}{\let\forest@next\forest@group@env}%
8481   \forest@next{#3}%
8482 }
8483 \def\forest@config#1{%
8484   \forest@defstages{stages}%
8485   \forestset{@config/.cd,#1}%
8486 }
8487 \def\forest@defstages#1{%
8488   \def\forest@stages{#1}%
8489 }
8490 \forestset{@config/.cd,
8491   %stages/.store in=\forest@stages,
8492   stages/.code={\forest@defstages{#1}},
8493   .unknown/.code={\PackageError{forest}{Unknown config option for forest environment/command.}{In Forest v2.0
8494 }

```

```

8495 \def\forest@group@env#1{\forest@env{#1}}
8496 \newif\ifforest@externalize@tree@
8497 \newif\ifforest@was@tikzexternalwasenable
8498 \newcommand\forest@env[1]{%
8499   \let\forest@external@next\forest@begin
8500   \forest@was@tikzexternalwasenablefalse
8501   \ifdefined\tikzexternal@tikz@replacement
8502     \ifx\tikz\tikzexternal@tikz@replacement
8503       \forest@was@tikzexternalwasenabletrue
8504       \tikzexternaldisable
8505     \fi
8506   \fi
8507   \forest@externalize@tree@false
8508   \ifforest@external@
8509     \ifforest@was@tikzexternalwasenable
8510     \forest@env@
8511   \fi
8512   \fi
8513   \forest@standardnode@calibrate
8514   \forest@external@next{#1}%
8515 }
8516 \def\forest@env{%
8517   \iftikzexternalexportnext
8518     \tikzifexternalizing{%
8519       \let\forest@external@next\forest@begin@externalizing
8520     }{%
8521       \let\forest@external@next\forest@begin@externalize
8522     }%
8523   \else
8524     \tikzexternalexportnexttrue
8525   \fi
8526 }

```

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

```

8527 \long\def\forest@begin@externalizing#1{%
8528   \forest@external@setup{#1}%
8529   \let\forest@external@next\forest@begin
8530   \forest@externalize@inner@n=-1
8531   \ifforest@external@optimize@\forest@externalizing@maybeoptimize\fi
8532   \forest@external@next{#1}%
8533   \tikzexternalenable
8534 }
8535 \def\forest@externalizing@maybeoptimize{%
8536   \edef\forest@temp{\tikzexternalrealjob-forest-\forest@externalize@outer@n}%
8537   \edef\forest@marshal{%
8538     \noexpand\pgfutil@in@
8539     {\expandafter\detokenize\expandafter{\forest@temp}.}
8540     {\expandafter\detokenize\expandafter{\pgfactualjobname}.}%
8541   }\forest@marshal
8542   \ifpgfutil@in@
8543   \else
8544     \let\forest@external@next\@gobble
8545   \fi
8546 }

```

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

```

8547 \long\def\forest@begin@externalize#1{%
8548   \forest@external@setup{#1}%
8549   \iftikzexternal@file@isuptodate
8550     \setbox0=\hbox{%
8551       \csname forest@externalcheck@\forest@externalize@outer@n\endcsname

```

```

8552 }%
8553 \fi
8554 \iftikzexternal@file@isuptodate
8555 \csname forest@externalload@\forest@externalize@outer@n\endcsname
8556 \else
8557 \forest@externalize@tree@true
8558 \forest@externalize@inner@n=-1
8559 \forest@begin{#1}%
8560 \ifcsdef{forest@externalize@@\forest@externalize@id}{-}{%
8561 \immediate\write\forest@auxout{%
8562 \noexpand\forest@external
8563 {\forest@externalize@outer@n}%
8564 {\expandafter\detokenize\expandafter{\forest@externalize@id}}%
8565 {\expandonce\forest@externalize@checkimages}%
8566 {\expandonce\forest@externalize@loadimages}%
8567 }%
8568 }%
8569 \fi
8570 \tikzexternalenable
8571 }
8572 \def\forest@includeexternal@check#1{%
8573 \tikzsetnextfilename{#1}%
8574 \IfFileExists{\tikzexternal@filenameprefix/#1}{\tikzexternal@file@isuptodate true}{\tikzexternal@file@isuptodate false}%
8575 }
8576 \def\makehashother{\catcode'\# =12}%
8577 \long\def\forest@external@setup#1{%
8578 % set up \forest@externalize@id and \forest@externalize@outer@n
8579 % we need to deal with #s correctly (\write doubles them)
8580 \setbox0=\hbox{\makehashother\makeatletter
8581 \scantokens{\forest@temp@toks{#1}}\expandafter
8582 }%
8583 \expandafter\forest@temp@toks\expandafter{\the\forest@temp@toks}%
8584 \edef\forest@temp{\pgfkeysvalueof{/forest/external/context}}%
8585 \edef\forest@externalize@id{%
8586 \expandafter\detokenize\expandafter{\forest@temp}%
8587 @@%
8588 \expandafter\detokenize\expandafter{\the\forest@temp@toks}%
8589 }%
8590 \letcs\forest@externalize@outer@n\forest@externalize@@\forest@externalize@id}%
8591 \ifdefined\forest@externalize@outer@n
8592 \global\tikzexternal@file@isuptodate true
8593 \else
8594 \global\advance\forest@externalize@max@outer@n 1
8595 \edef\forest@externalize@outer@n{\the\forest@externalize@max@outer@n}%
8596 \global\tikzexternal@file@isuptodate false
8597 \fi
8598 \def\forest@externalize@loadimages{}%
8599 \def\forest@externalize@checkimages{}%
8600 }
8601 \newcount\forest@externalize@max@outer@n
8602 \global\forest@externalize@max@outer@n=0
8603 \newcount\forest@externalize@inner@n

```

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

