

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

v2.0.0

Sašo Živanović*

January 30, 2015

Contents

1	Package options	2
2	Patches	3
3	Utilities	3
3.1	Sorting	9
4	The bracket representation parser	13
4.1	The user interface macros	13
4.2	Parsing	14
4.3	The tree-structure interface	18
5	Nodes	18
5.1	Option setting and retrieval	18
5.2	Tree structure	21
5.3	Node options	29
5.3.1	Option-declaration mechanism	29
5.3.2	Registers	37
5.3.3	Declaring options	42
5.3.4	Option propagation	48
5.4	Aggregate functions	51
5.4.1	pgfmath extensions	52
5.5	Nodewalk	54
5.6	Dynamic tree	73
6	Stages	77
6.1	Typesetting nodes	80
6.2	Packing	81
6.2.1	Tiers	92
6.2.2	Node boundary	96
6.3	Compute absolute positions	100
6.4	Drawing the tree	101
7	Geometry	103
7.1	Projections	103
7.2	Break path	106
7.3	Get tight edge of path	108
7.4	Get rectangle/band edge	114
7.5	Distance between paths	115
7.6	Utilities	117

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

8	The outer UI	119
8.1	Externalization	119
8.2	The forest environment	120
8.3	Standard node	123
8.4	ls coordinate system	125
8.5	Relative node names in TikZ	126
8.6	Anchors	127
9	Compatibility with previous versions	132

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

```

1 \ProvidesPackage{forest}[2015/01/30 v2.0.0 Drawing (linguistic) trees]
2
3 \RequirePackage{tikz}[2013/12/13]
4 \usetikzlibrary{shapes}
5 \usetikzlibrary{fit}
6 \usetikzlibrary{calc}
7 \usepgflibrary{intersections}
8
9 \RequirePackage{pgfplots}
10 \RequirePackage{etoolbox}[2010/08/21]
11 \RequirePackage{elocalloc}% for \locbox
12 \RequirePackage{environ}
13 \RequirePackage{xparse}
14
15 % \usepackage[trace]{trace-pgfkeys}
16
17 /forest is the root of the key hierarchy.
17 \pgfkeys{/forest/.is family}
18 \def\forestset#1{\pgfkeys{/forest}{#1}}
```

1 Package options

```

19 \newif\ifforest@external@
20 \newif\ifforest@tikz@shack
21 \newif\ifforest@install@keys@to@tikz@path@
22 \newif\ifforest@debug
23 \def\forest@compat{}
24 \forestset{package@options/.cd,
25   external/.is if=forest@external@,
26   tikz@shack/.is if=forest@tikz@shack,
27   tikz@install@keys/.is if=forest@install@keys@to@tikz@path@,
28   debug/.is if=forest@debug,
29   compat/.store in=\forest@compat,
30   compat/.default=most,
31   .unknown/.code={% load library
32     \eappto\forest@loadlibrarieslater{%
33       \noexpand\useforestlibrary{\pgfkeyscurrentname}%
34       \noexpand\forestapplylibrarydefaults{\pgfkeyscurrentname}%
35     }%
36   },
37 }
38 \forest@install@keys@to@tikz@path@true
39 \forest@tikz@shacktrue
40 \def\forest@loadlibrarieslater{}
41 \AtEndOfPackage{\forest@loadlibrarieslater}
42 \def\useforestlibrary#1{\forcsvlist\useforestlibrary@{#1}}
43 \def\useforestlibrary@#1{\RequirePackage{forest-lib-#1}}
```

```

44 \def\forestapplylibrarydefaults#1{\forcsvlist\forestapplylibrarydefaults@{#1}}
45 \def\forestapplylibrarydefaults@#1{\forestset{libraries/#1/defaults/.try}}
46 \NewDocumentCommand\ProvidesForestLibrary{m O{}}{\ProvidesPackage{forest-lib-#1}[#2]}
47 \ProcessPgfPackageOptions{/forest/package@options}

```

2 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.)

```

48 \def\forest@patch#1#2#3#4#5{%
49   % #1 = cs to be patched
50   % #2 = purpose of the patch
51   % #3 = macro arguments
52   % #4 = original code
53   % #5 = patched code
54   \csdef{forest@original@#1}#3{#4}%
55   \csdef{forest@patched@#1}#3{#5}%
56   \ifcsequal{#1}{forest@original@#1}{%
57     \csletcs{#1}{forest@patched@#1}%
58   }{%
59     \ifcsequal{#1}{forest@patched@#1}{% all is good, the patch is in!
60     }{%
61       \PackageWarning{forest}{Failed patching '\expandafter\string\csname #1\endcsname'. Purpose of the patch
62     }%
63   }%
64 }

```

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

```

65 \forest@patch{pgfgettransform}{fix a leaking space}{#1}{%
66   \edef#1{{\pgf@pt@aa}{\pgf@pt@ab}{\pgf@pt@ba}{\pgf@pt@bb}{\the\pgf@pt@x}{\the\pgf@pt@y}}
67 }{%
68   \edef#1{{\pgf@pt@aa}{\pgf@pt@ab}{\pgf@pt@ba}{\pgf@pt@bb}{\the\pgf@pt@x}{\the\pgf@pt@y}}%
69 }

```

3 Utilities

Escaping \ifs.

```

70 \long\def\@escapeif#1#2\fi{\fi#1}
71 \long\def\@escapeifif#1#2\fi#3\fi{\fi\fi#1}
72 \long\def\@escapeififif#1#2\fi#3\fi#4\fi{\fi\fi\fi#1}

```

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

```

73 \def\newloop#1{%
74   \count@=\escapechar
75   \escapechar=-1
76   \expandafter\newloop@parse@loopname\string#1\newloop@end
77   \escapechar=\count@
78 }%
79 {\lccode'7='1 \lccode'8='o \lccode'9='p
80   \lowercase{\gdef\newloop@parse@loopname#17889#2\newloop@end}%
81   \edef\newloop@marshal{%
82     \noexpand\csdef{#1loop#2}####1\expandafter\noexpand\csname #1repeat#2\endcsname{%
83       \noexpand\csdef{#1iterate#2}####1\relax\noexpand\expandafter\expandafter\noexpand\csname#1iterate#
84       \expandafter\noexpand\csname#1iterate#2\endcsname
85       \let\expandafter\noexpand\csname#1iterate#2\endcsname\relax
86     }%
87   }%
88   \newloop@marshal

```

```

89   }%
90   }%
91 }%

```

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.

```

92 \def\nnewsafeloop#1{%
93   \csdef{safeloop@#1}##1\saferepeat{%
94     \edef\safeloop@marshal{%
95       \noexpand\csdef{safeiterate@#1}{%
96         \unexpanded{##1}\relax
97         \noexpand\expandafter
98         \expandonce{\csname safeiterate@#1\endcsname}%
99         \noexpand\fi
100      }%
101    }\safeloop@marshal
102    \csuse{safeiterate@#1}%
103    \advance\nnewsafeloop@depth-1\relax
104    \cslet{safeiterate@#1}\relax
105  }%
106 }%
107 \newcount\safeloop@depth
108 \def\safeloop{%
109   \advance\safeloop@depth1
110   \ifcsdef{safeloop@the\safeloop@depth}{\expandafter\nnewsafeloop\expandafter{\the\safeloop@depth}}%
111   \csdef{safeloopn@the\safeloop@depth}{0}%
112   \csuse{safeloopn@the\safeloop@depth}%
113   \csdef{safeloopn@the\safeloop@depth}{\number\nnewsafeloopn@the\safeloop@depth+1}%
114 }
115 \let\saferepeat\fi
116 \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.

```

117 \def\nnewsafeRKloop#1{%
118   \csdef{safeRKloop@#1}##1\safeRKrepeat{%
119     \edef\safeRKloop@marshal{%
120       \noexpand\csdef{safeRKiterate@#1}{%
121         \unexpanded{##1}\relax
122         \noexpand\expandafter
123         \expandonce{\csname safeRKiterate@#1\endcsname}%
124         \noexpand\fi
125       }%
126     }\safeRKloop@marshal
127     \csuse{safeRKiterate@#1}%
128     \advance\nnewsafeRKloop@depth-1\relax
129     \cslet{safeRKiterate@#1}\relax
130   }%
131   \expandafter\nnewsafeRKloop\expandafter{\the\nnewsafeRKloop@depth\endcsname}
132 }%
133 \newcount\nnewsafeRKloop@depth
134 \def\nnewsafeRKloop{%
135   \advance\nnewsafeRKloop@depth1
136   \ifcsdef{nnewsafeRKloop@the\nnewsafeRKloop@depth}{\expandafter\nnewsafeRKloop\expandafter{\the\nnewsafeRKloop@depth}}%
137   \csdef{nnewsafeRKloopn@the\nnewsafeRKloop@depth}{0}%
138   \csuse{nnewsafeRKloopn@the\nnewsafeRKloop@depth}%
139   \csuse{nnewsafeRKloopn@the\nnewsafeRKloop@depth}%
140   \csdef{nnewsafeRKloopn@the\nnewsafeRKloop@depth}{\number\nnewsafeRKloopn@the\nnewsafeRKloop@depth+1}%
141 }
142 \let\nnewsafeRKrepeat\fi

```

```
143 \def\safeRKloopn{\csuse{safeRKloopn@the\safeRKloop@depth}}%
```

Additional loops (for embedding).

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

New counters, dimens, ifs.

```
145 \newdimen\forest@temp@dimen
```

```
146 \newcount\forest@temp@count
```

```
147 \newcount\forest@n
```

```
148 \newif\ifforest@temp
```

```
149 \newcount\forest@temp@global@count
```

```
150 \newtoks\forest@temp@toks
```

Appending and prepending to token lists.

```
151 \def\etotoks#1#2{\edef\pot@temp{#2}\expandafter#1\expandafter{\pot@temp}}
```

```
152 \def\apptotoks#1#2{\expandafter#1\expandafter{\the#1#2}}
```

```
153 \long\def\lapptotoks#1#2{\expandafter#1\expandafter{\the#1#2}}
```

```
154 \def\eapptotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter#1\expandafter\expandafter\expandafter{\exp
```

```
155 \def\pretotoks#1#2{\toks@={#2}\expandafter\expandafter\expandafter#1\expandafter\expandafter\expandafter{\exp
```

```
156 \def\epretotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter#1\expandafter\expandafter\expandafter{\exp
```

```
157 \def\gapptotoks#1#2{\expandafter\global\expandafter#1\expandafter{\the#1#2}}
```

```
158 \def\xapptotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter\global\expandafter\expandafter\exp
```

```
159 \def\gpretotoks#1#2{\toks@={#2}\expandafter\expandafter\expandafter\global\expandafter\expandafter\expandafter{\exp
```

```
160 \def\xpretotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter\global\expandafter\expandafter\exp
```

Expanding number arguments.

```
161 \def\expandnumberarg#1#2{\expandafter#1\expandafter{\number#2}}
```

```
162 \def\expandtwonumberargs#1#2#3{%
```

```
163 \expandafter\expandtwonumberargs@\expandafter#1\expandafter{\number#3}{#2}}
```

```
164 \def\expandtwonumberargs@#1#2#3{%
```

```
165 \expandafter#1\expandafter{\number#3}{#2}}
```

```
166 \def\expandthreenumberargs#1#2#3#4{%
```

```
167 \expandafter\expandthreenumberargs@\expandafter#1\expandafter{\number#4}{#2}{#3}}
```

```
168 \def\expandthreenumberargs@#1#2#3#4{%
```

```
169 \expandafter\expandthreenumberargs@@\expandafter#1\expandafter{\number#4}{#2}{#3}}
```

```
170 \def\expandthreenumberargs@@#1#2#3#4{%
```

```
171 \expandafter#1\expandafter{\number#4}{#2}{#3}}
```

A macro converting all non-letters in a string to `__`. #1 = string, #2 = receiving macro. Used for declaring pgfnath functions.

```
172 \def\forest@convert@others@to@underscores#1#2{%
```

```
173 \def\forest@cotu@result{}}%
```

```
174 \forest@cotu#1\forest@end
```

```
175 \let#2\forest@cotu@result
```

```
176 }
```

```
177 \def\forest@cotu{%
```

```
178 \futurelet\forest@cotu@nextchar\forest@cotu@checkforspace
```

```
179 }
```

```
180 \def\forest@cotu@checkforspace{%
```

```
181 \expandafter\ifx\space\forest@cotu@nextchar
```

```
182 \let\forest@cotu@next\forest@cotu@havespace
```

```
183 \else
```

```
184 \let\forest@cotu@next\forest@cotu@nospace
```

```
185 \fi
```

```
186 \forest@cotu@next
```

```
187 }
```

```
188 \def\forest@cotu@havespace#1{%
```

```
189 \appto\forest@cotu@result{_}%
```

```
190 \forest@cotu#1%
```

```
191 }
```

```
192 \def\forest@cotu@nospace{%
```

```
193 \ifx\forest@cotu@nextchar\forest@end
```

```
194 \@escapeif\@gobble
```

```

195 \else
196 \escapeif\forest@cotu@nospaceB
197 \fi
198 }
199 \def\forest@cotu@nospaceB{%
200 \ifcat\forest@cotu@nextchar a%
201 \let\forest@cotu@next\forest@cotu@have@alphanum
202 \else
203 \ifcat\forest@cotu@nextchar 0%
204 \let\forest@cotu@next\forest@cotu@have@alphanum
205 \else
206 \let\forest@cotu@next\forest@cotu@have@other
207 \fi
208 \fi
209 \forest@cotu@next
210 }
211 \def\forest@cotu@have@alphanum#1{%
212 \appto\forest@cotu@result{#1}%
213 \forest@cotu
214 }
215 \def\forest@cotu@have@other#1{%
216 \appto\forest@cotu@result{#1}%
217 \forest@cotu
218 }

```

Additional list macros.

```

219 \def\forest@listedel#1#2{% #1 = list, #2 = item
220 \edef\forest@marshal{\noexpand\forest@listdel\noexpand#1{#2}}%
221 \forest@marshal
222 }
223 \def\forest@listcsdel#1#2{%
224 \expandafter\forest@listdel\csname #1\endcsname{#2}%
225 }
226 \def\forest@listcsedel#1#2{%
227 \expandafter\forest@listedel\csname #1\endcsname{#2}%
228 }
229 \edef\forest@restorelistsepcatcode{\noexpand\catcode'\the\catcode'\relax}%
230 \catcode'\|=3
231 \gdef\forest@listdel#1#2{%
232 \def\forest@listedel@A##1|#2|##2\forest@END{%
233 \forest@listedel@B##1|##2\forest@END%|
234 }%
235 \def\forest@listedel@B|##1\forest@END{%|
236 \def#1{##1}%
237 }%
238 \expandafter\forest@listedel@A\expandafter|#1\forest@END%|
239 }
240 \forest@restorelistsepcatcode

```

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

```

241 \def\forest@strip@braces#1{%
242 \forest@strip@braces@A#1\forest@strip@braces@preend\forest@strip@braces@end
243 }
244 \def\forest@strip@braces@A#1#2\forest@strip@braces@end{%
245 #1\ifx\forest@strip@braces@preend#2\else\escapeif{\forest@strip@braces@A#2\forest@strip@braces@end}\fi
246 }

```

Utilities dealing with pgfkeys.

```

247 \def\forest@copycommandkey#1#2{% copies command of #1 into #2
248 \pgfkeysifdefined{#1/.cmd}{-}{%
249 \PackageError{forest}{Key #1 is not a command key}{-}%
250 }%

```

```

251 \pgfkeysgetvalue{#1/.@cmd}\forest@temp
252 \pgfkeyslet{#2/.@cmd}\forest@temp
253 \pgfkeysifdefined{#1/.@args}{%
254   \pgfkeysgetvalue{#1/.@args}\forest@temp
255   \pgfkeyslet{#2/.@args}\forest@temp
256 }{}%
257 \pgfkeysifdefined{#1/.@body}{%
258   \pgfkeysgetvalue{#1/.@body}\forest@temp
259   \pgfkeyslet{#2/.@body}\forest@temp
260 }{}%
261 \pgfkeysifdefined{#1/.@body}{%
262   \pgfkeysgetvalue{#1/.@body}\forest@temp
263   \pgfkeyslet{#2/.@body}\forest@temp
264 }{}%
265 \pgfkeysifdefined{#1/.@def}{%
266   \pgfkeysgetvalue{#1/.@def}\forest@temp
267   \pgfkeyslet{#2/.@def}\forest@temp
268 }{}%
269 }
270 \forestset{
271   copy command key/.code 2 args={\forest@copycommandkey{#1}{#2}},
272   autoforward/.code 2 args={\forest@autoforward{#1}{#2={#1={##1}}}{true}},
273   autoforward'/.code 2 args={\forest@autoforward{#1}{#2=#1, #2={#1={##1}}}{true}},
274   Autoforward/.code 2 args={\forest@autoforward{#1}{#2}{true}},
275   autoforward register/.code 2 args={\forest@autoforward{#1}{#2={#1={##1}}}{false}},
276   autoforward register'/.code 2 args={\forest@autoforward{#1}{#2=#1, #2={#1={##1}}}{false}},
277   Autoforward register/.code 2 args={\forest@autoforward{#1}{#2}{false}},
278   copy command key@if it exists/.code 2 args={%
279     \pgfkeysifdefined{#1/.@cmd}{%
280       \forest@copycommandkey{#1}{#2}%
281     }{}%
282   },
283   unautoforward/.style={
284     typeout={unautoforwarding #1},
285     copy command key@if it exists={/forest/autoforwarded #1}{/forest/#1},
286     copy command key@if it exists={/forest/autoforwarded #1+}{/forest/#1+},
287     copy command key@if it exists={/forest/autoforwarded #1-}{/forest/#1-},
288     copy command key@if it exists={/forest/autoforwarded #1*}{/forest/#1*},
289     copy command key@if it exists={/forest/autoforwarded #1:}{/forest/#1:},
290     copy command key@if it exists={/forest/autoforwarded #1'}{/forest/#1'},
291     copy command key@if it exists={/forest/autoforwarded #1+'}{/forest/#1+'},
292     copy command key@if it exists={/forest/autoforwarded #1-'}{/forest/#1-'},
293     copy command key@if it exists={/forest/autoforwarded #1*'}{/forest/#1*'},
294     copy command key@if it exists={/forest/autoforwarded #1:'}{/forest/#1:'},
295     copy command key@if it exists={/forest/autoforwarded +#1}{/forest/+ #1},
296   },
297   /handlers/.undef/.code={\csundef{pgfk@pgfkeyscurrentpath}},
298   undef option/.style={
299     /forest/#1/.undef,
300     /forest/#1/.@cmd/.undef,
301     /forest/#1+/.@cmd/.undef,
302     /forest/#1-/.@cmd/.undef,
303     /forest/#1*/.@cmd/.undef,
304     /forest/#1:/@cmd/.undef,
305     /forest/#1'/.@cmd/.undef,
306     /forest/#1+'/.@cmd/.undef,
307     /forest/#1-'/.@cmd/.undef,
308     /forest/#1*'/.@cmd/.undef,
309     /forest/#1:'/.@cmd/.undef,
310     /forest/+ #1/.@cmd/.undef,
311     /forest/TeX={\patchcmd{\forest@node@init}{\forest@init{#1}}{}{}},

```

```

312 },
313 undef register/.style={undef option={#1}},
314 }
315 \def\forest@autoforward#1#2#3{%
316   % #1 = option name
317   % #2 = code of a style taking one arg (new option value),
318   %       which expands to whatever should be done with the new value
319   %       autoforward(') adds to the keylist (arg#2)
320   % #3 = true=option, false=register
321   \forest@autoforward@createforwarder{}{#1}{}{#2}{#3}%
322   \forest@autoforward@createforwarder{}{#1}{+}{#2}{#3}%
323   \forest@autoforward@createforwarder{}{#1}{-}{#2}{#3}%
324   \forest@autoforward@createforwarder{}{#1}{*}{#2}{#3}%
325   \forest@autoforward@createforwarder{}{#1}{:}{#2}{#3}%
326   \forest@autoforward@createforwarder{}{#1}{'}{#2}{#3}%
327   \forest@autoforward@createforwarder{}{#1}{+'}{#2}{#3}%
328   \forest@autoforward@createforwarder{}{#1}{-'}{#2}{#3}%
329   \forest@autoforward@createforwarder{}{#1}{{'}}{#2}{#3}%
330   \forest@autoforward@createforwarder{}{#1}{:'}{#2}{#3}%
331   \forest@autoforward@createforwarder{+}{#1}{}{#2}{#3}%
332 }
333 \def\forest@autoforward@createforwarder#1#2#3#4#5{%
334   % #1=prefix, #2=option name, #3=suffix, #4=macro code (#2 above), #5=option or register
335   \pgfkeysifdefined{/forest/#1#2#3/.@cmd}{%
336     \forest@copycommandkey{/forest/#1#2#3}{/forest/autoforwarded #1#2#3}%
337     \pgfkeyssetvalue{/forest/autoforwarded #1#2#3/option@name}{#2}%
338     \pgfkeysdef{/forest/#1#2#3}{%
339       \pgfkeysalso{autoforwarded #1#2#3={##1}}%
340       \def\forest@temp@macro###1{#4}%
341       \csname forest@temp#5\endcsname
342       \edef\forest@temp@value{\ifforest@temp\expandafter\forest0v\expandafter{\expandafter\forest@setter@node
343         \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\pgfkeysalso\expand
344       }}%
345     }{}%
346 }
347 \def\forest@node@removekeysfromkeylist#1#2{% #1 = keys to remove, #2 = option name
348   \edef\forest@marshal{%
349     \noexpand\forest@removekeysfromkeylist{\unexpanded{#1}}{\forest0v{#2}}\noexpand\forest@temp@toks}\forest0
350     \forest0eset{#2}{\the\forest@temp@toks}%
351 }
352 \def\forest@removekeysfromkeylist#1#2#3{%
353   % #1 = keys to remove (a keylist: an empty value means remove a key with any value)
354   % #2 = keylist
355   % #3 = toks cs for result
356   \forest@temp@toks{}%
357   \def\forest@novalue{\forest@novalue}%
358   \pgfqkeys{/forest/remove@key@installer}{#1}%
359   \let\forest@novalue\pgfkeysnovaluertext
360   \pgfqkeys{/forest/remove@key}{#2}%
361   \pgfqkeys{/forest/remove@key@uninstaller}{#1}%
362   #3\forest@temp@toks
363 }
364 \def\forest@remove@key@novalue{\forest@remove@key@novalue}%
365 \forestset{
366   remove@key@installer/.unknown/.code={% #1 = (outer) value
367     \def\forest@temp{#1}%
368     \ifx\forest@temp\pgfkeysnovalue@text
369       \pgfkeysdef{/forest/remove@key/\pgfkeyscurrentname}{}%
370     \else
371       \ifx\forest@temp\forest@novalue
372         \expandafter\forest@remove@key@installer\defwithvalue\expandafter{\pgfkeyscurrentname}{\pgfkeysnovalu

```



```

373     \else
374     \expandafter\forest@remove@key@installer@defwithvalue\expandafter{\pgfkeyscurrentname}{#1}%
375     \fi
376 \fi
377 },
378 remove@key/.unknown/.code={% #1 = (inner) value
379     \expandafter\apptotoks\expandafter\forest@temp@toks\expandafter{\pgfkeyscurrentname={#1},}%
380 },
381 remove@key@uninstaller/.unknown/.code={%
382     \pgfkeyslet{/forest/remove@key/\pgfkeyscurrentname/.@cmd}\@undefined},
383 }
384 \def\forest@remove@key@installer@defwithvalue#1#2{% #1=key name, #2 = outer value
385     \pgfkeysdef{/forest/remove@key/#1}{% ##1 = inner value
386         \def\forest@temp@outer{#2}%
387         \def\forest@temp@inner{##1}%
388         \ifx\forest@temp@outer\forest@temp@inner
389             \else
390                 \apptotoks\forest@temp@toks{#1={##1},}%
391             \fi
392     }%
393 }

```

3.1 Sorting

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

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

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

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

```

394 \def\forest@sort#1#2#3#4#5{%
395     \let\forest@sort@cmp#1\relax
396     \let\forest@sort@let#2\relax
397     \let\forest@sort@direction#3\relax
398     \forest@@sort{#4}{#5}%
399 }

```

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.

```

400 \def\forest@quicksort@minarraylength{10000}
401 \def\forest@@sort#1#2{%
402     \ifnum#1<#2\relax\@escapeif{%
403         \forest@sort@m=#2
404         \advance\forest@sort@m -#1
405         \ifnum\forest@sort@m>\forest@quicksort@minarraylength\relax\@escapeif{%
406             \forest@quicksort{#1}{#2}%
407         }\else\@escapeif{%
408             \forest@insertionsort{#1}{#2}%
409         }\fi

```

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`.

```

410 } \fi
411 }

```

Various counters and macros needed by the sorting algorithms.

```

412 \newcount\forest@sort@m\newcount\forest@sort@k\newcount\forest@sort@p
413 \def\forest@sort@ascending{>}
414 \def\forest@sort@descending{<}
415 \def\forest@sort@cmp{%
416   \PackageError{sort}{You must define forest@sort@cmp function before calling
417     sort}{The macro must take two arguments, indices of the array
418     elements to be compared, and return '=' if the elements are equal
419     and '>'/ '<' if the first is greater /less than the secong element.}%
420 }
421 \def\forest@sort@cmp@gt{\def\forest@sort@cmp@result{>}}
422 \def\forest@sort@cmp@lt{\def\forest@sort@cmp@result{<}}
423 \def\forest@sort@cmp@eq{\def\forest@sort@cmp@result{=}}
424 \def\forest@sort@let{%
425   \PackageError{sort}{You must define forest@sort@let function before calling
426     sort}{The macro must take two arguments, indices of the array:
427     element 2 must be copied onto element 1.}%
428 }

```

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

```

429 \newloop\forest@sort@loop
430 \newloop\forest@sort@loopA
431 \def\forest@quicksort#1#2{%

```

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

```

432   \forest@sort@m=#2
433   \advance\forest@sort@m -#1
434   \ifodd\forest@sort@m\relax\advance\forest@sort@m1 \fi
435   \divide\forest@sort@m 2
436   \advance\forest@sort@m #1

```

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

```

437   \forest@sort@cmp{#1}{#2}%
438   \if\forest@sort@cmp@result=%
439     \forest@sort@p=#1
440   \else
441     \if\forest@sort@cmp@result>%
442       \forest@sort@p=#1\relax
443     \else
444       \forest@sort@p=#2\relax
445     \fi
446     \forest@sort@cmp{\the\forest@sort@p}{\the\forest@sort@m}%
447     \if\forest@sort@cmp@result<%
448       \else
449         \forest@sort@p=\the\forest@sort@m
450       \fi
451   \fi

```

Exchange the pivot and the first element.

```

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

```

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

```

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

```

Loop through the list.

```

454   \forest@sort@k=#1\relax
455   \forest@sort@loop
456   \ifnum\forest@sort@k<#2\relax
457     \advance\forest@sort@k 1

```

Compare the pivot and the current element.

```
458 \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.

```
459 \ifx\forest@sort@direction\forest@sort@cmp@result
460 \advance\forest@sort@m 1
461 \forest@sort@xch{\the\forest@sort@m}{\the\forest@sort@k}
462 \fi
463 \forest@sort@repeat
```

Move the pivot element into its final position.

```
464 \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.

```
465 \forest@sort@k=\forest@sort@m
466 \advance\forest@sort@k -1
467 \advance\forest@sort@m 1
468 \edef\forest@sort@marshal{%
469 \noexpand\forest@sort@{#1}{\the\forest@sort@k}%
470 \noexpand\forest@sort@{\the\forest@sort@m}{#2}%
471 }%
472 \forest@sort@marshal
473 }
474 % We defines the item-exchange macro in terms of the (user-provided)
475 % array let macro.
476 % \begin{macrocode}
477 \def\forest@sort@aux{aux}
478 \def\forest@sort@xch#1#2{%
479 \forest@sort@let{\forest@sort@aux}{#1}%
480 \forest@sort@let{#1}{#2}%
481 \forest@sort@let{#2}{\forest@sort@aux}%
482 }
```

Insertion sort.

```
483 \def\forest@insertionsort#1#2{%
484 \forest@sort@m=#1
485 \edef\forest@insertionsort@low{#1}%
486 \forest@sort@loopA
487 \ifnum\forest@sort@m<#2
488 \advance\forest@sort@m 1
489 \forest@insertionsort@qbody
490 \forest@sort@repeatA
491 }
492 \newif\ifforest@insertionsort@loop
493 \def\forest@insertionsort@qbody{%
494 \forest@sort@let{\forest@sort@aux}{\the\forest@sort@m}%
495 \forest@sort@k\forest@sort@m
496 \advance\forest@sort@k -1
497 \forest@insertionsort@looptrue
498 \forest@sort@loop
499 \ifforest@insertionsort@loop
500 \forest@insertionsort@qbody
501 \forest@sort@repeat
502 \advance\forest@sort@k 1
503 \forest@sort@let{\the\forest@sort@k}{\forest@sort@aux}%
504 }
505 \def\forest@insertionsort@qbody{%
506 \forest@sort@cmp{\the\forest@sort@k}{\forest@sort@aux}%
507 \ifx\forest@sort@direction\forest@sort@cmp@result\relax
```

```

508 \forest@sort@p=\forest@sort@k
509 \advance\forest@sort@p 1
510 \forest@sort@let{\the\forest@sort@p}{\the\forest@sort@k}%
511 \advance\forest@sort@k -1
512 \ifnum\forest@sort@k<\forest@insertionsort@low\relax
513 \forest@insertionsort@loopfalse
514 \fi
515 \else
516 \forest@insertionsort@loopfalse
517 \fi
518 }

```

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

Compare numbers.

```

519 \def\forest@sort@cmpnumcs#1#2{%
520 \ifnum\csname#1\endcsname>\csname#2\endcsname\relax
521 \forest@sort@cmp@gt
522 \else
523 \ifnum\csname#1\endcsname<\csname#2\endcsname\relax
524 \forest@sort@cmp@lt
525 \else
526 \forest@sort@cmp@eq
527 \fi
528 \fi
529 }

```

Compare dimensions.

```

530 \def\forest@sort@cmpdimcs#1#2{%
531 \ifdim\csname#1\endcsname>\csname#2\endcsname\relax
532 \forest@sort@cmp@gt
533 \else
534 \ifdim\csname#1\endcsname<\csname#2\endcsname\relax
535 \forest@sort@cmp@lt
536 \else
537 \forest@sort@cmp@eq
538 \fi
539 \fi
540 }

```

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

```

541 \def\forest@sort@cmptwodimcs#1#2#3#4{%
542 \ifdim\csname#1\endcsname>\csname#3\endcsname\relax
543 \forest@sort@cmp@gt
544 \else
545 \ifdim\csname#1\endcsname<\csname#3\endcsname\relax
546 \forest@sort@cmp@lt
547 \else
548 \ifdim\csname#2\endcsname>\csname#4\endcsname\relax
549 \forest@sort@cmp@gt
550 \else
551 \ifdim\csname#2\endcsname<\csname#4\endcsname\relax
552 \forest@sort@cmp@lt
553 \else
554 \forest@sort@cmp@eq
555 \fi
556 \fi
557 \fi
558 \fi
559 }

```

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).

```

560 \def\forest@reversearray#1#2#3{%
561   \let\forest@sort@let#1%
562   \c@pgf@countc=#2
563   \c@pgf@countd=#3
564   \advance\c@pgf@countd -1
565   \safeloop
566   \ifnum\c@pgf@countc<\c@pgf@countd\relax
567     \forest@sort@exch{\the\c@pgf@countc}{\the\c@pgf@countd}%
568     \advance\c@pgf@countc 1
569     \advance\c@pgf@countd -1
570   \saferepeat
571 }
```

4 The bracket representation parser

4.1 The user interface macros

Settings.

```

572 \def\bracketset#1{\pgfqkeys{/bracket}{#1}}%
573 \bracketset{%
574   /bracket/.is family,
575   /handlers/.let/.style={\pgfkeyscurrentpath/.code={\let#1##1}},
576   opening bracket/.let=\bracket@openingBracket,
577   closing bracket/.let=\bracket@closingBracket,
578   action character/.let=\bracket@actionCharacter,
579   opening bracket=[,
580   closing bracket=],
581   action character,
582   new node/.code n args={3}{% #1=preamble, #2=node spec, #3=cs receiving the id
583     \forest@node@new#3%
584     \forest@set{#3}{given options}{\forest@contentto=\unexpanded{#2}}%
585     \ifblank{#1}{}{%
586       \forest@rset{preamble}{#1}%
587     }%
588   },
589   set afterthought/.code 2 args={% #1=node id, #2=afterthought
590     \ifblank{#2}{}{\forest@appto{#1}{given options}{,afterthought={#2}}}%
591   }
592 }
```

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

```

593 \newtoks\bracket@content
594 \newtoks\bracket@afterthought
595 \def\bracketParse#1#2={%
596   \def\bracket@endParsingHook{#1}%
597   \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.)

```

598   \bracket@content={}%
599   \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.

```

600   \let\bracket@state\bracket@state@starting
```

```
601 \bracket@ignorepacestrue
```

By default, don't expand anything.

```
602 \bracket@expandtokensfalse
```

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

```
603 \def\bracket@parentNode{0}%
```

```
604 \def\bracket@rootNode{0}%
```

```
605 \def\bracket@newNode{0}%
```

```
606 \def\bracket@afterthoughtNode{0}%
```

Finally, we start the parser.

```
607 \bracket@Parse
```

```
608 }
```

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.

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

4.2 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_EX's (undelimited) argument parsing algorithm. If a space is found, remember that, eat it up, and restart the parsing.

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

```
611 \futurelet\bracket@next@token\bracket@Parse@checkForSpace
```

```
612 }
```

```
613 \def\bracket@Parse@checkForSpace{%
```

```
614 \expandafter\ifx\space\bracket@next@token\@escapeif{%
```

```
615 \ifbracket@ignorepaces\else
```

```
616 \bracket@haveSpacetrue
```

```
617 \fi
```

```
618 \expandafter\bracket@Parse\romannumeral-‘0%
```

```
619 }\else\@escapeif{%
```

```
620 \bracket@Parse@maybeexpand
```

```
621 }\fi
```

```
622 }
```

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`.

```
623 \newif\ifbracket@expandtokens
```

```
624 \def\bracket@Parse@maybeexpand{%
```

```
625 \ifbracket@expandtokens\@escapeif{%
```

```
626 \expandafter\bracket@Parse@peekAhead\romannumeral-‘0%
```

```
627 }\else\@escapeif{%
```

```
628 \bracket@Parse@peekAhead
```

```
629 }\fi
```

```
630 }
```

We then look ahead to see what's coming.

```
631 \def\bracket@Parse@peekAhead{%
```

```
632 \futurelet\bracket@next@token\bracket@Parse@checkForTeXGroup
```

```
633 }
```

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

```
634 \def\bracket@Parse@checkForTeXGroup{%
```

```
635 \ifx\bracket@next@token\bgroup%
```

```
636 \@escapeif{\bracket@Parse@appendGroup}%
```

```
637 \else
```

```
638 \@escapeif{\bracket@Parse@token}%
```

```
639 \fi
```

```
640 }
```

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.

```

641 \long\def\bracket@Parse@token#1{%
642   \ifx#1\bracket@openingBracket
643     \@escapeif{\bracket@Parse@openingBracketFound}%
644   \else
645     \@escapeif{%
646       \ifx#1\bracket@closingBracket
647         \@escapeif{\bracket@Parse@closingBracketFound}%
648       \else
649         \@escapeif{%
650           \ifx#1\bracket@actionCharacter
651             \@escapeif{\futurelet\bracket@next@token\bracket@Parse@actionCharacterFound}%
652           \else
653             \@escapeif{\bracket@Parse@appendToken#1}%
654           \fi
655         }%
656       \fi
657     }%
658   \fi
659 }

```

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

```

660 \newif\ifbracket@haveSpace
661 \newif\ifbracket@ignorespaces
662 \def\bracket@Parse@appendSpace{%
663   \ifbracket@haveSpace
664     \ifcase\bracket@state\relax
665       \eapptotoks\bracket@content\space
666     \or
667       \eapptotoks\bracket@afterthought\space
668     \or
669       \eapptotoks\bracket@afterthought\space
670     \fi
671   \bracket@haveSpacefalse
672 \fi
673 }
674 \long\def\bracket@Parse@appendToken#1{%
675   \bracket@Parse@appendSpace
676   \ifcase\bracket@state\relax
677     \lapptotoks\bracket@content{#1}%
678   \or
679     \lapptotoks\bracket@afterthought{#1}%
680   \or
681     \lapptotoks\bracket@afterthought{#1}%
682   \fi
683   \bracket@ignorespacesfalse
684   \bracket@Parse
685 }
686 \def\bracket@Parse@appendGroup#1{%
687   \bracket@Parse@appendSpace
688   \ifcase\bracket@state\relax
689     \apptotoks\bracket@content{{#1}}%
690   \or
691     \apptotoks\bracket@afterthought{{#1}}%
692   \or
693     \apptotoks\bracket@afterthought{{#1}}%
694   \fi
695   \bracket@ignorespacesfalse
696   \bracket@Parse

```

697 }

Declare states.

```
698 \def\bracket@state@inContent{0}
699 \def\bracket@state@inAfterthought{1}
700 \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.)

```
701 \def\bracket@Parse@openingBracketFound{%
702   \bracket@haveSpacefalse
703   \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.

```
704     \@escapeif{%
705       \bracket@createNode
706       \ifnum\bracket@parentNode=0 \else
707         \forest@node@Append{\bracket@parentNode}{\bracket@newNode}%
708       \fi
709       \let\bracket@parentNode\bracket@newNode
710       \bracket@Parse
711     }%
712   \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.

```
713     \@escapeif{%
714       \bracket@addAfterthought
715       \let\bracket@state\bracket@state@inContent
716       \bracket@Parse
717     }%
718   \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.

```
719     \@escapeif{%
720       \let\bracket@state\bracket@state@inContent
721       \bracket@Parse
722     }%
723   \fi
724 }
```

```
725 \def\bracket@Parse@closingBracketFound{%
726   \bracket@haveSpacefalse
727   \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.

```
728     \@escapeif{%
729       \bracket@createNode
730       \ifnum\bracket@parentNode=0
731         \@escapeif\bracket@endParsingHook
732       \else
733         \@escapeif{%
734           \let\bracket@afterthoughtNode\bracket@newNode
```



```

735      \let\bracket@state\bracket@state@inAfterthought
736      \forest@node@Append{\bracket@parentNode}{\bracket@newNode}%
737      \bracket@Parse
738    }%
739    \fi
740  }%
741  \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.

```

742  \@escapeif{%
743    \bracket@addAfterthought
744    \let\bracket@afterthoughtNode\bracket@parentNode
745    \edef\bracket@parentNode{\forestOve{\bracket@parentNode}{@parent}}%
746    \ifnum\bracket@parentNode=0
747      \expandafter\bracketEndParsingHook
748    \else
749      \expandafter\bracket@Parse
750    \fi
751  }%
752  \else % starting

```

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

```

753    \PackageError{forest}{You're attempting to start a bracket representation
754      with a closing bracket}{}%
755  \fi
756 }

```

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

```

757 \def\bracket@Parse@actionCharacterFound{%
  If a braced expression follows, its contents will be fully expanded.
758   \ifx\bracket@next@token\bgroup\@escapeif{%
759     \bracket@Parse@action@expandgroup
760   }\else\@escapeif{%
761     \bracket@Parse@action@notagroup
762   }\fi
763 }
764 \def\bracket@Parse@action@expandgroup#1{%
765   \edef\bracket@Parse@action@expandgroup@macro{#1}%
766   \expandafter\bracket@Parse\bracket@Parse@action@expandgroup@macro
767 }
768 \let\bracket@action@fullyexpandCharacter+
769 \let\bracket@action@dontexpandCharacter-
770 \let\bracket@action@executeCharacter!
771 \def\bracket@Parse@action@notagroup#1{%

```

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

```

772   \ifx#1\bracket@action@fullyexpandCharacter\@escapeif{%
773     \bracket@expandtokenstrue\bracket@Parse
774   }\else\@escapeif{%

```

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

```

775   \ifx#1\bracket@action@dontexpandCharacter\@escapeif{%
776     \bracket@expandtokensfalse\bracket@Parse
777   }\else\@escapeif{%

```

Inhibit expansion of the next token.

```

778   \ifx#10\@escapeif{%
779     \bracket@Parse@appendToken
780   }\else\@escapeif{%

```

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

```

781     \ifx#1\bracket@actionCharacter
782     \else\@escapeif{%
    Anything else will be expanded once.
783     \expandafter\bracket@Parse#1%
784     }\fi
785     }\fi
786     }\fi
787 }\fi
788 }
```

4.3 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.

```

789 \def\bracket@createNode{%
790   \ifnum\bracket@rootNode=0
791     % root node
792     \bracketset{new node/.expanded=%
793       {\the\bracket@afterthought}%
794       {\the\bracket@content}%
795       \noexpand\bracket@newNode
796     }%
797     \bracket@afterthought={}%
798     \let\bracket@rootNode\bracket@newNode
799     \expandafter\let\bracket@saveRootNodeTo\bracket@newNode
800   \else
801     % other nodes
802     \bracketset{new node/.expanded=%
803       {}%
804       {\the\bracket@content}%
805       \noexpand\bracket@newNode
806     }%
807   \fi
808   \bracket@content={}%
809 }
```

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

```

810 \def\bracket@addAfterthought{%
811   \bracketset{%
812     set afterthought/.expanded={\bracket@afterthoughtNode}{\the\bracket@afterthought}%
813   }%
814   \bracket@afterthought={}%
815 }
```

5 Nodes

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

5.1 Option setting and retrieval

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

```

816 % full expansion expands precisely to the value
817 \def\forestov#1{\expandafter\expandafter\expandafter\expandonce
```

```

818 \pgfkeysvalueof{/forest/@node/\forest@cn/#1}}
819 % full expansion expands all the way
820 \def\forestove#1{\pgfkeysvalueof{/forest/@node/\forest@cn/#1}}
821 % full expansion expands to the cs holding the value
822 \def\forestom#1{\expandafter\expandonce\expandafter{\pgfkeysvalueof{/forest/@node/\forest@cn/#1}}}
823 \def\forestoget#1#2{\pgfkeysgetvalue{/forest/@node/\forest@cn/#1}{#2}}
824 \def\forestolet#1#2{\pgfkeyslet{/forest/@node/\forest@cn/#1}{#2}}
825 \def\forestocslet#1#2{%
826   \edef\forest@marshal{%
827     \noexpand\pgfkeyslet{/forest/@node/\forest@cn/#1}{\expandonce{\csname#2\endcsname}}}%
828   }\forest@marshal
829 }
830 \def\forestoset#1#2{\pgfkeyssetvalue{/forest/@node/\forest@cn/#1}{#2}}
831 \def\forestoeset#1#2{%
832   \edef\forest@option@temp{%
833     \noexpand\pgfkeyssetvalue{/forest/@node/\forest@cn/#1}{#2}%
834   }\forest@option@temp
835 }
836 \def\forestooppto#1#2{%
837   \forestoeset{#1}{\forestov{#1}\unexpanded{#2}}}%
838 }
839 \def\forestoiddefined#1#2#3{%
840   \pgfkeysifdefined{/forest/@node/\forest@cn/#1}{#2}{#3}%
841 }

```

User macros for retrieving node options of the current node.

```

842 \let\forestoption\forestov
843 \let\foresteooption\forestove

```

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

```

844 \def\forestOv#1#2{\expandafter\expandafter\expandafter\expandonce
845   \pgfkeysvalueof{/forest/@node/#1/#2}}
846 \def\forestOve#1#2{\pgfkeysvalueof{/forest/@node/#1/#2}}
847 % full expansion expands to the cs holding the value
848 \def\forestOm#1#2{\expandafter\expandonce\expandafter{\pgfkeysvalueof{/forest/@node/#1/#2}}}
849 \def\forestOget#1#2#3{\pgfkeysgetvalue{/forest/@node/#1/#2}{#3}}
850 \def\forestOget#1#2#3{\pgfkeysgetvalue{/forest/@node/#1/#2}{#3}}
851 \def\forestOlet#1#2#3{\pgfkeyslet{/forest/@node/#1/#2}{#3}}
852 \def\forestOcslet#1#2#3{%
853   \edef\forest@marshal{%
854     \noexpand\pgfkeyslet{/forest/@node/#1/#2}{\expandonce{\csname#3\endcsname}}}%
855   }\forest@marshal
856 }
857 \def\forestOset#1#2#3{\pgfkeyssetvalue{/forest/@node/#1/#2}{#3}}
858 \def\forestOset#1#2#3{%
859   \edef\forestoption@temp{%
860     \noexpand\pgfkeyssetvalue{/forest/@node/#1/#2}{#3}%
861   }\forestoption@temp
862 }
863 \def\forestOappto#1#2#3{%
864   \forestOset{#1}{#2}{\forestOv{#1}{#2}\unexpanded{#3}}}%
865 }
866 \def\forestOeappto#1#2#3{%
867   \forestOset{#1}{#2}{\forestOv{#1}{#2}#3}%
868 }
869 \def\forestOpreto#1#2#3{%
870   \forestOset{#1}{#2}{\unexpanded{#3}\forestOv{#1}{#2}}}%
871 }
872 \def\forestOepreto#1#2#3{%
873   \forestOset{#1}{#2}{#3\forestOv{#1}{#2}}}%
874 }
875 \def\forestOifdefined#1#2#3#4{%

```

```

876 \pgfkeysifdefined{/forest/@node/#1/#2}{#3}{#4}%
877 }
878 \def\forest0let0#1#2#3#4{% option #2 of node #1 <-- option #4 of node #3
879 \forest0get{#3}{#4}\forestoption@temp
880 \forest0let{#1}{#2}\forestoption@temp}
881 \def\forest0leto#1#2#3{%
882 \forest0get{#3}\forestoption@temp
883 \forest0let{#1}{#2}\forestoption@temp}
884 \def\forestolet0#1#2#3{%
885 \forest0get{#2}{#3}\forestoption@temp
886 \forestolet{#1}\forestoption@temp}
887 \def\forestoleto#1#2{%
888 \forest0get{#2}\forestoption@temp
889 \forestolet{#1}\forestoption@temp}

Macros for retrieving/setting registers.

890 % full expansion expands precisely to the value
891 \def\forestrv#1\expandafter\expandafter\expandafter\expandonce
892 \pgfkeysvalueof{/forest/@node/register/#1}}
893 % full expansion expands all the way
894 \def\forestrv#1{\pgfkeysvalueof{/forest/@node/register/#1}}
895 % full expansion expands to the cs holding the value
896 \def\forestrm#1\expandafter\expandonce\expandafter{\pgfkeysvalueof{/forest/@node/register/#1}}
897 \def\forestrget#1#2{\pgfkeysgetvalue{/forest/@node/register/#1}{#2}}
898 \def\forestrlet#1#2{\pgfkeyslet{/forest/@node/register/#1}{#2}}
899 \def\forestrcslet#1#2{%
900 \edef\forest@marshal{%
901 \noexpand\pgfkeyslet{/forest/@node/register/#1}{\expandonce{\csname#2\endcsname}}}%
902 }\forest@marshal
903 }
904 \def\forestrset#1#2{\pgfkeyssetvalue{/forest/@node/register/#1}{#2}}
905 \def\forestreset#1#2{%
906 \edef\forest@option@temp{%
907 \noexpand\pgfkeyssetvalue{/forest/@node/register/#1}{#2}%
908 }\forest@option@temp
909 }
910 \def\forestrappto#1#2{%
911 \forestreset{#1}{\forestrv{#1}\unexpanded{#2}}}%
912 }
913 \def\forestrpreto#1#2{%
914 \forestreset{#1}{\unexpanded{#2}\forestrv{#1}}}%
915 }
916 \def\forestrifdefined#1#2#3{%
917 \pgfkeysifdefined{/forest/@node/register/#1}{#2}{#3}%
918 }

```

User macros for retrieving node options of the current node.

```

919 \def\forestregister#1{\forestrv{#1}}
920 \def\foresteregister#1{\forestrv{#1}}

```

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

```

921 \def\forest@node@init{%
922 \forestoset{@parent}{0}%
923 \forestoset{@previous}{0}% previous sibling
924 \forestoset{@next}{0}% next sibling
925 \forestoset{@first}{0}% primary child
926 \forestoset{@last}{0}% last child
927 }
928 \def\forestoinit#1{%
929 \pgfkeysgetvalue{/forest/#1}\forestoinit@temp
930 \forestolet{#1}\forestoinit@temp
931 }

```

```

932 \newcount\forest@node@maxid
933 \def\forest@node@new#1{% #1 = cs receiving the new node id
934   \advance\forest@node@maxid1
935   \forest@fornode{\the\forest@node@maxid}{%
936     \forest@node@init
937     \forest@node@setname{node@\forest@cn}%
938     \forest@initializefromstandardnode
939     \edef#1{\forest@cn}%
940   }%
941 }
942 \let\forest@init@orig\forest@init
943 \def\forest@node@copy#1#2{% #1=from node id, cs receiving the new node id
944   \advance\forest@node@maxid1
945   \def\forest@init##1{\ifstrequal{##1}{name}{\forest@set{name}{node@\forest@cn}}{\forest@let0{##1}{#1}{##1}}}
946   \forest@fornode{\the\forest@node@maxid}{%
947     \forest@node@init
948     \forest@node@setname{\forest@copy@name@template{\forest@ve{#1}{name}}}%
949     \edef#2{\forest@cn}%
950   }%
951   \let\forest@init\forest@init@orig
952 }
953 \forestset{
954   copy name template/.code={\def\forest@copy@name@template##1{#1}},
955   copy name template/.default={node@\the\forest@node@maxid},
956   copy name template
957 }
958 \def\forest@tree@copy#1#2{% #1=from node id, #2=cs receiving the new node id
959   \forest@node@copy{#1}\forest@node@copy@temp@id
960   \forest@fornode{\forest@node@copy@temp@id}{%
961     \expandafter\forest@tree@copy@\expandafter{\forest@node@copy@temp@id}{#1}%
962     \edef#2{\forest@cn}%
963   }%
964 }
965 \def\forest@tree@copy@#1#2{%
966   \forest@node@Foreachchild{#2}{%
967     \expandafter\forest@tree@copy\expandafter{\forest@cn}\forest@node@copy@temp@childid
968     \forest@node@Append{#1}{\forest@node@copy@temp@childid}%
969   }%
970 }

```

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.

```

971 \def\forest@cn{0}
972 \forest@node@init

```

5.2 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.

```

973 \def\forest@node@append#1{\expandtwonumberargs\forest@node@Append{\forest@cn}{#1}}
974 \def\forest@node@prepend#1{\expandtwonumberargs\forest@node@Insertafter{\forest@cn}{#1}{0}}
975 \def\forest@node@insertafter#1#2{%
976   \expandthreenumberargs\forest@node@Insertafter{\forest@cn}{#1}{#2}}
977 \def\forest@node@insertbefore#1#2{%
978   \expandthreenumberargs\forest@node@Insertafter{\forest@cn}{#1}{\forest@ve{#2}{@previous}}}%
979 }
980 \def\forest@node@remove{\expandnumberarg\forest@node@Remove{\forest@cn}}
981 \def\forest@node@Append#1#2{\expandtwonumberargs\forest@node@Append@{#1}{#2}}

```

```

982 \def\forest@node@Prepend#1#2{\expandtwonumberargs\forest@node@Insertafter{#1}{#2}{0}}
983 \def\forest@node@Insertafter#1#2#3{% #2 is inserted after #3
984   \expandthreenumberargs\forest@node@Insertafter@{#1}{#2}{#3}%
985 }
986 \def\forest@node@Insertbefore#1#2#3{% #2 is inserted before #3
987   \expandthreenumberargs\forest@node@Insertafter{#1}{#2}{\forestOve{#3}{@previous}}%
988 }
989 \def\forest@node@Remove#1{\expandnumberarg\forest@node@Remove@{#1}}
990 \def\forest@node@Insertafter@#1#2#3{%
991   \ifnum\forestOve{#2}{@parent}=0
992   \else
993     \PackageError{forest}{Insertafter(#1,#2,#3):
994       node #2 already has a parent (\forestOve{#2}{@parent})}{}%
995   \fi
996   \ifnum#3=0
997   \else
998     \ifnum#1=\forestOve{#3}{@parent}
999     \else
1000       \PackageError{forest}{Insertafter(#1,#2,#3): node #1 is not the parent of the
1001         intended sibling #3 (with parent \forestOve{#3}{@parent})}{}%
1002     \fi
1003   \fi
1004   \forestOset{#2}{@parent}{#1}%
1005   \forestOset{#2}{@previous}{#3}%
1006   \ifnum#3=0
1007     \forestOget{#1}{@first}\forest@node@temp
1008     \forestOset{#1}{@first}{#2}%
1009   \else
1010     \forestOget{#3}{@next}\forest@node@temp
1011     \forestOset{#3}{@next}{#2}%
1012   \fi
1013   \forestOset{#2}{@next}{\forest@node@temp}%
1014   \ifnum\forest@node@temp=0
1015     \forestOset{#1}{@last}{#2}%
1016   \else
1017     \forestOset{\forest@node@temp}{@previous}{#2}%
1018   \fi
1019 }
1020 \def\forest@node@Append@#1#2{%
1021   \ifnum\forestOve{#2}{@parent}=0
1022   \else
1023     \PackageError{forest}{Append(#1,#2):
1024       node #2 already has a parent (\forestOve{#2}{@parent})}{}%
1025   \fi
1026   \forestOset{#2}{@parent}{#1}%
1027   \forestOget{#1}{@last}\forest@node@temp
1028   \forestOset{#1}{@last}{#2}%
1029   \forestOset{#2}{@previous}{\forest@node@temp}%
1030   \ifnum\forest@node@temp=0
1031     \forestOset{#1}{@first}{#2}%
1032   \else
1033     \forestOset{\forest@node@temp}{@next}{#2}%
1034   \fi
1035 }
1036 \def\forest@node@Remove@#1{%
1037   \forestOget{#1}{@parent}\forest@node@temp@parent
1038   \ifnum\forest@node@temp@parent=0
1039   \else
1040     \forestOget{#1}{@previous}\forest@node@temp@previous
1041     \forestOget{#1}{@next}\forest@node@temp@next
1042     \ifnum\forest@node@temp@previous=0

```

```

1043     \forestOset{\forest@node@temp@parent}{@first}{\forest@node@temp@next}%
1044 \else
1045     \forestOset{\forest@node@temp@previous}{@next}{\forest@node@temp@next}%
1046 \fi
1047 \ifnum\forest@node@temp@next=0
1048     \forestOset{\forest@node@temp@parent}{@last}{\forest@node@temp@previous}%
1049 \else
1050     \forestOset{\forest@node@temp@next}{@previous}{\forest@node@temp@previous}%
1051 \fi
1052 \forestOset{#1}{@parent}{0}%
1053 \forestOset{#1}{@previous}{0}%
1054 \forestOset{#1}{@next}{0}%
1055 \fi
1056 }

Do some stuff and return to the current node.
1057 \def\forest@forthis#1{%
1058     \edef\forest@node@marshal{\unexpanded{#1}\def\noexpand\forest@cn}%
1059     \expandafter\forest@node@marshal\expandafter{\forest@cn}%
1060 }
1061 \def\forest@fornode#1#2{%
1062     \edef\forest@node@marshal{\edef\noexpand\forest@cn{#1}\unexpanded{#2}\def\noexpand\forest@cn}%
1063     \expandafter\forest@node@marshal\expandafter{\forest@cn}%
1064 }

Looping methods: children.
1065 \def\forest@node@foreachchild#1{\forest@node@Foreachchild{\forest@cn}{#1}}
1066 \def\forest@node@Foreachchild#1#2{%
1067     \forest@fornode{\forestOve{#1}{@first}}{\forest@node@@forselfandfollowingsiblings{#2}}%
1068 }
1069 \def\forest@node@@forselfandfollowingsiblings#1{%
1070     \ifnum\forest@cn=0
1071     \else
1072         \forest@forthis{#1}%
1073         \@escapeif{%
1074             \edef\forest@cn{\forestove{@next}}%
1075             \forest@node@@forselfandfollowingsiblings{#1}%
1076         }%
1077     \fi
1078 }
1079 \def\forest@node@@forselfandfollowingsiblings@reversed#1{%
1080     \ifnum\forest@cn=0
1081     \else
1082         \@escapeif{%
1083             \edef\forest@marshal{%
1084                 \noexpand\def\noexpand\forest@cn{\forestove{@next}}%
1085                 \noexpand\forest@node@@forselfandfollowingsiblings@reversed{\unexpanded{#1}}%
1086                 \noexpand\forest@fornode{\forest@cn}{\unexpanded{#1}}%
1087             }\forest@marshal
1088         }%
1089     \fi
1090 }
1091 \def\forest@node@foreachchild@reversed#1{\forest@node@Foreachchild@reversed{\forest@cn}{#1}}
1092 \def\forest@node@Foreachchild@reversed#1#2{%
1093     \forest@fornode{\forestOve{#1}{@last}}{\forest@node@@forselfandprecedingsiblings@reversed{#2}}%
1094 }
1095 \def\forest@node@@forselfandprecedingsiblings@reversed#1{%
1096     \ifnum\forest@cn=0
1097     \else
1098         \forest@forthis{#1}%
1099         \@escapeif{%
1100             \edef\forest@cn{\forestove{@previous}}%

```

```

1101 \forest@node@@forselfandprecedingsiblings@reversed{#1}%
1102 }%
1103 \fi
1104 }
1105 \def\forest@node@@forselfandprecedingsiblings#1{%
1106 \ifnum\forest@cn=0
1107 \else
1108 \@escapeif{%
1109 \edef\forest@marshal{%
1110 \noexpand\def\noexpand\forest@cn{\forestove{@previous}}%
1111 \noexpand\forest@node@@forselfandprecedingsiblings{\unexpanded{#1}}%
1112 \noexpand\forest@fornode{\forest@cn}{\unexpanded{#1}}%
1113 }\forest@marshal
1114 }%
1115 \fi
1116 }

```

Looping methods: (sub)tree and descendants.

```

1117 \def\forest@node@@foreach#1#2#3#4{%
1118 % #1 = do what
1119 % #2 = do that -1=before,1=after processing children
1120 % #3 & #4: normal or reversed order of children?
1121 % #3 = @first/@last
1122 % #4 = \forest@node@@forselfandfollowingsiblings / \forest@node@@forselfandprecedingsiblings@reversed
1123 \ifnum#2<0 \forest@forthis{#1}\fi
1124 \ifnum\forestove{#3}=0
1125 \else\@escapeif{%
1126 \forest@forthis{%
1127 \edef\forest@cn{\forestove{#3}}%
1128 #4{\forest@node@@foreach{#1}{#2}{#3}{#4}}%
1129 }%
1130 }\fi
1131 \ifnum#2>0 \forest@forthis{#1}\fi
1132 }
1133 \def\forest@node@foreach#1{%
1134 \forest@node@@foreach{#1}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1135 \def\forest@node@Foreach#1#2{%
1136 \forest@fornode{#1}{\forest@node@@foreach{#2}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1137 \def\forest@node@foreach@reversed#1{%
1138 \forest@node@@foreach{#1}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1139 \def\forest@node@Foreach@reversed#1#2{%
1140 \forest@fornode{#1}{\forest@node@@foreach{#2}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1141 \def\forest@node@foreach@childrenfirst#1{%
1142 \forest@node@@foreach{#1}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1143 \def\forest@node@Foreach@childrenfirst#1#2{%
1144 \forest@fornode{#1}{\forest@node@@foreach{#2}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1145 \def\forest@node@foreach@childrenfirst@reversed#1{%
1146 \forest@node@@foreach{#1}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1147 \def\forest@node@Foreach@childrenfirst@reversed#1#2{%
1148 \forest@fornode{#1}{\forest@node@@foreach{#2}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1149 \def\forest@node@foreachdescendant#1{%
1150 \forest@node@@foreachchild{\forest@node@@foreach{#1}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1151 \def\forest@node@Foreachdescendant#1#2{%
1152 \forest@node@Foreachchild{#1}{\forest@node@@foreach{#2}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1153 \def\forest@node@foreachdescendant@reversed#1{%
1154 \forest@node@@foreachchild@reversed{\forest@node@@foreach{#1}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1155 \def\forest@node@Foreachdescendant@reversed#1#2{%
1156 \forest@node@Foreachchild@reversed{#1}{\forest@node@@foreach{#2}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1157 \def\forest@node@foreachdescendant@childrenfirst#1{%
1158 \forest@node@@foreachchild{\forest@node@@foreach{#1}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1159 \def\forest@node@Foreachdescendant@childrenfirst#1#2{%

```



```

1160 \forest@node@Foreachchild{#1}{\forest@node@@foreach{#2}{1}{@first}{\forest@node@@forselfandfollowingsibling
1161 \def\forest@node@foreachdescendant@childrenfirst@reversed#1{%
1162 \forest@node@foreachchild@reversed{\forest@node@@foreach{#1}{1}{@last}{\forest@node@@forselfandprecedingsib
1163 \def\forest@node@Foreachdescendant@childrenfirst@reversed#1#2{%
1164 \forest@node@Foreachchild@reversed{#1}{\forest@node@@foreach{#2}{1}{@last}{\forest@node@@forselfandprecedin

Looping methods: breadth-first.

1165 \def\forest@node@foreach@breadthfirst#1#2{% #1 = max level, #2 = code
1166 \forest@node@Foreach@breadthfirst@{\forest@cn}{@first}{@next}{#1}{#2}}
1167 \def\forest@node@foreach@breadthfirst@reversed#1#2{% #1 = max level, #2 = code
1168 \forest@node@Foreach@breadthfirst@{\forest@cn}{@last}{@previous}{#1}{#2}}
1169 \def\forest@node@Foreach@breadthfirst#1#2#3{% #1 = node id, #2 = max level, #3 = code
1170 \forest@node@Foreach@breadthfirst@{#1}{@first}{@next}{#2}{#3}}
1171 \def\forest@node@Foreach@breadthfirst@reversed#1#2#3{% #1 = node id, #2 = max level, #3 = code
1172 \forest@node@Foreach@breadthfirst@{#1}{@last}{@previous}{#2}{#3}}
1173 \def\forest@node@Foreach@breadthfirst@#1#2#3#4#5{%
1174 % #1 = root node,
1175 % #2 = @first/@last, #3 = @next/@previous (must be in sync with #2),
1176 % #4 = max level (< 0 means infinite)
1177 % #5 = code to execute at each node
1178 \forest@node@Foreach@breadthfirst@processqueue{#1},{#2}{#3}{#4}{#5}%
1179 }
1180 \def\forest@node@Foreach@breadthfirst@processqueue#1#2#3#4#5{%
1181 % #1 = queue,
1182 % #2 = @first/@last, #3 = @next/@previous (must be in sync with #2),
1183 % #4 = max level (< 0 means infinite)
1184 % #5 = code to execute at each node
1185 \ifstrempy{#1}{}{%
1186 \forest@node@Foreach@breadthfirst@processqueue#1\forest@node@Foreach@breadthfirst@processqueue@
1187 {#2}{#3}{#4}{#5}%
1188 }%
1189 }
1190 \def\forest@node@Foreach@breadthfirst@processqueue#1,#2\forest@node@Foreach@breadthfirst@processqueue@#3#4#5
1191 % #1 = first,
1192 % #2 = rest,
1193 % #3 = @first/@last, #4 = next/previous (must be in sync with #2),
1194 % #5 = max level (< 0 means infinite)
1195 % #6 = code to execute at each node
1196 \forest@node@for{#1}{%
1197 #6%
1198 \ifnum#5<0
1199 \forest@node@getlistofchildren\forest@temp{#3}{#4}%
1200 \else
1201 \ifnum\forest@level>#5\relax
1202 \def\forest@temp{%
1203 \else
1204 \forest@node@getlistofchildren\forest@temp{#3}{#4}%
1205 \fi
1206 \fi
1207 \edef\forest@marshal{%
1208 \noexpand\forest@node@Foreach@breadthfirst@processqueue{\unexpanded{#2}\forest@temp}%
1209 {#3}{#4}{#5}{\unexpanded{#6}}%
1210 }\forest@marshal
1211 }%
1212 }
1213 \def\forest@node@getlistofchildren#1#2#3{% #1 = list cs, #2 = @first/@last, #3 = @next/@previous
1214 \forest@node@Getlistofchildren{\forest@cn}{#1}{#2}{#3}%
1215 }
1216 \def\forest@node@Getlistofchildren#1#2#3#4{% #1 = node, #2 = list cs, #3 = @first/@last, #4 = @next/@previous
1217 \def#2{}%
1218 \ifnum\forest@level=0

```

```

1219 \else
1220 \eappto#2{\forestOve{#1}{#3},}%
1221 \@escapeif{%
1222 \edef\forest@marshal{%
1223 \noexpand\forest@node@Getlistofchildren@{\forestOve{#1}{#3}}\noexpand#2{#4}%
1224 }\forest@marshal
1225 }%
1226 \fi
1227 }
1228 \def\forest@node@Getlistofchildren@#1#2#3{% #1 = node, #2 = list cs, #3 = @next/@previous
1229 \ifnum\forestOve{#1}{#3}=0
1230 \else
1231 \eappto#2{\forestOve{#1}{#3},}%
1232 \@escapeif{%
1233 \edef\forest@marshal{%
1234 \noexpand\forest@node@Getlistofchildren@{\forestOve{#1}{#3}}\noexpand#2{#3}%
1235 }\forest@marshal
1236 }%
1237 \fi
1238 }

Compute n, n', n children and level.
1239 \def\forest@node@Compute@numeric@ts@info@#1{%
1240 \forest@node@Foreach{#1}{\forest@node@@compute@numeric@ts@info}%
1241 \ifnum\forestOve{#1}{@parent}=0
1242 \else
1243 \forest@for@node{#1}{\forest@node@@compute@numeric@ts@info@nbar}%
1244 % hack: the parent of the node we called the update for gets +1 for n_children
1245 \edef\forest@node@temp{\forestOve{#1}{@parent}}%
1246 \forest@set{\forest@node@temp}{n children}{%
1247 \number\numexpr\forestOve{\forest@node@temp}{n children}-1%
1248 }%
1249 \fi
1250 \forest@node@Foreachdescendant{#1}{\forest@node@@compute@numeric@ts@info@nbar}%
1251 }
1252 \def\forest@node@@compute@numeric@ts@info{%
1253 \forest@set{n children}{0}%
1254 %
1255 \edef\forest@node@temp{\forestove{@previous}}%
1256 \ifnum\forest@node@temp=0
1257 \forest@set{n}{1}%
1258 \else
1259 \forest@set{n}{\number\numexpr\forestOve{\forest@node@temp}{n}+1}%
1260 \fi
1261 %
1262 \edef\forest@node@temp{\forestove{@parent}}%
1263 \ifnum\forest@node@temp=0
1264 \forest@set{n}{0}%
1265 \forest@set{n'}{0}%
1266 \forest@set{level}{0}%
1267 \else
1268 \forest@set{\forest@node@temp}{n children}{%
1269 \number\numexpr\forestOve{\forest@node@temp}{n children}+1%
1270 }%
1271 \forest@set{level}{%
1272 \number\numexpr\forestOve{\forest@node@temp}{level}+1%
1273 }%
1274 \fi
1275 }
1276 \def\forest@node@@compute@numeric@ts@info@nbar{%
1277 \forest@set{n'}{\number\numexpr\forestOve{\forestove{@parent}}{n children}-\forestove{n}+1}%

```

```

1278 }
1279 \def\forest@node@compute@numeric@ts@info#1{%
1280   \expandnumberarg\forest@node@Compute@numeric@ts@info@\forest@cn}%
1281 }
1282 \def\forest@node@Compute@numeric@ts@info#1{%
1283   \expandnumberarg\forest@node@Compute@numeric@ts@info@{#1}%
1284 }

Tree structure queries.

1285 \def\forest@node@rootid{%
1286   \expandnumberarg\forest@node@Rootid{\forest@cn}%
1287 }
1288 \def\forest@node@Rootid#1{% #1=node
1289   \ifnum\forest@ve{#1}{@parent}=0
1290     #1%
1291   \else
1292     \escapeif{\expandnumberarg\forest@node@Rootid{\forest@ve{#1}{@parent}}}%
1293   \fi
1294 }
1295 \def\forest@node@nthchildid#1{% #1=n
1296   \ifnum#1<1
1297     0%
1298   \else
1299     \expandnumberarg\forest@node@nthchildid@\number\forest@ve{@first}}{#1}%
1300   \fi
1301 }
1302 \def\forest@node@nthchildid@#1#2{%
1303   \ifnum#1=0
1304     0%
1305   \else
1306     \ifnum#2>1
1307       \escapeif{\expandtwonumberargs
1308         \forest@node@nthchildid@\forest@ve{#1}{@next}}{\numexpr#2-1}}%
1309     \else
1310       #1%
1311     \fi
1312   \fi
1313 }
1314 \def\forest@node@nbarthchildid#1{% #1=n
1315   \expandnumberarg\forest@node@nbarthchildid@\number\forest@ve{@last}}{#1}%
1316 }
1317 \def\forest@node@nbarthchildid@#1#2{%
1318   \ifnum#1=0
1319     0%
1320   \else
1321     \ifnum#2>1
1322       \escapeif{\expandtwonumberargs
1323         \forest@node@nbarthchildid@\forest@ve{#1}{@previous}}{\numexpr#2-1}}%
1324     \else
1325       #1%
1326     \fi
1327   \fi
1328 }
1329 \def\forest@node@nornbarthchildid#1{%
1330   \ifnum#1>0
1331     \forest@node@nthchildid{#1}%
1332   \else
1333     \ifnum#1<0
1334       \forest@node@nbarthchildid{-#1}%
1335     \else
1336       \forest@node@nornbarthchildid@error

```

```

1337 \fi
1338 \fi
1339 }
1340 \def\forest@node@nornbarthchildid@error{%
1341 \PackageError{forest}{In \string\forest@node@nornbarthchildid, n should !=0}{}}%
1342 }
1343 \def\forest@node@previousleafid{%
1344 \expandnumberarg\forest@node@Previousleafid{\forest@cn}%
1345 }
1346 \def\forest@node@Previousleafid#1{%
1347 \ifnum\forestOve{#1}{@previous}=0
1348 \escapeif{\expandnumberarg\forest@node@previousleafid@Goup{#1}}}%
1349 \else
1350 \expandnumberarg\forest@node@previousleafid@Gdown{\forestOve{#1}{@previous}}}%
1351 \fi
1352 }
1353 \def\forest@node@previousleafid@Goup#1{%
1354 \ifnum\forestOve{#1}{@parent}=0
1355 \PackageError{forest}{get previous leaf: this is the first leaf}{}}%
1356 \else
1357 \escapeif{\expandnumberarg\forest@node@Previousleafid{\forestOve{#1}{@parent}}}%
1358 \fi
1359 }
1360 \def\forest@node@previousleafid@Gdown#1{%
1361 \ifnum\forestOve{#1}{@last}=0
1362 #1%
1363 \else
1364 \escapeif{\expandnumberarg\forest@node@previousleafid@Gdown{\forestOve{#1}{@last}}}%
1365 \fi
1366 }
1367 \def\forest@node@nextleafid{%
1368 \expandnumberarg\forest@node@Nextleafid{\forest@cn}%
1369 }
1370 \def\forest@node@Nextleafid#1{%
1371 \ifnum\forestOve{#1}{@next}=0
1372 \escapeif{\expandnumberarg\forest@node@nextleafid@Goup{#1}}}%
1373 \else
1374 \expandnumberarg\forest@node@nextleafid@Gdown{\forestOve{#1}{@next}}}%
1375 \fi
1376 }
1377 \def\forest@node@nextleafid@Goup#1{%
1378 \ifnum\forestOve{#1}{@parent}=0
1379 \PackageError{forest}{get next leaf: this is the last leaf}{}}%
1380 \else
1381 \escapeif{\expandnumberarg\forest@node@Nextleafid{\forestOve{#1}{@parent}}}%
1382 \fi
1383 }
1384 \def\forest@node@nextleafid@Gdown#1{%
1385 \ifnum\forestOve{#1}{@first}=0
1386 #1%
1387 \else
1388 \escapeif{\expandnumberarg\forest@node@nextleafid@Gdown{\forestOve{#1}{@first}}}%
1389 \fi
1390 }
1391
1392
1393
1394 \def\forest@node@linearnextid{%
1395 \ifnum\forestove{@first}=0
1396 \expandafter\forest@node@linearnextnotdescendantid
1397 \else

```

```

1398 \forestove{@first}%
1399 \fi
1400 }
1401 \def\forest@node@linearnextnotdescendantid{%
1402 \expandnumberarg\forest@node@Linearnextnotdescendantid{\forest@cn}%
1403 }
1404 \def\forest@node@Linearnextnotdescendantid#1{%
1405 \ifnum\forestOve{#1}{@next}=0
1406 \ifnum\forestOve{#1}{@parent}=0
1407 0%
1408 \else
1409 \@escapeifif{\expandnumberarg\forest@node@Linearnextnotdescendantid{\forestOve{#1}{@parent}}}%
1410 \fi
1411 \else
1412 \forestOve{#1}{@next}%
1413 \fi
1414 }
1415
1416
1417 \def\forest@node@linearpreviousid{%
1418 \ifnum\forestove{@previous}=0
1419 \forestove{@parent}%
1420 \else
1421 \forest@node@previousleafid
1422 \fi
1423 }
1424 \def\forest@ifancestorof#1{% is the current node an ancestor of #1? Yes: #2, no: #3
1425 \expandnumberarg\forest@ifancestorof@{\forestOve{#1}{@parent}}%
1426 }
1427 \def\forest@ifancestorof@#1#2#3{%
1428 \ifnum#1=0
1429 \def\forest@ifancestorof@next{\@secondoftwo}%
1430 \else
1431 \ifnum\forest@cn=#1
1432 \def\forest@ifancestorof@next{\@firstoftwo}%
1433 \else
1434 \def\forest@ifancestorof@next{\expandnumberarg\forest@ifancestorof@{\forestOve{#1}{@parent}}}%
1435 \fi
1436 \fi
1437 \forest@ifancestorof@next{#2}{#3}%
1438 }

```

5.3 Node options

5.3.1 Option-declaration mechanism

Common code for declaring options.

```

1439 \def\forest@declarehandler#1#2#3{%#1=handler for specific type,#2=option name,#3=default value
1440 \pgfkeyssetvalue{/forest/#2}{#3}%
1441 \appto\forest@node@init{\forest@init{#2}}%
1442 \pgfkeyssetvalue{/forest/#2/node@or@reg}{\forest@cn}%
1443 \forest@convert@others@to@underscores{#2}\forest@pgfmathoptionname
1444 \edef\forest@marshal{%
1445 \noexpand#1{/forest/#2}{/forest}{#2}{\forest@pgfmathoptionname}%
1446 }\forest@marshal
1447 }
1448 \def\forest@def@with@pgfeov#1#2{% \pgfeov mustn't occur in the arg of the .code handler!!!
1449 \long\def#1##1\pgfeov{#2}%
1450 }

```

Option-declaration handlers.

```

1451 \def\forest@declaretoks@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1452   \forest@declaretoks@handler@A{#1}{#2}{#3}{#4}{}%
1453 }
1454 \def\forest@declarekeylist@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1455   \forest@declaretoks@handler@A{#1}{#2}{#3}{#4}{,}%
1456   \forest@copycommandkey{#1}{#1'}%
1457   \pgfkeyssetvalue{#1'/option@name}{#3}%
1458   \forest@copycommandkey{#1+}{#1}%
1459   \pgfkeysalso{#1-/.code={%
1460     \forest@fornode{\forest@setter@node}{%
1461       \forest@node@removekeysfromkeylist{##1}{#3}%
1462     }}%
1463   \pgfkeyssetvalue{#1-/.option@name}{#3}%
1464 }
1465 \def\forest@declaretoks@handler@A#1#2#3#4#5{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
1466   \pgfkeysalso{%
1467     #1/.code={\forest@set{\forest@setter@node}{#3}{##1}},
1468     #1+/.code={\forest@appto{\forest@setter@node}{#3}{#5##1}},
1469     #2/+##3/.code={\forest@preto{\forest@setter@node}{#3}{##1#5}},
1470     #2/if #3/.code n args={3}{%
1471       \forest@toget{#3}\forest@temp@option@value
1472       \edef\forest@temp@compared@value{\unexpanded{##1}}%
1473       \ifx\forest@temp@option@value\forest@temp@compared@value
1474         \pgfkeysalso{##2}%
1475       \else
1476         \pgfkeysalso{##3}%
1477       \fi
1478     },
1479     #2/if in #3/.code n args={3}{%
1480       \forest@toget{#3}\forest@temp@option@value
1481       \edef\forest@temp@compared@value{\unexpanded{##1}}%
1482       \expandafter\expandafter\expandafter\pgfutil@in@\expandafter\expandafter\expandafter{\expandafter\forest@
1483       \ifpgfutil@in@
1484         \pgfkeysalso{##2}%
1485       \else
1486         \pgfkeysalso{##3}%
1487       \fi
1488     },
1489     #2/where #3/.style n args={3}{for tree={#2/if #3={##1}{##2}{##3}}},
1490     #2/where in #3/.style n args={3}{for tree={#2/if in #3={##1}{##2}{##3}}}%
1491   }%
1492   \pgfkeyssetvalue{#1'/option@name}{#3}%
1493   \pgfkeyssetvalue{#1+/.option@name}{#3}%
1494   \pgfkeyssetvalue{#2/+##3/.option@name}{#3}%
1495   \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@toks{##1}{#3}}%
1496 }
1497 \def\forest@declareautowrappedtoks@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
1498   \forest@declaretoks@handler{#1}{#2}{#3}{#4}%
1499   \forest@copycommandkey{#1}{#1'}%
1500   \pgfkeysalso{#1/.style={#1'/.wrap value={##1}}}%
1501   \pgfkeyssetvalue{#1'/option@name}{#3}%
1502   \forest@copycommandkey{#1+}{#1+'}%
1503   \pgfkeysalso{#1+/.style={#1+'/.wrap value={##1}}}%
1504   \pgfkeyssetvalue{#1+'/.option@name}{#3}%
1505   \forest@copycommandkey{#2/+##3}{#2/+##3'}%
1506   \pgfkeysalso{#2/+##3/.style={#2/+##3'/.wrap value={##1}}}%
1507   \pgfkeyssetvalue{#2/+##3'/option@name}{#3}%
1508 }
1509 \def\forest@declarereadonlydimen@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1510   \pgfkeysalso{%
1511     #2/if #3/.code n args={3}{%

```

```

1512 \forestoget{#3}\forest@temp@option@value
1513 \ifdim\forest@temp@option@value=##1\relax
1514 \pgfkeysalso{##2}%
1515 \else
1516 \pgfkeysalso{##3}%
1517 \fi
1518 },
1519 #2/where #3/.style n args={3}{for tree={#2/if #3={##1}{##2}{##3}}},
1520 }%
1521 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@dimen{##1}{#3}}%
1522 }
1523 \def\forest@declaredimen@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1524 \forest@declarereadonlydimen@handler{#1}{#2}{#3}{#4}%
1525 \pgfkeysalso{%
1526 #1/.code={%
1527 \pgfmathsetlengthmacro\forest@temp{##1}%
1528 \forestOlet{\forest@setter@node}{#3}\forest@temp
1529 },
1530 #1+/.code={%
1531 \pgfmathsetlengthmacro\forest@temp{##1}%
1532 \pgfutil@tempdima=\forestove{#3}
1533 \advance\pgfutil@tempdima\forest@temp\relax
1534 \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1535 },
1536 #1-/.code={%
1537 \pgfmathsetlengthmacro\forest@temp{##1}%
1538 \pgfutil@tempdima=\forestove{#3}
1539 \advance\pgfutil@tempdima-\forest@temp\relax
1540 \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1541 },
1542 #1*/.style={%
1543 #1={#4()}*(##1)}%
1544 },
1545 #1:/ .style={%
1546 #1={#4()/(##1)}%
1547 },
1548 #1'/.code={%
1549 \pgfutil@tempdima=##1\relax
1550 \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1551 },
1552 #1'+/.code={%
1553 \pgfutil@tempdima=\forestove{#3}\relax
1554 \advance\pgfutil@tempdima##1\relax
1555 \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1556 },
1557 #1'-/.code={%
1558 \pgfutil@tempdima=\forestove{#3}\relax
1559 \advance\pgfutil@tempdima-##1\relax
1560 \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1561 },
1562 #1'*/.style={%
1563 \pgfutil@tempdima=\forestove{#3}\relax
1564 \multiply\pgfutil@tempdima##1\relax
1565 \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1566 },
1567 #1':/.style={%
1568 \pgfutil@tempdima=\forestove{#3}\relax
1569 \divide\pgfutil@tempdima##1\relax
1570 \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1571 },
1572 }%

```

```

1573 \pgfkeyssetvalue{#1/option@name}{#3}%
1574 \pgfkeyssetvalue{#1+/option@name}{#3}%
1575 \pgfkeyssetvalue{#1-/option@name}{#3}%
1576 \pgfkeyssetvalue{#1*/option@name}{#3}%
1577 \pgfkeyssetvalue{#1:/option@name}{#3}%
1578 \pgfkeyssetvalue{#1'/option@name}{#3}%
1579 \pgfkeyssetvalue{#1'+/option@name}{#3}%
1580 \pgfkeyssetvalue{#1'-/option@name}{#3}%
1581 \pgfkeyssetvalue{#1'*/option@name}{#3}%
1582 \pgfkeyssetvalue{#1':/option@name}{#3}%
1583 }
1584 \def\forest@declarereadonlycount@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1585 \pgfkeysalso{
1586 #2/if #3/.code n args={3}{%
1587 \foresttoget{#3}\forest@temp@option@value
1588 \ifnum\forest@temp@option@value=##1\relax
1589 \pgfkeysalso{##2}%
1590 \else
1591 \pgfkeysalso{##3}%
1592 \fi
1593 },
1594 #2/where #3/.style n args={3}{for tree={#2/if #3={##1}{##2}{##3}}},
1595 }%
1596 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@count{##1}{#3}}%
1597 }
1598 \def\forest@declarecount@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1599 \forest@declarereadonlycount@handler{#1}{#2}{#3}{#4}%
1600 \pgfkeysalso{
1601 #1/.code={%
1602 \pgfmathtruncatemacro\forest@temp{##1}%
1603 \forestOlet{\forest@setter@node}{#3}\forest@temp
1604 },
1605 #1+/.code={%
1606 \pgfmathtruncatemacro\forest@temp{##1}%
1607 \c@pgf@counta=\forestove{#3}\relax
1608 \advance\c@pgf@counta\forest@temp\relax
1609 \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1610 },
1611 #1-/.code={%
1612 \pgfmathtruncatemacro\forest@temp{##1}%
1613 \c@pgf@counta=\forestove{#3}\relax
1614 \advance\c@pgf@counta-\forest@temp\relax
1615 \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1616 },
1617 #1*/.code={%
1618 \pgfmathtruncatemacro\forest@temp{##1}%
1619 \c@pgf@counta=\forestove{#3}\relax
1620 \multiply\c@pgf@counta\forest@temp\relax
1621 \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1622 },
1623 #1:/.code={%
1624 \pgfmathtruncatemacro\forest@temp{##1}%
1625 \c@pgf@counta=\forestove{#3}\relax
1626 \divide\c@pgf@counta\forest@temp\relax
1627 \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1628 },
1629 #1'/.code={%
1630 \c@pgf@counta=##1\relax
1631 \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1632 },
1633 #1'+/.code={%

```



```

1634 \c@pgf@counta=\forestove{#3}\relax
1635 \advance\c@pgf@counta##1\relax
1636 \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1637 },
1638 #1'-.code={%
1639 \c@pgf@counta=\forestove{#3}\relax
1640 \advance\c@pgf@counta-##1\relax
1641 \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1642 },
1643 #1'*/.style={%
1644 \c@pgf@counta=\forestove{#3}\relax
1645 \multiply\c@pgf@counta##1\relax
1646 \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1647 },
1648 #1':/.style={%
1649 \c@pgf@counta=\forestove{#3}\relax
1650 \divide\c@pgf@counta##1\relax
1651 \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1652 },
1653 }%
1654 \pgfkeyssetvalue{#1/option@name}{#3}%
1655 \pgfkeyssetvalue{#1+/option@name}{#3}%
1656 \pgfkeyssetvalue{#1-/option@name}{#3}%
1657 \pgfkeyssetvalue{#1*/option@name}{#3}%
1658 \pgfkeyssetvalue{#1:/option@name}{#3}%
1659 \pgfkeyssetvalue{#1'/option@name}{#3}%
1660 \pgfkeyssetvalue{#1'+/option@name}{#3}%
1661 \pgfkeyssetvalue{#1'-/option@name}{#3}%
1662 \pgfkeyssetvalue{#1'*/option@name}{#3}%
1663 \pgfkeyssetvalue{#1':/option@name}{#3}%
1664 }
1665 \def\forest@declareboolean@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1666 \pgfkeysalso{%
1667 #1/.code={%
1668 \ifstrequal{##1}{1}{%
1669 \forestOset{\forest@setter@node}{#3}{1}%
1670 }{%
1671 \ifstrequal{##1}{0}{%
1672 \forestOset{\forest@setter@node}{#3}{0}%
1673 }{%
1674 \pgfmathifthenelse{##1}{1}{0}%
1675 \forestOlet{\forest@setter@node}{#3}\pgfmathresult
1676 }%
1677 }%
1678 },
1679 #1/.default=1,
1680 #2/not #3/.code={\forestOset{\forest@setter@node}{#3}{0}},
1681 #2/if #3/.code 2 args={%
1682 \forestoget{#3}\forest@temp@option@value
1683 \ifnum\forest@temp@option@value=1
1684 \pgfkeysalso{##1}%
1685 \else
1686 \pgfkeysalso{##2}%
1687 \fi
1688 },
1689 #2/where #3/.style 2 args={for tree={#2/if #3={##1}{##2}}}
1690 }%
1691 \pgfkeyssetvalue{#1/option@name}{#3}%
1692 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@count{##1}{#3}}%
1693 }
1694 \forestset{

```

```

1695 declare toks/.code 2 args={%
1696   \forest@declarehandler\forest@declaretoks@handler{#1}{#2}%
1697 },
1698 declare autowrapped toks/.code 2 args={%
1699   \forest@declarehandler\forest@declareautowrappedtoks@handler{#1}{#2}%
1700 },
1701 declare keylist/.code 2 args={%
1702   \forest@declarehandler\forest@declarekeylist@handler{#1}{#2}%
1703 },
1704 declare readonly dimen/.code={%
1705   \forest@declarehandler\forest@declarereadonlydimen@handler{#1}{}%
1706 },
1707 declare dimen/.code 2 args={%
1708   \forest@declarehandler\forest@declaredimen@handler{#1}{#2}%
1709 },
1710 declare readonly count/.code={%
1711   \forest@declarehandler\forest@declarereadonlycount@handler{#1}{}%
1712 },
1713 declare count/.code 2 args={%
1714   \forest@declarehandler\forest@declarecount@handler{#1}{#2}%
1715 },
1716 declare boolean/.code 2 args={%
1717   \forest@declarehandler\forest@declareboolean@handler{#1}{#2}%
1718 },
1719 /handlers/.restore default value/.code={%
1720   \edef\forest@handlers@currentpath{\pgfkeyscurrentpath}%
1721   \pgfkeysgetvalue{\pgfkeyscurrentpath/option@name}\forest@currentoptionname
1722   \pgfkeysgetvalue{/forest/\forest@currentoptionname}\forest@temp
1723   \expandafter\pgfkeysalso\expandafter{\forest@handlers@currentpath/.expand once=\forest@temp}%
1724 },
1725 /handlers/.pgfmath/.code={%
1726   \pgfmathparse{#1}%
1727   \pgfkeysalso{\pgfkeyscurrentpath/.expand once=\pgfmathresult}%
1728 },
1729 /handlers/.wrap value/.code={%
1730   \edef\forest@handlers@wrap@currentpath{\pgfkeyscurrentpath}%
1731   \pgfkeysgetvalue{\forest@handlers@wrap@currentpath/option@name}\forest@currentoptionname
1732   \forest@get{\pgfkeysvalueof{/forest/\forest@currentoptionname/node@or@reg}}{\forest@currentoptionname}\forest@temp
1733   \forest@def@with@pgfeov\forest@wrap@code{#1}%
1734   \expandafter\edef\expandafter\forest@wrapped@value\expandafter{\expandafter\expandonce\expandafter{\expandafter\forest@temp@toks}}
1735   \pgfkeysalso{\forest@handlers@wrap@currentpath/.expand once=\forest@wrapped@value}%
1736 },
1737 /handlers/.option/.code={%
1738   \edef\forest@marshal{%
1739     \noexpand\pgfkeysalso{\pgfkeyscurrentpath={\forestoption{#1}}}%
1740   }\forest@marshal
1741 },
1742 /handlers/.register/.code={%
1743   \edef\forest@marshal{%
1744     \noexpand\pgfkeysalso{\pgfkeyscurrentpath={\forestregister{#1}}}%
1745   }\forest@marshal
1746 },
1747 /handlers/.wrap pgfmath arg/.code 2 args={%
1748   \pgfmathparse{#2}\let\forest@wrap@arg\pgfmathresult
1749   \edef\forest@wrap@args{\expandonce\forest@wrap@arg}%
1750   \def\forest@wrap@code##1{#1}%
1751   \expandafter\expandafter\expandafter\forest@temp@toks\expandafter\expandafter\expandafter{\expandafter\forest@temp@toks}}
1752   \pgfkeysalso{\pgfkeyscurrentpath/.expand once=\the\forest@temp@toks}%
1753 },
1754 /handlers/.wrap 2 pgfmath args/.code n args={3}{%
1755   \pgfmathparse{#2}\let\forest@wrap@arg\pgfmathresult

```