```

8604 \long\def\forest@external#1#2#3#4{% #1=n,#2=context+source code,#3=update check code, #4=load code
8605 \ifnum\forest@externalize@max@outer@n<#1
8606 \global\forest@externalize@max@outer@n=#1
8607 \fi
8608 \global\csdef{forest@externalize@@\detokenize{#2}}{#1}%
8609 \global\csdef{forest@externalcheck@#1}{#3}%
8610 \global\csdef{forest@externalload@#1}{#4}%

```

```

8611 \tikzifexternalizing{}{%
8612   \immediate\write\forest@auxout{%
8613     \noexpand\forest@external{#1}%
8614     {\expandafter\detokenize\expandafter{#2}}%
8615     {\unexpanded{#3}}%
8616     {\unexpanded{#4}}%
8617   }%
8618 }%
8619 }

```

These two macros include the external picture.

```

8620 \def\forest@includeexternal#1{%
8621   \edef\forest@temp{\pgfkeysvalueof{/forest/external/context}}%
8622   %\typeout{forest: Including external picture '#1' for forest context+code: '\expandafter\detokenize\expandafter{#2}}%
8623   {%
8624     %\def\pgf@declaredraftimage##1##2{\def\pgf@image{\hbox{}}}%
8625     \tikzsetnextfilename{#1}%
8626     \tikzexternalenable
8627     \tikz{}%
8628   }%
8629 }
8630 \def\forest@includeexternal@box#1#2{%
8631   \global\setbox#1=\hbox{\forest@includeexternal{#2}}%
8632 }

```

This code runs the bracket parser and stage processing.

```

8633 \long\def\forest@begin#1{%
8634   \iffalse{\fi\forest@parsebracket#1}%
8635 }
8636 \def\forest@parsebracket{%
8637   \bracketParse{\forest@get@root@afterthought}\forest@root=%
8638 }
8639 \def\forest@get@root@afterthought{%
8640   \expandafter\forest@get@root@afterthought@\expandafter{\iffalse}\fi
8641 }
8642 \long\def\forest@get@root@afterthought@#1{%
8643   \ifblank{#1}{}%
8644   \forest@oappto{\forest@root}{given options}{,afterthought={\unexpanded{#1}}}%
8645   }%
8646   \forest@do
8647 }
8648 \def\forest@do{%
8649   \forest@node@Compute@numeric@ts@info{\forest@root}%
8650   \expandafter\forestset\expandafter{\forest@stages}%
8651   \ifforest@was@tikzexternalwasenable
8652     \tikzexternalenable
8653   \fi
8654 }

```

10.3 Standard node

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

```

8655 \def\forest@standardnode@new{%
8656   \advance\forest@node@maxid1
8657   \forest@fornode{\the\forest@node@maxid}{%
8658     \forest@node@init
8659     \forestoeset{id}{\forest@cn}%
8660     \forest@node@setname@silent{standard node}%
8661   }%
8662 }

```

```

8663 \def\forest@standardnode@calibrate{%
8664   \forest@fornode{\forest@node@Nametoid{standard node}}{%
8665     \edef\forest@environment{\forest@environment@formula}%
8666     \forest@get{previous@environment}\forest@previous@environment
8667     \ifx\forest@environment\forest@previous@environment\else
8668       \forest@let{previous@environment}\forest@environment
8669       \forest@node@typeset
8670       \forest@get{calibration@procedure}\forest@temp
8671       \expandafter\forest@set\expandafter{\forest@temp}%
8672     \fi
8673   }%
8674 }

```

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