```

1756 \pgfmathparse{#3}\let\forest@wrap@arg@ii\pgfmathresult
1757 \edef\forest@wrap@args{\expandonce\forest@wrap@arg@i{\expandonce\forest@wrap@arg@ii}}%
1758 \def\forest@wrap@code##1##2{#1}%
1759 \expandafter\expandafter\expandafter\def\expandafter\expandafter\expandafter\forest@wrapped\expandafter\exp
1760 \pgfkeysalso{\pgfkeyscurrentpath/.expand once=\forest@wrapped}%
1761 },
1762 /handlers/.wrap 3 pgfmath args/.code n args={4}{%
1763 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{-}{-}{-}{-}{3}%
1764 \forest@wrap@n@pgfmath@do{#1}{3}},
1765 /handlers/.wrap 4 pgfmath args/.code n args={5}{%
1766 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{-}{-}{-}{4}%
1767 \forest@wrap@n@pgfmath@do{#1}{4}},
1768 /handlers/.wrap 5 pgfmath args/.code n args={6}{%
1769 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{-}{-}{5}%
1770 \forest@wrap@n@pgfmath@do{#1}{5}},
1771 /handlers/.wrap 6 pgfmath args/.code n args={7}{%
1772 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{#7}{-}{6}%
1773 \forest@wrap@n@pgfmath@do{#1}{6}},
1774 /handlers/.wrap 7 pgfmath args/.code n args={8}{%
1775 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{#7}{#8}{-}{7}%
1776 \forest@wrap@n@pgfmath@do{#1}{7}},
1777 /handlers/.wrap 8 pgfmath args/.code n args={9}{%
1778 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}{8}%
1779 \forest@wrap@n@pgfmath@do{#1}{8}},
1780 /handlers/.process args/.code={%
1781 \forest@processargs#1\forest@eov\forest@END
1782 },
1783 }
1784 \def\forest@wrap@n@pgfmath@args#1#2#3#4#5#6#7#8#9{%
1785 \pgfmathparse{#1}\let\forest@wrap@arg@i\pgfmathresult
1786 \ifnum#9>1 \pgfmathparse{#2}\let\forest@wrap@arg@ii\pgfmathresult\fi
1787 \ifnum#9>2 \pgfmathparse{#3}\let\forest@wrap@arg@iii\pgfmathresult\fi
1788 \ifnum#9>3 \pgfmathparse{#4}\let\forest@wrap@arg@iv\pgfmathresult\fi
1789 \ifnum#9>4 \pgfmathparse{#5}\let\forest@wrap@arg@v\pgfmathresult\fi
1790 \ifnum#9>5 \pgfmathparse{#6}\let\forest@wrap@arg@vi\pgfmathresult\fi
1791 \ifnum#9>6 \pgfmathparse{#7}\let\forest@wrap@arg@vii\pgfmathresult\fi
1792 \ifnum#9>7 \pgfmathparse{#8}\let\forest@wrap@arg@viii\pgfmathresult\fi
1793 \edef\forest@wrap@args{%
1794 {\expandonce\forest@wrap@arg@i}
1795 \ifnum#9>1 {\expandonce\forest@wrap@arg@ii}\fi
1796 \ifnum#9>2 {\expandonce\forest@wrap@arg@iii}\fi
1797 \ifnum#9>3 {\expandonce\forest@wrap@arg@iv}\fi
1798 \ifnum#9>4 {\expandonce\forest@wrap@arg@v}\fi
1799 \ifnum#9>5 {\expandonce\forest@wrap@arg@vi}\fi
1800 \ifnum#9>6 {\expandonce\forest@wrap@arg@vii}\fi
1801 \ifnum#9>7 {\expandonce\forest@wrap@arg@viii}\fi
1802 }%
1803 }
1804 \def\forest@wrap@n@pgfmath@do#1#2{%
1805 \ifcase#2\relax
1806 \or\def\forest@wrap@code##1{#1}%
1807 \or\def\forest@wrap@code##1##2{#1}%
1808 \or\def\forest@wrap@code##1##2##3{#1}%
1809 \or\def\forest@wrap@code##1##2##3##4{#1}%
1810 \or\def\forest@wrap@code##1##2##3##4##5{#1}%
1811 \or\def\forest@wrap@code##1##2##3##4##5##6{#1}%
1812 \or\def\forest@wrap@code##1##2##3##4##5##6##7{#1}%
1813 \or\def\forest@wrap@code##1##2##3##4##5##6##7##8{#1}%
1814 \fi
1815 \expandafter\expandafter\expandafter\def\expandafter\expandafter\expandafter\forest@wrapped\expandafter\exp
1816 \pgfkeysalso{\pgfkeyscurrentpath/.expand once=\forest@wrapped}%

```

```

1817 }
1818 \newtoks\forest@processargs@result
1819 \newtoks\forest@processargs@current
1820 \newif\ifforest@processargs@append
1821 \def\forest@eov{\forest@eov}
1822 \def\forest@processargs#1#2\forest@END{% #1 = processing instructions, #2 = args
1823   \forest@processargs@result{}}%
1824   \forest@processargs@current{}}%
1825   \forest@processargs@appendfalse
1826   \forest@processargs@getins#1.\forest@END#2\forest@END
1827 }
1828 \def\forest@processargs@maybeappend{%
1829   \ifforest@processargs@append
1830     \eapptotoks\forest@processargs@result{{\the\forest@processargs@current}}}%
1831     \forest@processargs@current{}}%
1832   \fi
1833 }
1834 \def\forest@processargs@getins#1#2\forest@END#3\forest@END{% #1 = first instruction, #2 = rest of instruction
1835   \csname forest@processargs@ins@#1\endcsname#2\forest@END#3\forest@END
1836 }
1837 \csdef\forest@processargs@ins@.\forest@END#1\forest@END{%
1838   \forest@processargs@maybeappend
1839   \forest@processargs@appremainingargs#1\forest@END
1840 }
1841 \def\forest@processargs@appremainingargs#1#2\forest@END{%
1842   \edef\forest@temp{\unexpanded{#1}}%
1843   \ifx\forest@temp\forest@eov
1844     \let\forest@processargs@next\forest@processargs@done
1845   \else
1846     \apptotoks\forest@processargs@result{{#1}}%
1847     \let\forest@processargs@next\forest@processargs@appremainingargs
1848   \fi
1849   \forest@processargs@next#2\forest@END
1850 }
1851 \def\forest@processargs@done#1\forest@END{%
1852   \pgfkeysalso{\pgfkeyscurrentpath/.expanded=\the\forest@processargs@result}%
1853 }
1854 \csdef\forest@processargs@ins@_#1\forest@END#2#3\forest@END{% no processing
1855   % #1 = rest of ins, #2 = first arg, #3 = rest of args
1856   \forest@processargs@maybeappend
1857   \forest@processargs@current{#2}%
1858   \forest@processargs@appendtrue
1859   \forest@processargs@getins#1\forest@END#3\forest@END
1860 }
1861 \csdef\forest@processargs@ins@x#1\forest@END#2#3\forest@END{% expand
1862   % #1 = rest of ins, #2 = first arg, #3 = rest of args
1863   \forest@processargs@maybeappend
1864   \etotoks\forest@processargs@current{#2}%
1865   \forest@processargs@appendtrue
1866   \forest@processargs@getins#1\forest@END#3\forest@END
1867 }
1868 \csdef\forest@processargs@ins@o#1\forest@END#2#3\forest@END{% expand once
1869   % #1 = rest of ins, #2 = first arg, #3 = rest of args
1870   \forest@processargs@maybeappend
1871   \expandafter\forest@processargs@current\expandafter{#2}%
1872   \forest@processargs@appendtrue
1873   \forest@processargs@getins#1\forest@END#3\forest@END
1874 }
1875 \csdef\forest@processargs@ins@0#1\forest@END#2#3\forest@END{% option
1876   % #1 = rest of ins, #2 = first arg, #3 = rest of args
1877   \forest@processargs@maybeappend

```

```

1878 \etotoks\forest@processargs@current{\forestoption{#2}}%
1879 \forest@processargs@appendtrue
1880 \forest@processargs@getins#1\forest@END#3\forest@END
1881 }
1882 \csdef{forest@processargs@ins@R}#1\forest@END#2#3\forest@END{% register
1883 % #1 = rest of ins, #2 = first arg, #3 = rest of args
1884 \forest@processargs@maybeappend
1885 \etotoks\forest@processargs@current{\forestregister{#2}}%
1886 \forest@processargs@appendtrue
1887 \forest@processargs@getins#1\forest@END#3\forest@END
1888 }
1889 \csdef{forest@processargs@ins@P}#1\forest@END#2#3\forest@END{% pgfmath expression
1890 % #1 = rest of ins, #2 = first arg, #3 = rest of args
1891 \forest@processargs@maybeappend
1892 \pgfmathparse{#2}%
1893 \expandafter\forest@processargs@current\expandafter{\pgfmathresult}%
1894 \forest@processargs@appendtrue
1895 \forest@processargs@getins#1\forest@END#3\forest@END
1896 }
1897 \csdef{forest@processargs@ins@+}#1\forest@END#2\forest@END{% join processors
1898 \forest@processargs@appendfalse
1899 \edef\forest@marshal{%
1900   \unexpanded{\forest@processargs@getins#1\forest@END}{\the\forest@processargs@current}\unexpanded{#2\forest@END}
1901 }
1902 }
1903 \csdef{forest@processargs@ins@r}#1\forest@END#2#3\forest@END{% reverse keylist
1904 % #1 = rest of ins, #2 = first arg, #3 = rest of args
1905 \forest@processargs@maybeappend
1906 \forest@processargs@current{}%
1907 \pgfqkeys{/forest}{split={#2}{,}{reverse@keylist}}%
1908 \forest@processargs@appendtrue
1909 \forest@processargs@getins#1\forest@END#3\forest@END
1910 }
1911 \forestset{%
1912   reverse@keylist/.code={%
1913     \epretotoks\forest@processargs@current{#1,}%
1914   },
1915 }

```

5.3.2 Registers

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

```

1916 \def\forest@pgfmathhelper@register@toks#1#2{% #1 is discarded: it is present only for analogy with options
1917   \forestrget{#2}\pgfmathresult
1918 }
1919 \def\forest@pgfmathhelper@register@dimen#1#2{%
1920   \forestrget{#2}\forest@temp
1921   \pgfmathparse{+\forest@temp}%
1922 }
1923 \def\forest@pgfmathhelper@register@count#1#2{%
1924   \forestrget{#2}\forest@temp
1925   \pgfmathtruncatemacro\pgfmathresult{\forest@temp}%
1926 }
1927 \def\forest@declareregisterhandler#1#2{%#1=handler for specific type,#2=option name
1928   \pgfkeyssetvalue{/forest/#2/node@or@reg}{register}%
1929   \forest@convert@others@to@underscores{#2}\forest@pgfmathoptionname
1930   \edef\forest@marshal{%
1931     \noexpand#1{/forest/#2}{/forest}{#2}{\forest@pgfmathoptionname}%
1932   }
1933 }

```

```

1934 \def\forest@declaretoksregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmname
1935 \forest@declaretoksregister@handler@A{#1}{#2}{#3}{#4}{}%
1936 }
1937 \def\forest@declarekeylistregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmname
1938 \forest@declaretoksregister@handler@A{#1}{#2}{#3}{#4}{,}%
1939 \forest@copycommandkey{#1}{#1'}%
1940 \pgfkeyssetvalue{#1'/option@name}{#3}%
1941 \forest@copycommandkey{#1+}{#1}%
1942 \pgfkeysalso{#1-/.code={%
1943 \forest@fornode{register}{%
1944 \forest@node@removekeysfromkeylist{##1}{#3}%
1945 }}}%
1946 \pgfkeyssetvalue{#1-/.option@name}{#3}%
1947 }
1948 \def\forest@declaretoksregister@handler@A#1#2#3#4#5{% #1=key,#2=path,#3=name,#4=pgfmname,#5=infix
1949 \pgfkeysalso{%
1950 #1/.code={\forestset{#3}{##1}},
1951 #1+/.code={\forestrappto{#3}{#5##1}},
1952 #2/+#3/.code={\forestrpreto{#3}{##1#5}},
1953 #2/if #3/.code n args={3}{%
1954 \forestget{#3}\forest@temp@option@value
1955 \edef\forest@temp@compared@value{\unexpanded{##1}}%
1956 \ifx\forest@temp@option@value\forest@temp@compared@value
1957 \pgfkeysalso{##2}%
1958 \else
1959 \pgfkeysalso{##3}%
1960 \fi
1961 },
1962 #2/if in #3/.code n args={3}{%
1963 \forestget{#3}\forest@temp@option@value
1964 \edef\forest@temp@compared@value{\unexpanded{##1}}%
1965 \expandafter\expandafter\expandafter\pgfutil@in@\expandafter\expandafter\expandafter{\expandafter\forest@
1966 \ifpgfutil@in@
1967 \pgfkeysalso{##2}%
1968 \else
1969 \pgfkeysalso{##3}%
1970 \fi
1971 },
1972 }%
1973 \pgfkeyssetvalue{#1/.option@name}{#3}%
1974 \pgfkeyssetvalue{#1+/.option@name}{#3}%
1975 \pgfkeyssetvalue{#2/+#3/.option@name}{#3}%
1976 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@toks{##1}{#3}}%
1977 }
1978 \def\forest@declareautowrappedtoksregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmname,#5=infix
1979 \forest@declaretoksregister@handler{#1}{#2}{#3}{#4}%
1980 \forest@copycommandkey{#1}{#1'}%
1981 \pgfkeysalso{#1/.style={#1'/.wrap value={##1}}}%
1982 \pgfkeyssetvalue{#1'/option@name}{#3}%
1983 \forest@copycommandkey{#1+}{#1+}%
1984 \pgfkeysalso{#1+/.style={#1+/.wrap value={##1}}}%
1985 \pgfkeyssetvalue{#1+/.option@name}{#3}%
1986 \forest@copycommandkey{#2/+#3}{#2/+#3}%
1987 \pgfkeysalso{#2/+#3/.style={#2/+#3'/.wrap value={##1}}}%
1988 \pgfkeyssetvalue{#2/+#3'/option@name}{#3}%
1989 }
1990 \def\forest@declarereadonlydimenregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmname
1991 \pgfkeysalso{%
1992 #2/if #3/.code n args={3}{%
1993 \forestget{#3}\forest@temp@option@value
1994 \ifdim\forest@temp@option@value=##1\relax

```

```

1995     \pgfkeysalso{##2}%
1996     \else
1997     \pgfkeysalso{##3}%
1998     \fi
1999 },
2000 }%
2001 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@dimen{##1}{#3}}%
2002 }
2003 \def\forest@declaredimenregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2004 \forest@declarereadonlydimenregister@handler{#1}{#2}{#3}{#4}%
2005 \pgfkeysalso{%
2006   #1/.code={%
2007     \pgfmathsetlengthmacro\forest@temp{##1}%
2008     \forestrllet{#3}\forest@temp
2009   },
2010   #1+/.code={%
2011     \pgfmathsetlengthmacro\forest@temp{##1}%
2012     \pgfutil@tempdima=\forestrive{#3}
2013     \advance\pgfutil@tempdima\forest@temp\relax
2014     \forestreset{#3}{\the\pgfutil@tempdima}%
2015   },
2016   #1-/.code={%
2017     \pgfmathsetlengthmacro\forest@temp{##1}%
2018     \pgfutil@tempdima=\forestrive{#3}
2019     \advance\pgfutil@tempdima-\forest@temp\relax
2020     \forestreset{#3}{\the\pgfutil@tempdima}%
2021   },
2022   #1*/.style={%
2023     #1={#4()}*(##1)}%
2024   },
2025   #1:/ .style={%
2026     #1={#4()}/(##1)}%
2027   },
2028   #1'/.code={%
2029     \pgfutil@tempdima=##1\relax
2030     \forestreset{#3}{\the\pgfutil@tempdima}%
2031   },
2032   #1'+/.code={%
2033     \pgfutil@tempdima=\forestrive{#3}\relax
2034     \advance\pgfutil@tempdima##1\relax
2035     \forestreset{#3}{\the\pgfutil@tempdima}%
2036   },
2037   #1'-/.code={%
2038     \pgfutil@tempdima=\forestrive{#3}\relax
2039     \advance\pgfutil@tempdima-##1\relax
2040     \forestreset{#3}{\the\pgfutil@tempdima}%
2041   },
2042   #1'*/.style={%
2043     \pgfutil@tempdima=\forestrive{#3}\relax
2044     \multiply\pgfutil@tempdima##1\relax
2045     \forestreset{#3}{\the\pgfutil@tempdima}%
2046   },
2047   #1':/.style={%
2048     \pgfutil@tempdima=\forestrive{#3}\relax
2049     \divide\pgfutil@tempdima##1\relax
2050     \forestreset{#3}{\the\pgfutil@tempdima}%
2051   },
2052 }%
2053 \pgfkeyssetvalue{#1/option@name}{#3}%
2054 \pgfkeyssetvalue{#1+/option@name}{#3}%
2055 \pgfkeyssetvalue{#1-/option@name}{#3}%

```

```

2056 \pgfkeyssetvalue{#1*/option@name}{#3}%
2057 \pgfkeyssetvalue{#1:/option@name}{#3}%
2058 \pgfkeyssetvalue{#1'/option@name}{#3}%
2059 \pgfkeyssetvalue{#1'+/option@name}{#3}%
2060 \pgfkeyssetvalue{#1'-/option@name}{#3}%
2061 \pgfkeyssetvalue{#1'*/option@name}{#3}%
2062 \pgfkeyssetvalue{#1':/option@name}{#3}%
2063 }
2064 \def\forest@declarereadonlycountregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmname
2065 \pgfkeysalso{
2066 #2/if #3/.code n args={3}{%
2067 \forestget{#3}\forest@temp@option@value
2068 \ifnum\forest@temp@option@value=##1\relax
2069 \pgfkeysalso{##2}%
2070 \else
2071 \pgfkeysalso{##3}%
2072 \fi
2073 },
2074 }%
2075 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@count{##1}{#3}}%
2076 }
2077 \def\forest@declarecountregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmname
2078 \forest@declarereadonlycountregister@handler{#1}{#2}{#3}{#4}%
2079 \pgfkeysalso{
2080 #1/.code={%
2081 \pgfmathtruncatemacro\forest@temp{##1}%
2082 \forestrlt{#3}\forest@temp
2083 },
2084 #1+/.code={%
2085 \pgfmathtruncatemacro\forest@temp{##1}%
2086 \c@pgf@counta=\forestrve{#3}\relax
2087 \advance\c@pgf@counta\forest@temp\relax
2088 \forestreset{#3}{\the\c@pgf@counta}%
2089 },
2090 #1-/.code={%
2091 \pgfmathtruncatemacro\forest@temp{##1}%
2092 \c@pgf@counta=\forestrve{#3}\relax
2093 \advance\c@pgf@counta-\forest@temp\relax
2094 \forestreset{#3}{\the\c@pgf@counta}%
2095 },
2096 #1*/.code={%
2097 \pgfmathtruncatemacro\forest@temp{##1}%
2098 \c@pgf@counta=\forestrve{#3}\relax
2099 \multiply\c@pgf@counta\forest@temp\relax
2100 \forestreset{#3}{\the\c@pgf@counta}%
2101 },
2102 #1:/ .code={%
2103 \pgfmathtruncatemacro\forest@temp{##1}%
2104 \c@pgf@counta=\forestrve{#3}\relax
2105 \divide\c@pgf@counta\forest@temp\relax
2106 \forestreset{#3}{\the\c@pgf@counta}%
2107 },
2108 #1'/.code={%
2109 \c@pgf@counta=##1\relax
2110 \forestreset{#3}{\the\c@pgf@counta}%
2111 },
2112 #1'+/.code={%
2113 \c@pgf@counta=\forestrve{#3}\relax
2114 \advance\c@pgf@counta##1\relax
2115 \forestreset{#3}{\the\c@pgf@counta}%
2116 },

```



```

2117 #1'-.code={%
2118 \c@pgf@counta=\forestrv{#3}\relax
2119 \advance\c@pgf@counta-##1\relax
2120 \forestreset{#3}{\the\c@pgf@counta}%
2121 },
2122 #1'*/.style={%
2123 \c@pgf@counta=\forestrv{#3}\relax
2124 \multiply\c@pgf@counta##1\relax
2125 \forestreset{#3}{\the\c@pgf@counta}%
2126 },
2127 #1':/.style={%
2128 \c@pgf@counta=\forestrv{#3}\relax
2129 \divide\c@pgf@counta##1\relax
2130 \forestreset{#3}{\the\c@pgf@counta}%
2131 },
2132 }%
2133 \pgfkeyssetvalue{#1/option@name}{#3}%
2134 \pgfkeyssetvalue{#1+/option@name}{#3}%
2135 \pgfkeyssetvalue{#1-/option@name}{#3}%
2136 \pgfkeyssetvalue{#1*/option@name}{#3}%
2137 \pgfkeyssetvalue{#1:/option@name}{#3}%
2138 \pgfkeyssetvalue{#1'/option@name}{#3}%
2139 \pgfkeyssetvalue{#1'+/option@name}{#3}%
2140 \pgfkeyssetvalue{#1'-/option@name}{#3}%
2141 \pgfkeyssetvalue{#1'*/option@name}{#3}%
2142 \pgfkeyssetvalue{#1':/option@name}{#3}%
2143 }
2144 \def\forest@declarebooleanregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2145 \pgfkeysalso{%
2146 #1/.code={%
2147 \ifstrequal{##1}{1}{%
2148 \forestset{#3}{1}%
2149 }{%
2150 \ifstrequal{##1}{0}{%
2151 \forestset{#3}{0}%
2152 }{%
2153 \pgfmathifthenelse{##1}{1}{0}%
2154 \forestlet{#3}\pgfmathresult
2155 }%
2156 }%
2157 },
2158 #1/.default=1,
2159 #2/not #3/.code={\forestset{#3}{0}},
2160 #2/if #3/.code 2 args={%
2161 \forestget{#3}\forest@temp@option@value
2162 \ifnum\forest@temp@option@value=1
2163 \pgfkeysalso{##1}%
2164 \else
2165 \pgfkeysalso{##2}%
2166 \fi
2167 },
2168 }%
2169 \pgfkeyssetvalue{#1/option@name}{#3}%
2170 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@count{##1}{#3}}%
2171 }
2172 \forestset{
2173 declare toks register/.code={%
2174 \forest@declareregisterhandler\forest@declaretoksregister@handler{#1}%
2175 },
2176 declare autowrapped toks register/.code={%
2177 \forest@declareregisterhandler\forest@declareautowrappedtoksregister@handler{#1}%

```

```

2178 },
2179 declare keylist register/.code={%
2180   \forest@declareregisterhandler\forest@declarekeylistregister@handler{#1}%
2181 },
2182 declare dimen register/.code={%
2183   \forest@declareregisterhandler\forest@declaredimenregister@handler{#1}%
2184 },
2185 declare count register/.code={%
2186   \forest@declareregisterhandler\forest@declarecountregister@handler{#1}%
2187 },
2188 declare boolean register/.code={%
2189   \forest@declareregisterhandler\forest@declarebooleanregister@handler{#1}%
2190 },
2191 }

```

Declare some temporary registers.

```

2192 \forestset{
2193   declare toks register=temptoksa,temptoksa={},
2194   declare toks register=temptoksb,temptoksb={},
2195   declare toks register=temptoksc,temptoksc={},
2196   declare toks register=temptoksd,temptoksd={},
2197   declare keylist register=tempkeylista,tempkeylista'={},
2198   declare keylist register=tempkeylistb,tempkeylistb'={},
2199   declare keylist register=tempkeylistc,tempkeylistc'={},
2200   declare keylist register=tempkeylistd,tempkeylistd'={},
2201   declare dimen register=tempdima,tempdima'={0pt},
2202   declare dimen register=tempdimb,tempdimb'={0pt},
2203   declare dimen register=tempdimc,tempdimc'={0pt},
2204   declare dimen register=tempdimd,tempdimd'={0pt},
2205   declare dimen register=tempdimx,tempdimx'={0pt},
2206   declare dimen register=tempdimy,tempdimy'={0pt},
2207   declare dimen register=tempdiml,tempdiml'={0pt},
2208   declare dimen register=tempdims,tempdims'={0pt},
2209   declare count register=tempcounta,tempcounta'={0},
2210   declare count register=tempcountb,tempcountb'={0},
2211   declare count register=tempcountc,tempcountc'={0},
2212   declare count register=tempcountd,tempcountd'={0},
2213   declare boolean register=tempboola,tempboola={0},
2214   declare boolean register=tempboolb,tempboolb={0},
2215   declare boolean register=tempboolc,tempboolc={0},
2216   declare boolean register=tempboold,tempboold={0},
2217 }

```

5.3.3 Declaring options

```

2218 \def\forest@node@Nametoid#1{% #1 = name
2219   \csname forest@id@of@#1\endcsname
2220 }
2221 \def\forest@node@ifnamedefined#1#2#3{% #1 = name, #2=true,#3=false
2222   \ifcsvoid{forest@id@of@#1}{#3}{#2}%
2223 }
2224 \def\forest@node@setname#1{%
2225   \def\forest@temp@setname{y}%
2226   \def\forest@temp@silent{n}%
2227   \def\forest@temp@propagating{n}%
2228   \forest@node@setnameoralias{#1}%
2229 }
2230 \def\forest@node@setname@silent#1{%
2231   \def\forest@temp@setname{y}%
2232   \def\forest@temp@silent{y}%
2233   \def\forest@temp@propagating{n}%

```

```

2234 \forest@node@setnameoralias{#1}%
2235 }
2236 \def\forest@node@setalias#1{%
2237 \def\forest@temp@setname{n}%
2238 \def\forest@temp@silent{n}%
2239 \def\forest@temp@propagating{n}%
2240 \forest@node@setnameoralias{#1}%
2241 }
2242 \def\forest@node@setalias@silent#1{%
2243 \def\forest@temp@setname{n}%
2244 \def\forest@temp@silent{y}%
2245 \def\forest@temp@propagating{n}%
2246 \forest@node@setnameoralias{#1}%
2247 }
2248 \def\forest@node@setnameoralias#1{%
2249 \ifstrempy{#1}{%
2250 \forest@node@setnameoralias{node@\forest@cn}%
2251 }{%
2252 \forest@node@ifnamedefined{#1}{%
2253 \if y\forest@temp@propagating
2254 % this will find a unique name, eventually:
2255 \@escapeif{\forest@node@setnameoralias{#1@\forest@cn}}%
2256 \else\@escapeif{%
2257 \if y\forest@temp@setname
2258 \@escapeif{\forest@node@setnameoralias@nameclash{#1}}%
2259 \else\@escapeif{% setting an alias: clashing with alias is not a problem
2260 \forest@get{\forest@node@Nametoid{#1}}{name}\forest@temp
2261 \expandafter\ifstrequal\expandafter{\forest@temp}{#1}{%
2262 \forest@node@setnameoralias@nameclash{#1}%
2263 }{%
2264 \forest@node@setnameoralias@do{#1}%
2265 }%
2266 }\fi
2267 }\fi
2268 }{%
2269 \forest@node@setnameoralias@do{#1}%
2270 }%
2271 }%
2272 }
2273 \def\forest@node@setnameoralias@nameclash#1{%
2274 \if y\forest@temp@silent
2275 \forest@fornode{\forest@node@Nametoid{#1}}{%
2276 \def\forest@temp@propagating{y}%
2277 \forest@node@setnameoralias{}}%
2278 }%
2279 \forest@node@setnameoralias@do{#1}%
2280 \else
2281 \PackageError{forest}{Node name "#1" is already used}{}%
2282 \fi
2283 }
2284 \def\forest@node@setnameoralias@do#1{%
2285 \if y\forest@temp@setname
2286 \csdef{forest@id@of@\forest@ve{name}}{}}%
2287 \forest@set{name}{#1}%
2288 \fi
2289 \csdef{forest@id@of@#1}{\forest@cn}%
2290 }
2291 \forestset{
2292 TeX/.code={#1},
2293 TeX'/.code={\appto\forest@externalize@loadimages{#1}#1},
2294 TeX''/.code={\appto\forest@externalize@loadimages{#1}},

```

```

2295 options/.code={\forestset{#1}},
2296 typeout/.style={TeX={\typeout{#1}}},
2297 declare toks={name}{},
2298 name/.code={% override the default setter
2299   \forest@fornode{\forest@setter@node}{\forest@node@setname{#1}}%
2300 },
2301 name/.default={},
2302 name'/.code={% override the default setter
2303   \forest@fornode{\forest@setter@node}{\forest@node@setname@silent{#1}}%
2304 },
2305 name'/.default={},
2306 alias/.code={\forest@fornode{\forest@setter@node}{\forest@node@setalias{#1}}},
2307 alias'/.code={\forest@fornode{\forest@setter@node}{\forest@node@setalias@silent{#1}}},
2308 begin draw/.code={\begin{tikzpicture}},
2309 end draw/.code={\end{tikzpicture}},
2310 declare keylist register=default preamble,
2311 default preamble'={},
2312 declare keylist register=preamble,
2313 preamble'={},
2314 declare autowrapped toks={content}{},
2315 % #1 = which option to split, #2 = separator (one char!), #3 = receiving options
2316 %%% begin listing region: split_option
2317 split option/.style n args=3{split/.process args={0}{#1}{#2}{#3}}
2318 %%% end listing region: split_option
2319 ,
2320 split register/.style n args=3{% #1 = which register to split, #2 = separator (one char!), #3 = receiving o
2321   split/.process args={R}{#1}{#2}{#3},
2322 },
2323 TeX={%
2324   \def\forest@split@sourcevalues{%
2325     \def\forest@split@sourcevalue{%
2326       \def\forest@split@receivingoptions{%
2327         \def\forest@split@receivingoption{%
2328         },
2329 split/.code n args=3{% #1 = string to split, #2 = separator (one char!), #3 = receiving options
2330   \forest@saveandrestoremacro\forest@split@sourcevalues{%
2331     \forest@saveandrestoremacro\forest@split@sourcevalue{%
2332       \forest@saveandrestoremacro\forest@split@receivingoptions{%
2333         \forest@saveandrestoremacro\forest@split@receivingoption{%
2334           \def\forest@split@sourcevalues{#1#2}%
2335           \edef\forest@split@receivingoptions{#3,%
2336           \def\forest@split@receivingoption{%
2337             \safeloop
2338               \expandafter\forest@split\expandafter{\forest@split@sourcevalues}{#2}\forest@split@sourcevalue\
2339             \ifdefempty\forest@split@receivingoptions{}%
2340               \expandafter\forest@split\expandafter{\forest@split@receivingoptions}{,}\forest@temp\forest@s
2341             \ifdefempty\forest@temp{ }\let\forest@split@receivingoption\forest@temp\def\forest@temp{ }%
2342             }%
2343             \edef\forest@marshal{%
2344               \noexpand\pgfkeysalso{\forest@split@receivingoption={\expandonce{\forest@split@sourcevalue}}}%
2345             }\forest@marshal
2346             \ifdefempty\forest@split@sourcevalues{\forest@tempfalse}{\forest@temptrue}%
2347             \ifforest@temp
2348               \saferepeat
2349             }}}}%
2350 },
2351 declare count={grow}{270},
2352 TeX={% a hack for grow-reversed connection, and compass-based grow specification
2353   \forest@copycommandkey{/forest/grow}{/forest/grow@@}%
2354   %\pgfkeysgetvalue{/forest/grow/.@cmd}\forest@temp
2355   %\pgfkeyslet{/forest/grow@@/.@cmd}\forest@temp

```

```

2356 },
2357 grow/.style={grow@={#1},reversed=0},
2358 grow'/.style={grow@={#1},reversed=1},
2359 grow''/.style={grow@={#1}},
2360 grow@/.is choice,
2361 grow@/east/.style={/forest/grow@@=0},
2362 grow@/north east/.style={/forest/grow@@=45},
2363 grow@/north/.style={/forest/grow@@=90},
2364 grow@/north west/.style={/forest/grow@@=135},
2365 grow@/west/.style={/forest/grow@@=180},
2366 grow@/south west/.style={/forest/grow@@=225},
2367 grow@/south/.style={/forest/grow@@=270},
2368 grow@/south east/.style={/forest/grow@@=315},
2369 grow@/.unknown/.code={\let\forest@temp@grow\pgfkeyscurrentname
2370 \pgfkeysalso{/forest/grow@@/.expand once=\forest@temp@grow}},
2371 declare boolean={reversed}{0},
2372 declare toks={parent anchor}{},
2373 declare toks={child anchor}{},
2374 declare toks={anchor}{base},
2375 Autoforward={anchor}{
2376     node options-=anchor,
2377     node options+= {anchor={##1}}
2378 },
2379 anchor'/.style={anchor@no@compass=true,anchor=#1},
2380 anchor+/.style={anchor@no@compass=true,anchor+=#1},
2381 anchor-/.style={anchor@no@compass=true,anchor-=#1},
2382 anchor*/.style={anchor@no@compass=true,anchor*=#1},
2383 anchor:/.style={anchor@no@compass=true,anchor:=#1},
2384 anchor'+/.style={anchor@no@compass=true,anchor'+=#1},
2385 anchor'-/.style={anchor@no@compass=true,anchor'=-#1},
2386 anchor'*/.style={anchor@no@compass=true,anchor'*=#1},
2387 anchor':/.style={anchor@no@compass=true,anchor':=#1},
2388 % /tikz/forest anchor/.style={
2389 %     /forest/TeX={\forestanchortotikzanchor{#1}\forest@temp@anchor},
2390 %     anchor/.expand once=\forest@temp@anchor
2391 % },
2392 declare toks={calign}{midpoint},
2393 TeX={%
2394     \forest@copycommandkey{/forest/calign}{/forest/calign'}%
2395 },
2396 calign/.is choice,
2397 calign/child/.style={calign'=child},
2398 calign/first/.style={calign'=child,calign primary child=1},
2399 calign/last/.style={calign'=child,calign primary child=-1},
2400 calign with current/.style={for parent/.wrap pgfmath arg={calign=child,calign primary child=##1}{n}},
2401 calign with current edge/.style={for parent/.wrap pgfmath arg={calign=child edge,calign primary child=##1}{n}},
2402 calign/child edge/.style={calign'=child edge},
2403 calign/midpoint/.style={calign'=midpoint},
2404 calign/center/.style={calign'=midpoint,calign primary child=1,calign secondary child=-1},
2405 calign/edge midpoint/.style={calign'=edge midpoint},
2406 calign/fixed angles/.style={calign'=fixed angles},
2407 calign/fixed edge angles/.style={calign'=fixed edge angles},
2408 calign/.unknown/.code={\PackageError{forest}{unknown calign '\pgfkeyscurrentname'}{}}},
2409 declare count={calign primary child}{1},
2410 declare count={calign secondary child}{-1},
2411 declare count={calign primary angle}{-35},
2412 declare count={calign secondary angle}{35},
2413 calign child/.style={calign primary child={#1}},
2414 calign angle/.style={calign primary angle={-#1},calign secondary angle={#1}},
2415 declare toks={tier}{},
2416 declare toks={fit}{tight},

```

```

2417 declare boolean={ignore}{0},
2418 declare boolean={ignore edge}{0},
2419 no edge/.style={edge'={},ignore edge},
2420 declare keylist={edge}{draw},
2421 declare toks={edge path}{%
2422   \noexpand\path[\forestoption{edge}]%
2423   (\forestOve{\forestove{@parent}}{name}.parent anchor)--(\forestove{name}.child anchor)
2424   % =
2425   % (!u.parent anchor)--(.child anchor)\forestoption{edge label};
2426   \forestoption{edge label}};%
2427 },
2428 edge path'/.style={
2429   edge path={%
2430     \noexpand\path[\forestoption{edge}]%
2431     #1%
2432     \forestoption{edge label};
2433   }
2434 },
2435 declare toks={edge label}{},
2436 declare boolean={phantom}{0},
2437 baseline/.style={alias={forest@baseline@node}},
2438 declare readonly count={n},
2439 declare readonly count={n'},
2440 declare readonly count={n children},
2441 declare readonly count={level},
2442 declare dimen=x{0pt},
2443 declare dimen=y{0pt},
2444 declare dimen={s}{0pt},
2445 declare dimen={l}{6ex}, % just in case: should be set by the calibration
2446 declare dimen={s sep}{0.6666em},
2447 declare dimen={l sep}{1ex}, % just in case: calibration!
2448 declare keylist={node options}{anchor=base},
2449 declare toks={tikz}{},
2450 afterthought/.style={tikz+=#{1}},
2451 label/.style={tikz={\path[late options={%
2452   name=\forestoption{name},label=#{1}}];}},
2453 pin/.style={tikz={\path[late options={%
2454   name=\forestoption{name},pin=#{1}}];}},
2455 declare toks={content format}{\forestoption{content}},
2456 plain content/.style={content format={\forestoption{content}}},
2457 math content/.style={content format={\noexpand\ensuremath{\forestoption{content}}}},
2458 declare toks={node format}{%
2459   \noexpand\node
2460   (\forestoption{name})%
2461   [\forestoption{node options}]%
2462   {\foresteoption{content format}};%
2463 },
2464 node format'/.style={
2465   node format={\noexpand\node(\forestoption{name})#1;}
2466 },
2467 tabular@environment/.style={content format={%
2468   \noexpand\begin{tabular}[\forestoption{base}]{\forestoption{align}}%
2469   \forestoption{content}%
2470   \noexpand\end{tabular}%
2471 }},
2472 declare toks={align}{},
2473 TeX={%
2474   \forest@copycommandkey{/forest/align}{/forest/align'}%
2475   %\pgfkeysgetvalue{/forest/align/.@cmd}\forest@temp
2476   %\pgfkeyslet{/forest/align'/.@cmd}\forest@temp
2477 },

```

```

2478 align/.is choice,
2479 align/.unknown/.code={%
2480   \edef\forest@marshal{%
2481     \noexpand\pgfkeysalso{%
2482       align'={\pgfkeyscurrentname},%
2483       tabular@environment
2484     }%
2485   }\forest@marshal
2486 },
2487 align/center/.style={align'={@{}c@{}}},tabular@environment},
2488 align/left/.style={align'={@{}l@{}}},tabular@environment},
2489 align/right/.style={align'={@{}r@{}}},tabular@environment},
2490 declare toks={base}{t},
2491 TeX={%
2492   \forest@copycommandkey{/forest/base}{/forest/base'}%
2493   %\pgfkeysgetvalue{/forest/base/.@cmd}\forest@temp
2494   %\pgfkeyslet{/forest/base'/.@cmd}\forest@temp
2495 },
2496 base/.is choice,
2497 base/top/.style={base'=t},
2498 base/bottom/.style={base'=b},
2499 base/.unknown/.style={base'/.expand once=\pgfkeyscurrentname},
2500 unknown to/.store in=\forest@unknownto,
2501 unknown to=node options,
2502 unknown key error/.code={\PackageError{forest}{Unknown keyval: \detokenize{#1}}{}},
2503 content to/.store in=\forest@contentto,
2504 content to=content,
2505 .unknown/.code={%
2506   \expandafter\pgfutil@in@\expandafter.\expandafter{\pgfkeyscurrentname}%
2507   \ifpgfutil@in@
2508     \expandafter\forest@relatednode@option@setter\pgfkeyscurrentname=#1\forest@END
2509   \else
2510     \edef\forest@marshal{%
2511       \noexpand\pgfkeysalso{\forest@unknownto={\pgfkeyscurrentname=\unexpanded{#1}}}%
2512     }\forest@marshal
2513   \fi
2514 },
2515 get node boundary/.code={%
2516   \forestoget{@boundary}\forest@node@boundary
2517   \def#1{%
2518     \forest@extendpath#1\forest@node@boundary{\pgfpoint{\forestove{x}}{\forestove{y}}}%
2519   },
2520 % get min l tree boundary/.code={%
2521 %   \forest@get@tree@boundary{negative}{\the\numexpr\forestove{grow}-90\relax}#1},
2522 % get max l tree boundary/.code={%
2523 %   \forest@get@tree@boundary{positive}{\the\numexpr\forestove{grow}-90\relax}#1},
2524 get min s tree boundary/.code={%
2525   \forest@get@tree@boundary{negative}{\forestove{grow}}#1},
2526 get max s tree boundary/.code={%
2527   \forest@get@tree@boundary{positive}{\forestove{grow}}#1},
2528 use as bounding box/.style={%
2529   before drawing tree={
2530     tikz+/.expanded={%
2531       \noexpand\pgfresetboundingbox
2532       \noexpand\useasboundingbox
2533       ($(.anchor)+(\forestoption{min x},\forestoption{min y}))$)
2534       rectangle
2535       ($(.anchor)+(\forestoption{max x},\forestoption{max y}))$)
2536     };
2537   }
2538 }

```

```

2539 },
2540 use as bounding box'/.style={%
2541     before drawing tree={
2542         tikz+/.expanded={%
2543             \noexpand\pgfresetboundingbox
2544             \noexpand\useasboundingbox
2545             ($(.anchor)+(\forestoption{min x}+\pgfkeysvalueof{/pgf/outer xsep}/2+\pgfkeysvalueof{/pgf/inner xsep})
2546             rectangle
2547             ($(.anchor)+(\forestoption{max x}-\pgfkeysvalueof{/pgf/outer xsep}/2-\pgfkeysvalueof{/pgf/inner xsep})
2548             ;
2549         }
2550     }
2551 },
2552 }%
2553 \def\forest@iftikzkey#1#2#3{% #1 = key name, #2 = true code, #3 = false code
2554     \forest@temptrue
2555     \pgfkeysifdefined{/tikz/\pgfkeyscurrentname}{}%
2556     \pgfkeysifdefined{/tikz/\pgfkeyscurrentname/.@cmd}{}%
2557     \pgfkeysifdefined{/pgf/\pgfkeyscurrentname}{}%
2558     \pgfkeysifdefined{/pgf/\pgfkeyscurrentname/.@cmd}{}%
2559     \forest@tempfalse
2560     }}}}
2561 \ifforest@temp\@escapeif{#2}\else\@escapeif{#3}\fi
2562 }
2563 \def\forest@ifoptionortikzkey#1#2#3{% #1 = key name, #2 = true code, #3 = false code
2564     \forest@temptrue
2565     \pgfkeysifdefined{/forest/\pgfkeyscurrentname}{}%
2566     \pgfkeysifdefined{/forest/\pgfkeyscurrentname/.@cmd}{}%
2567     \forest@iftikzkey{#1}{}%
2568     }}}}
2569 \ifforest@temp\@escapeif{#2}\else\@escapeif{#3}\fi
2570 }
2571 \def\forest@get@tree@boundary#1#2#3{% #1=pos/neg, #2=grow, #3=receiving cs
2572     \def#3{}%
2573     \forest@node@getedge{#1}{#2}\forest@temp@boundary
2574     \forest@extendpath#3\forest@temp@boundary{\pgfpoint{\forestove{x}}{\forestove{y}}}%
2575 }
2576 \def\forest@setter@node{\forest@cn}%
2577 \def\forest@relatednode@option@compat@ignoreinvalidsteps#1{#1}
2578 \def\forest@relatednode@option@setter#1.#2=#3\forest@END{%
2579     \forest@forthis{%
2580         \forest@relatednode@option@compat@ignoreinvalidsteps{%
2581             \forest@nameandgo{#1}%
2582             \let\forest@setter@node\forest@cn
2583         }%
2584     }%
2585     \ifnum\forest@setter@node=0
2586     \else
2587         \forestset{#2={#3}}%
2588     \fi
2589     \def\forest@setter@node{\forest@cn}%
2590 }%
2591 \def\forest@split#1#2#3#4{% #1=list (assuming that the list is nonempty and finishes with the separator), #2
2592     \def\forest@split@@##1#2##2\forest@split@@##3##4{\def##3{##1}\def##4{##2}}%
2593     \forest@split@@##1\forest@split@@##3{##4}}

```

5.3.4 Option propagation

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

```

2594 \forestset{
2595     for tree'/.style 2 args={#1,for children={for tree'={#1}{#2}},#2},

```



```

2596 if/.code n args={3}{%
2597   \pgfmathparse{#1}%
2598   \ifnum\pgfmathresult=0
2599     \@escapeif{\pgfkeysalso{#3}}%
2600   \else
2601     \@escapeif{\pgfkeysalso{#2}}%
2602   \fi
2603 },
2604 if nodewalk valid/.code n args={3}{%
2605   \edef\forest@marshal{%
2606     \unexpanded{\forest@forthis{%
2607       \forest@nodewalk{%
2608         on invalid={fake}{#1},
2609         TeX={\global\let\forest@global@temp\forest@cn}
2610       }}%
2611     }%
2612     \def\forest@cn{\forest@cn}\unexpanded{%
2613       \ifnum\forest@global@temp=0
2614         \@escapeif{\pgfkeysalso{#3}}%
2615       \else
2616         \@escapeif{\pgfkeysalso{#2}}%
2617       \fi}%
2618   }\forest@marshal
2619 },
2620 if nodewalk empty/.code n args={3}{%
2621   \forest@forthis{%
2622     \forest@nodewalk{%
2623       on invalid={fake}{#1},
2624       TeX={\global\let\forest@global@temp\forest@nodewalk@cn},
2625     }}%
2626   }%
2627   \ifnum\forest@global@temp=0
2628     \@escapeif{\pgfkeysalso{#2}}%
2629   \else
2630     \@escapeif{\pgfkeysalso{#3}}%
2631   \fi
2632 },
2633 where/.style n args={3}{for tree={if={#1}{#2}{#3}}},
2634 where nodewalk valid/.style n args={3}{for tree={if nodewalk valid={#1}{#2}{#3}}},
2635 where nodewalk empty/.style n args={3}{for tree={if nodewalk empty={#1}{#2}{#3}}},
2636 repeat/.code 2 args={%
2637   \pgfmathtruncatemacro\forest@temp{#1}%
2638   \expandafter\forest@repeatkey\expandafter{\forest@temp}{#2}%
2639 },
2640 until/.code 2 args={%
2641   \ifstrempy{#1}{%
2642     \forest@untilkey{\ifnum\forest@cn=0\else\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
2643   }{%
2644     \forest@untilkey{\pgfmathifthenelse{#1}{"\noexpand\forestloopbreak"}{""}\pgfmathresult}{#2}%
2645   }%
2646 },
2647 while/.code 2 args={%
2648   \ifstrempy{#1}{%
2649     \forest@untilkey{\ifnum\forest@cn=0\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
2650   }{%
2651     \forest@untilkey{\pgfmathifthenelse{not(#1)}{"\noexpand\forestloopbreak"}{""}\pgfmathresult}{#2}%
2652   }%
2653 },
2654 do until/.code 2 args={%
2655   \ifstrempy{#1}{%
2656     \forest@dountilkey{\ifnum\forest@cn=0\else\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%

```

```

2657 }{%
2658   \forest@dountilkey{\pgfmathifthenelse{#1}{\noexpand\forestloopbreak}{""}\pgfmathresult}{#2}%
2659 }%
2660 },
2661 do while/.code 2 args={%
2662   \ifstrempy{#1}{%
2663     \forest@dountilkey{\ifnum\forest@cn=0\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
2664   }{%
2665     \forest@dountilkey{\pgfmathifthenelse{not(#1)}{\noexpand\forestloopbreak}{""}\pgfmathresult}{#2}%
2666   }%
2667 },
2668 until nodewalk valid/.code 2 args={%
2669   \forest@untilkey{\forest@forthis{%
2670     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\else\forestloopbreak\fi}}{}}{#2}
2671   },
2672 while nodewalk valid/.code 2 args={%
2673   \forest@untilkey{\forest@forthis{%
2674     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\forestloopbreak\fi}}{}}{#2}
2675   },
2676 do until nodewalk valid/.code 2 args={%
2677   \forest@dountilkey{\forest@forthis{%
2678     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\else\forestloopbreak\fi}}{}}{#2}
2679   },
2680 do while nodewalk valid/.code 2 args={%
2681   \forest@dountilkey{\forest@forthis{%
2682     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\forestloopbreak\fi}}{}}{#2}
2683   },
2684 until nodewalk empty/.code 2 args={%
2685   \forest@untilkey{\forest@forthis{%
2686     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\forestloopbreak\fi}}{}}
2687   },
2688 while nodewalk empty/.code 2 args={%
2689   \forest@untilkey{\forest@forthis{%
2690     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\else\forestloopbreak\fi}
2691   },
2692 do until nodewalk empty/.code 2 args={%
2693   \forest@dountilkey{\forest@forthis{%
2694     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\forestloopbreak\fi}}{}}
2695   },
2696 do while nodewalk empty/.code 2 args={%
2697   \forest@dountilkey{\forest@forthis{%
2698     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\else\forestloopbreak\fi}
2699   },
2700 break/.code={\forestloopBreak{#1}},
2701 break/.default=0,
2702 }
2703 \def\forest@repeatkey#1#2{%
2704   \safeRKloop
2705   \ifnum\safeRKloopn>#1\relax
2706     \csuse{safeRKbreak@\the\safeRKloop@depth true}%
2707   \fi
2708   \expandafter\unless\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
2709     \pgfkeysalso{#2}%
2710   \safeRKrepeat
2711 }
2712 \def\forest@untilkey#1#2{% #1 = condition, #2 = keys
2713   \safeRKloop
2714   #1%
2715   \expandafter\unless\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
2716     \pgfkeysalso{#2}%
2717   \safeRKrepeat

```

```

2718 }
2719 \def\forest@dountilkey#1#2{% #1 = condition, #2 = keys
2720   \safeRKloop
2721   \pgfkeysalso{#2}%
2722   #1%
2723   \expandafter\unless\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
2724   \safeRKrepeat
2725 }
2726 \def\forestloopbreak{%
2727   \csname safeRKbreak@\the\safeRKloop@depth true\endcsname
2728 }
2729 \def\forestloopBreak#1{%
2730   \csname safeRKbreak@\number\numexpr\the\safeRKloop@depth-#1\relax true\endcsname
2731 }
2732 \def\forestloopcount{%
2733   \csname safeRKloopn@\number\numexpr\the\safeRKloop@depth\endcsname
2734 }
2735 \def\forestloopCount#1{%
2736   \csname safeRKloopn@\number\numexpr\the\safeRKloop@depth-#1\endcsname
2737 }
2738 \pgfmathdeclarefunction{forestloopcount}{1}{%
2739   \edef\pgfmathresult{\forestloopCount{\ifstreempty{#1}{0}{#1}}}%
2740 }
2741 \forest@copycommandkey{/forest/repeat}{/forest/nodewalk/repeat}
2742 \forest@copycommandkey{/forest/while}{/forest/nodewalk/while}
2743 \forest@copycommandkey{/forest/do while}{/forest/nodewalk/do while}
2744 \forest@copycommandkey{/forest/until}{/forest/nodewalk/until}
2745 \forest@copycommandkey{/forest/do until}{/forest/nodewalk/do until}
2746 \forest@copycommandkey{/forest/if}{/forest/nodewalk/if}
2747 \forest@copycommandkey{/forest/if nodewalk valid}{/forest/nodewalk/if nodewalk valid}
2748 %

```

5.4 Aggregate functions

```

2749 \forestset{
2750   aggregate postparse/.is choice,
2751   aggregate postparse/int/.code={%
2752     \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@toint},
2753   aggregate postparse/none/.code={%
2754     \let\forest@aggregate@pgfmathpostparse\relax},
2755   aggregate postparse/print/.code={%
2756     \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@print},
2757   aggregate postparse/macro/.code={%
2758     \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@usemacro},
2759   aggregate postparse macro/.store in=\forest@aggregate@pgfmathpostparse@macro,
2760 }
2761 \def\forest@aggregate@pgfmathpostparse@print{%
2762   \pgfmathprintnumberto{\pgfmathresult}{\pgfmathresult}%
2763 }
2764 \def\forest@aggregate@pgfmathpostparse@toint{%
2765   \expandafter\forest@split\expandafter{\pgfmathresult.}{.}\pgfmathresult\forest@temp
2766 }
2767 \def\forest@aggregate@pgfmathpostparse@usemacro{%
2768   \forest@aggregate@pgfmathpostparse@macro
2769 }
2770 \let\forest@aggregate@pgfmathpostparse\relax
2771 \pgfkeys{
2772   /handlers/.aggregate/.code n args=4{%
2773     % #1 = start value
2774     % #2 = pgfmath expression for every step

```

```

2775 % #3 = pgfmath expression for after walk
2776 % #4 = nodewalk
2777 \edef\forest@marshal{%
2778   \unexpanded{\forest@aggregate{#1}{#2}{#3}{#4}}{\pgfkeyscurrentpath}%
2779 } \forest@marshal
2780 },
2781 /handlers/.sum/.style 2 args={/handlers/.aggregate={0}{(##1)+(##1)}{##1}{#2}},
2782 /handlers/.count/.style={/handlers/.aggregate={0}{1+(##1)}{##1}{#1}},
2783 /handlers/.average/.style 2 args={/handlers/.aggregate={0}{(##1)+(##1)}{##1/\forestregister{aggregate n}}{#2}},
2784 /handlers/.product/.style 2 args={/handlers/.aggregate={1}{(##1)*(##1)}{##1}{#2}},
2785 /handlers/.min/.style 2 args={/handlers/.aggregate={}{min(##1,##1)}{##1}{#2}},
2786 /handlers/.max/.style 2 args={/handlers/.aggregate={}{max(##1,##1)}{##1}{#2}},
2787 /forest/declare count register={aggregate n},
2788 }
2789
2790 \def\forest@aggregate#1#2#3#4#5{%
2791   % #5 = current path
2792   \def\forest@aggregate@result{#1}%
2793   \forest@forthis{%
2794     \forestreset{aggregate n}{0}%
2795     \forest@nodewalk{#4}{%
2796       TeX={%
2797         \forestreset{aggregate n}{\number\numexpr\forestrv{aggregate n}+1}%
2798         \def\forest@marshal##1{\pgfmathparse{#2}}%
2799         \expandafter\forest@marshal\expandafter{\forest@aggregate@result}%
2800         \let\forest@aggregate@result\pgfmathresult
2801       }%
2802     }{%
2803     }%
2804     \def\forest@marshal##1{\pgfmathparse{#3}}%
2805     \expandafter\forest@marshal\expandafter{\forest@aggregate@result}%
2806     \let\forest@aggregate@result\pgfmathresult
2807     \let\forest@temp\pgfmathpostparse\pgfmathpostparse
2808     \let\pgfmathpostparse\forest@aggregate@pgfmathpostparse
2809     \pgfmathqparse{\forest@aggregate@result pt}%
2810     \let\pgfmathpostparse\forest@temp\pgfmathpostparse
2811     \let\forest@aggregate@result\pgfmathresult
2812     \pgfkeysalso{#5/.expand once=\forest@aggregate@result}%
2813 }

```

5.4.1 pgfmath extensions

```

2814 \pgfmathdeclarefunction{strequal}{2}{%
2815   \ifstrequal{#1}{#2}{\def\pgfmathresult{1}}{\def\pgfmathresult{0}}%
2816 }
2817 \pgfmathdeclarefunction{instr}{2}{%
2818   \pgfutil@in@{#1}{#2}%
2819   \ifpgfutil@in@\def\pgfmathresult{1}\else\def\pgfmathresult{0}\fi
2820 }
2821 \pgfmathdeclarefunction{strcat}{...}{%
2822   \edef\pgfmathresult{\forest@strip@braces{#1}}%
2823 }
2824 \pgfmathdeclarefunction{min_s}{2}{% #1 = node, #2 = context node (for growth rotation)
2825   \forest@forthis{%
2826     \forest@nameandgo{#1}%
2827     \forest@compute@minmax@ls{#2}%
2828     \pgfmathsetmacro\pgfmathresult{\forest@min@s}%
2829   }%
2830 }
2831 \pgfmathdeclarefunction{min_l}{2}{% #1 = node, #2 = context node (for growth rotation)
2832   \forest@forthis{%
2833     \forest@nameandgo{#1}%

```

```

2834 \forest@compute@minmax@ls{#2}%
2835 \pgfmathsetmacro\pgfmathresult{\forestove{min@1}}%
2836 }%
2837 }
2838 \pgfmathdeclarefunction{max_s}{2}{% #1 = node, #2 = context node (for growth rotation)
2839 \forest@forthis{%
2840 \forest@nameandgo{#1}%
2841 \forest@compute@minmax@ls{#2}%
2842 \pgfmathsetmacro\pgfmathresult{\forestove{max@s}}%
2843 }%
2844 }
2845 \pgfmathdeclarefunction{max_l}{2}{% #1 = node, #2 = context node (for growth rotation)
2846 \forest@forthis{%
2847 \forest@nameandgo{#1}%
2848 \forest@compute@minmax@ls{#2}%
2849 \pgfmathsetmacro\pgfmathresult{\forestove{max@l}}%
2850 }%
2851 }
2852 \def\forest@compute@minmax@ls#1{% #1 = nodewalk; in the context of which node?
2853 {%
2854 \pgftransformreset
2855 \forest@forthis{%
2856 \forest@nameandgo{#1}%
2857 \pgftransformrotate{-\forestove{grow}}}%
2858 }%
2859 \forestoget{min x}\forest@temp@minx
2860 \forestoget{min y}\forest@temp@miny
2861 \forestoget{max x}\forest@temp@maxx
2862 \forestoget{max y}\forest@temp@maxy
2863 \pgfpointransformed{\pgfqpoint{\forest@temp@minx}{\forest@temp@miny}}%
2864 \forestoeset{min@1}{\the\pgf@x}%
2865 \forestoeset{min@s}{\the\pgf@y}%
2866 \forestoeset{max@l}{\the\pgf@x}%
2867 \forestoeset{max@s}{\the\pgf@y}%
2868 \pgfpointransformed{\pgfqpoint{\forest@temp@minx}{\forest@temp@maxy}}%
2869 \ifdim\pgf@x<\forestove{min@1}\relax\forestoeset{min@1}{\the\pgf@x}\fi
2870 \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
2871 \ifdim\pgf@x>\forestove{max@l}\relax\forestoeset{max@l}{\the\pgf@x}\fi
2872 \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
2873 \pgfpointransformed{\pgfqpoint{\forest@temp@maxx}{\forest@temp@miny}}%
2874 \ifdim\pgf@x<\forestove{min@1}\relax\forestoeset{min@1}{\the\pgf@x}\fi
2875 \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
2876 \ifdim\pgf@x>\forestove{max@l}\relax\forestoeset{max@l}{\the\pgf@x}\fi
2877 \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
2878 \pgfpointransformed{\pgfqpoint{\forest@temp@maxx}{\forest@temp@maxy}}%
2879 \ifdim\pgf@x<\forestove{min@1}\relax\forestoeset{min@1}{\the\pgf@x}\fi
2880 \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
2881 \ifdim\pgf@x>\forestove{max@l}\relax\forestoeset{max@l}{\the\pgf@x}\fi
2882 \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
2883 % smuggle out
2884 \edef\forest@marshal{%
2885 \noexpand\forestoeset{min@1}{\forestove{min@1}}%
2886 \noexpand\forestoeset{min@s}{\forestove{min@s}}%
2887 \noexpand\forestoeset{max@l}{\forestove{max@l}}%
2888 \noexpand\forestoeset{max@s}{\forestove{max@s}}%
2889 }\expandafter
2890 }\forest@marshal
2891 }
2892 \def\forest@pgfmathhelper@attribute@toks#1#2{%
2893 \forest@forthis{%
2894 \forest@nameandgo{#1}%

```

```

2895 \ifnum\forest@cn=0
2896 \def\pgfmathresult{%
2897 \else
2898 \forestoget{#2}\pgfmathresult
2899 \fi
2900 }%
2901 }
2902 \def\forest@pgfmathhelper@attribute@dimen#1#2{%
2903 \forest@forthis{%
2904 \forest@nameandgo{#1}%
2905 \ifnum\forest@cn=0
2906 \def\pgfmathresult{0}%
2907 \else
2908 \forestoget{#2}\forest@temp
2909 \pgfmathparse{+\forest@temp}%
2910 \fi
2911 }%
2912 }
2913 \def\forest@pgfmathhelper@attribute@count#1#2{%
2914 \forest@forthis{%
2915 \forest@nameandgo{#1}%
2916 \ifnum\forest@cn=0
2917 \def\pgfmathresult{0}%
2918 \else
2919 \forestoget{#2}\forest@temp
2920 \pgfmathtruncatemacro\pgfmathresult{\forest@temp}%
2921 \fi
2922 }%
2923 }
2924 \pgfmathdeclarefunction{id}{1}{%
2925 \forest@forthis{%
2926 \forest@nameandgo{#1}%
2927 \let\pgfmathresult\forest@cn
2928 }%
2929 }
2930 \forestset{%
2931 if id/.code n args={3}{%
2932 \ifnum#1=\forest@cn\relax
2933 \pgfkeysalso{#2}%
2934 \else
2935 \pgfkeysalso{#3}%
2936 \fi
2937 },
2938 where id/.style n args={3}{for tree={if id={#1}{#2}{#3}}}%
2939 }

```

5.5 Nodewalk

Setup machinery.

```

2940 \def\forest@nodewalk@n{0}
2941 \def\forest@nodewalk@historyback{0,}
2942 \def\forest@nodewalk@historyforward{0,}
2943 \def\forest@nodewalk@origin{0}
2944 \def\forest@nodewalk@config@everystep@independent@before#1{% #1 = every step keylist
2945 \forestset{every step}{#1}%
2946 }
2947 \def\forest@nodewalk@config@everystep@independent@after{%
2948 \noexpand\forestset{every step}{\forestrv{every step}}%
2949 }
2950 \def\forest@nodewalk@config@history@independent@before{%
2951 \def\forest@nodewalk@n{0}%

```

```

2952 \edef\forest@nodewalk@origin{\forest@cn}%
2953 \def\forest@nodewalk@historyback{0,}%
2954 \def\forest@nodewalk@historyforward{0,}%
2955 }
2956 \def\forest@nodewalk@config@history@independent@after{%
2957 \edef\noexpand\forest@nodewalk@n{\expandonce{\forest@nodewalk@n}}%
2958 \edef\noexpand\forest@nodewalk@origin{\expandonce{\forest@nodewalk@origin}}%
2959 \edef\noexpand\forest@nodewalk@historyback{\expandonce{\forest@nodewalk@historyback}}%
2960 \edef\noexpand\forest@nodewalk@historyforward{\expandonce{\forest@nodewalk@historyforward}}%
2961 }
2962 \def\forest@nodewalk@config@everystep@shared@before#1{% #1 = every step keylist
2963 \def\forest@nodewalk@config@everystep@shared@after{}
2964 \def\forest@nodewalk@config@history@shared@before{}
2965 \def\forest@nodewalk@config@history@shared@after{}
2966 \def\forest@nodewalk@config@everystep@inherited@before#1{% #1 = every step keylist
2967 \let\forest@nodewalk@config@everystep@inherited@after\forest@nodewalk@config@everystep@independent@after
2968 \def\forest@nodewalk@config@history@inherited@before{}
2969 \let\forest@nodewalk@config@history@inherited@after\forest@nodewalk@config@history@independent@after
2970 \def\forest@nodewalk#1#2{% #1 = nodewalk, #2 = every step keylist
2971 \def\forest@nodewalk@config@everystep@method{independent}%
2972 \def\forest@nodewalk@config@history@method{independent}%
2973 \def\forest@nodewalk@config@oninvalid{inherited}%
2974 \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
2975 \forest@Nodewalk{#1}{#2}%
2976 }%
2977 }
2978 \def\forest@Nodewalk#1#2{% #1 = nodewalk, #2 = every step keylist
2979 \edef\forest@marshal{%
2980 \noexpand\pgfqkeys{/forest/nodewalk}{\unexpanded{#1}}%
2981 \csname forest@nodewalk@config@everystep@forest@nodewalk@config@everystep@method @after\endcsname
2982 \csname forest@nodewalk@config@history@forest@nodewalk@config@history@method @after\endcsname
2983 }%
2984 \csname forest@nodewalk@config@everystep@forest@nodewalk@config@everystep@method @before\endcsname{#2}%
2985 \csname forest@nodewalk@config@history@forest@nodewalk@config@history@method @before\endcsname
2986 \forest@nodewalk@fakefalse
2987 \forest@marshal
2988 }
2989 \pgfmathdeclarefunction{valid}{1}{%
2990 \forest@forthis{%
2991 \forest@nameandgo{#1}%
2992 \edef\pgfmathresult{\ifnum\forest@cn=0 0\else 1\fi}%
2993 }%
2994 }
2995 \pgfmathdeclarefunction{invalid}{1}{%
2996 \forest@forthis{%
2997 \forest@nameandgo{#1}%
2998 \edef\pgfmathresult{\ifnum\forest@cn=0 1\else 0\fi}%
2999 }%
3000 }
3001 \newif\ifforest@nodewalk@fake
3002 \def\forest@nodewalk@oninvalid{error}
3003 \def\forest@nodewalk@makestep{%
3004 \ifnum\forest@cn=0
3005 \csname forest@nodewalk@makestep@oninvalid@forest@nodewalk@config@oninvalid\endcsname
3006 \else
3007 \forest@nodewalk@makestep@
3008 \fi
3009 }
3010 \def\forest@nodewalk@makestep@oninvalid@step{\forest@nodewalk@makestep@}
3011 \def\forest@nodewalk@makestep@oninvalid@error{\PackageError{forest}{nodewalk stepped to the invalid node; sta
3012 \let\forest@nodewalk@makestep@oninvalid@fake\relax

```

```

3013 \def\forest@nodewalk@makestep@oninvalid@compatfake{%
3014   \forest@deprecated{last step in stack "\forest@nodewalk@currentstepname", which stepped on an invalid node;}
3015 }%
3016 \def\forest@nodewalk@makestep@oninvalid@inherited{\csname forest@nodewalk@makestep@oninvalid@\forest@nodewalk
3017 \def\forest@nodewalk@makestep@{%
3018   \ifforest@nodewalk@fake
3019   \else
3020     \edef\forest@nodewalk@n{\number\numexpr\forest@nodewalk@n+1}%
3021     \epreto\forest@nodewalk@historyback{\forest@cn,}%
3022     \def\forest@nodewalk@historyforward{0,}%
3023     \ifforestdebug\forest@nodewalk@makestep@debug\fi
3024     \forest@process@keylist@register{every step}%
3025   \fi
3026 }
3027 \def\forest@nodewalk@makestep@debug{%
3028   \edef\forest@marshal{%
3029     \noexpand\typeout{\ifforest@nodewalk@fake fake \fi "\forest@nodewalk@currentstepname" step to node id=\fo
3030   }\forest@marshal
3031 }%
3032 \forestset{
3033   debug/.is if=forestdebug,
3034 }
3035 \def\forest@handlers@savecurrentpath{%
3036   \edef\pgfkeyscurrentkey{\pgfkeyscurrentpath}%
3037   \let\forest@currentkey\pgfkeyscurrentkey
3038   \pgfkeys@split@path
3039   \edef\forest@currentpath{\pgfkeyscurrentpath}%
3040   \let\forest@currentname\pgfkeyscurrentname
3041 }
3042 \pgfkeys{/handlers/save current path/.code={\forest@handlers@savecurrentpath}}
3043 \newif\ifforest@nodewalkstephandler@style
3044 \newif\ifforest@nodewalkstephandler@autostep
3045 \newif\ifforest@nodewalkstephandler@stripfakesteps
3046 \newif\ifforest@nodewalkstephandler@muststartatvalidnode
3047 \newif\ifforest@nodewalkstephandler@makefor
3048 \let\forest@nodewalkstephandler@styletrueorfalse\forest@nodewalkstephandler@stylefalse
3049 \def\forest@nodewalk@currentstepname{}
3050 \forestset{
3051   /forest/define@step/style/.is if=forest@nodewalkstephandler@style,
3052   /forest/define@step/autostep/.is if=forest@nodewalkstephandler@autostep,
3053   % the following is useful because some macros use grouping (by \forest@forthis or similar) and therefore, a
3054   /forest/define@step/strip fake steps/.is if=forest@nodewalkstephandler@stripfakesteps,
3055   % this can never happen with autosteps ...
3056   /forest/define@step/autostep/.append code={%
3057     \ifforest@nodewalkstephandler@autostep
3058       \forest@nodewalkstephandler@stripfakestepsfalse
3059     \fi
3060   },
3061   /forest/define@step/must start at valid node/.is if=forest@nodewalkstephandler@muststartatvalidnode,
3062   /forest/define@step/n args/.store in=\forest@nodewalkstephandler@nargs,
3063   /forest/define@step/make for/.is if=forest@nodewalkstephandler@makefor,
3064   /forest/define@step/@bare/.style={strip fake steps=false,must start at valid node=false,make for=false},
3065   define long step/.code n args=3{%
3066     \forest@nodewalkstephandler@styletrueorfalse % true for end users; but in the package, most of steps are
3067     \forest@nodewalkstephandler@autostepfalse
3068     \forest@nodewalkstephandler@stripfakestepstrue
3069     \forest@nodewalkstephandler@muststartatvalidnodetrue % most steps can only start at a valid node
3070     \forest@nodewalkstephandler@makefortrue % make for prefix?
3071     \def\forest@nodewalkstephandler@nargs{0}%
3072     \pgfqkeys{/forest/define@step}{#2}%
3073     \forest@temp@toks{#3}% handler code

```



```

3074 \ifforest@nodewalkstephandler@style
3075 \expandafter\forest@temp@toks\expandafter{%
3076 \expandafter\pgfkeysalso\expandafter{\the\forest@temp@toks}%
3077 }%
3078 \fi
3079 \ifforest@nodewalkstephandler@autostep
3080 \apptotoks\forest@temp@toks{\forest@nodewalk@makestep}%
3081 \fi
3082 \ifforest@nodewalkstephandler@stripfakesteps
3083 \expandafter\forest@temp@toks\expandafter{\expandafter\forest@nodewalk@stripfakesteps\expandafter{\the\
3084 \fi
3085 \ifforest@nodewalkstephandler@muststartatvalidnode
3086 \edef\forest@marshal{%
3087 \noexpand\forest@temp@toks{%
3088 \unexpanded{%
3089 \ifnum\forest@cn=0
3090 \csname forest@nodewalk@start@oninvalid@\forest@nodewalk@oninvalid\endcsname{#1}%
3091 \else
3092 }%
3093 \noexpand\@escapeif{\the\forest@temp@toks}%
3094 \noexpand\fi
3095 }%
3096 }\forest@marshal
3097 \fi
3098 \pretotoks\forest@temp@toks{\appto\forest@nodewalk@currentstepname{,#1}}%
3099 \expandafter\forest@temp@toks\expandafter{\expandafter\forest@saveandrestoremacro\expandafter\forest@node
3100 \ifforestdebug
3101 \epretotoks\forest@temp@toks{\noexpand\typeout{Starting step "#1" from id=\forest@cn
3102 \ifnum\forest@nodewalkstephandler@nargs>0 with args ####1\fi
3103 \ifnum\forest@nodewalkstephandler@nargs>1 ,####2\fi
3104 \ifnum\forest@nodewalkstephandler@nargs>2 ,####3\fi
3105 \ifnum\forest@nodewalkstephandler@nargs>3 ,####4\fi
3106 \ifnum\forest@nodewalkstephandler@nargs>4 ,####5\fi
3107 \ifnum\forest@nodewalkstephandler@nargs>5 ,####6\fi
3108 \ifnum\forest@nodewalkstephandler@nargs>6 ,####7\fi
3109 \ifnum\forest@nodewalkstephandler@nargs>7 ,####8\fi
3110 \ifnum\forest@nodewalkstephandler@nargs>8 ,####9\fi
3111 }}%
3112 \fi
3113 \def\forest@temp{/forest/nodewalk/#1/.code}%
3114 \ifnum\forest@nodewalkstephandler@nargs<2
3115 \eappto\forest@temp{=}%
3116 \else\ifnum\forest@nodewalkstephandler@nargs=2
3117 \eappto\forest@temp{ 2 args=}%
3118 \else
3119 \eappto\forest@temp{ n args={\forest@nodewalkstephandler@nargs}}%
3120 \fi\fi
3121 \eappto\forest@temp{{\the\forest@temp@toks}}%
3122 \expandafter\pgfkeysalso\expandafter{\forest@temp}%
3123 \ifforest@nodewalkstephandler@makefor
3124 \ifnum\forest@nodewalkstephandler@nargs=0
3125 \forestset{%
3126 for #1/.code={\forest@forstepwrapper{#1}{##1}},
3127 }%
3128 \else\ifnum\forest@nodewalkstephandler@nargs=1
3129 \forestset{%
3130 for #1/.code 2 args={\forest@forstepwrapper{#1={##1}}{##2}},
3131 }%
3132 \else
3133 \forestset{%
3134 for #1/.code n args/.expanded=%

```

```

3135         {\number\numexpr\forest@nodewalkstephandler@nargs+1}%
3136         {\noexpand\forest@forstepwrapper{#1\ifnum\forest@nodewalkstephandler@nargs>0=\fi\forest@util@narg
3137         }%
3138         \fi\fi
3139         \fi
3140     },
3141 }

```

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

```

3142 \def\forest@forstepwrapper#1#2{\forest@forthis{\forest@nodewalk{#1}{#2}}}
3143 \def\forest@util@nargs#1#2#3{% #1 = prefix (#, ##, ...), #2 = n args, #3=start; returns {#start}{#start+1}...
3144     \ifnum#2>0 {#1\number\numexpr#3+1}\fi
3145     \ifnum#2>1 {#1\number\numexpr#3+2}\fi
3146     \ifnum#2>2 {#1\number\numexpr#3+3}\fi
3147     \ifnum#2>3 {#1\number\numexpr#3+4}\fi
3148     \ifnum#2>4 {#1\number\numexpr#3+5}\fi
3149     \ifnum#2>5 {#1\number\numexpr#3+6}\fi
3150     \ifnum#2>6 {#1\number\numexpr#3+7}\fi
3151     \ifnum#2>7 {#1\number\numexpr#3+8}\fi
3152     \ifnum#2>8 {#1\number\numexpr#3+9}\fi
3153 }
3154 \def\forest@nodewalk@start@oninvalid@fake#1{}
3155 \def\forest@nodewalk@start@oninvalid@real#1{}
3156 \def\forest@nodewalk@start@oninvalid@error#1{\PackageError{forest}{nodewalk step "#1" cannot start at the inv

```

Define long-form single-step walks.

```

3157 \forestset{
3158     define long step={current}{autostep}{},
3159     define long step={next}{autostep}{\edef\forest@cn{\forestove{@next}}},
3160     define long step={previous}{autostep}{\edef\forest@cn{\forestove{@previous}}},
3161     define long step={parent}{autostep}{\edef\forest@cn{\forestove{@parent}}},
3162     define long step={first}{autostep}{\edef\forest@cn{\forestove{@first}}},
3163     define long step={last}{autostep}{\edef\forest@cn{\forestove{@last}}},
3164     define long step={sibling}{autostep}{%
3165         \edef\forest@cn{%
3166             \ifnum\forestove{@previous}=0
3167                 \forestove{@next}%
3168             \else
3169                 \forestove{@previous}%
3170             \fi
3171         }%
3172     },
3173     define long step={next node}{autostep}{\edef\forest@cn{\forest@node@linearnextid}},
3174     define long step={previous node}{autostep}{\edef\forest@cn{\forest@node@linearpreviousid}},
3175     define long step={first leaf}{autostep}{%
3176         \safeloop
3177         \edef\forest@cn{\forestove{@first}}%
3178         \unless\ifnum\forestove{@first}=0
3179             \saferepeat
3180         },
3181     define long step={first leaf'}{autostep}{%
3182         \safeloop
3183         \unless\ifnum\forestove{@first}=0
3184             \edef\forest@cn{\forestove{@first}}%
3185             \saferepeat
3186         },
3187     define long step={last leaf}{autostep}{%
3188         \safeloop
3189         \edef\forest@cn{\forestove{@last}}%
3190         \unless\ifnum\forestove{@last}=0

```

```

3191 \saferepeat
3192 },
3193 define long step={last leaf'}{autostep}{%
3194 \safeloop
3195 \unless\ifnum\forestove{@last}=0
3196 \edef\forest@cn{\forestove{@last}}%
3197 \saferepeat
3198 },
3199 define long step={next leaf}{style,strip fake steps=false}{group={do until={n_children()==0}{next node}}},
3200 define long step={previous leaf}{style,strip fake steps=false}{group={do until={n_children()==0}{previous node}}},
3201 define long step={next on tier}{autostep}{%
3202 \def\forest@temp{#1}%
3203 \ifx\forest@temp\pgfkeysnovalue@text
3204 \foresttoget{tier}\forest@nodewalk@giventier
3205 \else
3206 \def\forest@nodewalk@giventier{#1}%
3207 \fi
3208 \edef\forest@cn{\forest@node@linearnextnotdescendantid}%
3209 \safeloop
3210 \foresttoget{tier}\forest@nodewalk@tier
3211 \unless\ifx\forest@nodewalk@tier\forest@nodewalk@giventier
3212 \edef\forest@cn{\forest@node@linearnextid}%
3213 \saferepeat
3214 },
3215 define long step={previous on tier}{autostep}{%
3216 \def\forest@temp{#1}%
3217 \ifx\forest@temp\pgfkeysnovalue@text
3218 \foresttoget{tier}\forest@nodewalk@giventier
3219 \else
3220 \def\forest@nodewalk@giventier{#1}%
3221 \fi
3222 \safeloop
3223 \edef\forest@cn{\forest@node@linearpreviousid}%
3224 \foresttoget{tier}\forest@nodewalk@tier
3225 \unless\ifx\forest@nodewalk@tier\forest@nodewalk@giventier
3226 \saferepeat
3227 },
3228 %
3229 define long step={root}{autostep,must start at valid node=false}{%
3230 \edef\forest@cn{\forest@node@rootid}},
3231 define long step={root'}{autostep,must start at valid node=false}{%
3232 \forest0ifdefined{\forest@root}{\parent}{\edef\forest@cn{\forest@root}}{\edef\forest@cn{0}}%
3233 },
3234 define long step={origin}{autostep,must start at valid node=false}{\edef\forest@cn{\forest@nodewalk@origin}}
3235 %
3236 define long step={n}{autostep,n args=1}{%
3237 \pgfmathtruncatemacro\forest@temp{n}{#1}%
3238 \edef\forest@cn{\forest@node@nthchildid{\forest@temp{n}}}%
3239 },
3240 define long step={n}{autostep,make for=false,n args=1}{%
3241 % Yes, twice. ;- )
3242 % n=1 and n(ext)
3243 \def\forest@nodewalk@temp{#1}%
3244 \ifx\forest@nodewalk@temp\pgfkeysnovalue@text
3245 \edef\forest@cn{\forestove{@next}}%
3246 \else
3247 \pgfmathtruncatemacro\forest@temp{n}{#1}%
3248 \edef\forest@cn{\forest@node@nthchildid{\forest@temp{n}}}%
3249 \fi
3250 },
3251 define long step={n'}{autostep,n args=1}{%

```

```

3252 \pgfmathtruncatemacro\forest@temp@n{#1}%
3253 \edef\forest@cn{\forest@node@nbarthchildid{\forest@temp@n}}%
3254 },
3255 define long step={to tier}{autostep,n args=1}{%
3256 \def\forest@nodewalk@giventier{#1}%
3257 \safeloop
3258 \foresttoget{tier}\forest@nodewalk@tier
3259 \unless\ifx\forest@nodewalk@tier\forest@nodewalk@giventier
3260 \foresttoget{@parent}\forest@cn
3261 \safererepeat
3262 },
3263 %
3264 define long step={name}{autostep,n args=1,must start at valid node=false}{%
3265 \edef\forest@cn{%
3266 \forest@node@ifnamedefined{#1}{\forest@node@Nametoid{#1}}{0}}%
3267 }%
3268 },
3269 define long step={id}{autostep,n args=1,must start at valid node=false}{%
3270 \forest@ifdefined{#1}{@parent}{\edef\forest@cn{#1}}{\edef\forest@cn{0}}%
3271 },
3272 define long step={Nodewalk}{n args=3,@bare}{% #1 = config, #2 = nodewalk
3273 \def\forest@nodewalk@config@everystep@method{independent}%
3274 \def\forest@nodewalk@config@history@method{shared}%
3275 \def\forest@nodewalk@config@oninvalid{inherited}%
3276 \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
3277 \pgfqkeys{/forest/nodewalk@config}{#1}%
3278 \forest@Nodewalk{#2}{#3}%
3279 }%
3280 },
3281 define long step={nodewalk}{n args=2,@bare}{% #1 = nodewalk, #2 = every step
3282 \forest@nodewalk{#1}{#2}%
3283 },
3284 define long step={nodewalk'}{n args=1,@bare}{% #1 = nodewalk, #2 = every step
3285 \def\forest@nodewalk@config@everystep@method{inherited}%
3286 \def\forest@nodewalk@config@history@method{independent}%
3287 \def\forest@nodewalk@config@oninvalid{inherited}%
3288 \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
3289 \forest@Nodewalk{#1}{}%
3290 }%
3291 },
3292 % these must be defined explicitly, as prefix "for" normally introduces the every-step keylist
3293 for nodewalk/.code 2 args={%
3294 \forest@forthis{\forest@nodewalk{#1}{#2}}},
3295 for nodewalk'/.code={%
3296 \def\forest@nodewalk@config@everystep@method{inherited}%
3297 \def\forest@nodewalk@config@history@method{independent}%
3298 \def\forest@nodewalk@config@oninvalid{inherited}%
3299 \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
3300 \forest@forthis{\forest@Nodewalk{#1}{}}}%
3301 }%
3302 },
3303 for Nodewalk/.code n args=3{% #1 = config, #2 = nodewalk, #3 = every-step
3304 \def\forest@nodewalk@config@everystep@method{inherited}%
3305 \def\forest@nodewalk@config@history@method{independent}%
3306 \def\forest@nodewalk@config@oninvalid{inherited}%
3307 \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
3308 \pgfqkeys{/forest/nodewalk@config}{#1}%
3309 \forest@forthis{\forest@Nodewalk{#2}{#3}}%
3310 }%
3311 },
3312 copy command key={/forest/for nodewalk}{/forest/nodewalk/for nodewalk},

```

```

3313 copy command key={/forest/for nodewalk'}{/forest/nodewalk/for nodewalk'},
3314 copy command key={/forest/for Nodewalk}{/forest/nodewalk/for Nodewalk},
3315 declare keylist register=every step,
3316 every step'={},
3317 %%% begin nodewalk config
3318 nodewalk@config/.cd,
3319 every@step/.is choice,
3320 every@step/independent/.code={},
3321 every@step/inherited/.code={},
3322 every@step/shared/.code={},
3323 every step/.store in=\forest@nodewalk@config@everystep@method,
3324 every step/.prefix style={every@step=#1},
3325 @history/independent/.code={},
3326 @history/inherited/.code={},
3327 @history/shared/.code={},
3328 history/.store in=\forest@nodewalk@config@history@method,
3329 history/.prefix style={@history=#1},
3330 on@invalid/.is choice,
3331 on@invalid/error/.code={},
3332 on@invalid/fake/.code={},
3333 on@invalid/step/.code={},
3334 on@invalid/inherited/.code={},
3335 on invalid/.store in=\forest@nodewalk@config@oninvalid,
3336 on invalid/.prefix style={on@invalid=#1},
3337 %%% end nodewalk config
3338 }
3339 \forestset{
3340   define long step={branch}{n args=1,@bare,style}{%
3341     branch@@build/.style={branch@build={##1}{}},
3342     @branch={#1},
3343   },
3344   define long step={branch'}{n args=1,@bare,style}{%
3345     branch@@build/.style/.expanded={branch@build={###1}{\forestregister{every step},}},
3346     @branch={#1},
3347   },
3348   nodewalk/@branch/.style={
3349     TeX={\forest@temp@toks{}},
3350     split/.process args={r}{#1}{,}{branch@@build},
3351     branch@do,
3352   },
3353   nodewalk/branch@build/.code 2 args={% #1 = nodewalk for this branch, #2 = perhaps every step keylist
3354     \ifstrepty{#1}{}%
3355       \expandafter\ifstrepty\expandafter{\the\forest@temp@toks}{%
3356         \edef\forest@marshal{%
3357           \noexpand\forest@temp@toks{for nodewalk={\unexpanded{#1}}{\forestregister{every step}}}%
3358         }\forest@marshal
3359       }{%
3360         \edef\forest@marshal{%
3361           \noexpand\forest@temp@toks{for nodewalk={\unexpanded{#1}}{#2\the\forest@temp@toks}}%
3362         }\forest@marshal
3363       }%
3364     }%
3365   },
3366   nodewalk/branch@do/.code={%
3367     \edef\forest@marshal{%
3368       \noexpand\pgfkeysalso{fake={\the\forest@temp@toks}}%
3369     }\forest@marshal
3370   },
3371   define long step={group}{autostep}{\forest@go{#1}},
3372   nodewalk/fake/.code={%
3373     \forest@saveandrestoreifcs{forest@nodewalk@fake}{%

```

```

3374     \forest@nodewalk@faketrue
3375     \pgfkeysalso{#1}%
3376 }%
3377 },
3378 nodewalk/real/.code={%
3379     \forest@saveandrestoreifcs{forest@nodewalk@fake}{%
3380         \forest@nodewalk@fakefalse
3381         \pgfkeysalso{#1}%
3382     }%
3383 },
3384 define long step={filter}{n args=2,@bare,style}{% #1 = nodewalk, #2 = condition
3385     nodewalk/.expanded={\unexpanded{#1}}{if={\unexpanded{#2}}{\forestregister{every step}}{}}{}}
3386 },
3387 on@invalid/.is choice,
3388 on@invalid/error/.code={},
3389 on@invalid/fake/.code={},
3390 on@invalid/step/.code={},
3391 on invalid/.code 2 args={%
3392     \pgfkeysalso{/forest/on@invalid={#1}}%
3393     \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
3394         \def\forest@nodewalk@oninvalid{#1}%
3395         \pgfkeysalso{#2}%
3396     }%
3397 },
3398 define long step={strip fake steps}{n args=1,@bare}{%
3399     \forest@nodewalk@stripfakesteps{\pgfkeysalso{#1}}},
3400 define long step={walk back}{n args=1,@bare}{%
3401     \pgfmathtruncatemacro\forest@temp@n{#1}%
3402     \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\ifnum\forest@cn>0}{\forest@nodewalk@back@updatehistory}
3403 },
3404 nodewalk/walk back/.default=1,
3405 define long step={jump back}{n args=1,@bare}{%
3406     \pgfmathtruncatemacro\forest@temp@n{(#1)+\ifnum\forest@cn=0 0\else1\fi}%
3407     \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\forest@temp@n-1}{\forest@nodewalk@back@updatehistory}
3408 },
3409 nodewalk/jump back/.default=1,
3410 define long step={back}{n args=1,@bare}{%
3411     \pgfmathtruncatemacro\forest@temp@n{#1}%
3412     \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\ifnum\forest@cn>0}{\forest@nodewalk@back@updatehistory}
3413 },
3414 nodewalk/back/.default=1,
3415 define long step={walk forward}{n args=1,@bare}{%
3416     \pgfmathtruncatemacro\forest@temp@n{#1}%
3417     \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{0}{\forest@temp@n-1}{\forest@nodewalk@forward@updatehistory}
3418 },
3419 nodewalk/walk forward/.default=1,
3420 define long step={jump forward}{n args=1,@bare}{%
3421     \pgfmathtruncatemacro\forest@temp@n{#1}%
3422     \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{\forest@temp@n-1}{\forest@nodewalk@forward@updatehistory}
3423 },
3424 nodewalk/jump forward/.default=1,
3425 define long step={forward}{n args=1,@bare}{%
3426     \pgfmathtruncatemacro\forest@temp@n{#1}%
3427     \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{0}{\forest@temp@n-1}{\forest@nodewalk@forward@updatehistory}
3428 },
3429 nodewalk/forward/.default=1,
3430 define long step={forward}{n args=1,@bare}{%
3431     \pgfmathtruncatemacro\forest@temp@n{#1}%
3432     \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{0}{\forest@temp@n-1}{\forest@nodewalk@forward@updatehistory}
3433 },
3434 },

```

```

3435 nodewalk/forward/.default=1,
3436 define long step={last valid'}{@bare}{%
3437   \ifnum\forest@cn=0
3438     \forest@nodewalk@tolastvalid
3439     \forest@nodewalk@makestep
3440   \fi
3441 },
3442 define long step={last valid'}{@bare}{%
3443   \forest@nodewalk@tolastvalid
3444 },
3445 define long step={reverse}{n args=1,@bare,make for}{%
3446   \forest@nodewalk{#1,TeX={%
3447     \global\let\forest@global@temp\forest@nodewalk@historyback
3448     \global\let\forest@global@tempn\forest@nodewalk@n
3449   }}{%
3450     \forest@nodewalk@walklist}{\forest@global@temp}{0}{\forest@global@tempn}{\let\forest@cn\forest@nodewalk@n
3451   },
3452 define long step={walk and reverse}{n args=1,@bare,make for}{%
3453   \edef\forest@marshal{%
3454     \noexpand\pgfkeysalso{\unexpanded{#1}}%
3455     \noexpand\forest@nodewalk@walklist}{\noexpand\forest@nodewalk@historyback}{0}{\noexpand\forest@nodewalk@n-
3456   }\forest@marshal
3457 },
3458 define long step={sort}{n args=1,@bare,make for}{%
3459   \forest@nodewalk{#1,TeX={%
3460     \global\let\forest@global@temp\forest@nodewalk@historyback
3461     \global\let\forest@global@tempn\forest@nodewalk@n
3462   }}{%
3463     \forest@nodewalk@sortlist{\forest@global@temp}{\forest@global@tempn}\forest@sort@ascending
3464   },
3465 define long step={sort'}{n args=1,@bare,make for}{%
3466   \forest@nodewalk{#1,TeX={%
3467     \global\let\forest@global@temp\forest@nodewalk@historyback
3468     \global\let\forest@global@tempn\forest@nodewalk@n
3469   }}{%
3470     \forest@nodewalk@sortlist{\forest@global@temp}{\forest@global@tempn}\forest@sort@descending
3471   },
3472 define long step={walk and sort}{n args=1,@bare,make for}{% walk as given, then walk sorted
3473   \edef\forest@marshal{%
3474     \noexpand\pgfkeysalso{\unexpanded{#1}}%
3475     \noexpand\forest@nodewalk@sortlist{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-
3476   }\forest@marshal
3477 },
3478 define long step={walk and sort'}{n args=1,@bare,make for}{%
3479   \edef\forest@marshal{%
3480     \noexpand\pgfkeysalso{\unexpanded{#1}}%
3481     \noexpand\forest@nodewalk@sortlist{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-
3482   }\forest@marshal
3483 },
3484 sort by/.store in=\forest@nodesort@by,
3485 define long step={save}{n args=2,@bare,make for}{% #1 = name, #2 = nodewalk
3486   \forest@forthis{%
3487     \forest@nodewalk{#2,TeX={%
3488       \global\let\forest@global@temp\forest@nodewalk@historyback
3489       \global\let\forest@global@tempn\forest@nodewalk@n
3490     }}{%
3491     }%
3492     \forest@nodewalk@walklist}{\forest@global@temp}{0}{\forest@global@tempn}\relax
3493     \csedef\forest@nodewalk@sav#1{\forest@nodewalk@walklist@walked}%
3494   },
3495 define long step={walk and save}{n args=2,@bare,make for}{% #1 = name, #2 = nodewalk

```