```

8675 \def\forestStandardNode[#1]#2#3#4{%
8676   \let\forest@standardnode@restoretikzexternal\relax
8677   \ifdefined\tikzexternaldisable
8678     \ifx\tikz\tikzexternal\tikz@replacement
8679       \tikzexternaldisable
8680       \let\forest@standardnode@restoretikzexternal\tikzexternalenable
8681     \fi
8682   \fi
8683   \forest@standardnode@new
8684   \forest@fornode{\forest@node@Nametoid{standard node}}{%
8685     \forest@set{content=#1}%
8686     \forest@set{environment@formula}{#2}%
8687     \edef\forest@temp{\unexpanded{#3}}%
8688     \forest@let{calibration@procedure}\forest@temp
8689     \def\forest@calibration@initializing@code{%
8690       \pgfkeys{/forest/initializing@code}{#4}%
8691       \forest@let{initializing@code}\forest@calibration@initializing@code
8692     \forest@standardnode@restoretikzexternal
8693   }
8694 }
8695 \forest@set{initializing@code/.unknown/.code={%
8696   \eappto\forest@calibration@initializing@code{%
8697     \noexpand\forest@get{\forest@node@Nametoid{standard node}}{\pgfkeyscurrentname}\noexpand\forest@temp
8698     \noexpand\forest@let{\pgfkeyscurrentname}\noexpand\forest@temp
8699   }%
8700 }
8701 }

```

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

```

8702 \def\forest@initializefromstandardnode{%
8703   \forest@ve{\forest@node@Nametoid{standard node}}{initializing@code}%
8704 }

```

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

option (such as draw or blue) as default, set it in #1 and export the (internal) option node options.)

```

8705 \forestStandardNode[dj]
8706   {%
8707     \forestOve{\forest@node@Nametoid{standard node}}{content},%
8708     \the\ht\strutbox,\the\pgflinewidth,%
8709     \pgfkeysvalueof{/pgf/inner ysep},\pgfkeysvalueof{/pgf/outer ysep},%
8710     \pgfkeysvalueof{/pgf/inner xsep},\pgfkeysvalueof{/pgf/outer xsep}%
8711   }
8712   {
8713     l sep'/.expanded={\the\dimexpr\the\ht\strutbox+\pgfkeysvalueof{/pgf/inner ysep}},
8714     l={l_sep()+abs(max_y()-min_y())+2*\pgfkeysvalueof{/pgf/outer ysep}},
8715     s sep'/.expanded={\the\dimexpr \pgfkeysvalueof{/pgf/inner xsep}*2}
8716   }
8717   {l sep,l,s sep}

```

10.4 ls coordinate system

```

8718 \pgfkeys{/forest/@cs}{%
8719   name/.code={%
8720     \edef\forest@cn{\forest@node@Nametoid{#1}}%
8721     \forest@forestcs@resetxy,
8722   id/.code={%
8723     \edef\forest@cn{#1}%
8724     \forest@forestcs@resetxy,
8725   go/.code={%
8726     \forest@go{#1}%
8727     \forest@forestcs@resetxy,
8728   anchor/.code={\forest@forestcs@anchor{#1}},
8729   l/.code={%
8730     \forestmathsetlengthmacro\forest@forestcs@l{#1}%
8731     \forest@forestcs@ls
8732   },
8733   s/.code={%
8734     \forestmathsetlengthmacro\forest@forestcs@s{#1}%
8735     \forest@forestcs@ls
8736   },
8737   .unknown/.code={%
8738     \expandafter\pgfutil@in@\expandafter.\expandafter{\pgfkeyscurrentname}%
8739     \ifpgfutil@in@
8740       \expandafter\forest@forestcs@namegoanchor\pgfkeyscurrentname\forest@end
8741     \else
8742       \expandafter\forest@nameandgo\expandafter{\pgfkeyscurrentname}%
8743       \forest@forestcs@resetxy
8744     \fi
8745   }
8746 }
8747 \def\forest@forestcs@resetxy{%
8748   \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
8749   \global\pgf@x\forestove{x}\relax
8750   \global\pgf@y\forestove{y}\relax
8751 }
8752 \def\forest@forestcs@ls{%
8753   \ifdefined\forest@forestcs@l
8754     \ifdefined\forest@forestcs@s
8755       {%
8756         \pgftransformreset
8757         \forest@pgfqtransformrotate{\forestove{grow}}%
8758         \pgfpointransformed{\pgfpoint{\forest@forestcs@l}{\forest@forestcs@s}}%
8759       }%
8760     \global\advance\pgf@x\forestove{x}%

```