```

3496 \edef\forest@marshal{%
3497 \noexpand\pgfkeysalso{\unexpanded{#2}}%
3498 \noexpand\forest@nodewalk@walklist@{\noexpand\forest@nodewalk@historyback}{0}{\noexpand\forest@nodewalk@
3499 }\forest@marshal
3500 \csedef{forest@nodewalk@samed@#1}{\forest@nodewalk@walklist@walked}%
3501 },
3502 nodewalk/save history/.code 2 args={% #1 = back, forward
3503 \csedef{forest@nodewalk@samed@#1}{\forest@nodewalk@historyback}%
3504 \csedef{forest@nodewalk@samed@#2}{\forest@nodewalk@historyforward}%
3505 },
3506 define long step={load}{n args=1,@bare,make for}{%
3507 \forest@nodewalk@walklist@{\csuse{forest@nodewalk@samed@#1}0,}{0}{-1}{\ifnum\forest@nodewalk@cn=0 \else\
3508 },
3509 if in saved nodewalk/.code n args=4{% is node #1 in nodewalk #2; yes: #3, no: #4
3510 \forest@forthis{%
3511 \forest@go{#1}%
3512 \edef\forest@marshal{%
3513 \noexpand\pgfutil@in@{,\forest@cn,}{,\csuse{forest@nodewalk@samed@#2},}%
3514 }\forest@marshal
3515 }%
3516 \ifpgfutil@in@
3517 \@escapeif{\pgfkeysalso{#3}}%
3518 \else
3519 \@escapeif{\pgfkeysalso{#4}}%
3520 \fi
3521 },
3522 where in saved nodewalk/.style n args=4{
3523 for tree={if in saved nodewalk={#1}{#2}{#3}{#4}}
3524 },
3525 nodewalk/options/.code={\forestset{#1}},
3526 nodewalk/TeX/.code={#1},
3527 nodewalk/TeX'/.code={\appto\forest@externalize@loadimages{#1}{#1},
3528 nodewalk/TeX''/.code={\appto\forest@externalize@loadimages{#1}},
3529 nodewalk/typeout/.style={TeX={\typeout{#1}}},
3530 % repeat is taken later from /forest/repeat
3531 }
3532 \def\forest@nodewalk@walklist#1#2#3#4#5{%
3533 % #1 = list of preceding, #2 = list to walk
3534 % #3 = from, #4 = to
3535 % #5 = every step code
3536 \let\forest@nodewalk@cn\forest@cn
3537 \edef\forest@marshal{%
3538 \noexpand\forest@nodewalk@walklist@{#1}{#2}{\number\numexpr#3}{\number\numexpr#4}{1}{0}{\unexpanded{#5}}%
3539 }\forest@marshal
3540 }
3541 \def\forest@nodewalk@walklist@#1#2#3#4#5#6#7{%
3542 % #1 = list of walked, #2 = list to walk
3543 % #3 = from, #4 = to
3544 % #5 = current step n, #6 = steps made
3545 % #7 = every step code
3546 \def\forest@nodewalk@walklist@walked{#1}%
3547 \def\forest@nodewalk@walklist@rest{#2}%
3548 \edef\forest@nodewalk@walklist@stepsmade{#6}%
3549 \ifnum#4<0
3550 \forest@temptrue
3551 \else
3552 \ifnum#5>#4\relax
3553 \forest@tempfalse
3554 \else
3555 \forest@temptrue
3556 \fi

```



```

3557 \fi
3558 \ifforest@temp
3559 \edef\forest@nodewalk@cn{\forest@csvlist@getfirst@{#2}}%
3560 \ifnum\forest@nodewalk@cn=0
3561 #7%
3562 \else
3563 \ifnum#5>#3\relax
3564 #7%
3565 \edef\forest@nodewalk@walklist@steps@made{\number\numexpr#6+1}%
3566 \fi
3567 \forest@csvlist@getfirstrest@{#2}\forest@nodewalk@cn\forest@nodewalk@walklist@rest
3568 \@escapeifif{%
3569 \edef\forest@marshal{%
3570 \noexpand\forest@nodewalk@walklist@
3571 {\forest@nodewalk@cn,#1}{\forest@nodewalk@walklist@rest}{#3}{#4}{\number\numexpr#5+1}{\forest@nod
3572 }\forest@marshal
3573 }%
3574 \fi
3575 \fi
3576 }
3577
3578 \def\forest@nodewalk@back@updatehistory{%
3579 \ifnum\forest@cn=0
3580 \let\forest@nodewalk@historyback\forest@nodewalk@walklist@rest
3581 \let\forest@nodewalk@historyforward\forest@nodewalk@walklist@walked
3582 \else
3583 \expandafter\forest@csvlist@getfirstrest\expandafter{\forest@nodewalk@walklist@walked}\forest@temp\forest@
3584 \edef\forest@nodewalk@historyback{\forest@temp,\forest@nodewalk@walklist@rest}%
3585 \fi
3586 }
3587 \def\forest@nodewalk@forward@updatehistory{%
3588 \let\forest@nodewalk@historyforward\forest@nodewalk@walklist@rest
3589 \let\forest@nodewalk@historyback\forest@nodewalk@walklist@walked
3590 }
3591 \def\forest@go#1{%
3592 \def\forest@nodewalk@config@everystep@method{independent}%
3593 \def\forest@nodewalk@config@history@method{inherited}%
3594 \def\forest@nodewalk@config@oninvalid{inherited}%
3595 \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
3596 \forest@nodewalk@{#1}{}%
3597 }%
3598 }
3599 \def\forest@csvlist@getfirst@#1{% assuming that the list is nonempty and finishes with a comma
3600 \forest@csvlist@getfirst@@#1\forest@csvlist@getfirst@@}
3601 \def\forest@csvlist@getfirst@@#1,#2\forest@csvlist@getfirst@@{#1}
3602 \def\forest@csvlist@getrest@#1{% assuming that the list is nonempty and finishes with a comma
3603 \forest@csvlist@getrest@@#1\forest@csvlist@getrest@@}
3604 \def\forest@csvlist@getrest@@#1,#2\forest@csvlist@getrest@@{#2}
3605 \def\forest@csvlist@getfirstrest@#1#2#3{% assuming that the list is nonempty and finishes with a comma
3606 % #1 = list, #2 = cs receiving first, #3 = cs receiving rest
3607 \forest@csvlist@getfirstrest@@#1\forest@csvlist@getfirstrest@@{#2}{#3}}
3608 \def\forest@csvlist@getfirstrest@@#1,#2\forest@csvlist@getfirstrest@@{#3#4}%
3609 \def#3{#1}%
3610 \def#4{#2}%
3611 }
3612 \def\forest@nodewalk@stripfakesteps#1{%
3613 % go to the last valid node if the walk contained any nodes, otherwise restore the current node
3614 \edef\forest@marshal{%
3615 \unexpanded{#1}%
3616 \noexpand\ifnum\noexpand\forest@nodewalk@n=\forest@nodewalk@n\relax
3617 \def\noexpand\forest@cn{\forest@cn}%

```

```

3618 \noexpand\else
3619 \unexpanded{%
3620 \edef\forest@cn{%
3621 \expandafter\forest@csvlist@getfirst@\expandafter{\forest@nodewalk@historyback}%
3622 }%
3623 }%
3624 \noexpand\fi
3625 }\forest@marshal
3626 }
3627 \def\forest@nodewalk@tolastvalid{%
3628 \ifnum\forest@cn=0
3629 \edef\forest@cn{\expandafter\forest@csvlist@getfirst@\expandafter{\forest@nodewalk@historyback}}%
3630 \ifnum\forest@cn=0
3631 \let\forest@cn\forest@nodewalk@origin
3632 \fi
3633 \fi
3634 }
3635 \def\forest@nodewalk@sortlist#1#2#3{%#1=list,#2=to,#3=asc/desc
3636 \edef\forest@nodewalksort@list{#1}%
3637 \expandafter\forest@nodewalk@sortlist@\expandafter{\number\numexpr#2-#3}%
3638 }
3639 \def\forest@nodewalk@sortlist@#1#2{%#1=to,#2=asc/desc
3640 \safeloop
3641 \unless\ifnum\safeloopn>#1\relax
3642 \expandafter\forest@csvlist@getfirstrest@\expandafter{\forest@nodewalksort@list}\forest@nodewalksort@cn\fi
3643 \csedef\forest@nodesort@\safeloopn{\forest@nodewalksort@cn}%
3644 \saferepeat
3645 \edef\forest@nodesort@sortkey{\forest@nodesort@by}%
3646 \forest@sort\forest@nodesort@cmpnodes\forest@nodesort@let#2{1}{#1}%
3647 \def\forest@nodewalksort@sorted{}%
3648 \safeloop
3649 \unless\ifnum\safeloopn>#1\relax
3650 \edef\forest@cn{\csname forest@nodesort@\safeloopn\endcsname}%
3651 \forest@nodewalk@makestep
3652 \saferepeat
3653 }

Find minimal/maximal node in a walk.

3654 \forestset{
3655 define long step={min}{n args=1,@bare,make for}{% the first min in the argument nodewalk
3656 \forest@nodewalk{#1,TeX={%
3657 \global\let\forest@global@temp\forest@nodewalk@historyback
3658 }}{}}%
3659 \forest@nodewalk@minmax{\forest@global@temp}{-1}{<}{\forest@nodewalk@minmax@node,}%
3660 },
3661 define long step={mins}{n args=1,@bare,make for}{% all mins in the argument nodewalk
3662 \forest@nodewalk{#1,TeX={%
3663 \global\let\forest@global@temp\forest@nodewalk@historyback
3664 }}{}}%
3665 \forest@nodewalk@minmax{\forest@global@temp}{-1}{<}{\forest@nodewalk@minmax@nodes}%
3666 },
3667 define long step={walk and min}{n args=1,@bare}{%
3668 \edef\forest@marshal{%
3669 \noexpand\pgfkeysalso{\unexpanded{#1}}%
3670 \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\forest@nodewalk@n}%
3671 }\forest@marshal
3672 },
3673 define long step={walk and mins}{n args=1,@bare}{%
3674 \edef\forest@marshal{%
3675 \noexpand\pgfkeysalso{\unexpanded{#1}}%
3676 \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\forest@nodewalk@n}%

```

```

3677     }\forest@marshal
3678 },
3679 define long step={min in nodewalk}{@bare}{% find the first min in the preceding nodewalk, step to it
3680     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{\forest@nodewalk@minmax@node,}%
3681 },
3682 define long step={mins in nodewalk}{@bare}{% find mins in the preceding nodewalk, step to mins
3683     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{\forest@nodewalk@minmax@nodes}%
3684 },
3685 define long step={min in nodewalk'}{@bare}{% find the first min in the preceding nodewalk, step to min in h
3686     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{}%
3687 },
3688 %
3689 define long step={max}{n args=1,@bare,make for}{% the first max in the argument nodewalk
3690     \forest@nodewalk{#1,TeX={%
3691         \global\let\forest@global@temp\forest@nodewalk@historyback
3692     }}{%
3693         \forest@nodewalk@minmax{\forest@global@temp}{-1}{>}{\forest@nodewalk@minmax@node,}%
3694     },
3695 define long step={maxs}{n args=1,@bare,make for}{% all maxs in the argument nodewalk
3696     \forest@nodewalk{#1,TeX={%
3697         \global\let\forest@global@temp\forest@nodewalk@historyback
3698     }}{%
3699         \forest@nodewalk@minmax{\forest@global@temp}{-1}{>}{\forest@nodewalk@minmax@nodes}%
3700     },
3701 define long step={walk and max}{n args=1,@bare}{%
3702     \edef\forest@marshal{%
3703         \noexpand\pgfkeysalso{\unexpanded{#1}}%
3704         \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
3705     }\forest@marshal
3706 },
3707 define long step={walk and maxs}{n args=1,@bare}{%
3708     \edef\forest@marshal{%
3709         \noexpand\pgfkeysalso{\unexpanded{#1}}%
3710         \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
3711     }\forest@marshal
3712 },
3713 define long step={max in nodewalk}{@bare}{% find the first max in the preceding nodewalk, step to it
3714     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{\forest@nodewalk@minmax@node,}%
3715 },
3716 define long step={maxs in nodewalk}{@bare}{% find maxs in the preceding nodewalk, step to maxs
3717     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{\forest@nodewalk@minmax@nodes}%
3718 },
3719 define long step={max in nodewalk'}{@bare}{% find the first max in the preceding nodewalk, step to max in h
3720     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{}%
3721 },
3722 }
3723
3724 \def\forest@nodewalk@minmax#1#2#3#4{%
3725     % #1 = list of nodes
3726     % #2 = max index in list (start with 1)
3727     % #3 = min/max = ascending/descending = </>
3728     % #4 = how many results? 1 = {\forest@nodewalk@minmax@node,}, all={\forest@nodewalk@minmax@nodes}, walk in
3729     \edef\forest@nodesort@sortkey{\forest@nodesort@by}%
3730     \edef\forest@nodewalk@minmax@N{\number\numexpr#2}%
3731     \edef\forest@nodewalk@minmax@n{%
3732         \edef\forest@nodewalk@minmax@list{#1}%
3733         \def\forest@nodewalk@minmax@nodes{%
3734             \def\forest@nodewalk@minmax@node{%
3735                 \ifdefempty{\forest@nodewalk@minmax@list}{%
3736                     }{%
3737                 \safeloop

```

```

3738 \expandafter\forest@csvglist@getfirstrest@\expandafter{\forest@nodewalk@minmax@list}\forest@nodewalk@min
3739 \ifnum\forest@nodewalk@minmax@cn=0 \else
3740 \ifdefempty{\forest@nodewalk@minmax@node}{%
3741 \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
3742 \edef\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,}%
3743 \edef\forest@nodewalk@minmax@n{\safeloopn}%
3744 }{%
3745 \csetdef{forest@nodesort@1}{\forest@nodewalk@minmax@node}%
3746 \csetdef{forest@nodesort@2}{\forest@nodewalk@minmax@cn}%
3747 \forest@nodesort@cmpnodes{2}{1}%
3748 \if=\forest@sort@cmp@result
3749 \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
3750 \epreto\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,}%
3751 \edef\forest@nodewalk@minmax@n{\safeloopn}%
3752 \else
3753 \if#3\forest@sort@cmp@result
3754 \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
3755 \edef\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,}%
3756 \edef\forest@nodewalk@minmax@n{\safeloopn}%
3757 \fi
3758 \fi
3759 }%
3760 \fi
3761 \ifdefempty{\forest@nodewalk@minmax@list}{\forest@tempfalse}{\forest@temptrue}%
3762 \ifnum\safeloopn=\forest@nodewalk@minmax@N\relax\forest@temptrue\fi
3763 \ifforest@temp
3764 \saferepeat
3765 \edef\forest@nodewalk@minmax@list{#4}%
3766 \ifdefempty\forest@nodewalk@minmax@list{%
3767 \forestset{nodewalk/jump back=\forest@nodewalk@minmax@n-1}% CHECK
3768 }{%
3769 \safeloop
3770 \expandafter\forest@csvglist@getfirstrest@\expandafter{\forest@nodewalk@minmax@list}\forest@cn\forest@
3771 \forest@nodewalk@makestep
3772 \ifdefempty{\forest@nodewalk@minmax@list}{\forest@tempfalse}{\forest@temptrue}%
3773 \ifforest@temp
3774 \saferepeat
3775 }%
3776 }%
3777 }

```

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.

```

3778 \newtoks\forest@nodewalk@shortsteps@resolution
3779 \newif\ifforest@nodewalk@areshortsteps
3780 \pgfqkeys{/forest/nodewalk}{
3781 .unknown/.code={%
3782 \forest@nodewalk@areshortstepsfalse
3783 \pgfkeysifdefined{/forest/\pgfkeyscurrentname/.cmd}{%
3784 }{%
3785 \ifx\pgfkeyscurrentvalue\pgfkeysnovalue@text % no value, so possibly short steps
3786 \forest@nodewalk@shortsteps@resolution{}%
3787 \forest@nodewalk@areshortstepstrue
3788 \expandafter\forest@nodewalk@shortsteps\pgfkeyscurrentname=====,% "=" and "," cannot be short st
3789 \fi
3790 }%
3791 \ifforest@nodewalk@areshortsteps
3792 \@escapeif{\expandafter\pgfkeysalso\expandafter{\the\forest@nodewalk@shortsteps@resolution}}%
3793 \else
3794 \@escapeif{\pgfkeysalso{/forest/\pgfkeyscurrentname={#1}}}%

```

```

3795 \fi
3796 },
3797 }
3798 \def\forest@nodewalk@shortsteps{%
3799 \futurelet\forest@nodewalk@nexttoken\forest@nodewalk@shortsteps@
3800 }
3801 \def\forest@nodewalk@shortsteps@{%
3802 \ifx\forest@nodewalk@nexttoken=%
3803 \let\forest@nodewalk@nextop\forest@nodewalk@shortsteps@end
3804 \else
3805 \ifx\forest@nodewalk@nexttoken\bgroup
3806 \letcs\forest@nodewalk@nextop{forest@shortstep@group}%
3807 \else
3808 \let\forest@nodewalk@nextop\forest@nodewalk@shortsteps@@
3809 \fi
3810 \fi
3811 \forest@nodewalk@nextop
3812 }
3813 \def\forest@nodewalk@shortsteps@@#1{%
3814 \ifcsdef{forest@shortstep@#1}{%
3815 \csname forest@shortstep@#1\endcsname
3816 }{%
3817 \forest@nodewalk@areshortstepsfalse
3818 \forest@nodewalk@shortsteps@end
3819 }%
3820 }
3821 % in the following definitions:
3822 % #1 = short step
3823 % #2 = (long) step, or a style in /forest/nodewalk (taking n args)
3824 \csdef{forest@nodewalk@defshortstep@0@args}#1#2{%
3825 \csdef{forest@shortstep@#1}{%
3826 \apptotoks\forest@nodewalk@shortsteps@resolution{,#2}%
3827 \forest@nodewalk@shortsteps}}
3828 \csdef{forest@nodewalk@defshortstep@1@args}#1#2{%
3829 \csdef{forest@shortstep@#1}##1{%
3830 \edef\forest@marshal####1{#2}%
3831 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}}%
3832 \forest@nodewalk@shortsteps}}
3833 \csdef{forest@nodewalk@defshortstep@2@args}#1#2{%
3834 \csdef{forest@shortstep@#1}##1##2{%
3835 \edef\forest@marshal####1####2{#2}%
3836 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}}%
3837 \forest@nodewalk@shortsteps}}
3838 \csdef{forest@nodewalk@defshortstep@3@args}#1#2{%
3839 \csdef{forest@shortstep@#1}##1##2##3{%
3840 \edef\forest@marshal####1####2####3{#2}%
3841 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}}%
3842 \forest@nodewalk@shortsteps}}
3843 \csdef{forest@nodewalk@defshortstep@4@args}#1#2{%
3844 \csdef{forest@shortstep@#1}##1##2##3##4{%
3845 \edef\forest@marshal####1####2####3####4{#2}%
3846 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}}%
3847 \forest@nodewalk@shortsteps}}
3848 \csdef{forest@nodewalk@defshortstep@5@args}#1#2{%
3849 \csdef{forest@shortstep@#1}##1##2##3##4##5{%
3850 \edef\forest@marshal####1####2####3####4####5{#2}%
3851 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}}%
3852 \forest@nodewalk@shortsteps}}
3853 \csdef{forest@nodewalk@defshortstep@6@args}#1#2{%
3854 \csdef{forest@shortstep@#1}##1##2##3##4##5##6{%
3855 \edef\forest@marshal####1####2####3####4####5####6{#2}%

```

```

3856 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}}%
3857 \forest@nodewalk@shortsteps}}
3858 \csdef{forest@nodewalk@defshortstep@7@args}#1#2{%
3859 \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7{%
3860 \edef\forest@marshal####1####2####3####4####5####6####7{#2}%
3861 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}}%
3862 \forest@nodewalk@shortsteps}}
3863 \csdef{forest@nodewalk@defshortstep@8@args}#1#2{%
3864 \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7##8{%
3865 \edef\forest@marshal####1####2####3####4####5####6####7####8{#2}%
3866 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}{##8}}%
3867 \forest@nodewalk@shortsteps}}
3868 \csdef{forest@nodewalk@defshortstep@9@args}#1#2{%
3869 \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7##8##9{%
3870 \edef\forest@marshal####1####2####3####4####5####6####7####8####9{#2}%
3871 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}{##8}{##9}}%
3872 \forest@nodewalk@shortsteps}}
3873 \forestset{
3874   define short step/.code n args=3{% #1 = short step, #2 = n args, #3 = long step
3875     \csname forest@nodewalk@defshortstep@#2@args\endcsname{#1}{#3}%
3876   },
3877 }
3878 \def\forest@nodewalk@shortsteps@end#1,{}

Define short-form steps.

3879 \forestset{
3880   define short step={group}{1}{group={#1}}, % {braces} are special
3881   define short step={p}{0}{previous},
3882   define short step={n}{0}{next},
3883   define short step={u}{0}{parent},
3884   define short step={s}{0}{sibling},
3885   define short step={c}{0}{current},
3886   define short step={o}{0}{origin},
3887   define short step={r}{0}{root},
3888   define short step={R}{0}{root'},
3889   define short step={P}{0}{previous leaf},
3890   define short step={N}{0}{next leaf},
3891   define short step={F}{0}{first leaf},
3892   define short step={L}{0}{last leaf},
3893   define short step={>}{0}{next on tier},
3894   define short step={<}{0}{previous on tier},
3895   define short step={1}{0}{n=1},
3896   define short step={2}{0}{n=2},
3897   define short step={3}{0}{n=3},
3898   define short step={4}{0}{n=4},
3899   define short step={5}{0}{n=5},
3900   define short step={6}{0}{n=6},
3901   define short step={7}{0}{n=7},
3902   define short step={8}{0}{n=8},
3903   define short step={9}{0}{n=9},
3904   define short step={l}{0}{last},
3905   define short step={b}{0}{back},
3906   define short step={f}{0}{forward},
3907   define short step={v}{0}{last valid},
3908   define short step={*}{2}{repeat={#1}{#2}},
3909   for 1/.style={for nodewalk={n=1}{#1}},
3910   for 2/.style={for nodewalk={n=2}{#1}},
3911   for 3/.style={for nodewalk={n=3}{#1}},
3912   for 4/.style={for nodewalk={n=4}{#1}},
3913   for 5/.style={for nodewalk={n=5}{#1}},
3914   for 6/.style={for nodewalk={n=6}{#1}},

```



```

3974 \pgfmathtruncatemacro\forest@temp{#1}%
3975 \edef\forest@marshal{%
3976 \noexpand\forest@node@foreach@breadthfirst@reversed
3977 {-1}%
3978 {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
3979 }\forest@marshal
3980 },
3981 define long step={level reversed<}{n args=1}{%
3982 \pgfmathtruncatemacro\forest@temp{(#1)-1}%
3983 \edef\forest@marshal{%
3984 \noexpand\forest@node@foreach@breadthfirst@reversed
3985 {\forest@temp}%
3986 {\noexpand\forest@nodewalk@makestep}%
3987 }\forest@marshal
3988 },
3989 %
3990 define long step={relative level}{n args=1}{%
3991 \pgfmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
3992 \edef\forest@marshal{%
3993 \noexpand\forest@node@foreach@breadthfirst
3994 {\forest@temp}%
3995 {\noexpand\ifnum\noexpand\forestove{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexp
3996 }\forest@marshal
3997 },
3998 define long step={relative level>}{n args=1}{%
3999 \pgfmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
4000 \edef\forest@marshal{%
4001 \noexpand\forest@node@foreach@breadthfirst
4002 {-1}%
4003 {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
4004 }\forest@marshal
4005 },
4006 define long step={relative level<}{n args=1}{%
4007 \pgfmathtruncatemacro\forest@temp{(#1)+\forestove{level}-1}%
4008 \edef\forest@marshal{%
4009 \noexpand\forest@node@foreach@breadthfirst
4010 {\forest@temp}%
4011 {\noexpand\forest@nodewalk@makestep}%
4012 }\forest@marshal
4013 },
4014 define long step={relative level reversed}{n args=1}{%
4015 \pgfmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
4016 \edef\forest@marshal{%
4017 \noexpand\forest@node@foreach@breadthfirst@reversed
4018 {\forest@temp}%
4019 {\noexpand\ifnum\noexpand\forestove{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexp
4020 }\forest@marshal
4021 },
4022 define long step={relative level reversed>}{n args=1}{%
4023 \pgfmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
4024 \edef\forest@marshal{%
4025 \noexpand\forest@node@foreach@breadthfirst@reversed
4026 {-1}%
4027 {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
4028 }\forest@marshal
4029 },
4030 define long step={relative level reversed<}{n args=1}{%
4031 \pgfmathtruncatemacro\forest@temp{(#1)+\forestove{level}-1}%
4032 \edef\forest@marshal{%
4033 \noexpand\forest@node@foreach@breadthfirst@reversed
4034 {\forest@temp}%

```



```

4035     {\noexpand\forest@nodewalk@makestep}%
4036   }\forest@marshal
4037 },
4038 define long step={children}{\forest@node@foreachchild{\forest@nodewalk@makestep}},
4039 define long step={children reversed}{\forest@node@foreachchild@reversed{\forest@nodewalk@makestep}},
4040 define long step={current and following siblings}{\forest@node@@forselfandfollowingsiblings{\forest@nodewalk@makestep}},
4041 define long step={following siblings}{style}{if nodewalk valid={next}{fake=next,current and following siblings}{\forest@node@@forsiblings{\forest@nodewalk@makestep}}},
4042 define long step={current and preceding siblings}{\forest@node@@forselfandprecedingsiblings{\forest@nodewalk@makestep}},
4043 define long step={preceding siblings}{style}{if nodewalk valid={previous}{fake=previous,current and preceding siblings}{\forest@node@@forsiblings{\forest@nodewalk@makestep}}},
4044 define long step={current and following siblings reversed}{\forest@node@@forselfandfollowingsiblings@reversed{\forest@nodewalk@makestep}},
4045 define long step={following siblings reversed}{style}{fake=next,current and following siblings reversed},
4046 define long step={current and preceding siblings reversed}{\forest@node@@forselfandprecedingsiblings@reversed{\forest@nodewalk@makestep}},
4047 define long step={preceding siblings reversed}{style}{fake=previous,current and preceding siblings reversed},
4048 define long step={siblings}{style}{for nodewalk'={preceding siblings},following siblings},
4049 define long step={siblings reversed}{style}{for nodewalk'={following siblings reversed},preceding siblings},
4050 define long step={ancestors}{style}{while={}{parent},last valid},
4051 define long step={current and ancestors}{style}{current,ancestors},
4052 define long step={following nodes}{style}{while={}{next node},last valid},
4053 define long step={preceding nodes}{style}{while={}{previous node},last valid},
4054 define long step={current and following nodes}{style}{current,following nodes},
4055 define long step={current and preceding nodes}{style}{current,preceding nodes},
4056 }
4057 \let\forest@nodewalkstephandler@styletrueorfalse\forest@nodewalkstephandler@styletrue

```

5.6 Dynamic tree

```

4058 \def\forest@last@node{0}
4059 \def\forest@nodehandleby@name@nodewalk@or@bracket#1{%
4060   \ifx\pgfkeysnovalue#1%
4061     \edef\forest@last@node{\forest@node@Nametoid{forest@last@node}}%
4062   \else
4063     \forest@nodehandleby@nnb@checkfirst#1\forest@END
4064   \fi
4065 }
4066 \def\forest@nodehandleby@nnb@checkfirst#1#2\forest@END{%
4067   \ifx[#1%
4068     \forest@create@node{#1#2}%
4069   \else
4070     \forest@forthis{%
4071       \forest@nameandgo{#1#2}%
4072       \ifnum\forest@cn=0
4073         \PackageError{forest}{Cannot use a dynamic key on the invalid node}{}%
4074       \fi
4075       \let\forest@last@node\forest@cn
4076     }%
4077   \fi
4078 }
4079 \def\forest@create@node#1{% #1=bracket representation
4080   \bracketParse{\forest@create@collectafterthought}%
4081   \forest@last@node=#1\forest@end@create@node
4082 }
4083 \def\forest@create@collectafterthought#1\forest@end@create@node{%
4084   \forest@node@Foreach{\forest@last@node}{%
4085     \forest@toletto{delay}{given options}%
4086     \forest@set{given options}{}%
4087   }%
4088   \forest@eappto{\forest@last@node}{delay}{,\unexpanded{#1}}%
4089   \forest@set{\forest@last@node}{given options}{delay={}}%
4090 }
4091 \def\forest@create@node@and@process@given@options#1{% #1=bracket representation

```

```

4092 \bracketParse{\forest@createandprocess@collectafterthought}%
4093 \forest@last@node=#1\forest@end@create@node
4094 }
4095 \def\forest@createandprocess@collectafterthought#1\forest@end@create@node{%
4096 \forest@node@Compute@numeric@ts@info{\forest@last@node}%
4097 \forest@saveandrestoremacro\forest@root{%
4098 \let\forest@root\forest@last@node
4099 \forestset{process keylist=given options}%
4100 }%
4101 }
4102 \def\forest@saveandrestoremacro#1#2{% #1 = the (zero-arg) macro to save before and restore after processing c
4103 \edef\forest@marshal{%
4104 \unexpanded{#2}%
4105 \noexpand\def\noexpand#1{\expandonce{#1}}%
4106 }\forest@marshal
4107 }
4108 \def\forest@saveandrestoreifcs#1#2{% #1 = the if cs to save before and restore after processing code in #2
4109 \edef\forest@marshal{%
4110 \unexpanded{#2}%
4111 \ifbool{#1}{\noexpand\setbool{#1}{true}}{\noexpand\setbool{#1}{false}}
4112 }\forest@marshal
4113 }
4114 \def\forest@saveandrestoretoks#1#2{% #1 = the toks to save before and restore after processing code in #2
4115 \edef\forest@marshal{%
4116 \unexpanded{#2}%
4117 \noexpand#1{\the#1}%
4118 }\forest@marshal
4119 }
4120 \def\forest@saveandrestoreregister#1#2{% #1 = the register to save before and restore after processing code i
4121 \edef\forest@marshal{%
4122 \unexpanded{#2}%
4123 \noexpand\forestregister{#1}{\forestregister{#1}}%
4124 }\forest@marshal
4125 }
4126 \def\forest@remove@node#1{%
4127 \forest@node@Remove{#1}%
4128 }
4129 \def\forest@append@node#1#2{%
4130 \forest@node@Remove{#2}%
4131 \forest@node@Append{#1}{#2}%
4132 }
4133 \def\forest@prepend@node#1#2{%
4134 \forest@node@Remove{#2}%
4135 \forest@node@Prepend{#1}{#2}%
4136 }
4137 \def\forest@insertafter@node#1#2{%
4138 \forest@node@Remove{#2}%
4139 \forest@node@Insertafter{\forestOve{#1}{@parent}}{#2}{#1}%
4140 }
4141 \def\forest@insertbefore@node#1#2{%
4142 \forest@node@Remove{#2}%
4143 \forest@node@Insertbefore{\forestOve{#1}{@parent}}{#2}{#1}%
4144 }
4145 \def\forest@set@root#1#2{%
4146 \def\forest@root{#2}%
4147 }
4148 \def\forest@appto@do@ynamics#1#2{%
4149 \forest@nodehandleby@name@nodewalk@or@bracket{#2}%
4150 \ifcase\forest@dynamics@copyhow\relax\or
4151 \forest@tree@copy{\forest@last@node}\forest@last@node
4152 \or

```

```

4153     \forest@node@copy{\forest@last@node}\forest@last@node
4154 \fi
4155 \forest@node@ifnamedefined{forest@last@node}{%
4156     \forest@depreto{\forest@last@node}{delay}
4157     {for id={\forest@node@Nametoid{forest@last@node}}{alias=forest@last@node},}%
4158 }{%
4159 \edef\forest@marshal{%
4160     \noexpand\apptotoks\noexpand\forest@do@ dynamics{%
4161         \noexpand#1{\forest@cn}{\forest@last@node}}%
4162 } \forest@marshal
4163 }
4164 \forestset{%
4165     create/.code={%
4166         \forest@create@node{#1}%
4167         \forest@for@node{\forest@last@node}{\forest@node@setalias{forest@last@node}}%
4168     },
4169     create'/.code={%
4170         \forest@create@node@and@process@given@options{#1}%
4171         \forest@for@node{\forest@last@node}{\forest@node@setalias{forest@last@node}}%
4172     },
4173     append/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@append@node{#1}},
4174     prepend/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@prepend@node{#1}},
4175     insert after/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@insertafter@node{#1}},
4176     insert before/.code={\def\forest@dynamics@copyhow{0}\forest@appto@do@dynamics\forest@insertbefore@node{#1}},
4177     append'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@append@node{#1}},
4178     prepend'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@prepend@node{#1}},
4179     insert after'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@insertafter@node{#1}},
4180     insert before'/.code={\def\forest@dynamics@copyhow{1}\forest@appto@do@dynamics\forest@insertbefore@node{#1}},
4181     append''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@append@node{#1}},
4182     prepend''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@prepend@node{#1}},
4183     insert after''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@insertafter@node{#1}},
4184     insert before''/.code={\def\forest@dynamics@copyhow{2}\forest@appto@do@dynamics\forest@insertbefore@node{#1}},
4185     remove/.code={%
4186         \pgfkeysalso{alias=forest@last@node}%
4187         \expandafter\apptotoks\expandafter\forest@do@dynamics\expandafter{%
4188             \expandafter\forest@remove@node\expandafter{\forest@cn}}%
4189     },
4190     set root/.code={%
4191         \def\forest@dynamics@copyhow{0}%
4192         \forest@appto@do@dynamics\forest@set@root{#1}%
4193     },
4194     replace by/.code={\forest@replaceby@code{#1}{insert after}},
4195     replace by'/.code={\forest@replaceby@code{#1}{insert after'}},
4196     replace by''/.code={\forest@replaceby@code{#1}{insert after''}},
4197     sort/.code={%
4198         \eapptotoks\forest@do@dynamics{%
4199             \noexpand\forest@nodesort
4200             \noexpand\forest@sort@ascending
4201             {\forest@cn}%
4202             {\expandonce{\forest@nodesort@by}}%
4203         }%
4204     },
4205     sort'/.code={%
4206         \eapptotoks\forest@do@dynamics{%
4207             \noexpand\forest@nodesort
4208             \noexpand\forest@sort@descending
4209             {\forest@cn}%
4210             {\expandonce{\forest@nodesort@by}}%
4211         }%
4212     },
4213     sort by/.store in=\forest@nodesort@by,

```

```

4214 }
4215 \def\forest@replaceby@code#1#2{%#1=node spec,#2=insert after['']['']}
4216 \ifnum\forest@ve{\parent}=0
4217 \pgfkeysalso{set root={#1}}%
4218 \else
4219 \pgfkeysalso{alias=forest@last@node,#2={#1}}%
4220 \eapptotoks\forest@do@dynamics{%
4221 \noexpand\ifnum\noexpand\forest@ve{\forest@cn}{\parent}=\forest@ve{\parent}
4222 \noexpand\forest@remove@node{\forest@cn}%
4223 \noexpand\fi
4224 }%
4225 \fi
4226 }
4227 \def\forest@nodesort#1#2#3{% #1 = direction, #2 = parent node, #3 = sort key
4228 \def\forest@nodesort@sortkey{#3}%
4229 \forest@for@node{#2}{\forest@nodesort@#1}%
4230 }
4231 \def\forest@nodesort@#1{%
4232 % prepare the array of child ids
4233 \c@pgf@counta=0
4234 \forest@toget{@first}\forest@nodesort@id
4235 \forest@loop
4236 \ifnum\forest@nodesort@id>0
4237 \advance\c@pgf@counta 1
4238 \csedef{forest@nodesort@the\c@pgf@counta}{\forest@nodesort@id}%
4239 \forest@get{\forest@nodesort@id}{@next}\forest@nodesort@id
4240 \forest@repeat
4241 % sort
4242 \forest@toget{n children}\forest@nodesort@n
4243 \forest@sort\forest@nodesort@cmpnodes\forest@nodesort@let#1{1}{\forest@nodesort@n}%
4244 % remove all children
4245 \forest@toget{@first}\forest@nodesort@id
4246 \forest@loop
4247 \ifnum\forest@nodesort@id>0
4248 \forest@node@Remove{\forest@nodesort@id}%
4249 \forest@toget{@first}\forest@nodesort@id
4250 \forest@repeat
4251 % insert the children in new order
4252 \c@pgf@counta=0
4253 \forest@loop
4254 \ifnum\c@pgf@counta<\forest@nodesort@n\relax
4255 \advance\c@pgf@counta 1
4256 \edef\temp{\csname forest@nodesort@the\c@pgf@counta\endcsname}%
4257 \forest@node@append{\csname forest@nodesort@the\c@pgf@counta\endcsname}%
4258 \forest@repeat
4259 }
4260 \def\forest@nodesort@cmpnodes#1#2{%
4261 \global\let\forest@nodesort@cmpresult\forest@sort@cmp@eq
4262 \foreach \forest@temp@pgfmath in \forest@nodesort@sortkey {%
4263 \forest@for@node{\csname forest@nodesort@#1\endcsname}{%
4264 \pgfmathparse{\forest@temp@pgfmath}\global\let\forest@global@tempa\pgfmathresult}%
4265 \forest@for@node{\csname forest@nodesort@#2\endcsname}{%
4266 \pgfmathparse{\forest@temp@pgfmath}\global\let\forest@global@tempb\pgfmathresult}%
4267 \ifdim\forest@global@tempa pt<\forest@global@tempb pt
4268 \global\let\forest@nodesort@cmpresult\forest@sort@cmp@lt
4269 \breakforeach
4270 \else
4271 \ifdim\forest@global@tempa pt>\forest@global@tempb pt
4272 \global\let\forest@nodesort@cmpresult\forest@sort@cmp@gt
4273 \breakforeach
4274 \fi

```

```

4275 \fi
4276 }%
4277 \forest@nodesort@cmpresult
4278 }
4279 \def\forest@nodesort@let#1#2{%
4280 \csletcs{forest@nodesort@#1}{forest@nodesort@#2}%
4281 }

```

6 Stages

```

4282 \def\forest@root{0}
4283 %%% begin listing region: stages
4284 \forestset{
4285   stages/.style={
4286     for root'={
4287       process keylist register=default preamble,
4288       process keylist register=preamble
4289     },
4290     process keylist=given options,
4291     process keylist=before typesetting nodes,
4292     typeset nodes stage,
4293     process keylist=before packing,
4294     pack stage,
4295     process keylist=before computing xy,
4296     compute xy stage,
4297     process keylist=before drawing tree,
4298     draw tree stage
4299   },
4300   typeset nodes stage/.style={for root'=typeset nodes},
4301   pack stage/.style={for root'=pack},
4302   compute xy stage/.style={for root'=compute xy},
4303   draw tree stage/.style={for root'=draw tree},
4304 }
4305 %%% end listing region: stages
4306 \forestset{
4307   process keylist/.code={%
4308     \forest@process@hook@keylist{#1}{#1 processing order/.try,processing order/.lastretry}},
4309   process keylist'/ .code 2 args={\forest@process@hook@keylist@nodynamics{#1}{#2}},
4310   process keylist''/.code 2 args={\forest@process@hook@keylist@{#1}{#2}},
4311   process keylist register/.code={\forest@process@keylist@register{#1}},
4312   process delayed/.code={%
4313     \forest@havedelayedoptions{@delay}{#1}%
4314     \ifforest@havedelayedoptions
4315       \forest@process@hook@keylist@nodynamics{@delay}{#1}%
4316     \fi
4317   },
4318   do dynamics/.code={%
4319     \the\forest@do@dynamics
4320     \forest@do@dynamics{}%
4321     \forest@node@Compute@numeric@ts@info{\forest@root}%
4322   },
4323   declare keylist={given options}{},
4324   declare keylist={before typesetting nodes}{},
4325   declare keylist={before packing}{},
4326   declare keylist={before packing node}{},
4327   declare keylist={after packing node}{},
4328   declare keylist={before computing xy}{},
4329   declare keylist={before drawing tree}{},
4330   declare keylist={delay}{},
4331   delay n/.style 2 args={if={#1==0}{#2}{delay@n={#1}{#2}}},
4332   delay@n/.style 2 args={

```

```

4333     if={#1==1}{delay={#2}}{delay={delay@n/.wrap pgfmath arg={{#1}{#2}}{#1-1}}}
4334 },
4335 if have delayed/.style 2 args={if have delayed'={processing order}{#1}{#2}},
4336 if have delayed'/.code n args=3{%
4337     \forest@havedelayedoptionsfalse
4338     \forest@forthis{%
4339         \forest@nodewalk{#1}{%
4340             TeX={%
4341                 \forest@get{delay}\forest@temp@delayed
4342                 \ifdefempty\forest@temp@delayed{}\forest@havedelayedoptionstrue}%
4343             }%
4344         }%
4345     }%
4346     \ifforest@havedelayedoptions\pgfkeysalso{#2}\else\pgfkeysalso{#3}\fi
4347 },
4348 typeset nodes/.code={%
4349     \forest@drawtree@preservenodeboxes@false
4350     \forest@nodewalk
4351         {typeset nodes processing order/.try,processing order/.lastretry}%
4352         {TeX={\forest@node@typeset}}%
4353 },
4354 typeset nodes'/.code={%
4355     \forest@drawtree@preservenodeboxes@true
4356     \forest@nodewalk
4357         {typeset nodes processing order/.try,processing order/.lastretry}%
4358         {TeX={\forest@node@typeset}}%
4359 },
4360 typeset node/.code={%
4361     \forest@drawtree@preservenodeboxes@false
4362     \forest@node@typeset
4363 },
4364 pack/.code={\forest@pack},
4365 pack'/.code={\forest@pack@onlythisnode},
4366 compute xy/.code={\forest@node@computeabsolute positions},
4367 draw tree box/.store in=\forest@drawtreebox,
4368 draw tree box,
4369 draw tree/.code={%
4370     \forest@drawtree@preservenodeboxes@false
4371     \forest@node@drawtree
4372 },
4373 draw tree'/.code={%
4374     \forest@drawtree@preservenodeboxes@true
4375     \forest@node@drawtree
4376 },
4377 %%% begin listing region: draw_tree_method
4378 draw tree method/.style={
4379     for nodewalk={
4380         draw tree nodes processing order/.try,
4381         draw tree processing order/.retry,
4382         processing order/.lastretry
4383     }{draw tree node},
4384     for nodewalk={
4385         draw tree edges processing order/.try,
4386         draw tree processing order/.retry,
4387         processing order/.lastretry
4388     }{draw tree edge},
4389     for nodewalk={
4390         draw tree tikz processing order/.try,
4391         draw tree processing order/.retry,
4392         processing order/.lastretry
4393     }{draw tree tikz}

```

```

4394 },
4395 %%% end listing region: draw_tree_method
4396 draw tree node/.code={\forest@draw@node},
4397 draw tree edge/.code={\forest@draw@edge},
4398 draw tree tikz/.code={\forest@draw@tikz},
4399 for nodewalk={processing order/.style={tree}}{ },
4400 %given options processing order/.style={processing order},
4401 %before typesetting nodes processing order/.style={processing order},
4402 %before packing processing order/.style={processing order},
4403 %before computing xy processing order/.style={processing order},
4404 %before drawing tree processing order/.style={processing order},
4405 }
4406 \newtoks\forest@do@dynamics
4407 \newif\ifforest@havedelayedoptions
4408 \def\forest@process@hook@keylist#1#2{%#1=keylist,#2=processing order nodewalk
4409   \safeloop
4410     \forest@for@node{\forest@root}{\forest@process@hook@keylist@{#1}{#2}}{%
4411       \expandafter\ifstremp\expandafter{\the\forest@do@dynamics}{ }{%
4412         \the\forest@do@dynamics
4413         \forest@do@dynamics={ }%
4414         \forest@node@Compute@numeric@ts@info{\forest@root}%
4415       }%
4416     \forest@for@node{\forest@root}{\forest@havedelayedoptions{#1}{#2}}{%
4417     \ifforest@havedelayedoptions
4418     \saferepeat
4419   }
4420 \def\forest@process@hook@keylist@nodynamics#1#2{%#1=keylist,#2=processing order nodewalk
4421   % note: this macro works on (nodewalk starting at) the current node
4422   \safeloop
4423     \forest@for@this{\forest@process@hook@keylist@{#1}{#2}}{%
4424     \forest@havedelayedoptions{#1}{#2}%
4425     \ifforest@havedelayedoptions
4426     \saferepeat
4427   }
4428 \def\forest@process@hook@keylist@#1#2{%#1=keylist,#2=processing order nodewalk
4429   \forest@nodewalk{#2}{%
4430     TeX={%
4431       \forest@toget{#1}\forest@temp@keys
4432       \ifdefvoid\forest@temp@keys{ }{%
4433         \forest@set{#1}{ }%
4434         \expandafter\forest@set\expandafter{\forest@temp@keys}%
4435       }%
4436     }%
4437   }%
4438 }
4439 \def\forest@process@keylist@register#1{%
4440   \edef\forest@marshal{%
4441     \noexpand\forest@set{\forest@register{#1}}%
4442   }\forest@marshal
4443 }

```

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

```

4444 \def\forest@havedelayedoptions#1#2{%#1 = keylist, #2=nodewalk
4445   \forest@havedelayedoptionsfalse
4446   \forest@for@this{%
4447     \forest@nodewalk{#2}{%
4448       TeX={%
4449         \forest@toget{delay}\forest@temp@delayed
4450         \ifdefempty\forest@temp@delayed{ }\forest@havedelayedoptionstrue}%
4451         \forest@tolet{#1}\forest@temp@delayed
4452         \forest@set{delay}{ }%

```

```

4453     }%
4454 }%
4455 }%
4456 }

```

6.1 Typesetting nodes

```

4457 \def\forest@node@typeset{%
4458   \let\forest@next\forest@node@typeset@
4459   \forestoifdefined{@box}{%
4460     \foresttoget{@box}\forest@temp
4461     \ifdefempty\forest@temp{%
4462       \locbox\forest@temp@box
4463       \forestolet{@box}\forest@temp@box
4464     }{%
4465       \ifforest@drawtree@preservenodeboxes@
4466       \let\forest@next\relax
4467     \fi
4468   }%
4469 }{%
4470   \locbox\forest@temp@box
4471   \forestolet{@box}\forest@temp@box
4472 }%
4473 \def\forest@node@typeset@restore{%
4474   \ifdefined\ifsa@tikz\forest@standalone@hack\fi
4475   \forest@next
4476   \forest@node@typeset@restore
4477 }
4478 \def\forest@standalone@hack{%
4479   \ifsa@tikz
4480     \let\forest@standalone@tikzpicture\tikzpicture
4481     \let\forest@standalone@endtikzpicture\endtikzpicture
4482     \let\tikzpicture\sa@orig@tikzpicture
4483     \let\endtikzpicture\sa@orig@endtikzpicture
4484     \def\forest@node@typeset@restore{%
4485       \let\tikzpicture\forest@standalone@tikzpicture
4486       \let\endtikzpicture\forest@standalone@endtikzpicture
4487     }%
4488     \fi
4489 }
4490 \newbox\forest@box
4491 \def\forest@pgf@notyetpositioned{not yet positionedPGFINTERNAL}
4492 \def\forest@node@typeset@{%
4493   \forestanchortotikzanchor{anchor}\forest@temp
4494   \edef\forest@marshal{%
4495     \noexpand\forestolet{anchor}\noexpand\forest@temp
4496     \noexpand\forest@node@typeset@@
4497     \noexpand\forestoset{anchor}{\forestov{anchor}}%
4498   }\forest@marshal
4499 }
4500 \def\forest@node@typeset@@{%
4501   \foresttoget{name}\forest@nodename
4502   \edef\forest@temp@nodeformat{\forestove{node format}}%
4503   \gdef\forest@smuggle{%
4504     \setbox0=\hbox{%
4505       \begin{tikzpicture}[%
4506         /forest/copy command key={/tikz/anchor}{/tikz/forest@orig@anchor},
4507         anchor/.style={%
4508           /forest/TeX={\forestanchortotikzanchor{##1}\forest@temp@anchor},
4509           forest@orig@anchor/.expand once=\forest@temp@anchor
4510         }%
4511       \pgfpositionnodelater{\forest@positionnodelater@save}%

```



```

4512     \forest@temp@nodeformat
4513     \pgfinterruptpath
4514     \pgfpointanchor{\forest@pgf@notyetpositioned\forest@nodename}{forestcomputenodeboundary}%
4515     \endpgfinterruptpath
4516     \end{tikzpicture}%
4517 }%
4518 \setbox\forestove{@box}=\box\forest@box % smuggle the box
4519 \forestolet{@boundary}\forest@global@boundary
4520 \forest@smuggle % ... and the rest
4521 }
4522
4523
4524 \forestset{
4525   declare readonly dimen={min x},
4526   declare readonly dimen={min y},
4527   declare readonly dimen={max x},
4528   declare readonly dimen={max y},
4529 }
4530 \def\forest@patch@enormouscoordinateboxbounds@plus#1{%
4531   \expandafter\ifstrequal\expandafter{#1}{16000.0pt}{\def#1{0.0pt}}{}%
4532 }
4533 \def\forest@patch@enormouscoordinateboxbounds@minus#1{%
4534   \expandafter\ifstrequal\expandafter{#1}{-16000.0pt}{\def#1{0.0pt}}{}%
4535 }
4536 \def\forest@positionnodelater@save{%
4537   \global\setbox\forest@box=\box\pgfpositionnodelaterbox
4538   \xappto\forest@smuggle{\noexpand\forestoset{later@name}{\pgfpositionnodelatername}}%
4539   % a bug in pgf? ---well, here's a patch
4540   \forest@patch@enormouscoordinateboxbounds@plus\pgfpositionnodelaterminx
4541   \forest@patch@enormouscoordinateboxbounds@plus\pgfpositionnodelaterminy
4542   \forest@patch@enormouscoordinateboxbounds@minus\pgfpositionnodelatermaxx
4543   \forest@patch@enormouscoordinateboxbounds@minus\pgfpositionnodelatermaxy
4544   % end of patch
4545   \xappto\forest@smuggle{\noexpand\forestoset{min x}{\pgfpositionnodelaterminx}}%
4546   \xappto\forest@smuggle{\noexpand\forestoset{min y}{\pgfpositionnodelaterminy}}%
4547   \xappto\forest@smuggle{\noexpand\forestoset{max x}{\pgfpositionnodelatermaxx}}%
4548   \xappto\forest@smuggle{\noexpand\forestoset{max y}{\pgfpositionnodelatermaxy}}%
4549 }
4550 \def\forest@node@forest@positionnodelater@restore{%
4551   \ifforest@drawtree@preservenodeboxes@
4552     \let\forest@boxorcopy\copy
4553   \else
4554     \let\forest@boxorcopy\box
4555   \fi
4556   \foresttoget{@box}\forest@temp
4557   \setbox\pgfpositionnodelaterbox=\forest@boxorcopy\forest@temp
4558   \edef\pgfpositionnodelatername{\forestove{later@name}}%
4559   \edef\pgfpositionnodelaterminx{\forestove{min x}}%
4560   \edef\pgfpositionnodelaterminy{\forestove{min y}}%
4561   \edef\pgfpositionnodelatermaxx{\forestove{max x}}%
4562   \edef\pgfpositionnodelatermaxy{\forestove{max y}}%
4563   \ifforest@drawtree@preservenodeboxes@
4564   \else
4565     \forestoset{@box}{}%
4566   \fi
4567 }

```

6.2 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.

```

4568 \def\forest@pack{%
4569   \forest@pack@computetiers
4570   \forest@pack@computegrowthuniformity
4571   \forest@@pack
4572 }
4573 \def\forest@@pack{%
4574   \ifnum\forestove{n children}>0
4575     \ifnum\forestove{uniform growth}>0
4576       \forest@pack@level@uniform
4577       \forest@pack@aligntiers@ofsubtree
4578       \forest@pack@sibling@uniform@recursive
4579     \else
4580       \forest@node@foreachchild{\forest@@pack}%
4581       \forest@process@hook@keylist@nodynamics{before packing node}{current}%
4582       \forest@pack@level@nonuniform
4583       \forest@pack@aligntiers
4584       \forest@pack@sibling@uniform@applyreversed
4585     \fi
4586   \fi
4587   \forest@toget{after packing node}\forest@temp@keys
4588   \forest@process@hook@keylist@nodynamics{after packing node}{current}%
4589 }
4590 % \forestset{recalculate tree boundary/.code={\forest@node@recalculate@edges}}
4591 % \def\forest@node@recalculate@edges{%
4592 %   \edef\forest@marshal{%
4593 %     \noexpand\forest@forthis{\noexpand\forest@node@getedges{\forestove{grow}}}%
4594 %   }\forest@marshal
4595 % }
4596 \def\forest@pack@onlythisnode{%
4597   \ifnum\forestove{n children}>0
4598     \forest@pack@computetiers
4599     \forest@pack@level@nonuniform
4600     \forest@pack@aligntiers
4601     \forest@node@foreachchild{\forest@set{s}{0pt}}%
4602     \forest@pack@sibling@uniform@applyreversed
4603   \fi
4604 }

```

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

```

4605 \def\forest@pack@computegrowthuniformity{%
4606   \forest@node@foreachchild{\forest@pack@computegrowthuniformity}%
4607   \edef\forest@pack@cgu@uniformity{%
4608     \ifnum\forestove{n children}=0
4609       2\else 1\fi
4610   }%
4611   \forest@toget{grow}\forest@pack@cgu@parentgrow
4612   \forest@node@foreachchild{%
4613     \ifnum\forestove{uniform growth}=0
4614       \def\forest@pack@cgu@uniformity{0}%
4615     \else
4616       \ifnum\forestove{uniform growth}=1
4617         \ifnum\forestove{grow}=\forest@pack@cgu@parentgrow\relax\else
4618           \def\forest@pack@cgu@uniformity{0}%
4619         \fi
4620       \fi
4621     \fi
4622   }%
4623   \forest@toget{before packing node}\forest@temp@a
4624   \forest@toget{after packing node}\forest@temp@b
4625   \expandafter\expandafter\expandafter\ifstrempy\expandafter\expandafter\expandafter{\expandafter\forest@temp@

```

```

4626 \forestolet{uniform growth}\forest@pack@cgu@uniformity
4627 }{%
4628 \forestoset{uniform growth}{0}%
4629 }%
4630 }

Pack children in the level dimension in a uniform tree.
4631 \def\forest@pack@level@uniform{%
4632 \let\forest@plu@minchildl\relax
4633 \forestogrow\forest@plu@grow
4634 \forest@node@foreachchild{%
4635 \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
4636 \advance\pgf@xa\forestove{1}\relax
4637 \ifx\forest@plu@minchildl\relax
4638 \edef\forest@plu@minchildl{\the\pgf@xa}%
4639 \else
4640 \ifdim\pgf@xa<\forest@plu@minchildl\relax
4641 \edef\forest@plu@minchildl{\the\pgf@xa}%
4642 \fi
4643 \fi
4644 }%
4645 \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
4646 \pgfutil@tempdima=\pgf@xb\relax
4647 \advance\pgfutil@tempdima-\forest@plu@minchildl\relax
4648 \advance\pgfutil@tempdima\forestove{1 sep}\relax
4649 \ifdim\pgfutil@tempdima>0pt
4650 \forest@node@foreachchild{%
4651 \forestoeset{1}{\the\dimexpr\forestove{1}+\the\pgfutil@tempdima}%
4652 }%
4653 \fi
4654 \forest@node@foreachchild{%
4655 \ifnum\forestove{n children}>0
4656 \forest@pack@level@uniform
4657 \fi
4658 }%
4659 }

Pack children in the level dimension in a non-uniform tree. (Expects the children to be fully packed.)
4660 \def\forest@pack@level@nonuniform{%
4661 \let\forest@plu@minchildl\relax
4662 \forestogrow\forest@plu@grow
4663 \forest@node@foreachchild{%
4664 \forest@node@getedge{negative}{\forest@plu@grow}{\forest@plnu@negativechildedge}%
4665 \forest@node@getedge{positive}{\forest@plu@grow}{\forest@plnu@positivechildedge}%
4666 \def\forest@plnu@childedge{\forest@plnu@negativechildedge\forest@plnu@positivechildedge}%
4667 \forest@path@getboundingrectangle@ls\forest@plnu@childedge{\forest@plu@grow}%
4668 \advance\pgf@xa\forestove{1}\relax
4669 \ifx\forest@plu@minchildl\relax
4670 \edef\forest@plu@minchildl{\the\pgf@xa}%
4671 \else
4672 \ifdim\pgf@xa<\forest@plu@minchildl\relax
4673 \edef\forest@plu@minchildl{\the\pgf@xa}%
4674 \fi
4675 \fi
4676 }%
4677 \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
4678 \pgfutil@tempdima=\pgf@xb\relax
4679 \advance\pgfutil@tempdima-\forest@plu@minchildl\relax
4680 \advance\pgfutil@tempdima\forestove{1 sep}\relax
4681 \ifdim\pgfutil@tempdima>0pt
4682 \forest@node@foreachchild{%
4683 \forestoeset{1}{\the\dimexpr\the\pgfutil@tempdima+\forestove{1}}%

```

```

4684 }%
4685 \fi
4686 }

Align tiers.
4687 \def\forest@pack@alignntiers{%
4688 \foresttoget{grow}\forest@temp@parentgrow
4689 \foresttoget{@tiers}\forest@temp@tiers
4690 \forlistloop\forest@pack@alignntier@\forest@temp@tiers
4691 }
4692 \def\forest@pack@alignntiers@ofsubtree{%
4693 \forest@node@foreach{\forest@pack@alignntiers}%
4694 }
4695 \def\forest@pack@alignntiers@computeabsl{%
4696 \forestoleto{abs@1}{1}%
4697 \forest@node@foreachdescendant{\forest@pack@alignntiers@computeabsl}%
4698 }
4699 \def\forest@pack@alignntiers@computeabsl@{%
4700 \forestoeset{abs@1}{\the\dimexpr\forestove{1}+\forestove{\forestove{@parent}}{abs@1}}%
4701 }
4702 \def\forest@pack@alignntier@#1{%
4703 \forest@pack@alignntiers@computeabsl
4704 \pgfutil@tempdima=-\maxdimen\relax
4705 \def\forest@temp@currenttier{#1}%
4706 \forest@node@foreach{%
4707 \foresttoget{tier}\forest@temp@tier
4708 \ifx\forest@temp@currenttier\forest@temp@tier
4709 \ifdim\pgfutil@tempdima<\forestove{abs@1}\relax
4710 \pgfutil@tempdima=\forestove{abs@1}\relax
4711 \fi
4712 \fi
4713 }%
4714 \ifdim\pgfutil@tempdima=-\maxdimen\relax\else
4715 \forest@node@foreach{%
4716 \foresttoget{tier}\forest@temp@tier
4717 \ifx\forest@temp@currenttier\forest@temp@tier
4718 \forestoeset{1}{\the\dimexpr\pgfutil@tempdima-\forestove{abs@1}+\forestove{1}}%
4719 \fi
4720 }%
4721 \fi
4722 }

```

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

```

4723 \def\forest@pack@sibling@uniform@recursive{%
4724 \forest@node@foreachchild{\forest@pack@sibling@uniform@recursive}%
4725 \forest@pack@sibling@uniform@applyreversed
4726 }

```

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

```

4727 \def\forest@pack@sibling@uniform@applyreversed{%
4728 \ifnum\forestove{n children}>1
4729 \ifnum\forestove{reversed}=0
4730 \forest@pack@sibling@uniform@main{first}{last}{next}{previous}%
4731 \else
4732 \forest@pack@sibling@uniform@main{last}{first}{previous}{next}%
4733 \fi
4734 \else
4735 \ifnum\forestove{n children}=1

```

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

```

4736 \csname forest@calign@\forestove{calign}\endcsname
4737 \fi

```

```

4738 \fi
4739 }

```

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

```

4740 \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.

```

4741 \forestoget{#1}\forest@child
4742 \edef\forest@marshal{%
4743 \noexpand\forest@fornode{\forestove{#1}}{%
4744 \noexpand\forest@node@getedge
4745 {positive}}%
4746 {\forestove{grow}}}%
4747 \noexpand\forest@temp@edge
4748 }%
4749 }\forest@marshal
4750 \forest@pack@pgfpoint@childposition\forest@child
4751 \let\forest@previous@positive@edge\pgfutil@empty
4752 \forest@extendpath\forest@previous@positive@edge\forest@temp@edge{}%
4753 \forest@get{\forest@child}{#3}\forest@child

```

Loop until the current child is the null node.

```

4754 \edef\forest@previous@child@s{0pt}%
4755 \safeloop
4756 \unless\ifnum\forest@child=0

```

Get the negative edge of the child.

```

4757 \edef\forest@temp{%
4758 \noexpand\forest@fornode{\forest@child}{%
4759 \noexpand\forest@node@getedge
4760 {negative}}%
4761 {\forestove{grow}}}%
4762 \noexpand\forest@temp@edge
4763 }%
4764 }\forest@temp

```

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

```

4765 \forest@pack@pgfpoint@childposition\forest@child

```

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

```

4766 \let\forest@child@negative@edge\pgfutil@empty
4767 \forest@extendpath\forest@child@negative@edge\forest@temp@edge{}%

```

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

```

4768 \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!)

```

4769 \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`.

```

4770 \ifx\forest@csdistance\relax
4771 %\forest@set{\forest@child}{s}{\forest@previous@child@s}%
4772 \else
4773 \advance\pgfutil@tempdimb-\forest@csdistance\relax
4774 \advance\pgfutil@tempdimb\forestove{s sep}\relax
4775 \forest@set{\forest@child}{s}{\the\dimexpr\forest@ove{\forest@child}{s}-\forest@csdistance+\forestove{s
4776 \fi

```

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

```
4777 \ifdim\forestOve{\forest@child}{s}<\forest@previous@child@s\relax
4778 \forestOset{\forest@child}{s}{\forest@previous@child@s}%
4779 \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.

```
4780 \forestOget{\forest@child}{s}\forest@child@s
4781 \edef\forest@previous@child@s{\forest@child@s}%
4782 \edef\forest@temp{%
4783 \noexpand\forest@fornode{\forest@child}{%
4784 \noexpand\forest@node@getedge
4785 {positive}%
4786 {\forestove{grow}}%
4787 \noexpand\forest@temp@edge
4788 }%
4789 }\forest@temp
4790 \forest@pack@pgfpoint@child@sposition\forest@child
4791 \forest@extendpath\forest@previous@positive@edge\forest@temp@edge{%
4792 \forest@getpositivetightedgeofpath\forest@previous@positive@edge\forest@previous@positive@edge
4793 \forestOget{\forest@child}{@#3}\forest@child
4794 \saferepeat
```

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

```
4795 \csname forest@calign@\forestove{calign}\endcsname
4796 }
```

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

```
4797 \def\forest@pack@pgfpoint@child@sposition#1{%
4798 {%
4799 \pgftransformreset
4800 \pgftransformrotate{\forestove{grow}}%
4801 \forest@fornode{#1}{%
4802 \pgfpointtransformed{\pgfpoint{\forestove{1}}{\forestove{s}}}%
4803 }%
4804 }%
4805 }
```

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

```
4806 \def\forest@pack@pgfpoint@positioningrow#1{%
4807 {%
4808 \pgftransformreset
4809 \pgftransformrotate{#1}%
4810 \pgfpointtransformed{\pgfpoint{\forestove{1}}{\forestove{s}}}%
4811 }%
4812 }
```

Child alignment.

```
4813 \def\forest@calign@s@shift#1{%
4814 \pgfutil@tempdima=#1\relax
4815 \forest@node@foreachchild{%
4816 \forestoeset{s}{\the\dimexpr\forestove{s}+\pgfutil@tempdima}%
4817 }%
4818 }
4819 \def\forest@calign@child{%
4820 \forest@calign@s@shift{-\forestOve{\forest@node@nornbarthchildid{\forestove{calign primary child}}}{s}}%
4821 }
4822 \csdef{forest@calign@child edge}{%
4823 {%
4824 \edef\forest@temp@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
4825 \pgftransformreset
4826 \pgftransformrotate{\forestove{grow}}%
```

```

4827 \pgfpointtransformed{\pgfqpoint{\forestOve{\forest@temp@child}{1}}{\forestOve{\forest@temp@child}{s}}}%
4828 \pgf@xa=\pgf@x\relax\pgf@ya=\pgf@y\relax
4829 \forest@Pointanchor{\forest@temp@child}{child anchor}%
4830 \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
4831 \forest@pointanchor{parent anchor}%
4832 \advance\pgf@xa-\pgf@x\relax\advance\pgf@ya-\pgf@y\relax
4833 \edef\forest@marshal{%
4834   \noexpand\pgftransformreset
4835   \noexpand\pgftransformrotate{-\forestove{grow}}%
4836   \noexpand\pgfpointtransformed{\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}%
4837 } \forest@marshal
4838 }%
4839 \forest@calign@s@shift{\the\dimexpr-\the\pgf@y}%
4840 }
4841 \csdef{forest@calign@midpoint}{%
4842   \forest@calign@s@shift{\the\dimexpr Opt -%
4843     (\forestOve{\forest@node@nornbarthchildid{\forestove{calign primary child}}}{s}%
4844     +\forestOve{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}{s}%
4845     )/2\relax
4846   }%
4847 }
4848 \csdef{forest@calign@edge midpoint}{%
4849   {%
4850     \edef\forest@temp@firstchild{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
4851     \edef\forest@temp@secondchild{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
4852     \pgftransformreset
4853     \pgftransformrotate{\forestove{grow}}%
4854     \pgfpointtransformed{\pgfqpoint{\forestOve{\forest@temp@firstchild}{1}}{\forestOve{\forest@temp@firstchild}{s}}}%
4855     \pgf@xa=\pgf@x\relax\pgf@ya=\pgf@y\relax
4856     \forest@Pointanchor{\forest@temp@firstchild}{child anchor}%
4857     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
4858     \edef\forest@marshal{%
4859       \noexpand\pgfpointtransformed{\noexpand\pgfqpoint{\forestOve{\forest@temp@secondchild}{1}}{\forestOve{\forest@temp@secondchild}{s}}}%
4860     } \forest@marshal
4861     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
4862     \forest@Pointanchor{\forest@temp@secondchild}{child anchor}%
4863     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
4864     \divide\pgf@xa2 \divide\pgf@ya2
4865     \edef\forest@marshal{%
4866       \noexpand\pgftransformreset
4867       \noexpand\pgftransformrotate{-\forestove{grow}}%
4868       \noexpand\pgfpointtransformed{\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}%
4869     } \forest@marshal
4870   }%
4871   \forest@calign@s@shift{\the\dimexpr-\the\pgf@y}%
4872 }

```

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.

```

4873 \csdef{forest@calign@fixed angles}{%
4874   \ifnum\forestove{n children}>1
4875     \edef\forest@ca@first@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
4876     \edef\forest@ca@second@child{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
4877     \ifnum\forestove{reversed}=1
4878       \let\forest@temp\forest@ca@first@child
4879       \let\forest@ca@first@child\forest@ca@second@child
4880       \let\forest@ca@second@child\forest@temp
4881     \fi
4882     \forest@get{\forest@ca@first@child}{1}\forest@ca@first@1

```

```

4883 \forestOget{\forest@ca@second@child}{1}\forest@ca@second@1
4884 \pgfmathsetlengthmacro\forest@ca@desired@s@distance{%
4885   tan(\forestove{calign secondary angle})*\forest@ca@second@1
4886   -tan(\forestove{calign primary angle})*\forest@ca@first@1
4887 }%
4888 \forestOget{\forest@ca@first@child}{s}\forest@ca@first@s
4889 \forestOget{\forest@ca@second@child}{s}\forest@ca@second@s
4890 \pgfmathsetlengthmacro\forest@ca@actual@s@distance{%
4891   \forest@ca@second@s-\forest@ca@first@s}%
4892 \ifdim\forest@ca@desired@s@distance>\forest@ca@actual@s@distance\relax
4893 \ifdim\forest@ca@actual@s@distance=0pt
4894   \pgfmathsetlength\pgfutil@tempdima{tan(\forestove{calign primary angle})*\forest@ca@second@1}%
4895   \pgfmathsetlength\pgfutil@tempdimb{\forest@ca@desired@s@distance/(\forestove{n children}-1)}%
4896   \forest@node@foreachchild{%
4897     \forestoet{s}{\the\pgfutil@tempdima}%
4898     \advance\pgfutil@tempdima\pgfutil@tempdimb
4899   }%
4900   \def\forest@calign@anchor{0pt}%
4901 \else
4902   \pgfmathsetmacro\forest@ca@ratio{%
4903     \forest@ca@desired@s@distance/\forest@ca@actual@s@distance}%
4904   \forest@node@foreachchild{%
4905     \pgfmathsetlengthmacro\forest@temp{\forest@ca@ratio*\forestove{s}}%
4906     \forestolet{s}\forest@temp
4907   }%
4908   \pgfmathsetlengthmacro\forest@calign@anchor{%
4909     -tan(\forestove{calign primary angle})*\forest@ca@first@1}%
4910   \fi
4911 \else
4912   \ifdim\forest@ca@desired@s@distance<\forest@ca@actual@s@distance\relax
4913     \pgfmathsetlengthmacro\forest@ca@ratio{%
4914       \forest@ca@actual@s@distance/\forest@ca@desired@s@distance}%
4915     \forest@node@foreachchild{%
4916       \pgfmathsetlengthmacro\forest@temp{\forest@ca@ratio*\forestove{1}}%
4917       \forestolet{1}\forest@temp
4918     }%
4919     \forestOget{\forest@ca@first@child}{1}\forest@ca@first@1
4920     \pgfmathsetlengthmacro\forest@calign@anchor{%
4921       -tan(\forestove{calign primary angle})*\forest@ca@first@1}%
4922     \fi
4923   \fi
4924   \forest@calign@s@shift{-\forest@calign@anchor}%
4925 \fi
4926 }
4927 \csdef{forest@calign@fixed edge angles}{%
4928   \ifnum\forestove{n children}>1
4929     \edef\forest@ca@first@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
4930     \edef\forest@ca@second@child{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
4931     \ifnum\forestove{reversed}=1
4932       \let\forest@temp\forest@ca@first@child
4933       \let\forest@ca@first@child\forest@ca@second@child
4934       \let\forest@ca@second@child\forest@temp
4935     \fi
4936     \forestOget{\forest@ca@first@child}{1}\forest@ca@first@1
4937     \forestOget{\forest@ca@second@child}{1}\forest@ca@second@1
4938     \forest@pointanchor{parent anchor}%
4939     \edef\forest@ca@parent@anchor@s{\the\pgf@x}%
4940     \edef\forest@ca@parent@anchor@l{\the\pgf@y}%
4941     \forest@Pointanchor{\forest@ca@first@child}{child anchor}%
4942     \edef\forest@ca@first@child@anchor@s{\the\pgf@x}%
4943     \edef\forest@ca@first@child@anchor@l{\the\pgf@y}%

```



```

4944 \forest@Pointanchor{\forest@ca@second@child}{child anchor}%
4945 \edef\forest@ca@second@child@anchor@s{\the\pgf{x}}%
4946 \edef\forest@ca@second@child@anchor@l{\the\pgf{y}}%
4947 \pgfmathsetlengthmacro\forest@ca@desired@second@edge@s{tan(\forestove{calign secondary angle})*%
4948 (\forest@ca@second@l-\forest@ca@second@child@anchor@l+\forest@ca@parent@anchor@l)}%
4949 \pgfmathsetlengthmacro\forest@ca@desired@first@edge@s{tan(\forestove{calign primary angle})*%
4950 (\forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@l)}%
4951 \pgfmathsetlengthmacro\forest@ca@desired@s@distance{\forest@ca@desired@second@edge@s-\forest@ca@desired@f
4952 \forest@get{\forest@ca@first@child}{s}\forest@ca@first@s
4953 \forest@get{\forest@ca@second@child}{s}\forest@ca@second@s
4954 \pgfmathsetlengthmacro\forest@ca@actual@s@distance{%
4955 \forest@ca@second@s+\forest@ca@second@child@anchor@s
4956 -\forest@ca@first@s-\forest@ca@first@child@anchor@s}%
4957 \ifdim\forest@ca@desired@s@distance>\forest@ca@actual@s@distance\relax
4958 \ifdim\forest@ca@actual@s@distance=0pt
4959 \forest@get{n children}\forest@temp@n@children
4960 \forest@node@foreachchild{%
4961 \forest@pointanchor{child anchor}%
4962 \edef\forest@temp@child@anchor@s{\the\pgf{x}}%
4963 \pgfmathsetlengthmacro\forest@temp{%
4964 \forest@ca@desired@first@edge@s+(\forestove{n}-1)*\forest@ca@desired@s@distance/(\forest@temp@n@
4965 \forest@let{s}\forest@temp
4966 }%
4967 \def\forest@calign@anchor{0pt}%
4968 \else
4969 \pgfmathsetmacro\forest@ca@ratio{%
4970 \forest@ca@desired@s@distance/\forest@ca@actual@s@distance}%
4971 \forest@node@foreachchild{%
4972 \forest@pointanchor{child anchor}%
4973 \edef\forest@temp@child@anchor@s{\the\pgf{x}}%
4974 \pgfmathsetlengthmacro\forest@temp{%
4975 \forest@ca@ratio*(%
4976 \forestove{s}-\forest@ca@first@s
4977 +\forest@temp@child@anchor@s-\forest@ca@first@child@anchor@s)%
4978 +\forest@ca@first@s
4979 +\forest@ca@first@child@anchor@s-\forest@temp@child@anchor@s}%
4980 \forest@let{s}\forest@temp
4981 }%
4982 \pgfmathsetlengthmacro\forest@calign@anchor{%
4983 -tan(\forestove{calign primary angle})*(\forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@
4984 +\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s
4985 }%
4986 \fi
4987 \else
4988 \ifdim\forest@ca@desired@s@distance<\forest@ca@actual@s@distance\relax
4989 \pgfmathsetlengthmacro\forest@ca@ratio{%
4990 \forest@ca@actual@s@distance/\forest@ca@desired@s@distance}%
4991 \forest@node@foreachchild{%
4992 \forest@pointanchor{child anchor}%
4993 \edef\forest@temp@child@anchor@l{\the\pgf{y}}%
4994 \pgfmathsetlengthmacro\forest@temp{%
4995 \forest@ca@ratio*(%
4996 \forestove{l}+\forest@ca@parent@anchor@l-\forest@temp@child@anchor@l)
4997 -\forest@ca@parent@anchor@l+\forest@temp@child@anchor@l}%
4998 \forest@let{l}\forest@temp
4999 }%
5000 \forest@get{\forest@ca@first@child}{l}\forest@ca@first@l
5001 \pgfmathsetlengthmacro\forest@calign@anchor{%
5002 -tan(\forestove{calign primary angle})*(\forest@ca@first@l+\forest@ca@parent@anchor@l-\forest@temp@
5003 +\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s
5004 }%

```

```

5005     \fi
5006     \fi
5007     \forest@calign@s@shift{-\forest@calign@anchor}%
5008     \fi
5009 }

```

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.

```

5010 \def\forest@node@getedge#1#2#3{%
5011     \forestoget{#1@edge@#2}#3%
5012     \ifx#3\relax
5013         \forest@node@foreachchild{%
5014             \forest@node@getedge{#1}{#2}{\forest@temp@edge}%
5015         }%
5016         \forest@forthis{\forest@node@getedges{#2}}%
5017         \forestoget{#1@edge@#2}#3%
5018     \fi
5019 }

```

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.

```

5020 \def\forest@node@getedges#1{%

```

Run the computation in a \TeX group.

```

5021     %{}%

```

Setup the grow line.

```

5022     \forest@setupgrowline{#1}%

```

Get the edge of the node itself.

```

5023     \ifnum\forestove{ignore}=0
5024         \forestoget{@boundary}\forest@node@boundary
5025     \else
5026         \def\forest@node@boundary{}%
5027     \fi
5028     \csname forest@getboth\forestove{fit}edgesofpath\endcsname
5029         \forest@node@boundary\forest@negative@node@edge\forest@positive@node@edge
5030     \forestolet{negative@edge@#1}\forest@negative@node@edge
5031     \forestolet{positive@edge@#1}\forest@positive@node@edge

```

Add the edges of the children.

```

5032     \get@edges@merge{negative}{#1}%
5033     \get@edges@merge{positive}{#1}%
5034     %}%
5035 }

```

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

```

5036 \def\forest@get@edges@merge#1#2{%
5037     \ifnum\forestove{n children}>0
5038         \forestoget{#1@edge@#2}\forest@node@edge

```

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

```

5039     \forest@pointanchor{parent anchor}%
5040     \edef\forest@getedge@pa@l{\the\pgf@x}%
5041     \edef\forest@getedge@pa@s{\the\pgf@y}%
5042     \eappto\forest@node@edge{\noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@pa@l}{\forest@getedge@pa@s}}

```

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

```

5043     \pgfgettransform\forest@temp@transform
5044     \pgftransformreset
5045     \pgftransformrotate{\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.

```

5046 \def\forest@all@edges{%
5047 \forest@node@foreachchild{%
5048 \forest@toget{#1@edge@#2}\forest@temp@edge
5049 \pgfpointtransformed{\pgfqpoint{\forest@ve{1}}{\forest@ve{s}}}%
5050 \forest@extendpath\forest@node@edge\forest@temp@edge}%
5051 \ifnum\forest@ve{ignore edge}=0
5052 \pgfpointadd
5053 {\pgfpointtransformed{\pgfqpoint{\forest@ve{1}}{\forest@ve{s}}}%
5054 {\forest@pointanchor{child anchor}}}%
5055 \pgfgetlastxy{\forest@getedge@ca@1}{\forest@getedge@ca@s}%
5056 \eappto\forest@all@edges{%
5057 \noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@pa@1}{\forest@getedge@pa@s}%
5058 \noexpand\pgfsyssoftpath@linetotoken{\forest@getedge@ca@1}{\forest@getedge@ca@s}%
5059 }%
5060 % this deals with potential overlap of the edges:
5061 \eappto\forest@node@edge{\noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@ca@1}{\forest@getedge@ca@s}%
5062 \fi
5063 }%
5064 \ifdefempty{\forest@all@edges}{}%
5065 \pgfintersectionofpaths{\pgfsetpath\forest@all@edges}{\pgfsetpath\forest@node@edge}%
5066 \def\forest@edgenode@intersections{%
5067 \forest@merge@intersectionloop
5068 \eappto\forest@node@edge{\expandonce{\forest@all@edges}\expandonce{\forest@edgenode@intersections}}%
5069 }%
5070 \pgfsettransform\forest@temp@transform

```

Process the path into an edge and store the edge.

```

5071 \csname forest@get#1\forest@ve{fit}edgeofpath\endcsname\forest@node@edge\forest@node@edge
5072 \forest@let{#1@edge@#2}\forest@node@edge
5073 \fi
5074 }
5075 %\newloop\forest@merge@loop
5076 \def\forest@merge@intersectionloop{%
5077 \c@pgf@counta=0
5078 \forest@loop
5079 \ifnum\c@pgf@counta<\pgfintersectionofsolutions\relax
5080 \advance\c@pgf@counta1
5081 \pgfpointintersectionsolution{\the\c@pgf@counta}%
5082 \eappto\forest@edgenode@intersections{\noexpand\pgfsyssoftpath@movetotoken
5083 {\the\pgf@x}{\the\pgf@y}}%
5084 \forest@repeat
5085 }

```

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

```

5086 \def\forest@node@getboundingrectangle@ls#1{%
5087 \forest@toget{@boundary}\forest@node@boundary
5088 \forest@path@getboundingrectangle@ls\forest@node@boundary{#1}%
5089 }

```

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).

```

5090 \def\forest@pgfpathtransformed#1{%
5091 \forest@save@pgfsyssoftpath@tokendef
5092 \let\pgfsyssoftpath@movetotoken\forest@pgfpathtransformed@moveto
5093 \let\pgfsyssoftpath@linetotoken\forest@pgfpathtransformed@lineto
5094 \pgfsyssoftpath@setcurrentpath\pgfutil@empty

```

```

5095 #1%
5096 \forest@restore@pgfsyssoftpath@tokendef
5097 }
5098 \def\forest@pgfpathtransformed@moveto#1#2{%
5099 \forest@pgfpathtransformed@op\pgfsyssoftpath@moveto{#1}{#2}%
5100 }
5101 \def\forest@pgfpathtransformed@lineto#1#2{%
5102 \forest@pgfpathtransformed@op\pgfsyssoftpath@lineto{#1}{#2}%
5103 }
5104 \def\forest@pgfpathtransformed@op#1#2#3{%
5105 \pgfpointransformed{\pgfqpoint{#2}{#3}}%
5106 \edef\forest@temp{%
5107 \noexpand#1{\the\pgf@x}{\the\pgf@y}%
5108 }%
5109 \forest@temp
5110 }

```

6.2.1 Tiers

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

```

5111 \def\forest@pack@computetiers{%
5112 {%
5113 \forest@pack@tiers@getalltiersinsubtree
5114 \forest@pack@tiers@computetierhierarchy
5115 \forest@pack@tiers@findcontainers
5116 \forest@pack@tiers@raisecontainers
5117 \forest@pack@tiers@computeprocessingorder
5118 \gdef\forest@smuggle{}%
5119 \forest@pack@tiers@write
5120 }%
5121 \forest@node@foreach{\forestoset{@tiers}{}}%
5122 \forest@smuggle
5123 }

```

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

```

5124 \def\forest@pack@tiers@getalltiersinsubtree{%
5125 \ifnum\forestove{n children}>0
5126 \forest@node@foreachchild{\forest@pack@tiers@getalltiersinsubtree}%
5127 \fi
5128 \foresttoget{tier}\forest@temp@mytier
5129 \def\forest@temp@mytiers{}%
5130 \ifdefempty\forest@temp@mytier{}{%
5131 \listead\forest@temp@mytiers\forest@temp@mytier
5132 }%
5133 \ifnum\forestove{n children}>0
5134 \forest@node@foreachchild{%
5135 \foresttoget{tiers}\forest@temp@tiers
5136 \forlistloop\forest@pack@tiers@forhandlerA\forest@temp@tiers
5137 }%
5138 \fi
5139 \forestolet{tiers}\forest@temp@mytiers
5140 }
5141 \def\forest@pack@tiers@forhandlerA#1{%
5142 \ifinlist{#1}\forest@temp@mytiers{}{%
5143 \listead\forest@temp@mytiers{#1}%
5144 }%
5145 }

```

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.

```

5146 \def\forest@pack@tiers@computetierhierarchy{%

```

```

5147 \def\forest@tiers@ancestors{%
5148 \forestoget{tiers}\forest@temp@mytiers
5149 \forlistloop\forest@pack@tiers@cth@init\forest@temp@mytiers
5150 \forest@pack@tiers@computetierhierarchy@
5151 }
5152 \def\forest@pack@tiers@cth@init#1{%
5153 \csdef{forest@tiers@higher@#1}{}%
5154 \csdef{forest@tiers@lower@#1}{}%
5155 }
5156 \def\forest@pack@tiers@computetierhierarchy@{%
5157 \forestoget{tier}\forest@temp@mytier
5158 \ifdefempty\forest@temp@mytier{}{%
5159 \forlistloop\forest@pack@tiers@forhandlerB\forest@tiers@ancestors
5160 \listead\forest@tiers@ancestors\forest@temp@mytier
5161 }%
5162 \forest@node@foreachchild{%
5163 \forest@pack@tiers@computetierhierarchy@
5164 }%
5165 \forestoget{tier}\forest@temp@mytier
5166 \ifdefempty\forest@temp@mytier{}{%
5167 \forest@listedel\forest@tiers@ancestors\forest@temp@mytier
5168 }%
5169 }
5170 \def\forest@pack@tiers@forhandlerB#1{%
5171 \def\forest@temp@tier{#1}%
5172 \ifx\forest@temp@tier\forest@temp@mytier
5173 \PackageError{forest}{Circular tier hierarchy (tier \forest@temp@mytier)}{}%
5174 \fi
5175 \ifinlistcs{#1}{forest@tiers@higher@\forest@temp@mytier}{}%
5176 \listcsadd{forest@tiers@higher@\forest@temp@mytier}{#1}%
5177 \xifinlistcs\forest@temp@mytier{forest@tiers@lower@#1}{}%
5178 \listcseadd{forest@tiers@lower@#1}{\forest@temp@mytier}%
5179 }
5180 \def\forest@pack@tiers@findcontainers{%
5181 \forestoget{tiers}\forest@temp@tiers
5182 \forlistloop\forest@pack@tiers@findcontainer\forest@temp@tiers
5183 }
5184 \def\forest@pack@tiers@findcontainer#1{%
5185 \def\forest@temp@tier{#1}%
5186 \forestoget{tier}\forest@temp@mytier
5187 \ifx\forest@temp@tier\forest@temp@mytier
5188 \csdef{forest@tiers@container@#1}{\forest@cn}%
5189 \else\@escapeif{%
5190 \forest@pack@tiers@findcontainerA{#1}%
5191 }\fi%
5192 }
5193 \def\forest@pack@tiers@findcontainerA#1{%
5194 \c@pgf@counta=0
5195 \forest@node@foreachchild{%
5196 \forestoget{tiers}\forest@temp@tiers
5197 \ifinlist{#1}\forest@temp@tiers{%
5198 \advance\c@pgf@counta 1
5199 \let\forest@temp@child\forest@cn
5200 }{}%
5201 }%
5202 \ifnum\c@pgf@counta>1
5203 \csdef{forest@tiers@container@#1}{\forest@cn}%
5204 \else\@escapeif{% surely =1
5205 \forest@fornode{\forest@temp@child}{%
5206 \forest@pack@tiers@findcontainer{#1}%
5207 }%

```

```

5208 } \fi
5209 }
5210 \def\forest@pack@tiers@raisecontainers{%
5211   \forestoget{tiers}\forest@temp@mytiers
5212   \forlistloop\forest@pack@tiers@rc@forhandlerA\forest@temp@mytiers
5213 }
5214 \def\forest@pack@tiers@rc@forhandlerA#1{%
5215   \edef\forest@tiers@temptier{#1}%
5216   \letcs\forest@tiers@containernodeoftier{forest@tiers@container@#1}%
5217   \letcs\forest@temp@lowertiers{forest@tiers@lower@#1}%
5218   \forlistloop\forest@pack@tiers@rc@forhandlerB\forest@temp@lowertiers
5219 }
5220 \def\forest@pack@tiers@rc@forhandlerB#1{%
5221   \letcs\forest@tiers@containernodeoflowertier{forest@tiers@container@#1}%
5222   \forestoget{\forest@tiers@containernodeoflowertier}{content}\lowercontent
5223   \forestoget{\forest@tiers@containernodeoftier}{content}\uppercontent
5224   \forest@fornode{\forest@tiers@containernodeoflowertier}{%
5225     \forest@ifancestorof
5226       {\forest@tiers@containernodeoftier}
5227       {\csletcs{forest@tiers@container@forest@tiers@temptier}{forest@tiers@container@#1}}%
5228     }%
5229   }%
5230 }
5231 \def\forest@pack@tiers@computeprocessingorder{%
5232   \def\forest@tiers@processingorder{}%
5233   \forestoget{tiers}\forest@tiers@cpo@tierstodo
5234   \safeloop
5235     \ifdefempty\forest@tiers@cpo@tierstodo{\forest@tempfalse}{\forest@temptrue}%
5236   \ifforest@temp
5237     \def\forest@tiers@cpo@tiersremaining{}%
5238     \def\forest@tiers@cpo@tiersindependent{}%
5239     \forlistloop\forest@pack@tiers@cpo@forhandlerA\forest@tiers@cpo@tierstodo
5240     \ifdefempty\forest@tiers@cpo@tiersindependent{%
5241       \PackageError{forest}{Circular tiers!}{-}{-}%
5242     \forlistloop\forest@pack@tiers@cpo@forhandlerB\forest@tiers@cpo@tiersremaining
5243     \let\forest@tiers@cpo@tierstodo\forest@tiers@cpo@tiersremaining
5244     \saferepeat
5245   }
5246   \def\forest@pack@tiers@cpo@forhandlerA#1{%
5247     \ifcsempy{forest@tiers@higher@#1}{%
5248       \listadd\forest@tiers@cpo@tiersindependent{#1}%
5249       \listadd\forest@tiers@processingorder{#1}%
5250     }{%
5251       \listadd\forest@tiers@cpo@tiersremaining{#1}%
5252     }%
5253   }
5254   \def\forest@pack@tiers@cpo@forhandlerB#1{%
5255     \def\forest@pack@tiers@cpo@aremainingtier{#1}%
5256     \forlistloop\forest@pack@tiers@cpo@forhandlerC\forest@tiers@cpo@tiersindependent
5257   }
5258   \def\forest@pack@tiers@cpo@forhandlerC#1{%
5259     \ifinlistcs{#1}{forest@tiers@higher@\forest@pack@tiers@cpo@aremainingtier}{%
5260       \forest@listcsdel{forest@tiers@higher@\forest@pack@tiers@cpo@aremainingtier}{#1}%
5261     }{}%
5262   }
5263   \def\forest@pack@tiers@write{%
5264     \forlistloop\forest@pack@tiers@write@forhandler\forest@tiers@processingorder
5265   }
5266   \def\forest@pack@tiers@write@forhandler#1{%
5267     \forest@fornode{\csname forest@tiers@container@#1\endcsname}{%
5268       \forest@pack@tiers@check{#1}%

```

```

5269 }%
5270 \xappto\forest@smuggle{%
5271   \noexpand\listadd
5272     \forest@om{\csname forest@tiers@container@#1\endcsname}{@tiers}%
5273     {#1}%
5274 }%
5275 }
5276 % checks if the tier is compatible with growth changes and calign=node/edge angle
5277 \def\forest@pack@tiers@check#1{%
5278   \def\forest@temp@currenttier{#1}%
5279   \forest@node@foreachdescendant{%
5280     \ifnum\forestove{grow}=\forestove{\forestove{@parent}}{grow}
5281     \else
5282       \forest@pack@tiers@check@grow
5283     \fi
5284     \ifnum\forestove{n children}>1
5285       \forest@toget{calign}\forest@temp
5286       \ifx\forest@temp\forest@pack@tiers@check@nodeangle
5287         \forest@pack@tiers@check@calign
5288       \fi
5289       \ifx\forest@temp\forest@pack@tiers@check@edgeangle
5290         \forest@pack@tiers@check@calign
5291       \fi
5292     \fi
5293   }%
5294 }
5295 \def\forest@pack@tiers@check@nodeangle{node angle}%
5296 \def\forest@pack@tiers@check@edgeangle{edge angle}%
5297 \def\forest@pack@tiers@check@grow{%
5298   \forest@toget{content}\forest@temp@content
5299   \let\forest@temp@currentnode\forest@cn
5300   \forest@node@foreachdescendant{%
5301     \forest@toget{tier}\forest@temp
5302     \ifx\forest@temp@currenttier\forest@temp
5303       \forest@pack@tiers@check@grow@error
5304     \fi
5305   }%
5306 }
5307 \def\forest@pack@tiers@check@grow@error{%
5308   \PackageError{forest}{Tree growth direction changes in node \forest@temp@currentnode\space
5309     (content: \forest@temp@content), while tier '\forest@temp' is specified for nodes both
5310     out- and inside the subtree rooted in node \forest@temp@currentnode. This will not work.}{}%
5311 }
5312 \def\forest@pack@tiers@check@calign{%
5313   \forest@node@foreachchild{%
5314     \forest@toget{tier}\forest@temp
5315     \ifx\forest@temp@currenttier\forest@temp
5316       \forest@pack@tiers@check@calign@warning
5317     \fi
5318   }%
5319 }
5320 \def\forest@pack@tiers@check@calign@warning{%
5321   \PackageWarning{forest}{Potential option conflict: node \forestove{@parent} (content:
5322     '\forestove{\forestove{@parent}}{calign}') was given 'calign=\forestove{calign}', while its
5323     child \forest@cn\space (content: '\forestove{content}') was given 'tier=\forestove{tier}'.
5324     The parent's 'calign' will only work if the child was the lowest node on its tier before the
5325     alignment.}%
5326 }