```

8761 \global\advance\pgf@y\forestove{y}%
8762 \fi
8763 \fi
8764 }
8765 \def\forest@forestcs@anchor#1{%
8766 \edef\forest@marshal{%
8767 \noexpand\forest@original@tikz@parse@node\relax
8768 (\forestove{name}\ifx\relax#1\relax\else.\fi#1)%
8769 }\forest@marshal
8770 }
8771 \def\forest@forestcs@namegoanchor#1.#2\forest@end{%
8772 \forest@nameandgo{#1}%
8773 \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
8774 \forest@forestcs@anchor{#2}%
8775 }
8776 \def\forest@cs@invalidnodeerror{%
8777 \PackageError{forest}{Attempt to refer to the invalid node by "forest cs"}{ }%
8778 }
8779 \tikzdeclarecoordinatesystem{forest}{%
8780 \forest@forthis{%
8781 \forest@forestcs@resetxy
8782 \ifdefined\forest@forestcs@l\undef\forest@forestcs@l\fi
8783 \ifdefined\forest@forestcs@s\undef\forest@forestcs@s\fi
8784 \pgfkeys{/forest/@cs}{#1}%
8785 }%
8786 }

```

10.5 Relative node names in TikZ

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

```

8787 \def\forest@tikz@parse@node#1(#2){%
8788 \pgfutil@in@.#2}%
8789 \ifpgfutil@in@
8790 \expandafter\forest@tikz@parse@node@checkiftikzname@withdot
8791 \else%
8792 \expandafter\forest@tikz@parse@node@checkiftikzname@withoutdot
8793 \fi%
8794 #1(#2)\forest@end
8795 }
8796 \def\forest@tikz@parse@node@checkiftikzname@withdot#1(#2.#3)\forest@end{%
8797 \forest@tikz@parse@node@checkiftikzname#1{#2}{.#3}}
8798 \def\forest@tikz@parse@node@checkiftikzname@withoutdot#1(#2)\forest@end{%
8799 \forest@tikz@parse@node@checkiftikzname#1{#2}{}}
8800 \def\forest@tikz@parse@node@checkiftikzname#1#2#3{%
8801 \expandafter\ifx\csname pgf@sh@ns@#2\endcsname\relax
8802 \forest@forthis{%
8803 \forest@nameandgo{#2}%
8804 \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
8805 \edef\forest@temp@relativenodename{\forestove{name}}%
8806 }%
8807 \else
8808 \def\forest@temp@relativenodename{#2}%
8809 \fi
8810 \expandafter\forest@original@tikz@parse@node\expandafter#1\expandafter(\forest@temp@relativenodename#3)%
8811 }
8812 \def\forest@nameandgo#1{%
8813 \pgfutil@in@!{#1}%
8814 \ifpgfutil@in@
8815 \forest@nameandgo@(#1)%
8816 \else
8817 \ifstrempy{#1}{ }\{\edef\forest@cn{\forest@node@Nametoid{#1}}\}%

```



```

8818 \fi
8819 }
8820 \def\forest@nameandgo@(#1!#2){%
8821 \ifstrempy{#1}{\edef\forest@cn{\forest@node@Nametoid{#1}}}%
8822 \forest@go{#2}%
8823 }

```

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.

```

8824 \forestset{
8825   declare count={rotate}{0},
8826   autoforward'={rotate}{node options},
8827 }

```

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

```

8828 \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`.

```

8829 \newif\ifforest@anchor@forwardtotikz

```

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

```

8830 \newif\ifforest@anchor@isborder

```

The UI macro.

```

8831 \def\forestanchortotikzanchor#1#2{% #1 = forest anchor, #2 = macro to receive the tikz anchor
8832 \forest@anchor@forwardtotikzfalse
8833 \forest@anchor@do{#1}{\forest@cn}%
8834 \let#2\forest@temp@anchor
8835 }

```

Generic anchors.

```

8836 \pgfdeclaregenericanchor{child anchor}{%
8837 \forest@anchor@forwardtotikztrue
8838 \forest@anchor@do{#1}{child anchor}{\forest@referencednodeid}%
8839 }
8840 \pgfdeclaregenericanchor{parent anchor}{%
8841 \forest@anchor@forwardtotikztrue
8842 \forest@anchor@do{#1}{parent anchor}{\forest@referencednodeid}%
8843 }
8844 \pgfdeclaregenericanchor{anchor}{%
8845 \forest@anchor@forwardtotikztrue
8846 \forest@anchor@do{#1}{anchor}{\forest@referencednodeid}%
8847 }
8848 \pgfdeclaregenericanchor{children}{%
8849 \forest@anchor@forwardtotikztrue
8850 \forest@anchor@do{#1}{children}{\forest@referencednodeid}%
8851 }
8852 \pgfdeclaregenericanchor{-children}{%
8853 \forest@anchor@forwardtotikztrue
8854 \forest@anchor@do{#1}{-children}{\forest@referencednodeid}%
8855 }
8856 \pgfdeclaregenericanchor{children first}{%
8857 \forest@anchor@forwardtotikztrue
8858 \forest@anchor@do{#1}{children first}{\forest@referencednodeid}%
8859 }
8860 \pgfdeclaregenericanchor{-children first}{%