```

6.2.2 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.

```

5327 \pgfdeclaregenericanchor{forestcomputenodeboundary}{%
5328   \letcs\forest@temp@boundary@macro{forest@compute@node@boundary@#1}%
5329   \ifcsname forest@compute@node@boundary@#1\endcsname
5330     \csname forest@compute@node@boundary@#1\endcsname
5331   \else
5332     \forest@compute@node@boundary@rectangle
5333   \fi
5334   \pgfsyssoftpath@getcurrentpath\forest@temp
5335   \global\let\forest@global@boundary\forest@temp
5336 }
5337 \def\forest@mt#1{%
5338   \expandafter\pgfpointanchor\expandafter{\pgfreferencednodename}{#1}%
5339   \pgfsyssoftpath@moveto{\the\pgf@x}{\the\pgf@y}%
5340 }%
5341 \def\forest@lt#1{%
5342   \expandafter\pgfpointanchor\expandafter{\pgfreferencednodename}{#1}%
5343   \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
5344 }%
5345 \def\forest@compute@node@boundary@coordinate{%
5346   \forest@mt{center}%
5347 }
5348 \def\forest@compute@node@boundary@circle{%
5349   \forest@mt{east}%
5350   \forest@lt{north east}%
5351   \forest@lt{north}%
5352   \forest@lt{north west}%
5353   \forest@lt{west}%
5354   \forest@lt{south west}%
5355   \forest@lt{south}%
5356   \forest@lt{south east}%
5357   \forest@lt{east}%
5358 }
5359 \def\forest@compute@node@boundary@rectangle{%
5360   \forest@mt{south west}%
5361   \forest@lt{south east}%
5362   \forest@lt{north east}%
5363   \forest@lt{north west}%
5364   \forest@lt{south west}%
5365 }
5366 \def\forest@compute@node@boundary@diamond{%
5367   \forest@mt{east}%
5368   \forest@lt{north}%
5369   \forest@lt{west}%
5370   \forest@lt{south}%
5371   \forest@lt{east}%
5372 }
5373 \let\forest@compute@node@boundary@ellipse\forest@compute@node@boundary@circle
5374 \def\forest@compute@node@boundary@trapezium{%
5375   \forest@mt{top right corner}%
5376   \forest@lt{top left corner}%
5377   \forest@lt{bottom left corner}%
5378   \forest@lt{bottom right corner}%
5379   \forest@lt{top right corner}%
5380 }
5381 \def\forest@compute@node@boundary@semicircle{%
5382   \forest@mt{arc start}%
5383   \forest@lt{north}%

```



```

5384 \forest@lt{east}%
5385 \forest@lt{north east}%
5386 \forest@lt{apex}%
5387 \forest@lt{north west}%
5388 \forest@lt{west}%
5389 \forest@lt{arc end}%
5390 \forest@lt{arc start}%
5391 }
5392 %\newloop\forest@computenodeboundary@loop
5393 \csdef{forest@compute@node@boundary@regular polygon}{%
5394 \forest@mt{corner 1}%
5395 \c@pgf@counta=\sides\relax
5396 \forest@loop
5397 \ifnum\c@pgf@counta>0
5398 \forest@lt{corner \the\c@pgf@counta}%
5399 \advance\c@pgf@counta-1
5400 \forest@repeat
5401 }%
5402 \def\forest@compute@node@boundary@star{%
5403 \forest@mt{outer point 1}%
5404 \c@pgf@counta=\totalstarpoints\relax
5405 \divide\c@pgf@counta2
5406 \forest@loop
5407 \ifnum\c@pgf@counta>0
5408 \forest@lt{inner point \the\c@pgf@counta}%
5409 \forest@lt{outer point \the\c@pgf@counta}%
5410 \advance\c@pgf@counta-1
5411 \forest@repeat
5412 }%
5413 \csdef{forest@compute@node@boundary@isosceles triangle}{%
5414 \forest@mt{apex}%
5415 \forest@lt{left corner}%
5416 \forest@lt{right corner}%
5417 \forest@lt{apex}%
5418 }
5419 \def\forest@compute@node@boundary@kite{%
5420 \forest@mt{upper vertex}%
5421 \forest@lt{left vertex}%
5422 \forest@lt{lower vertex}%
5423 \forest@lt{right vertex}%
5424 \forest@lt{upper vertex}%
5425 }
5426 \def\forest@compute@node@boundary@dart{%
5427 \forest@mt{tip}%
5428 \forest@lt{left tail}%
5429 \forest@lt{tail center}%
5430 \forest@lt{right tail}%
5431 \forest@lt{tip}%
5432 }
5433 \csdef{forest@compute@node@boundary@circular sector}{%
5434 \forest@mt{sector center}%
5435 \forest@lt{arc start}%
5436 \forest@lt{arc center}%
5437 \forest@lt{arc end}%
5438 \forest@lt{sector center}%
5439 }
5440 \def\forest@compute@node@boundary@cylinder{%
5441 \forest@mt{top}%
5442 \forest@lt{after top}%
5443 \forest@lt{before bottom}%
5444 \forest@lt{bottom}%

```

```

5445 \forest@lt{after bottom}%
5446 \forest@lt{before top}%
5447 \forest@lt{top}%
5448 }
5449 \cslet{forest@compute@node@boundary@forbidden sign}\forest@compute@node@boundary@circle
5450 \cslet{forest@compute@node@boundary@magnifying glass}\forest@compute@node@boundary@circle
5451 \def\forest@compute@node@boundary@cloud{%
5452   \getradii
5453   \forest@mt{puff 1}%
5454   \c@pgf@counta=\puffs\relax
5455   \forest@loop
5456   \ifnum\c@pgf@counta>0
5457     \forest@lt{puff \the\c@pgf@counta}%
5458     \advance\c@pgf@counta-1
5459   \forest@repeat
5460 }
5461 \def\forest@compute@node@boundary@starburst{
5462   \calculatestarburstpoints
5463   \forest@mt{outer point 1}%
5464   \c@pgf@counta=\totalpoints\relax
5465   \divide\c@pgf@counta2
5466   \forest@loop
5467   \ifnum\c@pgf@counta>0
5468     \forest@lt{inner point \the\c@pgf@counta}%
5469     \forest@lt{outer point \the\c@pgf@counta}%
5470     \advance\c@pgf@counta-1
5471   \forest@repeat
5472 }%
5473 \def\forest@compute@node@boundary@signal{%
5474   \forest@mt{east}%
5475   \forest@lt{south east}%
5476   \forest@lt{south west}%
5477   \forest@lt{west}%
5478   \forest@lt{north west}%
5479   \forest@lt{north east}%
5480   \forest@lt{east}%
5481 }
5482 \def\forest@compute@node@boundary@tape{%
5483   \forest@mt{north east}%
5484   \forest@lt{60}%
5485   \forest@lt{north}%
5486   \forest@lt{120}%
5487   \forest@lt{north west}%
5488   \forest@lt{south west}%
5489   \forest@lt{240}%
5490   \forest@lt{south}%
5491   \forest@lt{310}%
5492   \forest@lt{south east}%
5493   \forest@lt{north east}%
5494 }
5495 \csdef{forest@compute@node@boundary@single arrow}{%
5496   \forest@mt{tip}%
5497   \forest@lt{after tip}%
5498   \forest@lt{after head}%
5499   \forest@lt{before tail}%
5500   \forest@lt{after tail}%
5501   \forest@lt{before head}%
5502   \forest@lt{before tip}%
5503   \forest@lt{tip}%
5504 }
5505 \csdef{forest@compute@node@boundary@double arrow}{%

```

```

5506 \forest@mt{tip 1}%
5507 \forest@lt{after tip 1}%
5508 \forest@lt{after head 1}%
5509 \forest@lt{before head 2}%
5510 \forest@lt{before tip 2}%
5511 \forest@mt{tip 2}%
5512 \forest@lt{after tip 2}%
5513 \forest@lt{after head 2}%
5514 \forest@lt{before head 1}%
5515 \forest@lt{before tip 1}%
5516 \forest@lt{tip 1}%
5517 }
5518 \csdef{forest@compute@node@boundary@arrow box}{%
5519 \forest@mt{before north arrow}%
5520 \forest@lt{before north arrow head}%
5521 \forest@lt{before north arrow tip}%
5522 \forest@lt{north arrow tip}%
5523 \forest@lt{after north arrow tip}%
5524 \forest@lt{after north arrow head}%
5525 \forest@lt{after north arrow}%
5526 \forest@lt{north east}%
5527 \forest@lt{before east arrow}%
5528 \forest@lt{before east arrow head}%
5529 \forest@lt{before east arrow tip}%
5530 \forest@lt{east arrow tip}%
5531 \forest@lt{after east arrow tip}%
5532 \forest@lt{after east arrow head}%
5533 \forest@lt{after east arrow}%
5534 \forest@lt{south east}%
5535 \forest@lt{before south arrow}%
5536 \forest@lt{before south arrow head}%
5537 \forest@lt{before south arrow tip}%
5538 \forest@lt{south arrow tip}%
5539 \forest@lt{after south arrow tip}%
5540 \forest@lt{after south arrow head}%
5541 \forest@lt{after south arrow}%
5542 \forest@lt{south west}%
5543 \forest@lt{before west arrow}%
5544 \forest@lt{before west arrow head}%
5545 \forest@lt{before west arrow tip}%
5546 \forest@lt{west arrow tip}%
5547 \forest@lt{after west arrow tip}%
5548 \forest@lt{after west arrow head}%
5549 \forest@lt{after west arrow}%
5550 \forest@lt{north west}%
5551 \forest@lt{before north arrow}%
5552 }
5553 \cslet{forest@compute@node@boundary@circle split}\forest@compute@node@boundary@circle
5554 \cslet{forest@compute@node@boundary@circle solidus}\forest@compute@node@boundary@circle
5555 \cslet{forest@compute@node@boundary@ellipse split}\forest@compute@node@boundary@ellipse
5556 \cslet{forest@compute@node@boundary@rectangle split}\forest@compute@node@boundary@rectangle
5557 \def\forest@compute@node@boundary@@callout{%
5558 \beforecalloutpointer
5559 \pgfsyssoftpath@moveto{\the\pgf@x}{\the\pgf@y}%
5560 \calloutpointeranchor
5561 \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
5562 \aftercalloutpointer
5563 \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
5564 }
5565 \csdef{forest@compute@node@boundary@rectangle callout}{%
5566 \forest@compute@node@boundary@rectangle

```

```

5567 \rectanglecalloutpoints
5568 \forest@compute@node@boundary@@callout
5569 }
5570 \csdef{forest@compute@node@boundary@ellipse callout}{%
5571 \forest@compute@node@boundary@ellipse
5572 \ellipsecalloutpoints
5573 \forest@compute@node@boundary@@callout
5574 }
5575 \csdef{forest@compute@node@boundary@cloud callout}{%
5576 \forest@compute@node@boundary@cloud
5577 % at least a first approx...
5578 \forest@mt{center}%
5579 \forest@lt{pointer}%
5580 }%
5581 \csdef{forest@compute@node@boundary@cross out}{%
5582 \forest@mt{south east}%
5583 \forest@lt{north west}%
5584 \forest@mt{south west}%
5585 \forest@lt{north east}%
5586 }%
5587 \csdef{forest@compute@node@boundary@strike out}{%
5588 \forest@mt{north east}%
5589 \forest@lt{south west}%
5590 }%
5591 \csdef{forest@compute@node@boundary@rounded rectangle}{%
5592 \forest@mt{east}%
5593 \forest@lt{north east}%
5594 \forest@lt{north}%
5595 \forest@lt{north west}%
5596 \forest@lt{west}%
5597 \forest@lt{south west}%
5598 \forest@lt{south}%
5599 \forest@lt{south east}%
5600 \forest@lt{east}%
5601 }%
5602 \csdef{forest@compute@node@boundary@chamfered rectangle}{%
5603 \forest@mt{before south west}%
5604 \forest@mt{after south west}%
5605 \forest@lt{before south east}%
5606 \forest@lt{after south east}%
5607 \forest@lt{before north east}%
5608 \forest@lt{after north east}%
5609 \forest@lt{before north west}%
5610 \forest@lt{after north west}%
5611 \forest@lt{before south west}%
5612 }%

```

6.3 Compute absolute positions

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

```

5613 \def\forest@node@computeabsolutepositions{%
5614 \edef\forest@marshal{%
5615 \noexpand\forest@node@foreachchild{%
5616 \noexpand\forest@node@computeabsolutepositions@{\forestove{x}}{\forestove{y}}{\forestove{grow}}}%
5617 }%
5618 }\forest@marshal
5619 }
5620 \def\forest@node@computeabsolutepositions@#1#2#3{%
5621 \pgfpointadd{\pgfpoint{#1}{#2}}{%

```

```

5622 \pgfpointadd{\pgfpolar{#3}{\forestove{1}}}{\pgfpolar{90 + #3}{\forestove{s}}}}%
5623 \pgfgetlastxy\forest@temp@x\forest@temp@y
5624 \forestolet{x}\forest@temp@x
5625 \forestolet{y}\forest@temp@y
5626 \edef\forest@marshal{%
5627 \noexpand\forest@node@foreachchild{%
5628 \noexpand\forest@node@computeabsolute positions@{\forest@temp@x}{\forest@temp@y}{\forestove{grow}}}%
5629 }%
5630 }\forest@marshal
5631 }

```

6.4 Drawing the tree

```

5632 \newif\ifforest@drawtree@preservenodeboxes@
5633 \def\forest@node@drawtree{%
5634 \expandafter\ifstrequal\expandafter{\forest@drawtreebox}{\pgfkeysnovalue}{%
5635 \let\forest@drawtree@beginbox\relax
5636 \let\forest@drawtree@endbox\relax
5637 }{%
5638 \edef\forest@drawtree@beginbox{\global\setbox\forest@drawtreebox=\hbox\bgroup}%
5639 \let\forest@drawtree@endbox\egroup
5640 }%
5641 \ifforest@external@
5642 \ifforest@externalize@tree@
5643 \forest@temptrue
5644 \else
5645 \tikzifexternalizing{%
5646 \ifforest@was@tikzexternalwasenable
5647 \forest@temptrue
5648 \pgfkeys{/tikz/external/optimize=false}%
5649 \let\forest@drawtree@beginbox\relax
5650 \let\forest@drawtree@endbox\relax
5651 \else
5652 \forest@tempfalse
5653 \fi
5654 }{%
5655 \forest@tempfalse
5656 }%
5657 \fi
5658 \ifforest@temp
5659 \advance\forest@externalize@inner@n 1
5660 \edef\forest@externalize@filename{%
5661 \tikzexternalrealjob-forest-\forest@externalize@outer@n
5662 \ifnum\forest@externalize@inner@n=0 \else.\the\forest@externalize@inner@n\fi}%
5663 \expandafter\tikzsetnextfilename\expandafter{\forest@externalize@filename}%
5664 \tikzexternalenable
5665 \pgfkeysalso{/tikz/external/remake next,/tikz/external/export next}%
5666 \fi
5667 \ifforest@externalize@tree@
5668 \typeout{forest: Invoking a recursive call to generate the external picture
5669 '\forest@externalize@filename' for the following context+code:
5670 '\expandafter\detokenize\expandafter{\forest@externalize@id}'}%
5671 \fi
5672 \fi
5673 %
5674 \ifforesttikzcshack
5675 \let\forest@original@tikz@parse@node\tikz@parse@node
5676 \let\tikz@parse@node\forest@tikz@parse@node
5677 \fi
5678 \pgfkeysgetvalue{/forest/begin draw/.@cmd}\forest@temp@begindraw

```

```

5679 \pgfkeysgetvalue{/forest/end draw/.@cmd}\forest@temp@enddraw
5680 \edef\forest@marshal{%
5681   \noexpand\forest@drawtree@beginbox
5682   \expandonce{\forest@temp@begindraw\pgfkeysnovalue\pgfeov}%
5683   \noexpand\forest@node@drawtree@
5684   \expandonce{\forest@temp@enddraw\pgfkeysnovalue\pgfeov}%
5685   \noexpand\forest@drawtree@endbox
5686 } \forest@marshal
5687 \ifforesttikzcsack
5688   \let\tikz@parse@node\forest@original@tikz@parse@node
5689 \fi
5690 %
5691 \ifforest@external@
5692   \ifforest@externalize@tree@
5693     \tikzexternaldisable
5694     \eappto\forest@externalize@checkimages{%
5695       \noexpand\forest@includeexternal@check{\forest@externalize@filename}%
5696     }%
5697     \expandafter\ifstrequal\expandafter{\forest@drawtreebox}{\pgfkeysnovalue}{%
5698       \eappto\forest@externalize@loadimages{%
5699         \noexpand\forest@includeexternal{\forest@externalize@filename}%
5700       }%
5701     }{%
5702       \eappto\forest@externalize@loadimages{%
5703         \noexpand\forest@includeexternal@box\forest@drawtreebox{\forest@externalize@filename}%
5704       }%
5705     }%
5706   \fi
5707 \fi
5708 }
5709 \def\forest@node@drawtree@{%
5710   \forest@forthis{\forestset{draw tree method}}%
5711   \forest@node@ifnamedefined{forest@baseline@node}{%
5712     \edef\forest@temp{%
5713       \noexpand\pgfsetbaselinepointlater{%
5714         \noexpand\pgfpointanchor
5715         {\forest@ve{\forest@node@Nametoid{forest@baseline@node}}{name}}
5716         {\forest@ve{\forest@node@Nametoid{forest@baseline@node}}{anchor}}
5717       }%
5718     } \forest@temp
5719   }{}%
5720 }
5721 \def\forest@draw@node{%
5722   \ifnum\forest@ve{phantom}=0
5723     \forest@node@forest@position@node@later@restore
5724     \ifforest@drawtree@preservenodeboxes@
5725       \pgfnodealias{forest@temp}{\forest@ve{later@name}}%
5726     \fi
5727     \pgfpositionnodenow{\pgfpqpoint{\forest@ve{x}}{\forest@ve{y}}}%
5728     \ifforest@drawtree@preservenodeboxes@
5729       \pgfnodealias{\forest@ve{later@name}}{\forest@temp}%
5730     \fi
5731   \fi
5732 }
5733 \def\forest@draw@edge{%
5734   \ifnum\forest@cn=\forest@root\relax\else
5735     \ifnum\forest@ve{phantom}=0
5736       \ifnum\forest@ve{\forest@ve{@parent}}{phantom}=0
5737         \edef\forest@temp{\forest@ve{edge path}}%
5738       \forest@temp
5739     \fi

```

```

5740 \fi
5741 \fi
5742 }
5743 \def\forest@draw@tikz{%
5744 \forest@ve{tikz}%
5745 }

```

7 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.

```

5746 \newdimen\forest@xg
5747 \newdimen\forest@yg
5748 \newdimen\forest@xs
5749 \newdimen\forest@ys
5750 \def\forest@setupgrowline#1{%
5751 \edef\forest@grow{#1}%
5752 \pgfpointpolar\forest@grow{1pt}%
5753 \forest@xg=\pgf@x
5754 \forest@yg=\pgf@y
5755 \forest@xs=-\pgf@y
5756 \forest@ys=\pgf@x
5757 }

```

7.1 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`.

```

5758 \def\forest@pgfpointprojectiontogrowline#1{%
5759 \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.

```

5760 \pgfutil@tempdima=\pgf@sys@tonumber{\pgf@x}\forest@xg%
5761 \advance\pgfutil@tempdima by\pgf@sys@tonumber{\pgf@y}\forest@yg%

```

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

```

5762 \global\pgf@x=\pgf@sys@tonumber{\pgfutil@tempdima}\forest@xg%
5763 \global\pgf@y=\pgf@sys@tonumber{\pgfutil@tempdima}\forest@yg%
5764 }}

```

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.

```

5765 \def\forest@distancetogrowline#1#2{%
5766 \pgf@process{#2}%
5767 #1=\pgf@sys@tonumber{\pgf@x}\forest@xs\relax
5768 \advance#1 by\pgf@sys@tonumber{\pgf@y}\forest@ys\relax
5769 }

```

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

The following macro projects all segment edges (“points”) of a simple² path `#1` onto the grow line. The result is an array of tuples (x_o, y_o, x_p, y_p) , where x_o and y_o stand for the *original* point, and x_p and y_p stand for its *projection*. The prefix of the array is given by `#2`. If the array already exists, the new

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

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`.

```

5770 \let\forest@pp@n\relax
5771 \def\forest@projectpath@togrowline#1#2{%
5772   \edef\forest@pp@prefix{#2}%
5773   \forest@save@pgfsyssoftpath@tokendefs
5774   \let\pgfsyssoftpath@movetotoken\forest@projectpath@processpoint
5775   \let\pgfsyssoftpath@linetotoken\forest@projectpath@processpoint
5776   \c@pgf@counta=0
5777   #1%
5778   \csedef{#2n}{\the\c@pgf@counta}%
5779   \forest@restore@pgfsyssoftpath@tokendefs
5780 }

```

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

```

5781 \def\forest@projectpath@processpoint#1#2{%
5782   \pgfqpoint{#1}{#2}%
5783   \expandafter\edef\c@pgf@counta\forest@pp@prefix\the\c@pgf@counta xo\endcsname{\the\pgf@x}%
5784   \expandafter\edef\c@pgf@counta\forest@pp@prefix\the\c@pgf@counta yo\endcsname{\the\pgf@y}%
5785   \forest@pgfpointprojectiontogrowline{}%
5786   \expandafter\edef\c@pgf@counta\forest@pp@prefix\the\c@pgf@counta xp\endcsname{\the\pgf@x}%
5787   \expandafter\edef\c@pgf@counta\forest@pp@prefix\the\c@pgf@counta yp\endcsname{\the\pgf@y}%
5788   \advance\c@pgf@counta 1\relax
5789 }

```

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

```

5790 \def\forest@sortprojections#1{%
5791   % todo: optimize in cases when we know that the array is actually a
5792   % merger of sorted arrays; when does this happen? in
5793   % distance_between_paths, and when merging the edges of the parent
5794   % and its children in a uniform growth tree
5795   \edef\forest@ppi@inputprefix{#1}%
5796   \c@pgf@counta=\c@pgf@counta\endcsname\relax
5797   \advance\c@pgf@counta -1
5798   \forest@sort\forest@ppiraw@cmp\forest@ppiraw@let\forest@sort@ascending{0}{\the\c@pgf@counta}%
5799 }

```

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

- Array of projections (prefix #2)
 - its items are tuples (x,y) (the array is sorted by x and y), and
 - an inner array of original points (prefix #2N@, where N is the index of the item in array #2. The items of #2N@ are x, y and d: x and y are the coordinates of the original point; d is its distance to the grow line. The inner array is not sorted.
- A dictionary #2: keys are the coordinates (x,y) of the original points; a value is the index of the original point's projection in array #2.³

```

5800 \def\forest@processprojectioninfo#1#2{%
5801   \edef\forest@ppi@inputprefix{#1}%

```

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

```

5802   \c@pgf@counta=0
5803   \c@pgf@countb=-1
5804   \safeloop
5805   \ifnum\c@pgf@counta<\c@pgf@counta\endcsname\relax

```

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

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

```

5806 \letcs\forest@xo{#1\the\c@pgf@counta xo}%
5807 \letcs\forest@yo{#1\the\c@pgf@counta yo}%
5808 \letcs\forest@xp{#1\the\c@pgf@counta xp}%
5809 \letcs\forest@yp{#1\the\c@pgf@counta yp}%
5810 \ifnum\c@pgf@countb<0
5811 \forest@equaltotolerancefalse
5812 \else
5813 \forest@equaltotolerance
5814 {\pgfqpoint\forest@xp\forest@yp}%
5815 {\pgfqpoint
5816 {\csname#2\the\c@pgf@countb x\endcsname}%
5817 {\csname#2\the\c@pgf@countb y\endcsname}%
5818 }%
5819 \fi
5820 \ifforest@equaltotolerance\else

```

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

```

5821 \advance\c@pgf@countb 1
5822 \cslet{#2\the\c@pgf@countb x}\forest@xp
5823 \cslet{#2\the\c@pgf@countb y}\forest@yp
5824 \csdef{#2\the\c@pgf@countb @n}{0}%
5825 \fi

```

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

```

5826 % todo: this is ugly!
5827 \ifdefined\forest@xo\ifx\forest@xo\relax\else
5828 \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.

```

5829 \forest@append@point@to@inner@array
5830 \forest@xo\forest@yo
5831 {#2\the\c@pgf@countb @}%

```

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

```

5832 \csedef{#2(\forest@xo,\forest@yo)}{\the\c@pgf@countb}%
5833 \fi\fi
5834 \fi\fi

```

Clean-up the raw array item.

```

5835 \cslet{#1\the\c@pgf@counta xo}\relax
5836 \cslet{#1\the\c@pgf@counta yo}\relax
5837 \cslet{#1\the\c@pgf@counta xp}\relax
5838 \cslet{#1\the\c@pgf@counta yp}\relax
5839 \advance\c@pgf@counta 1
5840 \saferepeat

```

Clean up the raw array length.

```

5841 \cslet{#1n}\relax

```

Store the length of the outer result array.

```

5842 \advance\c@pgf@countb 1
5843 \csedef{#2n}{\the\c@pgf@countb}%
5844 }

```

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

```

5845 \def\forest@ppiraw@let#1#2{%
5846 \csletcs{\forest@ppi@inputprefix#1xo}{\forest@ppi@inputprefix#2xo}%
5847 \csletcs{\forest@ppi@inputprefix#1yo}{\forest@ppi@inputprefix#2yo}%
5848 \csletcs{\forest@ppi@inputprefix#1xp}{\forest@ppi@inputprefix#2xp}%
5849 \csletcs{\forest@ppi@inputprefix#1yp}{\forest@ppi@inputprefix#2yp}%
5850 }

```

Item comparison macro for quicksorting the raw projection data.

```
5851 \def\forest@ppiraw@cmp#1#2{%
5852   \forest@sort@cmptwodimcs
5853   {\forest@ppi@inputprefix#1xp}{\forest@ppi@inputprefix#1yp}%
5854   {\forest@ppi@inputprefix#2xp}{\forest@ppi@inputprefix#2yp}%
5855 }
```

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

```
5856 \def\forest@append@point@to@inner@array#1#2#3{%
5857   \c@pgf@countc=\csname#3n\endcsname\relax
5858   \csedef{#3\the\c@pgf@countc x}{#1}%
5859   \csedef{#3\the\c@pgf@countc y}{#2}%
5860   \forest@distancetogrowline\pgfutil@tempdima{\pgfqpoint#1#2}%
5861   \csedef{#3\the\c@pgf@countc d}{\the\pgfutil@tempdima}%
5862   \advance\c@pgf@countc 1
5863   \csedef{#3n}{\the\c@pgf@countc}%
5864 }
```

7.2 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.

```
5865 \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.

```
5866   \edef\forest@bp@prefix{#2}%
5867   \forest@save@pgfsyssoftpath@tokendefs
5868   \let\pgfsyssoftpath@movetotoken\forest@breakpath@processfirstpoint
5869   \let\pgfsyssoftpath@linetotoken\forest@breakpath@processfirstpoint
5870   %\pgfusepath{}% empty the current path. ok?
5871   #1%
5872   \forest@restore@pgfsyssoftpath@tokendefs
5873   \pgfsyssoftpath@getcurrentpath#3%
5874 }
```

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.)

```
5875 \def\forest@breakpath@processfirstpoint#1#2{%
5876   \forest@breakpath@processmoveto{#1}{#2}%
5877   \let\pgfsyssoftpath@movetotoken\forest@breakpath@processmoveto
5878   \let\pgfsyssoftpath@linetotoken\forest@breakpath@processlineto
5879 }
```

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.

```
5880 \def\forest@breakpath@processmoveto#1#2{%
5881   \pgfsyssoftpath@moveto{#1}{#2}%
5882   \def\forest@previous@x{#1}%
5883   \def\forest@previous@y{#2}%
5884   \expandafter\let\expandafter\forest@previous@i
5885     \csname\forest@bp@prefix(#1,#2)\endcsname
5886   \expandafter\let\expandafter\forest@previous@px
5887     \csname\forest@bp@prefix\forest@previous@i x\endcsname
```

```

5888 \expandafter\let\expandafter\forest@previous@py
5889 \csname\forest@bp@prefix\forest@previous@i y\endcsname
5890 }

```

This is the heart of the path-breaking procedure.

```

5891 \def\forest@breakpath@processlineto#1#2{%

```

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

```

5892 \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.)

```

5893 \expandafter\let\expandafter\forest@i
5894 \csname\forest@bp@prefix(#1,#2)\endcsname
5895 \expandafter\let\expandafter\forest@px
5896 \csname\forest@bp@prefix\forest@i x\endcsname
5897 \expandafter\let\expandafter\forest@py
5898 \csname\forest@bp@prefix\forest@i y\endcsname

```

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

```

5899 \forest@equaltotolerance
5900 {\pgfqpoint{\forest@previous@px}{\forest@previous@py}}%
5901 {\pgfqpoint{\forest@px}{\forest@py}}%
5902 \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.

```

5903 \let\forest@breakpath@op\pgfsyssoftpath@moveto
5904 \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.

```

5905 \forest@temp@count=\forest@previous@i\relax
5906 \ifnum\forest@previous@i<\forest@i\relax
5907 \def\forest@breakpath@step{1}%
5908 \def\forest@breakpath@test{\forest@temp@count<\forest@i\relax}%
5909 \else
5910 \def\forest@breakpath@step{-1}%
5911 \def\forest@breakpath@test{\forest@temp@count>\forest@i\relax}%
5912 \fi

```

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

```

5913 \safeloop
5914 \advance\forest@temp@count\forest@breakpath@step\relax
5915 \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.)

```

5916 \pgfpointintersectionoflines
5917 {\pgfqpoint
5918 {\csname\forest@bp@prefix\the\forest@temp@count x\endcsname}%
5919 {\csname\forest@bp@prefix\the\forest@temp@count y\endcsname}%
5920 }%
5921 {\pgfpointadd
5922 {\pgfqpoint
5923 {\csname\forest@bp@prefix\the\forest@temp@count x\endcsname}%
5924 {\csname\forest@bp@prefix\the\forest@temp@count y\endcsname}%
5925 }%
5926 {\pgfqpoint{\forest@xs}{\forest@ys}}%
5927 }%
5928 {\pgfqpoint{\forest@previous@x}{\forest@previous@y}}%
5929 {\pgfqpoint{#1}{#2}}%

```

Break the segment at the intersection.

```
5930 \pgfgetlastxy\forest@lastx\forest@lasty
5931 \pgfsyssoftpath@lineto\forest@lastx\forest@lasty
```

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

```
5932 \forest@append@point@to@inner@array
5933 \forest@lastx\forest@lasty
5934 {\forest@bp@prefix\the\forest@temp@count @}%
```

Cache the projection of the new segment edge.

```
5935 \csedef{\forest@bp@prefix(\the\pgf@x,\the\pgf@y)}{\the\forest@temp@count}%
5936 \saferepeat
5937 \fi
```

Add the current point.

```
5938 \forest@breakpath@op{#1}{#2}%
```

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

```
5939 \def\forest@previous@x{#1}%
5940 \def\forest@previous@y{#2}%
5941 \let\forest@previous@i\forest@i
5942 \let\forest@previous@px\forest@px
5943 \let\forest@previous@py\forest@py
5944 }
```

7.3 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.

```
5945 \def\forest@getnegativetightedgeofpath#1#2{%
5946 \forest@get@onetightedgeofpath#1\forest@sort@ascending#2}
5947 \def\forest@getpositivetightedgeofpath#1#2{%
5948 \forest@get@onetightedgeofpath#1\forest@sort@descending#2}
5949 \def\forest@get@onetightedgeofpath#1#2#3{%
5950 {%
5951 \forest@get@one@tightedgeofpath#1#2\forest@gep@edge
5952 \global\let\forest@gep@global@edge\forest@gep@edge
5953 }%
5954 \let#3\forest@gep@global@edge
5955 }
5956 \def\forest@get@one@tightedgeofpath#1#2#3{%
```

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

```
5957 \forest@projectpathtogrowline#1\forest@pp@}%
5958 \forest@sortprojections\forest@pp@}%
5959 \forest@processprojectioninfo\forest@pp@}\forest@pi@}%
```

Break the path.

```
5960 \forest@breakpath#1\forest@pi@\forest@brokenpath
```

⁴For the definition of negative/positive side, see `forest@distancetogrowline` in §7.1

Compile some more useful information.

```
5961 \forest@sort@inner@arrays{forest@pi@}#2%
5962 \forest@pathtodict\forest@brokenpath{forest@pi@}%
```

The auxiliary data is set up: do the work!

```
5963 \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).

```
5964 \forest@simplifypath\forest@edge#3%
5965 }
```

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

```
5966 \def\forest@getbothtightededgesofpath#1#2#3{%
5967   {%
5968     \forest@get@one@tightedgeofpath#1\forest@sort@ascending\forest@gep@firstedge
```

Reverse the order of items in the inner arrays.

```
5969     \c@pgf@counta=0
5970     \forest@loop
5971     \ifnum\c@pgf@counta<\forest@pi@n\relax
5972       \forest@ppi@deflet{forest@pi@\the\c@pgf@counta @}%
5973       \forest@reversearray\forest@ppi@let
5974       {0}%
5975       {\csname forest@pi@\the\c@pgf@counta @n\endcsname}%
5976       \advance\c@pgf@counta 1
5977     \forest@repeat
```

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

```
5978     \forest@gettightedgeofpath@getedge\forest@edge
5979     \forest@simplifypath\forest@edge\forest@gep@secondedge
```

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

```
5980     \global\let\forest@gep@global@firstedge\forest@gep@firstedge
5981     \global\let\forest@gep@global@secondedge\forest@gep@secondedge
5982   }%
5983   \let#2\forest@gep@global@firstedge
5984   \let#3\forest@gep@global@secondedge
5985 }
```

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

```
5986 \def\forest@sort@inner@arrays#1#2{%
5987   \c@pgf@counta=0
5988   \safeloop
5989   \ifnum\c@pgf@counta<\csname#1n\endcsname
5990     \c@pgf@countb=\csname#1\the\c@pgf@counta @n\endcsname\relax
5991     \ifnum\c@pgf@countb>1
5992       \advance\c@pgf@countb -1
5993       \forest@ppi@deflet{#1\the\c@pgf@counta @}%
5994       \forest@ppi@defcmp{#1\the\c@pgf@counta @}%
5995       \forest@sort\forest@ppi@cmp\forest@ppi@let#2{0}{\the\c@pgf@countb}%
5996     \fi
5997     \advance\c@pgf@counta 1
5998   \saferepeat
5999 }
```

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.

```
6000 \def\forest@ppi@deflet#1{%
6001   \edef\forest@ppi@let##1##2{%
6002     \noexpand\csletcs{#1##1x}{#1##2x}%
6003     \noexpand\csletcs{#1##1y}{#1##2y}%
```

```

6004 \noexpand\csletcs{#1##1d}{#1##2d}%
6005 }%
6006 }

```

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.

```

6007 \def\forest@ppi@defcmp#1{%
6008 \edef\forest@ppi@cmp##1##2{%
6009 \noexpand\forest@sort@cmpdimcs{#1##1d}{#1##2d}%
6010 }%
6011 }

```

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).

```

6012 \let\forest@inpath\advance

```

This macro is just a wrapper to process the path.

```

6013 \def\forest@pathtodict#1#2{%
6014 \edef\forest@pathtodict@prefix{#2}%
6015 \forest@save@pgfsyssoftpath@tokendefs
6016 \let\pgfsyssoftpath@movetotoken\forest@pathtodict@movetoop
6017 \let\pgfsyssoftpath@linetotoken\forest@pathtodict@linetoop
6018 \def\forest@pathtodict@subpathstart{}%
6019 #1%
6020 \forest@restore@pgfsyssoftpath@tokendefs
6021 }

```

When a move-to operation is encountered:

```

6022 \def\forest@pathtodict@movetoop#1#2{%

```

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

```

6023 \def\forest@pathtodict@subpathstart{(#1,#2)-}%
6024 }

```

When a line-to operation is encountered:

```

6025 \def\forest@pathtodict@linetoop#1#2{%

```

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

```

6026 \if\relax\forest@pathtodict@subpathstart\relax\else
6027 \let\forest@pathtodict@from\forest@pathtodict@subpathstart
6028 \fi

```

Mark the segment as existing.

```

6029 \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.

```

6030 \def\forest@pathtodict@from{(#1,#2)-}%
6031 \def\forest@pathtodict@subpathstart{}%
6032 }

```

In this macro, the edge is actually computed.

```

6033 \def\forest@gettightedgeofpath@getedge#1{% cs to store the edge into

```

Clear the path and the last projection.

```

6034 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
6035 \let\forest@last@x\relax
6036 \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.)

```

6037 \c@pgf@counta=0
6038 \forest@temp@count=\forest@pi@n\relax
6039 \advance\forest@temp@count -1

```

```

6040 \edef\forest@nminusone{\the\forest@temp@count}%
6041 \safeloop
6042 \ifnum\c@pgf@counta<\forest@nminusone\relax
6043   \forest@gettightedgeofpath@getedge@loopa
6044 \saferepeat

```

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

```

6045 \ifnum\forest@nminusone<\forest@n\relax\else
6046   \ifnum\csname forest@pi@\forest@nminusone @n\endcsname>0
6047     \forest@gettightedgeofpath@maybemoveto{\forest@nminusone}{0}%
6048   \fi
6049 \fi
6050 \pgfsyssoftpath@getcurrentpath#1%
6051 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
6052 }

```

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.

```

6053 \def\forest@gettightedgeofpath@getedge@loopa{%
6054   \ifnum\csname forest@pi@\the\c@pgf@counta @n\endcsname>0

```

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

```

6055   \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.

```

6056   \c@pgf@countb=0
6057   \safeloop
6058   \ifnum\c@pgf@countb<\csname forest@pi@\the\c@pgf@counta @n\endcsname\relax
6059     \forest@gettightedgeofpath@getedge@loopb
6060   \saferepeat
6061 \fi
6062 \advance\c@pgf@counta 1
6063 }

```

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

```

6064 \def\forest@gettightedgeofpath@getedge@loopb{%
6065   \c@pgf@countc=0
6066   \advance\c@pgf@counta 1
6067   \edef\forest@aplusone{\the\c@pgf@counta}%
6068   \advance\c@pgf@counta -1
6069   \safeloop
6070   \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!

```

6071   \forest@tempfalse
6072   \expandafter\ifx\csname forest@pi@(%
6073     \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb x\endcsname,%
6074     \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb y\endcsname)--(%
6075     \csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname,%
6076     \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname)%
6077   \endcsname\forest@inpath

```

```

6078         \forest@temptrue
6079     \else
6080         \expandafter\ifx\csname forest@pi@(%
6081             \csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname,%
6082             \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname)--(%
6083             \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb x\endcsname,%
6084             \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb y\endcsname)%
6085             \endcsname\forest@inpath
6086         \forest@temptrue
6087     \fi
6088 \fi
6089 \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.

```

6090     \forest@gettightedgeofpath@maybemoveto{\the\c@pgf@counta}{\the\c@pgf@countb}%

```

Second, create a line to the end point.

```

6091     \edef\forest@last@x{%
6092         \csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname}%
6093     \edef\forest@last@y{%
6094         \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname}%
6095     \pgfsyssoftpath@lineto\forest@last@x\forest@last@y

```

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

```

6096         \c@pgf@countc=\csname forest@pi@\forest@aplusone @n\endcsname
6097         \c@pgf@countb=\csname forest@pi@\the\c@pgf@counta @n\endcsname
6098     \fi
6099     \advance\c@pgf@countc 1
6100     \saferepeat
6101     \advance\c@pgf@countb 1
6102 }

```

`\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.

```

6103 \def\forest@gettightedgeofpath@maybemoveto#1#2{%
6104     \forest@temptrue
6105     \ifx\forest@last@x\relax\else
6106         \ifdim\forest@last@x=\csname forest@pi@#1@#2x\endcsname\relax
6107         \ifdim\forest@last@y=\csname forest@pi@#1@#2y\endcsname\relax
6108             \forest@tempfalse
6109         \fi
6110     \fi
6111 \fi
6112 \ifforest@temp
6113     \edef\forest@last@x{\csname forest@pi@#1@#2x\endcsname}%
6114     \edef\forest@last@y{\csname forest@pi@#1@#2y\endcsname}%
6115     \pgfsyssoftpath@moveto\forest@last@x\forest@last@y
6116 \fi
6117 }

```

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

```

6118 \def\forest@simplifypath#1#2{%
6119     \pgfsyssoftpath@setcurrentpath\pgfutil@empty
6120     \forest@save\pgfsyssoftpath@tokendefs
6121     \let\pgfsyssoftpath@movetotoken\forest@simplifypath@moveto
6122     \let\pgfsyssoftpath@linetotoken\forest@simplifypath@lineto

```



```

6123 \let\forest@last@x\relax
6124 \let\forest@last@y\relax
6125 \let\forest@last@atan\relax
6126 #1%
6127 \ifx\forest@last@x\relax\else
6128   \ifx\forest@last@atan\relax\else
6129     \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
6130   \fi
6131 \fi
6132 \forest@restore@pgfsyssoftpath@tokendef
6133 \pgfsyssoftpath@getcurrentpath#2%
6134 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
6135 }

```

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.

```

6136 \def\forest@simplifypath@moveto#1#2{%
6137   \ifx\forest@last@x\relax\else
6138     \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
6139   \fi
6140   \pgfsyssoftpath@moveto{#1}{#2}%
6141   \def\forest@last@x{#1}%
6142   \def\forest@last@y{#2}%
6143   \let\forest@last@atan\relax
6144 }

```

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

```

6145 \def\forest@getedgeofpath@precision{1pt}

```

When a line-to is encountered...

```

6146 \def\forest@simplifypath@lineto#1#2{%
6147   \ifx\forest@last@x\relax

```

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

```

6148   \def\forest@last@x{#1}%
6149   \def\forest@last@y{#2}%
6150   \let\forest@last@atan\relax
6151 \else

```

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

```

6152   \pgfpointdiff{\pgfpoint{#1}{#2}}{\pgfpoint{\forest@last@x}{\forest@last@y}}%
6153   \ifdim\pgf@x<\pgfintersectiontolerance
6154     \ifdim-\pgf@x<\pgfintersectiontolerance
6155       \pgf@x=0pt
6156     \fi
6157   \fi
6158   \csname pgfmathatan2\endcsname{\pgf@x}{\pgf@y}%
6159   \let\forest@current@atan\pgfmathresult
6160   \ifx\forest@last@atan\relax

```

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

```

6161     \def\forest@last@x{#1}%
6162     \def\forest@last@y{#2}%
6163     \let\forest@last@atan\forest@current@atan
6164   \else

```

Otherwise, compare the first and the current slope.

```

6165     \pgfutil@tempdima=\forest@current@atan pt
6166     \advance\pgfutil@tempdima -\forest@last@atan pt
6167     \ifdim\pgfutil@tempdima<0pt\relax
6168       \multiply\pgfutil@tempdima -1

```

```

6169     \fi
6170     \ifdim\pgfutil@tempdima<\forest@getedgeofpath@precision\relax
6171     \else

```

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

```

6172         \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
6173         \let\forest@last@atan\forest@current@atan
6174     \fi

```

In any event, update the last point.

```

6175         \def\forest@last@x{#1}%
6176         \def\forest@last@y{#2}%
6177     \fi
6178 \fi
6179 }

```

7.4 Get rectangle/band edge

```

6180 \def\forest@getnegativerectangleedgeofpath#1#2{%
6181   \forest@getnegativerectangleorbandedgeofpath{#1}{#2}{\the\pgf@xb}}
6182 \def\forest@getpositiverectangleedgeofpath#1#2{%
6183   \forest@getpositiverectangleorbandedgeofpath{#1}{#2}{\the\pgf@xb}}
6184 \def\forest@getbothrectangleedgesofpath#1#2#3{%
6185   \forest@getbothrectangleorbandedgesofpath{#1}{#2}{#3}{\the\pgf@xb}}
6186 \def\forest@bandlength{5000pt} % something large (ca. 180cm), but still manageable for TeX without producing
6187 \def\forest@getnegativerectangleorbandedgeofpath#1#2{%
6188   \forest@getnegativerectangleorbandedgeofpath{#1}{#2}{\forest@bandlength}}
6189 \def\forest@getpositiverectangleorbandedgeofpath#1#2{%
6190   \forest@getpositiverectangleorbandedgeofpath{#1}{#2}{\forest@bandlength}}
6191 \def\forest@getbothbandedgesofpath#1#2#3{%
6192   \forest@getbothrectangleorbandedgesofpath{#1}{#2}{#3}{\forest@bandlength}}
6193 \def\forest@getnegativerectangleorbandedgeofpath#1#2#3{%
6194   \forest@path@getboundingrectangle@ls#1{\forest@grow}%
6195   \edef\forest@gre@path{%
6196     \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@ya}%
6197     \noexpand\pgfsyssoftpath@linetotoken{#3}{\the\pgf@ya}%
6198   }%
6199   {%
6200     \pgftransformreset
6201     \pgftransformrotate{\forest@grow}%
6202     \forest@pgfpathtransformed\forest@gre@path
6203   }%
6204   \pgfsyssoftpath@getcurrentpath#2%
6205 }
6206 \def\forest@getpositiverectangleorbandedgeofpath#1#2#3{%
6207   \forest@path@getboundingrectangle@ls#1{\forest@grow}%
6208   \edef\forest@gre@path{%
6209     \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@yb}%
6210     \noexpand\pgfsyssoftpath@linetotoken{#3}{\the\pgf@yb}%
6211   }%
6212   {%
6213     \pgftransformreset
6214     \pgftransformrotate{\forest@grow}%
6215     \forest@pgfpathtransformed\forest@gre@path
6216   }%
6217   \pgfsyssoftpath@getcurrentpath#2%
6218 }
6219 \def\forest@getbothrectangleorbandedgesofpath#1#2#3#4{%
6220   \forest@path@getboundingrectangle@ls#1{\forest@grow}%
6221   \edef\forest@gre@negpath{%
6222     \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@ya}%

```

```

6223 \noexpand\pgfsyssoftpath@linetotoken{#4}{\the\pgf@ya}%
6224 }%
6225 \edef\forest@gre@pospath{%
6226 \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@yb}%
6227 \noexpand\pgfsyssoftpath@linetotoken{#4}{\the\pgf@yb}%
6228 }%
6229 {%
6230 \pgftransformreset
6231 \pgftransformrotate{\forest@grow}%
6232 \forest@pgfpathtransformed\forest@gre@negpath
6233 }%
6234 \pgfsyssoftpath@getcurrentpath#2%
6235 {%
6236 \pgftransformreset
6237 \pgftransformrotate{\forest@grow}%
6238 \forest@pgfpathtransformed\forest@gre@pospath
6239 }%
6240 \pgfsyssoftpath@getcurrentpath#3%
6241 }

```

7.5 Distance between paths

Another crucial part of the package.