```

```

8861 \forest@anchor@forwardtotikztrue
8862 \forest@anchor@do{#1}{-children first}{\forest@referencednodeid}%
8863 }
8864 \pgfdeclaregenericanchor{first}{%
8865 \forest@anchor@forwardtotikztrue
8866 \forest@anchor@do{#1}{first}{\forest@referencednodeid}%
8867 }
8868 \pgfdeclaregenericanchor{parent first}{%
8869 \forest@anchor@forwardtotikztrue
8870 \forest@anchor@do{#1}{parent first}{\forest@referencednodeid}%
8871 }
8872 \pgfdeclaregenericanchor{-parent first}{%
8873 \forest@anchor@forwardtotikztrue
8874 \forest@anchor@do{#1}{-parent first}{\forest@referencednodeid}%
8875 }
8876 \pgfdeclaregenericanchor{parent}{%
8877 \forest@anchor@forwardtotikztrue
8878 \forest@anchor@do{#1}{parent}{\forest@referencednodeid}%
8879 }
8880 \pgfdeclaregenericanchor{-parent}{%
8881 \forest@anchor@forwardtotikztrue
8882 \forest@anchor@do{#1}{-parent}{\forest@referencednodeid}%
8883 }
8884 \pgfdeclaregenericanchor{parent last}{%
8885 \forest@anchor@forwardtotikztrue
8886 \forest@anchor@do{#1}{parent last}{\forest@referencednodeid}%
8887 }
8888 \pgfdeclaregenericanchor{-parent last}{%
8889 \forest@anchor@forwardtotikztrue
8890 \forest@anchor@do{#1}{-parent last}{\forest@referencednodeid}%
8891 }
8892 \pgfdeclaregenericanchor{last}{%
8893 \forest@anchor@forwardtotikztrue
8894 \forest@anchor@do{#1}{last}{\forest@referencednodeid}%
8895 }
8896 \pgfdeclaregenericanchor{children last}{%
8897 \forest@anchor@forwardtotikztrue
8898 \forest@anchor@do{#1}{children last}{\forest@referencednodeid}%
8899 }
8900 \pgfdeclaregenericanchor{-children last}{%
8901 \forest@anchor@forwardtotikztrue
8902 \forest@anchor@do{#1}{-children last}{\forest@referencednodeid}%
8903 }
8904 \pgfdeclaregenericanchor{children'}{%
8905 \forest@anchor@forwardtotikztrue
8906 \forest@anchor@do{#1}{children'}{\forest@referencednodeid}%
8907 }
8908 \pgfdeclaregenericanchor{-children'}{%
8909 \forest@anchor@forwardtotikztrue
8910 \forest@anchor@do{#1}{-children'}{\forest@referencednodeid}%
8911 }
8912 \pgfdeclaregenericanchor{children first'}{%
8913 \forest@anchor@forwardtotikztrue
8914 \forest@anchor@do{#1}{children first'}{\forest@referencednodeid}%
8915 }
8916 \pgfdeclaregenericanchor{-children first'}{%
8917 \forest@anchor@forwardtotikztrue
8918 \forest@anchor@do{#1}{-children first'}{\forest@referencednodeid}%
8919 }
8920 \pgfdeclaregenericanchor{first'}{%
8921 \forest@anchor@forwardtotikztrue

```

```

8922 \forest@anchor@do{#1}{first'}{\forest@referencednodeid}%
8923 }
8924 \pgfdeclaregenericanchor{parent first'}{%
8925 \forest@anchor@forwardtotikztrue
8926 \forest@anchor@do{#1}{parent first'}{\forest@referencednodeid}%
8927 }
8928 \pgfdeclaregenericanchor{-parent first'}{%
8929 \forest@anchor@forwardtotikztrue
8930 \forest@anchor@do{#1}{-parent first'}{\forest@referencednodeid}%
8931 }
8932 \pgfdeclaregenericanchor{parent'}{%
8933 \forest@anchor@forwardtotikztrue
8934 \forest@anchor@do{#1}{parent'}{\forest@referencednodeid}%
8935 }
8936 \pgfdeclaregenericanchor{-parent'}{%
8937 \forest@anchor@forwardtotikztrue
8938 \forest@anchor@do{#1}{-parent'}{\forest@referencednodeid}%
8939 }
8940 \pgfdeclaregenericanchor{parent last'}{%
8941 \forest@anchor@forwardtotikztrue
8942 \forest@anchor@do{#1}{parent last'}{\forest@referencednodeid}%
8943 }
8944 \pgfdeclaregenericanchor{-parent last'}{%
8945 \forest@anchor@forwardtotikztrue
8946 \forest@anchor@do{#1}{-parent last'}{\forest@referencednodeid}%
8947 }
8948 \pgfdeclaregenericanchor{last'}{%
8949 \forest@anchor@forwardtotikztrue
8950 \forest@anchor@do{#1}{last'}{\forest@referencednodeid}%
8951 }
8952 \pgfdeclaregenericanchor{children last'}{%
8953 \forest@anchor@forwardtotikztrue
8954 \forest@anchor@do{#1}{children last'}{\forest@referencednodeid}%
8955 }
8956 \pgfdeclaregenericanchor{-children last'}{%
8957 \forest@anchor@forwardtotikztrue
8958 \forest@anchor@do{#1}{-children last'}{\forest@referencednodeid}%
8959 }

```

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

```

8960 \def\forest@anchor@do#1#2#3{% #1 = shape name, #2 = (potentially) forest anchor, #3 = node id
8961 \forest@fornode{#3}{%
8962 \def\forest@temp@anchor{#2}%
8963 \forest@anchor@snapbordertocompassfalse
8964 \forest@anchor@isborderfalse
8965 \forest@anchor@to@tikz@anchor
8966 \forest@anchor@border@to@compass
8967 \ifforest@anchor@forwardtotikz
8968 \forest@anchor@forward{#1}%
8969 \else
8970 \fi
8971 }%
8972 }

```

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

```

8973 \def\forest@anchor@to@tikz@anchor{%
8974 \ifcsdef{forest@anchor@@\forest@temp@anchor}{%
8975 \csuse{forest@anchor@@\forest@temp@anchor}%
8976 \forest@anchor@to@tikz@anchor
8977 }{}%
8978 }

```

Actual computation.

```

8979 \csdef{forest@anchor@@parent anchor}{%
8980   \foresttoget{parent anchor}\forest@temp@anchor}
8981 \csdef{forest@anchor@@child anchor}{%
8982   \foresttoget{child anchor}\forest@temp@anchor}
8983 \csdef{forest@anchor@@anchor}{%
8984   \foresttoget{anchor}\forest@temp@anchor}
8985 \csdef{forest@anchor@@children'}{%
8986   \forest@anchor@isbordertrue
8987   \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}}%
8988 }
8989 \csdef{forest@anchor@@-children'}{%
8990   \forest@anchor@isbordertrue
8991   \edef\forest@temp@anchor{\number\numexpr 180+\forestove{grow}-\forestove{rotate}}%
8992 }
8993 \csdef{forest@anchor@@parent'}{%
8994   \forest@anchor@isbordertrue
8995   \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestove{\forestove{@parent}}\gro
8996   \edef\forest@temp@anchor{\number\numexpr\forest@temp@grow-\forestove{rotate}+180}%
8997 }
8998 \csdef{forest@anchor@@-parent'}{%
8999   \forest@anchor@isbordertrue
9000   \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestove{\forestove{@parent}}\gro
9001   \edef\forest@temp@anchor{\number\numexpr\forest@temp@grow-\forestove{rotate}}%
9002 }
9003 \csdef{forest@anchor@@first'}{%
9004   \forest@anchor@isbordertrue
9005   \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=0 -\e
9006 }
9007 \csdef{forest@anchor@@last'}{%
9008   \forest@anchor@isbordertrue
9009   \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=0 +\e
9010 }
9011 \csdef{forest@anchor@@parent first'}{%
9012   \forest@anchor@isbordertrue
9013   \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestove{\forestove{@parent}}\gro
9014   \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestove{\forestove{@pare
9015   \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}+180}%
9016   \edef\forest@temp@anchor@first{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@revers
9017   \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@first}\forest@temp@anchor
9018 }
9019 \csdef{forest@anchor@@-parent first'}{%
9020   \forest@anchor@isbordertrue
9021   \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestove{\forestove{@parent}}\gro
9022   \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestove{\forestove{@pare
9023   \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}}%
9024   \edef\forest@temp@anchor@first{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@revers
9025   \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@first}\forest@temp@anchor
9026 }
9027 \csdef{forest@anchor@@parent last'}{%
9028   \forest@anchor@isbordertrue
9029   \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestove{\forestove{@parent}}\gro
9030   \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestove{\forestove{@pare
9031   \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}+180}%
9032   \edef\forest@temp@anchor@last{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@reverse
9033   \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@last}\forest@temp@anchor
9034 }
9035 \csdef{forest@anchor@@-parent last'}{%
9036   \forest@anchor@isbordertrue
9037   \edef\forest@temp@grow{\ifnum\forestove{@parent}=0 \forestove{grow}\else\forestove{\forestove{@parent}}\gro