```

6242 \def\forest@distance@between@edge@paths#1#2#3{%
6243 % #1, #2 = (edge) paths
6244 %
6245 % project paths
6246 \forest@projectpathtogrowline#1{\forest@p1@}%
6247 \forest@projectpathtogrowline#2{\forest@p2@}%
6248 % merge projections (the lists are sorted already, because edge
6249 % paths are |sorted|)
6250 \forest@dbep@mergeprojections
6251 {\forest@p1@}{\forest@p2@}%
6252 {\forest@P1@}{\forest@P2@}%
6253 % process projections
6254 \forest@processprojectioninfo{\forest@P1@}{\forest@PI1@}%
6255 \forest@processprojectioninfo{\forest@P2@}{\forest@PI2@}%
6256 % break paths
6257 \forest@breakpath#1{\forest@PI1@}\forest@broken@one
6258 \forest@breakpath#2{\forest@PI2@}\forest@broken@two
6259 % sort inner arrays ---optimize: it's enough to find max and min
6260 \forest@sort@inner@arrays{\forest@PI1@}\forest@sort@descending
6261 \forest@sort@inner@arrays{\forest@PI2@}\forest@sort@ascending
6262 % compute the distance
6263 \let\forest@distance\relax
6264 \c@pgf@countc=0
6265 \forest@loop
6266 \ifnum\c@pgf@countc<\csname forest@PI1@n\endcsname\relax
6267 \ifnum\csname forest@PI1@the\c@pgf@countc @n\endcsname=0 \else
6268 \ifnum\csname forest@PI2@the\c@pgf@countc @n\endcsname=0 \else
6269 \pgfutil@tempdima=\csname forest@PI2@the\c@pgf@countc @0d\endcsname\relax
6270 \advance\pgfutil@tempdima -\csname forest@PI1@the\c@pgf@countc @0d\endcsname\relax
6271 \ifx\forest@distance\relax
6272 \edef\forest@distance{\the\pgfutil@tempdima}%
6273 \else
6274 \ifdim\pgfutil@tempdima<\forest@distance\relax
6275 \edef\forest@distance{\the\pgfutil@tempdima}%
6276 \fi
6277 \fi
6278 \fi
6279 \fi

```

```

6280 \advance\c@pgf@countc 1
6281 \forest@repeat
6282 \let#3\forest@distance
6283 }
6284 % merge projections: we need two projection arrays, both containing
6285 % projection points from both paths, but each with the original
6286 % points from only one path
6287 \def\forest@dbep@mergeprojections#1#2#3#4{%
6288 % TODO: optimize: v bistvu ni treba sortirat, ker je edge path e sortiran
6289 \forest@sortprojections{#1}%
6290 \forest@sortprojections{#2}%
6291 \c@pgf@counta=0
6292 \c@pgf@countb=0
6293 \c@pgf@countc=0
6294 \edef\forest@input@prefix@one{#1}%
6295 \edef\forest@input@prefix@two{#2}%
6296 \edef\forest@output@prefix@one{#3}%
6297 \edef\forest@output@prefix@two{#4}%
6298 \forest@dbep@mp@iterate
6299 \csedef{#3n}{\the\c@pgf@countc}%
6300 \csedef{#4n}{\the\c@pgf@countc}%
6301 }
6302 \def\forest@dbep@mp@iterate{%
6303 \let\forest@dbep@mp@next\forest@dbep@mp@iterate
6304 \ifnum\c@pgf@counta<\csname\forest@input@prefix@one n\endcsname\relax
6305 \ifnum\c@pgf@countb<\csname\forest@input@prefix@two n\endcsname\relax
6306 \let\forest@dbep@mp@next\forest@dbep@mp@do
6307 \else
6308 \let\forest@dbep@mp@next\forest@dbep@mp@iteratefirst
6309 \fi
6310 \else
6311 \ifnum\c@pgf@countb<\csname\forest@input@prefix@two n\endcsname\relax
6312 \let\forest@dbep@mp@next\forest@dbep@mp@iteratesecond
6313 \else
6314 \let\forest@dbep@mp@next\relax
6315 \fi
6316 \fi
6317 \forest@dbep@mp@next
6318 }
6319 \def\forest@dbep@mp@do{%
6320 \forest@sort@cmptwodimcs%
6321 {\forest@input@prefix@one\the\c@pgf@counta xp}%
6322 {\forest@input@prefix@one\the\c@pgf@counta yp}%
6323 {\forest@input@prefix@two\the\c@pgf@countb xp}%
6324 {\forest@input@prefix@two\the\c@pgf@countb yp}%
6325 \if\forest@sort@cmp@result=%
6326 \forest@dbep@mp@@store@p\forest@input@prefix@one\c@pgf@counta
6327 \forest@dbep@mp@@store@o\forest@input@prefix@one
6328 \c@pgf@counta\forest@output@prefix@one
6329 \forest@dbep@mp@@store@o\forest@input@prefix@two
6330 \c@pgf@countb\forest@output@prefix@two
6331 \advance\c@pgf@counta 1
6332 \advance\c@pgf@countb 1
6333 \else
6334 \if\forest@sort@cmp@result>%
6335 \forest@dbep@mp@@store@p\forest@input@prefix@two\c@pgf@countb
6336 \forest@dbep@mp@@store@o\forest@input@prefix@two
6337 \c@pgf@countb\forest@output@prefix@two
6338 \advance\c@pgf@countb 1
6339 \else%<
6340 \forest@dbep@mp@@store@p\forest@input@prefix@one\c@pgf@counta

```

```

6341 \forest@dbep@mp@@store@o\forest@input@prefix@one
6342 \c@pgf@counta\forest@output@prefix@one
6343 \advance\c@pgf@counta 1
6344 \fi
6345 \fi
6346 \advance\c@pgf@countc 1
6347 \forest@dbep@mp@iterate
6348 }
6349 \def\forest@dbep@mp@@store@p#1#2{%
6350 \csletcs
6351 {\forest@output@prefix@one\the\c@pgf@countc xp}%
6352 {#1\the#2xp}%
6353 \csletcs
6354 {\forest@output@prefix@one\the\c@pgf@countc yp}%
6355 {#1\the#2yp}%
6356 \csletcs
6357 {\forest@output@prefix@two\the\c@pgf@countc xp}%
6358 {#1\the#2xp}%
6359 \csletcs
6360 {\forest@output@prefix@two\the\c@pgf@countc yp}%
6361 {#1\the#2yp}%
6362 }
6363 \def\forest@dbep@mp@@store@o#1#2#3{%
6364 \csletcs{#3\the\c@pgf@countc xo}{#1\the#2xo}%
6365 \csletcs{#3\the\c@pgf@countc yo}{#1\the#2yo}%
6366 }
6367 \def\forest@dbep@mp@iteratefirst{%
6368 \forest@dbep@mp@iterateone\forest@input@prefix@one\c@pgf@counta\forest@output@prefix@one
6369 }
6370 \def\forest@dbep@mp@iteratesecond{%
6371 \forest@dbep@mp@iterateone\forest@input@prefix@two\c@pgf@countb\forest@output@prefix@two
6372 }
6373 \def\forest@dbep@mp@iterateone#1#2#3{%
6374 \forest@loop
6375 \ifnum#2<\csname#1n\endcsname\relax
6376 \forest@dbep@mp@@store@p#1#2%
6377 \forest@dbep@mp@@store@o#1#2#3%
6378 \advance\c@pgf@countc 1
6379 \advance#21
6380 \forest@repeat
6381 }

```

7.6 Utilities

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

```

6382 \newif\ifforest@equaltotolerance
6383 \def\forest@equaltotolerance#1#2{%
6384 \pgfpointdiff{#1}{#2}%
6385 \ifdim\pgf@x<0pt \multiply\pgf@x -1 \fi
6386 \ifdim\pgf@y<0pt \multiply\pgf@y -1 \fi
6387 \global\forest@equaltotolerancefalse
6388 \ifdim\pgf@x<\pgfintersectiontolerance\relax
6389 \ifdim\pgf@y<\pgfintersectiontolerance\relax
6390 \global\forest@equaltolerancetrue
6391 \fi
6392 \fi
6393 }}

```

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

```

6394 \def\forest@save@pgfsyssoftpath@tokendefs{%
6395   \let\forest@origmovetotoken\pgfsyssoftpath@movetotoken
6396   \let\forest@origlinetotoken\pgfsyssoftpath@linetotoken
6397   \let\forest@origcurvetosupportatoken\pgfsyssoftpath@curvetosupportatoken
6398   \let\forest@origcurvetosupportbtoken\pgfsyssoftpath@curvetosupportbtoken
6399   \let\forest@origcurvetotoken\pgfsyssoftpath@curvetototoken
6400   \let\forest@origrectcornertoken\pgfsyssoftpath@rectcornertoken
6401   \let\forest@origrectsizetoken\pgfsyssoftpath@rectsizetoken
6402   \let\forest@origclosepathtoken\pgfsyssoftpath@closepathtoken
6403   \let\pgfsyssoftpath@movetotoken\forest@badtoken
6404   \let\pgfsyssoftpath@linetotoken\forest@badtoken
6405   \let\pgfsyssoftpath@curvetosupportatoken\forest@badtoken
6406   \let\pgfsyssoftpath@curvetosupportbtoken\forest@badtoken
6407   \let\pgfsyssoftpath@curvetototoken\forest@badtoken
6408   \let\pgfsyssoftpath@rectcornertoken\forest@badtoken
6409   \let\pgfsyssoftpath@rectsizetoken\forest@badtoken
6410   \let\pgfsyssoftpath@closepathtoken\forest@badtoken
6411 }
6412 \def\forest@badtoken{%
6413   \PackageError{forest}{This token should not be in this path}{}%
6414 }
6415 \def\forest@restore@pgfsyssoftpath@tokendefs{%
6416   \let\pgfsyssoftpath@movetotoken\forest@origmovetotoken
6417   \let\pgfsyssoftpath@linetotoken\forest@origlinetotoken
6418   \let\pgfsyssoftpath@curvetosupportatoken\forest@origcurvetosupportatoken
6419   \let\pgfsyssoftpath@curvetosupportbtoken\forest@origcurvetosupportbtoken
6420   \let\pgfsyssoftpath@curvetototoken\forest@origcurvetotoken
6421   \let\pgfsyssoftpath@rectcornertoken\forest@origrectcornertoken
6422   \let\pgfsyssoftpath@rectsizetoken\forest@origrectsizetoken
6423   \let\pgfsyssoftpath@closepathtoken\forest@origclosepathtoken
6424 }

```

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

```

6425 \def\forest@extendpath#1#2#3{%
6426   \pgf@process{#3}%
6427   \pgfsyssoftpath@setcurrentpath#1%
6428   \forest@save@pgfsyssoftpath@tokendefs
6429   \let\pgfsyssoftpath@movetotoken\forest@extendpath@moveto
6430   \let\pgfsyssoftpath@linetotoken\forest@extendpath@lineto
6431   #2%
6432   \forest@restore@pgfsyssoftpath@tokendefs
6433   \pgfsyssoftpath@getcurrentpath#1%
6434 }
6435 \def\forest@extendpath@moveto#1#2{%
6436   \forest@extendpath@do{#1}{#2}\pgfsyssoftpath@moveto
6437 }
6438 \def\forest@extendpath@lineto#1#2{%
6439   \forest@extendpath@do{#1}{#2}\pgfsyssoftpath@lineto
6440 }
6441 \def\forest@extendpath@do#1#2#3{%
6442   {%
6443     \advance\pgf@x #1
6444     \advance\pgf@y #2
6445     #3{\the\pgf@x}{\the\pgf@y}%
6446   }%
6447 }

```

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.)

```

6448 \def\forest@path@getboundingrectangle@ls#1#2{%
6449   {%

```

```

6450 \pgftransformreset
6451 \pgftransformrotate{-(#2)}%
6452 \forest@pgfpathtransformed#1%
6453 }%
6454 \pgfsyssoftpath@getcurrentpath\forest@gbr@rotatedpath
6455 \forest@path@getboundingrectangle@xy\forest@gbr@rotatedpath
6456 }
6457 \def\forest@path@getboundingrectangle@xy#1{%
6458 \forest@save@pgfsyssoftpath@tokendefs
6459 \let\pgfsyssoftpath@movetotoken\forest@gbr@firstpoint
6460 \let\pgfsyssoftpath@linetotoken\forest@gbr@firstpoint
6461 #1%
6462 \forest@restore@pgfsyssoftpath@tokendefs
6463 }
6464 \def\forest@gbr@firstpoint#1#2{%
6465 \pgf@xa=#1 \pgf@xb=#1 \pgf@ya=#2 \pgf@yb=#2
6466 \let\pgfsyssoftpath@movetotoken\forest@gbr@point
6467 \let\pgfsyssoftpath@linetotoken\forest@gbr@point
6468 }
6469 \def\forest@gbr@point#1#2{%
6470 \ifdim#1<\pgf@xa\relax\pgf@xa=#1 \fi
6471 \ifdim#1>\pgf@xb\relax\pgf@xb=#1 \fi
6472 \ifdim#2<\pgf@ya\relax\pgf@ya=#2 \fi
6473 \ifdim#2>\pgf@yb\relax\pgf@yb=#2 \fi
6474 }

```

8 The outer UI

8.1 Externalization

```

6475 \pgfkeys{/forest/external/.cd,
6476 copy command/.initial={cp "\source" "\target"},
6477 optimize/.is if=forest@external@optimize@,
6478 context/.initial={%
6479 \forest@Ove{\csname forest@id@of@standard node\endcsname}{environment@formula}},
6480 depends on macro/.style={context/.append/.expanded={%
6481 \expandafter\detokenize\expandafter{#1}}},
6482 }
6483 \def\forest@external@copy#1#2{%
6484 \pgfkeysgetvalue{/forest/external/copy command}\forest@copy@command
6485 \ifx\forest@copy@command\pgfkeysnovalue\else
6486 \IfFileExists{#1}{%
6487 {%
6488 \def\source{#1}%
6489 \def\target{#2}%
6490 \immediate\write18{\forest@copy@command}%
6491 }%
6492 }{}%
6493 \fi
6494 }
6495 \newif\ifforest@external@optimize@
6496 \forest@external@optimize@true
6497 \ifforest@install@keys@to@tikz@path@
6498 \tikzset{
6499 fit to/.style={
6500 /forest/for nodewalk=%
6501 {TeX={\def\forest@fitto{}}},#1}%
6502 {TeX={\eappto\forest@fitto{(\forest@name)}}}},
6503 fit/.expanded={\forest@fitto}
6504 },

```

```

6505 }
6506 \fi
6507 \ifforest@external@
6508 \ifdefined\tikzexternal@tikz@replacement\else
6509 \usetikzlibrary{external}%
6510 \fi
6511 \pgfkeys{%
6512   /tikz/external/failed ref warnings for={},
6513   /pgf/images/aux in dpth=false,
6514 }%
6515 \tikzifexternalizing{}{%
6516   \forest@external@copy{\jobname.aux}{\jobname.aux.copy}%
6517 }%
6518 \AtBeginDocument{%
6519   \tikzifexternalizing{%
6520     \IfFileExists{\tikzexternalrealjob.aux.copy}{%
6521       \makeatletter
6522       \input \tikzexternalrealjob.aux.copy
6523       \makeatother
6524     }{}%
6525   }{%
6526     \newwrite\forest@auxout
6527     \immediate\openout\forest@auxout=\tikzexternalrealjob.for.tmp
6528   }%
6529   \IfFileExists{\tikzexternalrealjob.for}{%
6530     {%
6531       \makehashother\makeatletter
6532       \input \tikzexternalrealjob.for
6533     }%
6534   }{}%
6535 }%
6536 \AtEndDocument{%
6537   \tikzifexternalizing{}{%
6538     \immediate\closeout\forest@auxout
6539     \forest@external@copy{\jobname.for.tmp}{\jobname.for}%
6540   }%
6541 }%
6542 \fi

```

8.2 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.

```

6543 \NewDocumentEnvironment{forest}{D()}{stages}}{%
6544   \forest@defstages{#1}%
6545   \Collect@Body
6546   \forest@env
6547 }{}
6548 \NewDocumentCommand{\Forest}{s D()}{stages} m}{%
6549   \forest@defstages{#2}%
6550   \IfBooleanTF{#1}{\let\forest@next\forest@env}{\let\forest@next\forest@group@env}%
6551   \forest@next{#3}%
6552 }
6553 \def\forest@defstages#1{%
6554   \def\forest@stages{#1}%
6555 }
6556 \def\forest@group@env#1{{\forest@env{#1}}}
6557 \newif\ifforest@externalize@tree@
6558 \newif\ifforest@was@tikzexternalwasenable
6559 \newcommand\forest@env[1]{%

```



```

6560 \let\forest@external@next\forest@begin
6561 \forest@was@tikzexternalwasenablefalse
6562 \ifdefined\tikzexternal@tikz@replacement
6563 \ifx\tikz\tikzexternal@tikz@replacement
6564 \forest@was@tikzexternalwasenabletrue
6565 \tikzexternaldisable
6566 \fi
6567 \fi
6568 \forest@externalize@tree@false
6569 \ifforest@external@
6570 \ifforest@was@tikzexternalwasenable
6571 \forest@env@
6572 \fi
6573 \fi
6574 \forest@standardnode@calibrate
6575 \forest@external@next{#1}%
6576 }
6577 \def\forest@env@{%
6578 \iftikzexternalexportnext
6579 \tikzifexternalizing{%
6580 \let\forest@external@next\forest@begin@externalizing
6581 }{%
6582 \let\forest@external@next\forest@begin@externalize
6583 }%
6584 \else
6585 \tikzexternalexportnexttrue
6586 \fi
6587 }

```

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

```

6588 \long\def\forest@begin@externalizing#1{%
6589 \forest@external@setup{#1}%
6590 \let\forest@external@next\forest@begin
6591 \forest@externalize@inner@n=-1
6592 \ifforest@external@optimize@\forest@externalizing@maybeoptimize\fi
6593 \forest@external@next{#1}%
6594 \tikzexternalenable
6595 }
6596 \def\forest@externalizing@maybeoptimize{%
6597 \edef\forest@temp{\tikzexternalrealjob-forest-\forest@externalize@outer@n}%
6598 \edef\forest@marshal{%
6599 \noexpand\pgfutil@in@
6600 {\expandafter\detokenize\expandafter{\forest@temp}.}
6601 {\expandafter\detokenize\expandafter{\pgfactualjobname}.}%
6602 }\forest@marshal
6603 \ifpgfutil@in@
6604 \else
6605 \let\forest@external@next@gobble
6606 \fi
6607 }

```

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

```

6608 \long\def\forest@begin@externalize#1{%
6609 \forest@external@setup{#1}%
6610 \iftikzexternal@file@isuptodate
6611 \setbox0=\hbox{%
6612 \csname forest@externalcheck@\forest@externalize@outer@n\endcsname
6613 }%
6614 \fi
6615 \iftikzexternal@file@isuptodate
6616 \csname forest@externalload@\forest@externalize@outer@n\endcsname

```

```

6617 \else
6618   \forest@externalize@tree@true
6619   \forest@externalize@inner@n=-1
6620   \forest@begin{#1}%
6621   \ifcsdef{forest@externalize@@\forest@externalize@id}{\{%
6622     \immediate\write\forest@auxout{%
6623       \noexpand\forest@external
6624       {\forest@externalize@outer@n}%
6625       {\expandafter\detokenize\expandafter{\forest@externalize@id}}%
6626       {\expandonce\forest@externalize@checkimages}%
6627       {\expandonce\forest@externalize@loadimages}%
6628     }%
6629   }%
6630 \fi
6631 \tikzexternalenable
6632 }
6633 \def\forest@includeexternal@check#1{%
6634   \tikzsetnextfilename{#1}%
6635   \IfFileExists{\tikzexternal@filenameprefix/#1}{\tikzexternal@file@isuptodatetrue}{\tikzexternal@file@isuptodatefalse}
6636 }
6637 \def\makehashother{\catcode'\# =12}%
6638 \long\def\forest@external@setup#1{%
6639   % set up \forest@externalize@id and \forest@externalize@outer@n
6640   % we need to deal with #s correctly (\write doubles them)
6641   \setbox0=\hbox{\makehashother\makeatletter
6642     \scantokens{\forest@temp@toks{#1}}\expandafter
6643   }%
6644   \expandafter\forest@temp@toks\expandafter{\the\forest@temp@toks}%
6645   \edef\forest@temp{\pgfkeysvalueof{/forest/external/context}}%
6646   \edef\forest@externalize@id{%
6647     \expandafter\detokenize\expandafter{\forest@temp}%
6648     @@%
6649     \expandafter\detokenize\expandafter{\the\forest@temp@toks}%
6650   }%
6651   \letcs\forest@externalize@outer@n\forest@externalize@@\forest@externalize@id}%
6652   \ifdefined\forest@externalize@outer@n
6653     \global\tikzexternal@file@isuptodatetrue
6654   \else
6655     \global\advance\forest@externalize@max@outer@n 1
6656     \edef\forest@externalize@outer@n{\the\forest@externalize@max@outer@n}%
6657     \global\tikzexternal@file@isuptodatefalse
6658   \fi
6659   \def\forest@externalize@loadimages{}%
6660   \def\forest@externalize@checkimages{}%
6661 }
6662 \newcount\forest@externalize@max@outer@n
6663 \global\forest@externalize@max@outer@n=0
6664 \newcount\forest@externalize@inner@n

```

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

```

6665 \long\def\forest@external#1#2#3#4{% #1=n,#2=context+source code,#3=update check code, #4=load code
6666   \ifnum\forest@externalize@max@outer@n<#1
6667     \global\forest@externalize@max@outer@n=#1
6668   \fi
6669   \global\csdef{forest@externalize@@\detokenize{#2}}{#1}%
6670   \global\csdef{forest@externalcheck@#1}{#3}%
6671   \global\csdef{forest@externalload@#1}{#4}%
6672   \tikzifexternalizing{\{%
6673     \immediate\write\forest@auxout{%
6674       \noexpand\forest@external{#1}%
6675       {\expandafter\detokenize\expandafter{#2}}%

```

```

6676      {\unexpanded{#3}}%
6677      {\unexpanded{#4}}%
6678    }%
6679  }%
6680 }

```

```

6681 \def\forest@includeexternal#1{%
6682   \edef\forest@temp{\pgfkeysvalueof{/forest/external/context}}%
6683   %\typeout{forest: Including external picture '#1' for forest context+code: '\expandafter\detokenize\expandafter\includegraphics[width=\forest@temp]{#1}}%
6684   {%
6685     %\def\pgf@declaredraftimage##1##2{\def\pgf@image{\hbox{}}}%
6686     \tikzsetnextfilename{#1}%
6687     \tikzexternalenable
6688     \tikz{}%
6689   }%
6690 }
6691 \def\forest@includeexternal@box#1#2{%
6692   \global\setbox#1=\hbox{\forest@includeexternal{#2}}%
6693 }

```

```

6694 \long\def\forest@begin#1{%
6695   \iffalse{\fi\forest@parsebracket#1}%
6696 }
6697 \def\forest@parsebracket{%
6698   \bracketParse{\forest@get@root@afterthought}\forest@root=%
6699 }
6700 \def\forest@get@root@afterthought{%
6701   \expandafter\forest@get@root@afterthought\expandafter{\iffalse}\fi
6702 }
6703 \long\def\forest@get@root@afterthought#1{%
6704   \ifblank{#1}{\}{%
6705     \forest@oappto{\forest@root}{given options}{,afterthought={\unexpanded{#1}}}%
6706   }%
6707   \forest@do
6708 }
6709 \def\forest@do{%
6710   \forest@node@Compute@numeric@ts@info{\forest@root}%
6711   \expandafter\forestset\expandafter{\forest@stages}%
6712   \ifforest@was@tikzexternalwasenable
6713     \tikzexternalenable
6714   \fi
6715 }

```

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

```

6728     \foresttolet{previous@environment}\forest@environment
6729     \forest@node@typeset
6730     \foresttoget{calibration@procedure}\forest@temp
6731     \expandafter\forestset\expandafter{\forest@temp}%
6732   \fi
6733 }%
6734 }

```

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.)

```

6735 \def\forestStandardNode[#1]#2#3#4{%
6736   \let\forest@standardnode@restoretikzexternal\relax
6737   \ifdefined\tikzexternaldisable
6738     \ifx\tikz\tikzexternal\tikz@replacement
6739       \tikzexternaldisable
6740       \let\forest@standardnode@restoretikzexternal\tikzexternalenable
6741     \fi
6742   \fi
6743   \forest@standardnode@new
6744   \forest@fornode{\forest@node@Nametoid{standard node}}{%
6745     \forestset{content=#1}%
6746     \forestset{environment@formula}{#2}%
6747     \edef\forest@temp{\unexpanded{#3}}%
6748     \foresttolet{calibration@procedure}\forest@temp
6749     \def\forest@calibration@initializing@code{%
6750       \pgfkeys{/forest/initializing@code}{#4}%
6751     \foresttolet{initializing@code}\forest@calibration@initializing@code
6752     \forest@standardnode@restoretikzexternal
6753   }
6754 }
6755 \forestset{initializing@code/.unknown/.code={%
6756   \eappto\forest@calibration@initializing@code{%
6757     \noexpand\forestOget{\forest@node@Nametoid{standard node}}{\pgfkeyscurrentname}\noexpand\forest@temp
6758     \noexpand\foresttolet{\pgfkeyscurrentname}\noexpand\forest@temp
6759   }%
6760 }
6761 }

```

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

```

6762 \def\forest@initializefromstandardnode{%
6763   \forestOve{\forest@node@Nametoid{standard node}}{initializing@code}%
6764 }

```

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`.)

```

6765 \forestStandardNode[dj]
6766 {%
6767   \forestOve{\forest@node@Nametoid{standard node}}{content},%
6768   \the\ht\strutbox,\the\pgflinewidth,%

```

```

6769 \pgfkeysvalueof{/pgf/inner ysep},\pgfkeysvalueof{/pgf/outer ysep},%
6770 \pgfkeysvalueof{/pgf/inner xsep},\pgfkeysvalueof{/pgf/outer xsep}%
6771 }
6772 {
6773 l sep={\the\ht\strutbox+\pgfkeysvalueof{/pgf/inner ysep}},
6774 l={l_sep()+abs(max_y()-min_y())+2*\pgfkeysvalueof{/pgf/outer ysep}},
6775 s sep={2*\pgfkeysvalueof{/pgf/inner xsep}}
6776 }
6777 {l sep,l,s sep}

```

8.4 ls coordinate system

```

6778 \pgfkeys{/forest/@cs}{%
6779 name/.code={%
6780 \edef\forest@cn{\forest@node@Nametoid{#1}}%
6781 \forest@forestcs@resetxy},
6782 id/.code={%
6783 \edef\forest@cn{#1}%
6784 \forest@forestcs@resetxy},
6785 go/.code={%
6786 \forest@go{#1}%
6787 \forest@forestcs@resetxy},
6788 anchor/.code={\forest@forestcs@anchor{#1}},
6789 l/.code={%
6790 \pgfmathsetlengthmacro\forest@forestcs@l{#1}%
6791 \forest@forestcs@ls
6792 },
6793 s/.code={%
6794 \pgfmathsetlengthmacro\forest@forestcs@s{#1}%
6795 \forest@forestcs@ls
6796 },
6797 .unknown/.code={%
6798 \expandafter\pgfutil@in@\expandafter.\expandafter{\pgfkeyscurrentname}%
6799 \ifpgfutil@in@
6800 \expandafter\forest@forestcs@namegoanchor\pgfkeyscurrentname\forest@end
6801 \else
6802 \expandafter\forest@nameandgo\expandafter{\pgfkeyscurrentname}%
6803 \forest@forestcs@resetxy
6804 \fi
6805 }
6806 }
6807 \def\forest@forestcs@resetxy{%
6808 \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
6809 \global\pgf@x\forestove{x}%
6810 \global\pgf@y\forestove{y}%
6811 }
6812 \def\forest@forestcs@ls{%
6813 \ifdefined\forest@forestcs@l
6814 \ifdefined\forest@forestcs@s
6815 {%
6816 \pgftransformreset
6817 \pgftransformrotate{\forestove{grow}}%
6818 \pgfpointransformed{\pgfpoint{\forest@forestcs@l}{\forest@forestcs@s}}%
6819 }%
6820 \global\advance\pgf@x\forestove{x}%
6821 \global\advance\pgf@y\forestove{y}%
6822 \fi
6823 \fi
6824 }
6825 \def\forest@forestcs@anchor#1{%

```

```

6826 \edef\forest@marshal{%
6827   \noexpand\forest@original@tikz@parse@node\relax
6828   (\forestove{name}\ifx\relax#1\relax\else.\fi#1)%
6829 }\forest@marshal
6830 }
6831 \def\forest@forestcs@namegoanchor#1.#2\forest@end{%
6832   \forest@nameandgo{#1}%
6833   \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
6834   \forest@forestcs@anchor{#2}%
6835 }
6836 \def\forest@cs@invalidnodeerror{%
6837   \PackageError{forest}{Attempt to refer to the invalid node by "forest cs"}%
6838 }
6839 \tikzdeclarecoordinatesystem{forest}{%
6840   \forest@forthis{%
6841     \forest@forestcs@resetxy
6842     \ifdefined\forest@forestcs@l\undef\forest@forestcs@l\fi
6843     \ifdefined\forest@forestcs@s\undef\forest@forestcs@s\fi
6844     \pgfqkeys{/forest/@cs}{#1}%
6845   }%
6846 }

```

8.5 Relative node names in TikZ

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

```

6847 \def\forest@tikz@parse@node#1(#2){%
6848   \pgfutil@in@.#2}%
6849   \ifpgfutil@in@
6850     \expandafter\forest@tikz@parse@node@checkiftikzname@withdot
6851   \else%
6852     \expandafter\forest@tikz@parse@node@checkiftikzname@withoutdot
6853   \fi%
6854   #1(#2)\forest@end
6855 }
6856 \def\forest@tikz@parse@node@checkiftikzname@withdot#1(#2.#3)\forest@end{%
6857   \forest@tikz@parse@node@checkiftikzname#1{#2}{.#3}}
6858 \def\forest@tikz@parse@node@checkiftikzname@withoutdot#1(#2)\forest@end{%
6859   \forest@tikz@parse@node@checkiftikzname#1{#2}{}}
6860 \def\forest@tikz@parse@node@checkiftikzname#1#2#3{%
6861   \expandafter\ifx\csname pgf@sh@ns@#2\endcsname\relax
6862     \forest@forthis{%
6863       \forest@nameandgo{#2}%
6864       \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
6865       \edef\forest@temp@relativenodename{\forestove{name}}%
6866     }%
6867   \else
6868     \def\forest@temp@relativenodename{#2}%
6869   \fi
6870   \expandafter\forest@original@tikz@parse@node\expandafter#1\expandafter(\forest@temp@relativenodename#3)%
6871 }
6872 \def\forest@nameandgo#1{%
6873   \pgfutil@in@!{#1}%
6874   \ifpgfutil@in@
6875     \forest@nameandgo@(#1)%
6876   \else
6877     \ifstrempy{#1}{-}{\edef\forest@cn{\forest@node@Nametoid{#1}}}%
6878   \fi
6879 }
6880 \def\forest@nameandgo@(#1!#2){%
6881   \ifstrempy{#1}{-}{\edef\forest@cn{\forest@node@Nametoid{#1}}}%
6882   \forest@go{#2}%

```

6883 }

8.6 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.

```
6884 \forestset{
6885   declare count={rotate}{0},
6886   autoforward'={rotate}{node options},
6887 }
```

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

```
6888 \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.

```
6889 \newif\ifforest@anchor@forwardtotikz
```

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

```
6890 \newif\ifforest@anchor@isborder
```

The UI macro.

```
6891 \def\forestanchortotikzanchor#1#2{% #1 = forest anchor, #2 = macro to receive the tikz anchor
6892   \forest@anchor@forwardtotikzfalse
6893   \forest@anchor@do{#1}{\forest@cn}%
6894   \let#2\forest@temp@anchor
6895 }
```

Generic anchors.

```
6896 \pgfdeclaregenericanchor{child anchor}{%
6897   \forest@anchor@forwardtotikztrue
6898   \forest@anchor@do{#1}{child anchor}{\forest@referencednodeid}%
6899 }
6900 \pgfdeclaregenericanchor{parent anchor}{%
6901   \forest@anchor@forwardtotikztrue
6902   \forest@anchor@do{#1}{parent anchor}{\forest@referencednodeid}%
6903 }
6904 \pgfdeclaregenericanchor{anchor}{%
6905   \forest@anchor@forwardtotikztrue
6906   \forest@anchor@do{#1}{anchor}{\forest@referencednodeid}%
6907 }
6908 \pgfdeclaregenericanchor{children}{%
6909   \forest@anchor@forwardtotikztrue
6910   \forest@anchor@do{#1}{children}{\forest@referencednodeid}%
6911 }
6912 \pgfdeclaregenericanchor{children first}{%
6913   \forest@anchor@forwardtotikztrue
6914   \forest@anchor@do{#1}{children first}{\forest@referencednodeid}%
6915 }
6916 \pgfdeclaregenericanchor{first}{%
6917   \forest@anchor@forwardtotikztrue
6918   \forest@anchor@do{#1}{first}{\forest@referencednodeid}%
6919 }
6920 \pgfdeclaregenericanchor{parent first}{%
6921   \forest@anchor@forwardtotikztrue
6922   \forest@anchor@do{#1}{parent first}{\forest@referencednodeid}%
6923 }
6924 \pgfdeclaregenericanchor{parent}{%
6925   \forest@anchor@forwardtotikztrue
```

```

6926 \forest@anchor@do{#1}{parent}{\forest@referencednodeid}%
6927 }
6928 \pgfdeclaregenericanchor{parent last}{%
6929 \forest@anchor@forwardtotikztrue
6930 \forest@anchor@do{#1}{parent last}{\forest@referencednodeid}%
6931 }
6932 \pgfdeclaregenericanchor{last}{%
6933 \forest@anchor@forwardtotikztrue
6934 \forest@anchor@do{#1}{last}{\forest@referencednodeid}%
6935 }
6936 \pgfdeclaregenericanchor{children last}{%
6937 \forest@anchor@forwardtotikztrue
6938 \forest@anchor@do{#1}{children last}{\forest@referencednodeid}%
6939 }
6940 \pgfdeclaregenericanchor{children'}{%
6941 \forest@anchor@forwardtotikztrue
6942 \forest@anchor@do{#1}{children'}{\forest@referencednodeid}%
6943 }
6944 \pgfdeclaregenericanchor{children first'}{%
6945 \forest@anchor@forwardtotikztrue
6946 \forest@anchor@do{#1}{children first'}{\forest@referencednodeid}%
6947 }
6948 \pgfdeclaregenericanchor{first'}{%
6949 \forest@anchor@forwardtotikztrue
6950 \forest@anchor@do{#1}{first'}{\forest@referencednodeid}%
6951 }
6952 \pgfdeclaregenericanchor{parent first'}{%
6953 \forest@anchor@forwardtotikztrue
6954 \forest@anchor@do{#1}{parent first'}{\forest@referencednodeid}%
6955 }
6956 \pgfdeclaregenericanchor{parent'}{%
6957 \forest@anchor@forwardtotikztrue
6958 \forest@anchor@do{#1}{parent'}{\forest@referencednodeid}%
6959 }
6960 \pgfdeclaregenericanchor{parent last'}{%
6961 \forest@anchor@forwardtotikztrue
6962 \forest@anchor@do{#1}{parent last'}{\forest@referencednodeid}%
6963 }
6964 \pgfdeclaregenericanchor{last'}{%
6965 \forest@anchor@forwardtotikztrue
6966 \forest@anchor@do{#1}{last'}{\forest@referencednodeid}%
6967 }
6968 \pgfdeclaregenericanchor{children last'}{%
6969 \forest@anchor@forwardtotikztrue
6970 \forest@anchor@do{#1}{children last'}{\forest@referencednodeid}%
6971 }

```

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

```

6972 \def\forest@anchor@do#1#2#3{% #1 = shape name, #2 = (potentially) forest anchor, #3 = node id
6973 \forest@for node{#3}{%
6974 \def\forest@temp@anchor{#2}%
6975 \forest@anchor@snapbordertocompassfalse
6976 \forest@anchor@isborderfalse
6977 \forest@anchor@to@tikz@anchor
6978 \forest@anchor@border@to@compass
6979 \ifforest@anchor@forwardtotikz
6980 \forest@anchor@forward{#1}%
6981 \else
6982 \fi
6983 }%
6984 }

```


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

```

6985 \def\forest@anchor@to@tikz@anchor{%
6986   \ifcsdef{forest@anchor@@\forest@temp@anchor}{%
6987     \csuse{forest@anchor@@\forest@temp@anchor}%
6988     \forest@anchor@to@tikz@anchor
6989   }{}%
6990 }

Actual computation.

6991 \csdef{forest@anchor@@parent anchor}{%
6992   \foresttoget{parent anchor}\forest@temp@anchor}
6993 \csdef{forest@anchor@@child anchor}{%
6994   \foresttoget{child anchor}\forest@temp@anchor}
6995 \csdef{forest@anchor@@anchor}{%
6996   \foresttoget{anchor}\forest@temp@anchor}
6997 \csdef{forest@anchor@@children'}{%
6998   \forest@anchor@isbordertrue
6999   \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}}%
7000 }
7001 \csdef{forest@anchor@@parent'}{%
7002   \forest@anchor@isbordertrue
7003   \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}+180}%
7004 }
7005 \csdef{forest@anchor@@first'}{%
7006   \forest@anchor@isbordertrue
7007   \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=0 -\e
7008 }
7009 \csdef{forest@anchor@@last'}{%
7010   \forest@anchor@isbordertrue
7011   \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=0 +\e
7012 }
7013 \csdef{forest@anchor@@parent first'}{%
7014   \forest@anchor@isbordertrue
7015   \edef\forest@temp@anchor@parent{\number\numexpr\forestove{grow}-\forestove{rotate}+180}%
7016   \edef\forest@temp@anchor@first{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=
7017   \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@first}\forest@temp@anchor
7018 }
7019 \csdef{forest@anchor@@parent last'}{%
7020   \forest@anchor@isbordertrue
7021   \edef\forest@temp@anchor@parent{\number\numexpr\forestove{grow}-\forestove{rotate}+180}%
7022   \edef\forest@temp@anchor@last{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=
7023   \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@last}\forest@temp@anchor
7024 }
7025 \csdef{forest@anchor@@children first'}{%
7026   \forest@anchor@isbordertrue
7027   \edef\forest@temp@anchor@first{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=
7028   \forest@getaverageangle{\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@first}\forest@temp@anchor
7029 }
7030 \csdef{forest@anchor@@children last'}{%
7031   \forest@anchor@isbordertrue
7032   \edef\forest@temp@anchor@last{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=
7033   \forest@getaverageangle{\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@last}\forest@temp@anchor
7034 }
7035 \csdef{forest@anchor@@children}{%
7036   \forest@anchor@snapbordertocompasstrue
7037   \csuse{forest@anchor@@children'}%
7038 }
7039 \csdef{forest@anchor@@parent}{%
7040   \forest@anchor@snapbordertocompasstrue
7041   \csuse{forest@anchor@@parent'}%
7042 }

```

```

7043 \csdef{forest@anchor@@first}{%
7044   \forest@anchor@snapbordertocompasstrue
7045   \csuse{forest@anchor@@first'}%
7046 }
7047 \csdef{forest@anchor@@last}{%
7048   \forest@anchor@snapbordertocompasstrue
7049   \csuse{forest@anchor@@last'}%
7050 }
7051 \csdef{forest@anchor@@parent first}{%
7052   \forest@anchor@snapbordertocompasstrue
7053   \csuse{forest@anchor@@parent first'}%
7054 }
7055 \csdef{forest@anchor@@parent last}{%
7056   \forest@anchor@snapbordertocompasstrue
7057   \csuse{forest@anchor@@parent last'}%
7058 }
7059 \csdef{forest@anchor@@children first}{%
7060   \forest@anchor@snapbordertocompasstrue
7061   \csuse{forest@anchor@@children first'}%
7062 }
7063 \csdef{forest@anchor@@children last}{%
7064   \forest@anchor@snapbordertocompasstrue
7065   \csuse{forest@anchor@@children last'}%
7066 }

```

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>.

```

7067 \def\forest@getaverageangle#1#2#3{%
7068   \edef\forest@temp{\number\numexpr #1-#2+540}%
7069   \pgfmathMod{\forest@temp}{360}\pgfmathtruncatemacro\pgfmathresult{\pgfmathresult}
7070   \edef\forest@temp{360+#2+((\pgfmathresult-180)/2)}%
7071   \pgfmathMod{\forest@temp}{360}\pgfmathtruncatemacro#3{\pgfmathresult}%
7072 }

```

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

```

7073 \def\forest@anchor@border@to@compass{%
7074   \ifforest@anchor@isborder
7075     \ifforest@anchor@snapbordertocompass
7076       \forest@anchor@snap@border@to@compass
7077     \else
7078       \pgfmathMod{\forest@temp@anchor}{360}%
7079       \pgfmathtruncatemacro\forest@temp@anchor{\pgfmathresult}%
7080     \fi
7081     \ifforest@anchor@forwardtotikz
7082       \ifcsname pgf@anchor%
7083         @\csname pgf@sh@ns@\pgfreferencednodename\endcsname
7084         @\csname forest@compass@\forest@temp@anchor\endcsname
7085         \endcsname
7086       \letcs\forest@temp@anchor{forest@compass@\forest@temp@anchor}%
7087     \fi
7088   \else
7089     \ifforest@anchor@snapbordertocompass
7090       \letcs\forest@temp@anchor{forest@compass@\forest@temp@anchor}%
7091     \fi
7092   \fi
7093 }
7094 }
7095 \csdef{forest@compass@0}{east}
7096 \csdef{forest@compass@45}{north east}

```

```

7097 \csdef{forest@compass@90}{north}
7098 \csdef{forest@compass@135}{north west}
7099 \csdef{forest@compass@180}{west}
7100 \csdef{forest@compass@225}{south west}
7101 \csdef{forest@compass@270}{south}
7102 \csdef{forest@compass@315}{south east}
7103 \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).

```

7104 \def\forest@anchor@snap@border@to@compass{%
7105   \pgfmathMod{\forest@temp@anchor}{360}%
7106   \pgfmathdivide{\pgfmathresult}{45}%
7107   \pgfmathround{\pgfmathresult}%
7108   \pgfmathmultiply{\pgfmathresult}{45}%
7109   \pgfmathtruncatemacro\forest@temp@anchor{\pgfmathresult}%
7110 }

```

This macro forwards the resulting anchor to TikZ.

```

7111 \def\forest@anchor@forward#1{% #1 = shape name
7112   \ifdefempty\forest@temp@anchor{%
7113     \pgf@sh@reanchor{#1}{center}%
7114     \xdef\forest@hack@tikzshapeborder{%
7115       \noexpand\tikz@shapebordertrue
7116       \def\noexpand\tikz@shapeborder@name{\pgfreferencednodename}%
7117     }\aftergroup\forest@hack@tikzshapeborder
7118   }{%
7119     \pgf@sh@reanchor{#1}{\forest@temp@anchor}%
7120   }%
7121 }

```

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

```

7122 \def\forest@referencednodeid{\forest@node@Nametoid{\forest@referencednodename}}%
7123 \def\forest@referencednodename{%
7124   \expandafter\expandafter\expandafter\forest@referencednodename@\expandafter\pgfreferencednodename\forest@pgf@notyetpositioned
7125 }
7126 \expandafter\def\expandafter\forest@referencednodename@\expandafter#\expandafter1\forest@pgf@notyetpositioned
7127   \if\relax#1\relax\forest@referencednodename@stripafter#2\relax\fi
7128   \if\relax#2\relax#1\fi
7129 }
7130 \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.

```

7131 \def\forest@Pointanchor#1#2{% #1 = node id, #2 = anchor
7132   {%
7133     \def\forest@pa@temp@name{name}%
7134     \forest0ifdefined{#1}{@box}{%
7135       \forest0get{#1}{@box}\forest@temp
7136       \ifdefempty\forest@temp{%
7137         \def\forest@pa@temp@name{later@name}%
7138       }%
7139     }{%
7140       \setbox0\hbox{%
7141         \begin{pgfpicture}%
7142           \if\relax\forest0ve{#1}{#2}\relax
7143             \pgfpointanchor{\forest0ve{#1}{\forest@pa@temp@name}}{center}%
7144           \else
7145             \pgfpointanchor{\forest0ve{#1}{\forest@pa@temp@name}}{\forest0ve{#1}{#2}}%
7146           \fi
7147         \xdef\forest@global@marshal{%

```

```

7148         \noexpand\global\noexpand\pgf@x=\the\pgf@x\relax
7149         \noexpand\global\noexpand\pgf@y=\the\pgf@y
7150     }%
7151     \end{pgfpicture}%
7152 }%
7153 }%
7154 \forest@global@marshal
7155 }
7156 \def\forest@pointanchor#1{% #1 = anchor
7157     \forest@Pointanchor{\forest@cn}{#1}%
7158 }

```

9 Compatibility with previous versions

```

7159 \ifdefempty{\forest@compat}{%
7160     \RequirePackage{forest-compat}
7161 }

```