```

```

9038 \edef\forest@temp@reversed{\ifnum\forestove{@parent}=0 \forestove{reversed}\else\forestove{\forestove{@parent}}
9039 \edef\forest@temp@anchor@parent{\number\numexpr\forest@temp@grow-\forestove{rotate}}}%
9040 \edef\forest@temp@anchor@last{\number\numexpr\forest@temp@grow-\forestove{rotate}\ifnum\forest@temp@reverse
9041 \forest@getaverageangle\forest@temp@anchor@parent}{\forest@temp@anchor@last}\forest@temp@anchor
9042 }
9043 \csdef{forest@anchor@@children first'}{%
9044 \forest@anchor@isbordertrue
9045 \edef\forest@temp@anchor@first{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}
9046 \forest@getaverageangle\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@first}\forest@temp@anchor
9047 }
9048 \csdef{forest@anchor@@-children first'}{%
9049 \forest@anchor@isbordertrue
9050 \edef\forest@temp@anchor@first{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}
9051 \forest@getaverageangle{180+\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@first}\forest@temp@anc
9052 }
9053 \csdef{forest@anchor@@children last'}{%
9054 \forest@anchor@isbordertrue
9055 \edef\forest@temp@anchor@last{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=
9056 \forest@getaverageangle\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@last}\forest@temp@anchor
9057 }
9058 \csdef{forest@anchor@@-children last'}{%
9059 \forest@anchor@isbordertrue
9060 \edef\forest@temp@anchor@last{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=
9061 \forest@getaverageangle{180+\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@last}\forest@temp@anch
9062 }
9063 \csdef{forest@anchor@@children}{%
9064 \forest@anchor@snapbordertocompasstrue
9065 \csuse{forest@anchor@@children'}%
9066 }
9067 \csdef{forest@anchor@@-children}{%
9068 \forest@anchor@snapbordertocompasstrue
9069 \csuse{forest@anchor@@-children'}%
9070 }
9071 \csdef{forest@anchor@@parent}{%
9072 \forest@anchor@snapbordertocompasstrue
9073 \csuse{forest@anchor@@parent'}%
9074 }
9075 \csdef{forest@anchor@@-parent}{%
9076 \forest@anchor@snapbordertocompasstrue
9077 \csuse{forest@anchor@@-parent'}%
9078 }
9079 \csdef{forest@anchor@@first}{%
9080 \forest@anchor@snapbordertocompasstrue
9081 \csuse{forest@anchor@@first'}%
9082 }
9083 \csdef{forest@anchor@@last}{%
9084 \forest@anchor@snapbordertocompasstrue
9085 \csuse{forest@anchor@@last'}%
9086 }
9087 \csdef{forest@anchor@@parent first}{%
9088 \forest@anchor@snapbordertocompasstrue
9089 \csuse{forest@anchor@@parent first'}%
9090 }
9091 \csdef{forest@anchor@@-parent first}{%
9092 \forest@anchor@snapbordertocompasstrue
9093 \csuse{forest@anchor@@-parent first'}%
9094 }
9095 \csdef{forest@anchor@@parent last}{%
9096 \forest@anchor@snapbordertocompasstrue
9097 \csuse{forest@anchor@@parent last'}%
9098 }

```

```

9099 \csdef{forest@anchor@@-parent last}{%
9100   \forest@anchor@snapbordertocompasstrue
9101   \csuse{forest@anchor@@-parent last'}%
9102 }
9103 \csdef{forest@anchor@@children first}{%
9104   \forest@anchor@snapbordertocompasstrue
9105   \csuse{forest@anchor@@children first'}%
9106 }
9107 \csdef{forest@anchor@@-children first}{%
9108   \forest@anchor@snapbordertocompasstrue
9109   \csuse{forest@anchor@@-children first'}%
9110 }
9111 \csdef{forest@anchor@@children last}{%
9112   \forest@anchor@snapbordertocompasstrue
9113   \csuse{forest@anchor@@children last'}%
9114 }
9115 \csdef{forest@anchor@@-children last}{%
9116   \forest@anchor@snapbordertocompasstrue
9117   \csuse{forest@anchor@@-children last'}%
9118 }

```

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>.

```

9119 \def\forest@getaverageangle#1#2#3{%
9120   \edef\forest@temp{\number\numexpr #1-#2+540}%
9121   \expandafter\pgfmathMod@\expandafter{\forest@temp}{360}%
9122   \forest@truncatepgfmathresult
9123   \edef\forest@temp{\number\numexpr 360+#2+((\pgfmathresult-180)/2)}%
9124   \expandafter\pgfmathMod@\expandafter{\forest@temp}{360}%
9125   \forest@truncatepgfmathresult
9126   \let#3\pgfmathresult
9127 }
9128 \def\forest@truncatepgfmathresult{%
9129   \afterassignment\forest@gobbletoEND
9130   \forest@temp@count=\pgfmathresult\forest@END
9131   \def\pgfmathresult{\the\forest@temp@count}%
9132 }
9133 \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.

```

9134 \def\forest@anchor@border@to@compass{%
9135   \ifforest@anchor@isborder % snap to 45 deg, to range 0-360
9136     \ifforest@anchor@snapbordertocompass
9137       \forest@anchor@snap@border@to@compass
9138     \else % to range 0-360
9139       \pgfmathMod@{\forest@temp@anchor}{360}%
9140       \forest@truncatepgfmathresult
9141       \let\forest@temp@anchor\pgfmathresult
9142     \fi
9143     \ifforest@anchor@snapbordertocompass
9144       \ifforest@anchor@forwardtotikz
9145         \ifcsname pgf@anchor%
9146           @\csname pgf@sh@ns@\pgfreferencednodename\endcsname
9147           @\csname forest@compass@\forest@temp@anchor\endcsname
9148         \endcsname
9149         \letcs\forest@temp@anchor{forest@compass@\forest@temp@anchor}%
9150       \fi
9151     \else
9152       \letcs\forest@temp@anchor{forest@compass@\forest@temp@anchor}%

```

```

9153     \fi
9154     \fi
9155     \fi
9156 }
9157 \csdef{forest@compass@0}{east}
9158 \csdef{forest@compass@45}{north east}
9159 \csdef{forest@compass@90}{north}
9160 \csdef{forest@compass@135}{north west}
9161 \csdef{forest@compass@180}{west}
9162 \csdef{forest@compass@225}{south west}
9163 \csdef{forest@compass@270}{south}
9164 \csdef{forest@compass@315}{south east}
9165 \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).

```

9166 \def\forest@anchor@snap@border@to@compass{%
9167   \pgfmathMod@{\forest@temp@anchor}{360}%
9168   \pgfmathdivide@{\pgfmathresult}{45}%
9169   \pgfmathround@{\pgfmathresult}%
9170   \pgfmathmultiply@{\pgfmathresult}{45}%
9171   \forest@truncatepgfmathresult
9172   \let\forest@temp@anchor\pgfmathresult
9173 }

```

This macro forwards the resulting anchor to TikZ.

```

9174 \def\forest@anchor@forward#1{% #1 = shape name
9175   \ifdefempty\forest@temp@anchor{%
9176     \pgf@sh@reanchor{#1}{center}%
9177     \xdef\forest@hack@tikzshapeborder{%
9178       \noexpand\tikz@shapebordertrue
9179       \def\noexpand\tikz@shapeborder@name{\pgfreferencednodename}%
9180     }\aftergroup\forest@hack@tikzshapeborder
9181   }{%
9182     \pgf@sh@reanchor{#1}{\forest@temp@anchor}%
9183   }%
9184 }

```

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

```

9185 \def\forest@referencednodeid{\forest@node@Nametoid{\forest@referencednodename}}%
9186 \def\forest@referencednodename{%
9187   \expandafter\expandafter\expandafter\forest@referencednodename\expandafter\pgfreferencednodename\forest@pgf@notyetpositioned
9188 }
9189 \expandafter\def\expandafter\forest@referencednodename\expandafter#\expandafter1\forest@pgf@notyetpositioned
9190 \if\relax#1\relax\forest@referencednodename@stripafter#2\relax\fi
9191 \if\relax#2\relax#1\fi
9192 }
9193 \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.

```

9194 \def\forest@pointanchor#1{% #1 = anchor
9195   \forest@Pointanchor{\forest@cn}{#1}%
9196 }
9197 \def\forest@Pointanchor#1#2{% #1 = node id, #2 = anchor
9198   \def\forest@pa@temp@name{name}%
9199   \forest@ifdefined{#1}{@box}{%
9200     \forest@get{#1}{@box}\forest@temp
9201     \ifdefempty\forest@temp{%
9202       \def\forest@pa@temp@name{later@name}%
9203     }%

```

```

9204 }{}%
9205 \setbox0\hbox{%
9206   \begin{pgfpicture}%
9207     \if\relax\forestOve{#1}{#2}\relax
9208       \pgfpointanchor{\forestOve{#1}{\forest@pa@temp@name}}{center}%
9209     \else
9210       \pgfpointanchor{\forestOve{#1}{\forest@pa@temp@name}}{\forestOve{#1}{#2}}%
9211     \fi
9212     \xdef\forest@global@marshal{%
9213       \noexpand\global\noexpand\pgf@x=\the\pgf@x\relax
9214       \noexpand\global\noexpand\pgf@y=\the\pgf@y\relax\relax
9215     }%
9216   \end{pgfpicture}%
9217 }%
9218 \forest@global@marshal
9219 }

```

11 Compatibility with previous versions

```

9220 \ifdefempty{\forest@compat}{}{%
9221   \RequirePackage{forest-compat}
9222 }

```